Quantifying Bounds in Strategy Logic[*]

Nathanaël Fijalkow
CNRS, LaBRI
Bordeaux, France
Alan Turing Institute of data science
London, United Kingdom
nfijalkow@turing.ac.uk

Bastien Maubert
University of Naples "Federico II"
Naples, Italy
bastien.maubert@gmail.com

Aniello Murano
University of Naples "Federico II"
Naples, Italy
murano@na.infn.it

Sasha Rubin
University of Naples "Federico II"
Naples, Italy
sasha.rubin@unina.it

Program synthesis automatically constructs programs from specifications. Strategy Logic is a powerful specification language whose goal is to give theoretical foundations for program synthesis in a multi-agent setting. One limitation of Strategy Logic is that it is purely qualitative. For instance it cannot specify quantitative properties of executions such as "every request is quickly granted", or quantitative properties of trees such as "most executions of the system terminate". In this work, we extend Strategy Logic to include quantitative aspects in a way that can express bounds on "how quickly" and "how many". We define Prompt Strategy Logic, which encompasses Prompt LTL (itself an extension of LTL with a prompt eventuality temporal operator), and we define Bounded-Outcome Strategy Logic which has a bounded quantifier on paths. We supply a general technique, based on the study of automata with counters, that solves the model-checking problems for both these logics.

1 Introduction

In order to reason about strategic aspects in distributed systems, temporal logics of programs (such as LTL [34], CTL [5] and CTL* [18]) have been extended with operators expressing the existence of strategies for coalitions of components. Among the most successful proposals are Alternating-time Temporal Logic (ATL) [3] and, more recently, the more expressive Strategy Logic (SL) [12, 32]. Both logics can express the existence of strategies for coalitions that ensure some temporal properties against all possible behaviours of the remaining components. Moreover, if such strategies exist, one can also obtain witnessing finite-state strategies. As a result, synthesising reactive systems from temporal specifications [35, 27, 28] can be reduced to model checking such strategic logics.

Although quite expressive, for instance Strategy Logic can express important game-theoretic concepts such as the existence of Nash equilibria, such logics can only express qualitative properties. On the other hand important properties of distributed systems, such as bounding the maximal number of steps between an event and its reaction, are quantitative. Parametric extensions of temporal logics have been introduced to capture such properties.

A simple way to extend temporal operators is to annotate them with constant bounds, e.g., $\mathbf{F}^{\leq k}\varphi$ says that φ holds within k steps where $k \in \mathbb{N}$ is a constant. However, one may not know such bounds or care for their exact value when writing the specification (or it may not be practical to compute the bound). Instead, one may replace the constants by variables N and ask about the possible valuations of

[*]This work has recently been submitted to a conference, but is not yet published.

Submitted to:
SR 2018

the variables that make the formula true. For instance, PROMPT-LTL [2, 29] is an extension of LTL with the operator $\mathbf{F}^{\leq N}$ where N is a variable. The model-checking problem asks if there exists a valuation of the variable N such that the formula holds. In order to reason about and synthesise strategies that ensure such parametric properties, we introduce "Prompt Strategy Logic", an extension of SL with the $\mathbf{F}^{\leq N}$ operator. For instance, the formula $\exists s_1(a_1,s_1)\forall s_2(a_2,s_2)\exists N\mathbf{A}\mathbf{G}\mathbf{F}^{\leq N}p$ expresses that there exists a strategy for agent a_1 such that for all strategies of agent a_2 there is a bound N (that can depend on the strategy for a_2) such that in all outcomes the atom p holds at least once every N steps.

Another way to parameterise temporal logics is to bound the number of paths to express, for instance, how well a linear-time temporal property holds, thus giving a measure of "coverage". We introduce "Bounding-Outcome Strategy Logic" which extends Strategy Logic with a *bounded outcome quantifier* $\mathbf{A}^{\leq N}$ which allows one to express that all but N outcomes satisfy some property. For instance, the formula $\exists s(a,s)\exists N\mathbf{A}^{\leq N}\mathbf{G}\mathbf{F}p$ expresses that there exists a strategy for agent a such that for all but finitely many outcomes, the atom p holds infinitely often.

The algorithmic contribution of this paper is a solution to the model-checking problem for both these logics (and their combination), which resorts to the theory of regular cost functions. A cost function is an equivalence class of mappings from the domain (e.g., infinite words) to $\mathbb{N}\cup\{\infty\}$ with an equivalence relation that, intuitively speaking, forgets the precise values and focuses on boundedness [13, 15].

Our results allow us to solve a problem left open in [9] that considers games with two players and a third player called "nature" (indicating that it is uncontrollable), and asks whether there is a strategy for player 1 (having very general linear-time objectives) such that for all strategies of player 2, in the resulting tree (i.e., where nature's strategy is not fixed), the number of plays in which player 1 does not achieve her objective is "small". In particular, in case the linear-time objective is the LTL formula ψ and "small" is instantiated to mean "finite", our main result allows one to solve this problem by reducing to model checking Bounding-outcome Strategy Logic formula $\exists s_1(a_1,s_1)\forall s_2(a_2,s_2)\exists N\mathbf{A}^{\leq N}\neg\psi$. In fact our automata construction can be adapted to deal with all omega-regular objectives.

Related work. Parametric-LTL [2] extends LTL with operators of the form $\mathbf{F}^{\leq x}$ and $\mathbf{G}^{\leq x}$, where x is a variable. The interpretation of $\mathbf{F}^{\leq x}\psi$ is that ψ holds within x steps, and the interpretation of $\mathbf{G}^{\leq x}$ is that ψ holds for at least the next x steps. That paper studies variations on the classic decision problems, e.g., model checking asks to decide if there is a valuation of the variables $x_1,\cdots,x_k$ such that the formula $\varphi(x_1,\cdots,x_k)$ holds in the given structure. Note that for this problem, the formula is equivalent to one in which all variables are replaced by a single variable.

Parametric-LTL has been studied in the context of open systems and games. For instance, [36] studies the problem of synthesising a strategy for an agent with a parametric-LTL objective in a turn-based graph-game against an adversarial environment. A number of variations are studied, e.g., decide whether there exists a valuation (resp. for all valuations) of the variables such that there exists a strategy for the agent that enforces the given parametric-LTL formula.

Promptness in multi-agent systems was first studied in [4], which introduces PROMPT-ATL* and studies its model-checking problem. We remark that the formula of Prompt Strategy Logic mentioned above is not a formula of PROMPT-ATL* because the bound N can depend on the strategy of agent a_2, which is not possible in PROMPT-ATL*.

Promptness has also been studied in relation with classic infinitary winning conditions in games on graphs. In bounded parity games, even colours represent requests, odd colours represent grants, and the objective of the player is to ensure that every request is promptly followed by a larger grant [11, 33] (see also Example 1). Such winning conditions have been generalised to games with costs in [20, 21], leading to the construction of efficient algorithms for synthesising controllers with prompt specifications.

Promptness in automata can be studied using various notions of automata with counters that only affect the acceptance condition. For instance, a run in a prompt Büchi-automaton is successful if there is a bound on the time between visits to the Büchi set. The expressive power, the cost of translating between such automata, and decision problems such as containment have been studied in [1, 11].

The theory of regular cost functions [13, 15] defines automata and logics able to express boundedness properties in various settings. For instance, the logics PROMPT-LTL, PLTL and kTL are in some precise sense subsumed by the $\mathsf{LTL}^{\leq}$ logic from [25], which extends LTL with a bounded until $\varphi \mathbf{U}^{\leq N} \varphi'$ allowing φ not to hold in at most N (possibly nonconsecutive) places before φ' holds. A decision procedure for this logic has been given through the compilation into cost automata on words. In this paper, we rely on several results from the theory of regular cost functions, and develop some new ones for the study of Prompt Strategy Logic and Bounding-outcome Strategy Logic. A major open problem in the theory of regular cost functions over infinite trees is the equivalence between general cost automata. To handle the bounded until operator in branching-time logics one would need to first prove this equivalence, which has been proved to be beyond our reach today [19]. In this work we rely on a weaker version of this equivalence for distance automata.

To the best of our knowledge, the only previous works on quantitative extensions of Strategy Logic consider games with counters and allow for the expression of constraints on their values in formulas. The model-checking problem for these logics is undecidable, even when restricted to the case of *energy constraints*, which can only state that the counters remain above certain thresholds [22]. For the Boolean Goal fragment of Strategy Logic in the case of one counter, the problem is still open [8, 22]. The present work thus provides the first decidable quantitative extension of Strategy Logic.

Plan. In Section 2 we recall Branching-time Strategy Logic. We introduce and motivate our two quantitative extensions, PROMPT-SL and BOSL, in Section 3 and Section 4 respectively. In Section 5 we solve their model-checking problem by introducing the intermediary logic BOUND-QCTL* and developing an automata construction based on automata with counters.

2 Branching-time Strategy Logic

We first recall Branching-time Strategy Logic [24], a variant of Strategy Logic [32]. We fix a number of parameters: AP is a finite set of *atomic propositions*, Ag is a finite set of *agents* or *players*, Act is a finite set of *actions*, and Var is a finite set of *strategy variables*. The alphabet is $\Sigma = 2^{\mathrm{AP}}$.

Notations. A *finite* (resp. *infinite*) *word* over Σ is an element of Σ^* (resp. Σ^ω). The *length* of a finite word $w = w_0 w_1 \dots w_n$ is $|w| = n+1$, and $\mathrm{last}(w) = w_n$ is its last letter. Given a finite (resp. infinite) word w and $0 \leq i < |w|$ (resp. $i \in \mathbb{N}$), we let w_i be the letter at position i in w, $w_{\leq i}$ is the prefix of w that ends at position i and $w_{\geq i}$ is the suffix of w that starts at position i. We write $w \preccurlyeq w'$ if w is a prefix of w'. The cardinal of a set S is written $\mathrm{Card}(S)$.

2.1 Games

We start with classic notions related to concurrent games on graphs.

Definition 1 (Game) *A* concurrent game structure *(or* game *for short) is a structure $G = (V, v_0, \Delta, \ell)$ where V is the set of* vertices*, $v_0 \in V$ is the* initial vertex*, $\Delta : V \times Act^{Ag} \to V$ is the* transition function*, and $\ell : V \to \Sigma$ is the* labelling function.

Joint actions. In a vertex $v \in V$, each player $a \in \mathrm{Ag}$ chooses an action $c(a) \in \mathrm{Act}$, and the game proceeds to the vertex $\Delta(v, \mathbf{c})$, where $\mathbf{c} \in \mathrm{Act}^{\mathrm{Ag}}$ stands for the *joint action* $(c(a))_{a \in \mathrm{Ag}}$. Given a joint action $\mathbf{c} = (c(a))_{a \in \mathrm{Ag}}$ and $a \in \mathrm{Ag}$, we let $\mathbf{c}(a)$ denote $c(a)$.
Plays and strategies. A *finite* (resp. *infinite*) *play* is a finite (resp. infinite) word $\rho = v_0 \dots v_n$ (resp. $\pi = v_0 v_1 \dots$) such that for every i such that $0 \leq i < |\rho| - 1$ (resp. $i \geq 0$), there exists a joint action $\mathbf{c}$ such that $\Delta(v_i, \mathbf{c}) = v_{i+1}$. A *strategy* is a partial function $\sigma : V^+ \rightharpoonup \mathrm{Act}$ mapping each finite play to an action, and Strat is the set of all strategies.
Assignments. An *assignment* is a partial function $\chi : \mathrm{Ag} \cup \mathrm{Var} \rightharpoonup \mathrm{Strat}$, assigning to each player and variable in its domain a strategy. For an assignment χ, a player a and a strategy σ, $\chi[a \mapsto \sigma]$ is the assignment of domain $dom(\chi) \cup \{a\}$ that maps a to σ and is equal to χ on the rest of its domain, and $\chi[s \mapsto \sigma]$ is defined similarly, where s is a variable; also, $\chi[a \mapsto ?]$ is the assignment of domain $dom(\chi) \setminus \{a\}$, on which it is equal to χ.
Outcomes. For an assignment χ and a finite play ρ, we let $\mathrm{Out}(\chi, \rho)$ be the set of infinite plays that start with ρ and are then extended by letting players follow the strategies assigned by χ. Formally, $\mathrm{Out}(\chi, \rho)$ is the set of plays of the form $\rho \cdot v_1 v_2 \dots$ such that for all $i \geq 0$, there exists $\mathbf{c}$ such that for all $a \in dom(\chi) \cap \mathrm{Ag}$, $\mathbf{c}_a \in \chi(a)(\rho \cdot v_1 \dots v_i)$ and $v_{i+1} = \Delta(v_i, \mathbf{c})$, with $v_0 = \mathrm{last}(\rho)$.

2.2 BSL syntax

The core of Branching-time Strategy Logic, on which we build Prompt Strategy Logic and Bounding-outcome Strategy Logic, is the full branching-time temporal logic CTL*. This differs from usual variants of Strategy Logic which are based on the linear-time temporal logic LTL. The main difference is the introduction of an *outcome quantifier* which quantifies on outcomes of the currently fixed strategies. While in SL temporal operators could only be evaluated in contexts where all agents were assigned a strategy, this outcome quantifier allows for evaluation of (branching-time) temporal properties on partial assignments of strategies to agents. We recall Branching-time Strategy Logic, introduced in [24], which has the same expressive power as SL but allows to express branching-time properties without resorting to computationally expensive strategy quantifications.

In addition to usual boolean connectives and temporal operators, we have four syntactic constructs:

- strategy quantification: $\exists s \varphi$, which means "there exists a strategy s such that φ holds",
- assigning a strategy to a player: $(a, s)\varphi$, which is interpreted as "when the agent a plays according to s, φ holds",
- unbinding a player: $(a, ?)\varphi$, which is interpreted as "φ holds after agent a has been unbound from her strategy, if any", and
- quantifying over outcomes: $\mathbf{A}\psi$, which reads as "ψ holds in all outcomes of the strategies currently assigned to agents".

The difference between BSL and SL lies in the last two constructs. Note that unbinding agents was irrelevant in linear-time SL, where assignments need to be total to evaluate temporal properties.

Definition 2 (BSL **syntax**) *The set of* BSL *formulas is the set of state formulas given by the grammar:*

$$\begin{array}{ll} \textit{State formulas:} & \varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists s \varphi \mid (a, s)\varphi \mid (a, ?)\varphi \mid \mathbf{A}\psi \\ \textit{Path formulas:} & \psi ::= \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathbf{X}\psi \mid \psi \mathbf{U} \psi, \end{array}$$

where $p \in AP, a \in Ag$ and $s \in Var$.

We use classic abbreviations $\top = p \vee \neg p$, $\mathbf{F}\psi = \top \mathbf{U} \psi$, $\mathbf{G}\psi = \neg \mathbf{F} \neg \psi$ and $\forall s \varphi = \neg \exists s \neg \varphi$.

2.3 BSL semantics

Given a formula $\varphi \in$ BSL, an assignment is *variable-complete for* φ if its domain contains all free strategy variables of φ.

Definition 3 (BSL **semantics)** *The semantics of a state formula is defined on a game G, an assignment χ variable-complete for φ, and a finite play ρ. For a path formula ψ, the finite play is replaced with an infinite play π and an index $i \in \mathbb{N}$. The definition is as follows (classic boolean cases are as follows):*

$$\begin{array}{lll}
G,\chi,\rho \models p & \textit{if} & p \in \ell(last(\rho)) \\
G,\chi,\rho \models \exists s\varphi & \textit{if} & \textit{there exists } \sigma \in Strat \textit{ s.t. } G,\chi[s \mapsto \sigma],\rho \models \varphi \\
G,\chi,\rho \models (a,s)\varphi & \textit{if} & G,\chi[a \mapsto \chi(s)],\rho \models \varphi \\
G,\chi,\rho \models (a,?)\varphi & \textit{if} & G,\chi[a \mapsto ?],\rho \models \varphi \\
G,\chi,\rho \models \mathbf{A}\psi & \textit{if} & \textit{for all } \pi \in \mathrm{Out}(\chi,\rho),\ G,\chi,\pi,|\rho|-1 \models \varphi \\
G,\chi,\pi,i \models \varphi & \textit{if} & G,\chi,\pi_{\leq i} \models \varphi \\
G,\chi,\pi,i \models \mathbf{X}\psi & \textit{if} & G,\chi,\pi,i+1 \models \psi \\
G,\chi,\pi,i \models \psi\mathbf{U}\psi' & \textit{if} & \exists j \geq i \textit{ s.t. } G,\chi,\pi,j \models \psi' \textit{ and } \forall k \textit{ s.t. } i \leq k < j,\ G,\chi,\pi,k \models \psi
\end{array}$$

BSL has the same expressivity as SL, and there are linear translations in both directions [24]. More precisely, the translation from BSL to SL is linear in the size of the formula times the number of players; indeed, the outcome quantifier is simulated in SL by a strategy quantification and a binding for each player who is not currently bound to a strategy. This translation may thus increase the nesting and alternation depth of strategy quantifiers in the formula, which is known to increase the complexity of the model-checking problem [12, 32].

3 Prompt Strategy Logic

In this section we introduce PROMPT-SL, an extension of both BSL and PROMPT-LTL.

3.1 PROMPT-SL syntax

The syntax of PROMPT-SL extends that of branching-time strategy logic BSL with two additional constructs, where N is a variable over natural numbers:

- a bounded version of the classical "eventually" operator written $\mathbf{F}^{\leq N}$, and
- an existential quantification on the values of variable N, written $\exists N$.

As in PROMPT-LTL, the formula $\mathbf{F}^{\leq N}\psi$ states that ψ will hold in at most N steps from the present. For a formula φ of PROMPT-SL there is a unique *bound variable* N: indeed, in the spirit of PROMPT-LTL where a unique bound must exist for all prompt-eventualities, formulas of our logic cannot use more than one bound variable. However, in PROMPT-SL, existential quantification on N is part of the syntax, which allows to freely combine quantification on the (unique) bound variable N with other operators of the logic. In particular one can express the existence of a unique bound that should work for all strategies, or instead that the bound may depend on the strategy (see Example 1).

Definition 4 (PROMPT-SL **syntax)** *The syntax of* PROMPT-SL *formulas is as follows:*

$$\begin{array}{ll}
\textit{State formulas:} & \varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists s\varphi \mid (a,s)\varphi \mid (a,?)\varphi \mid \mathbf{A}\psi \mid \exists N\varphi \\
\textit{Path formulas:} & \psi ::= \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathbf{X}\psi \mid \psi\mathbf{U}\psi \mid \mathbf{F}^{\leq N}\psi
\end{array}$$

where $p \in AP$, $s \in Var$, $a \in Ag$ and N is a fixed bounding variable. A PROMPT-SL sentence *is a state formula in which every $\mathbf{F}^{\leq N}$ is in the scope of some $\exists N$, and $\mathbf{F}^{\leq N}$ and $\exists N$ always appear positively, i.e. under an even number of negations.*

3.2 PROMPT-SL semantics

We now define the semantics of PROMPT-SL.

Definition 5 (PROMPT-SL **semantics)** *The semantics is defined inductively as follows, where φ (resp. ψ) is a* cost-SL *state (resp. path) formula, G is a game, χ is an assignment variable-complete for φ (resp. ψ), ρ is a finite play, π an infinite one, $i \in \mathbb{N}$ is a point in time and $n \in \mathbb{N}$ is a bound. We only give the definitions for the new operators, the others are as in Definition 3, with the bound n carrying over.*

$$\begin{array}{lll} G,\chi,\rho,n \models \exists N\varphi & \textit{if} & \textit{there exists } n' \in \mathbb{N} \textit{ such that } G,\chi,\rho,n' \models \varphi \\ G,\chi,\pi,i,n \models \mathbf{F}^{\leq N}\psi & \textit{if} & \textit{there exists } j \in [i,n] \textit{ such that } G,\chi,\pi,j,n \models \psi. \end{array}$$

Example 1 *In bounded parity games [11, 33] the odd colours represent requests and even colours represent grants, and the objective of the player a_1 is to ensure against player a_2 that every request is promptly followed by a larger grant. Solving such games can be cast as a model-checking problem of the* PROMPT-SL *formula*

$$\exists s_1(a_1,s_1)\forall s_2(a_2,s_2)\exists N\mathbf{AG}\left[\bigwedge_{c\ odd} c \to \mathbf{F}^{\leq N} \bigvee_{d>c\ even} d\right]$$

on the structure in which every vertex is labelled by its color. The finitary parity condition relaxes the constraint by only requiring requests that appear infinitely often to be promptly granted, and solving such games can be reduced to model-checking the PROMPT-SL *formula*

$$\exists s_1(a_1,s_1)\forall s_2(a_2,s_2)\exists N\mathbf{AG}\left[\bigwedge_{c\ odd} (c \wedge \mathbf{GF}c) \to \mathbf{F}^{\leq N} \bigvee_{d>c\ even} d\right].$$

Observe that in both definitions, the bound on the delay between requests and grants can depend on the opponent's strategy. We could express uniform *variants of these objectives by moving the quantification on the bound $\exists N$ before the quantification on opponent's strategies $\forall s_2$. Such games are studied in the context of the theory of regular cost functions [13, 15, 14], and their relationship to the non-uniform variants has been investigated in [10]. The solution to the model-checking problem for* PROMPT-SL *that we present here allows us to solve both types of games, uniform and non-uniform.*

4 Bounding-outcomes Strategy Logic

Bounding-outcomes Strategy Logic, or BOSL, is our second quantitative extension of Strategy Logic.

4.1 BOSL syntax

The syntax of BOSL extends that of strategy logic BSL with two additional constructs:

- a bounded version of the outcome quantifier written $\mathbf{A}^{\leq N}$,
- an existential quantification on the values of variable N, written $\exists N$.

BOSL can also be seen as PROMPT-SL without the bounded eventually $\mathbf{F}^{\leq N}$ but with the novel bounded outcome quantifier $\mathbf{A}^{\leq N}$. While formula $\mathbf{A}\psi$ states that ψ holds in all outcomes of the current assignment, $\mathbf{A}^{\leq N}\psi$ states that ψ holds in all of these outcomes *except for at most N of them.*

Definition 6 (BOSL **syntax)** *The syntax of* BOSL *formulas is given by the following grammar:*

$$\begin{array}{rl} \textit{State formulas:} & \varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists s\varphi \mid (a,s)\varphi \mid (a,?)\varphi \mid \mathbf{A}\psi \mid \mathbf{A}^{\leq N}\psi \mid \exists N\varphi \\ \textit{Path formulas:} & \psi ::= \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathbf{X}\psi \mid \psi\mathbf{U}\psi \end{array}$$

where $p \in AP$, $s \in Var$, $a \in Ag$ and N is a fixed bounding variable. A BOSL sentence *is a state-formula in which every $\mathbf{A}^{\leq N}$ is in the scope of some $\exists N$, and where $\mathbf{A}^{\leq N}$ and $\exists N$ always appear positively, i.e. under an even number of negations.*

4.2 BOSL semantics

Definition 7 (BOSL **semantics)** *We only give the definition for the new operator $\mathbf{A}^{\leq N}$, the others are as in Definition 5.*

$$G,\chi,\rho,n \models \mathbf{A}^{\leq N}\psi \quad \textit{if} \quad \mathrm{Card}(\{\pi \in \mathrm{Out}(\rho,\chi) : G,\chi,\pi,|\rho|-1,n \not\models \psi\}) \leq n$$

Example 2 *In [9] Carayol and Serre consider games with two players and a third player called "nature". The usual semantics is for nature to be a random player, in which case we are interested in the probability of satisfying a linear-time objective ψ. More specifically, one can write the following Strategy Logic formula extended with a probability operator $\exists s_1(a_1,s_1)\forall s_2(a_2,s_2)\mathbb{P}(\psi) = 1$, stating that player 1 has a strategy ensuring to win almost all paths. Paper [9] suggests other formalisations for the third player, of topological, measure-theoretic, and combinatorial nature, and provides general reductions. For instance, one may fix a constant N and write the formula $\exists s_1(a_1,s_1)\forall s_2(a_2,s_2)\mathbf{A}^{\leq N}\neg\psi$, stating that player a_1 has a strategy ensuring to win all but N paths. If N is a constant the above question is solved in [9]. However the latter work leaves open the question of ensuring that player a_1 wins all but a bounded number of paths, which is expressible with the Bounding-outcome Strategy Logic formula $\exists s_1(a_1,s_1)\forall s_2(a_2,s_2)\exists N\mathbf{A}^{\leq N}\neg\psi$. In this paper we show that the model-checking problem for Bounding-outcome Strategy Logic is decidable, thereby giving a solution to this question.*

5 Model checking

In this section we solve the model-checking problem for both PROMPT-SL and BOSL with a uniform approach which, in fact, works also for the combination of the two logics. As done in [31, 6, 7] for various strategic logics, we go through an adequate extension of QCTL*, itself an extension of CTL* with second-order quantification. This approach makes automata constructions and their proof of correctness easier and clearer. In our case we define an extension of QCTL* called BOUND-QCTL*, which contains the bounded eventually $\mathbf{F}^{\leq N}$ from PROMPT-LTL and PROMPT-SL, a bounded path quantifier $\mathbf{A}^{\leq N}$ similar to the bounded outcome quantifier from BOSL, and the quantifier on bounds $\exists N$ present in both PROMPT-SL and BOSL. We then recall definitions and results about cost automata, that we use to solve the model-checking problem for BOUND-QCTL*. We finally solve the model-checking problem for both PROMPT-SL and BOSL by reducing them to model checking BOUND-QCTL*.

5.1 Bound Quantified QCTL*

We define Bound Quantified CTL*, or BOUND-QCTL*, which extends PROMPT-LTL to the branching-time setting and adds quantification on atomic propositions. One can also see it as an extension of Quantified CTL* [31] with the bounded eventually operator and a bounded version of the universal path quantifier. Unlike PROMPT-LTL, but similarly to our PROMPT-SL and BOSL, a quantifier on the bound variable is also part of the syntax.

5.1.1 BOUND-QCTL* syntax

Definition 8 *The syntax of* BOUND-QCTL* *is defined by the following grammar:*

$$\varphi = p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{A}\psi \mid \mathbf{A}^{\leq N}\psi \mid \exists p\, \varphi \mid \exists N \varphi$$
$$\psi = \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathbf{X}\psi \mid \psi\mathbf{U}\psi \mid \mathbf{F}^{\leq N}\psi$$

where $p \in AP$, and N is a fixed bounding variable.

As usual, formulas of type φ are called *state formulas*, those of type ψ are called *path formulas*, and QCTL* consists of all the state formulas defined by the grammar. We further distinguish between *positive formulas*, in which operators $\mathbf{F}^{\leq N}$, $\mathbf{A}^{\leq N}$ and $\exists N$ appear only positively (under an even number of negations), and *negative formulas*, in which operators $\mathbf{F}^{\leq N}$, $\mathbf{A}^{\leq N}$ and $\exists N$ appear only negatively (under an odd number of negations). A BOUND-QCTL* *sentence* is a positive formula such that all operators $\mathbf{F}^{\leq N}$ and $\mathbf{A}^{\leq N}$ in the formula are in the scope of some $\exists N$.

5.1.2 BOUND-QCTL* semantics

BOUND-QCTL* formulas are evaluated on (unfoldings of) Kripke structures.

Definition 9 *A (finite)* Kripke structure *is a tuple $\mathcal{S} = (S, s_0, R, \ell)$, where S is a finite set of* states, *$s_0 \in S$ is an* initial state, *$R \subseteq S \times S$ is a left-total* transition relation[1], *and $\ell : S \to \Sigma$ is a* labelling function.

A path *in $\mathcal{S}$ is a finite word λ over S such that for all i, $(\lambda_i, \lambda_{i+1}) \in R$. For $s \in S$, we let $Paths(s) \subseteq S^+$ be the set of all paths that start in s.*

Trees. Let S be a finite set of *directions* and Σ a set of *labels*. A (Σ, S)-*tree* (or simply *tree*) is a pair $t = (\tau, \ell)$ where $\ell : \tau \to \Sigma$ is a *labelling* and $\tau \subseteq S^+$ is the *domain* such that:

- there exists $r \in S^+$, called the *root* of τ, such that each $u \in \tau$ starts with r, *i.e.* $r \preccurlyeq u$,
- if $u \cdot s \in \tau$ and $u \cdot s \neq r$, then $u \in \tau$,
- if $u \in \tau$ then there exists $s \in S$ such that $u \cdot s \in \tau$.

The elements of τ are called *nodes*. If $u \cdot s \in \tau$, we say that $u \cdot s$ is a *child* of u. A *branch* in t is an infinite sequence of nodes λ such that for all i, λ_{i+1} is a child of λ_i, and Branches(t, u) is the set of branches that start in node u. We let Branches(t) denote the set of branches that start in the root. If S is a singleton, a tree becomes an infinite word.

Definition 10 *The* tree unfolding of a Kripke structure *$\mathcal{S}$ from state s is the tree $t_{\mathcal{S}}(s) = (Paths(s), \ell')$ with $\ell'(u) = \ell(last(u))$. We may write $t_{\mathcal{S}}$ for $t_{\mathcal{S}}(s_0)$, the unfolding from the initial state.*

[1] *i.e.*, for all $s \in S$, there exists s' such that $(s, s') \in R$.

Projection, subtrees and regular trees. Given two trees t, t' and a proposition p, we write $t \equiv_p t'$ if they have same domain τ and for all p' in AP such that $p' \neq p$, for all u in τ, we have $p' \in \ell(u)$ if, and only if, $p' \in \ell'(u)$. Given a tree $t = (\tau, \ell)$ and a node $u \in \tau$, we define the *subtree of t rooted in u* as the tree $t_u = (\tau_u, \ell')$ where $\tau_u = \{v \in S^+ : u \preccurlyeq v\}$ and ℓ' is ℓ restricted to τ_u. A tree t is said *regular* if it is the unfolding of a finite Kripke structure.

Definition 11 *The semantics $t, u, n \models \varphi$ and $t, \lambda, n \models \psi$ are defined inductively, where φ is a* BOUND-QCTL* *state formula, ψ is a* BOUND-QCTL* *path formula, $t = (\tau, \ell)$ is a tree, u is a node, λ is a branch in t, and n in $\mathbb{N}$ a bound (we omit the inductive cases for classic* CTL* *operators):*

$$\begin{array}{lll} t,u,n \models \mathbf{A}^{\leq N}\psi & \textit{if} & \mathrm{Card}(\{\lambda \in \mathit{Branches}(t,u) : t,\lambda,n \not\models \psi\}) \leq n \\ t,u,n \models \exists p\,\varphi & \textit{if} & \exists t' \equiv_p t \textit{ such that } t',u,n \models \varphi \\ t,u,n \models \exists N\varphi & \textit{if} & \exists n' \in \mathbb{N} \textit{ such that } t,u,n' \models \varphi, \\ t,\lambda,n \models \mathbf{F}^{\leq N}\psi & \textit{if} & \exists j \textit{ such that } 0 \leq j \leq n \textit{ and } t,\lambda_{\geq j},n \models \psi \end{array}$$

The *value* $[\![\varphi]\!]_{\mathbf{inf}}(t)$ (resp. $[\![\varphi]\!]_{\mathbf{sup}}(t)$) of a positive (resp. negative) state formula φ on a tree t with root r is defined as $[\![\varphi]\!]_{\mathbf{inf}}(t) = \inf\{n \in \mathbb{N} : t,r,n \models \varphi\}$ and $[\![\varphi]\!]_{\mathbf{sup}}(t) = \sup\{n \in \mathbb{N} : t,r,n \models \varphi\}$, with the usual convention that $\inf \emptyset = \omega$ and $\sup \emptyset = 0$. In case it is not a positive or negative formula, its value is undefined. We remark that $\{n \in \mathbb{N} : t,r,n \models \varphi\}$ is upward (resp. downward) closed if φ is positive (resp. negative). The value of a sentence Φ is always either 0 or ω (sentences are positive formulas in which N is always quantified), and given a Kripke structure $\mathscr{S}$, we write $\mathscr{S} \models \Phi$ if $[\![\Phi]\!]_{\mathbf{inf}}(t_{\mathscr{S}}) = 0$.

5.2 Regular cost functions

In this section we develop the theory of regular cost functions over trees for distance automata. To this end we define and study the two dual models of **distance** and $\overline{\textbf{distance}}$-automata for recognising cost functions [13], referred to as cost automata.

Let E be a set of structures (such as infinite words or trees). We define an equivalence relation $\approx$ on functions $E \to \mathbb{N} \cup \{\infty\}$ by $f \approx g$ if for all $X \subseteq E$, $f(X)$ is bounded if, and only if, $g(X)$ is bounded. A *cost function* over E is an equivalence class of the relation $\approx$.

In Section 5.2.1 we define cost games whose objectives may refer to a single counter that, in each step, can be incremented or left unchanged. In Section 5.2.2 we define automata whose semantics are given using cost games. We introduce **distance**-automata and their duals $\overline{\textbf{distance}}$-automata that compute functions $E \to \mathbb{N} \cup \{\infty\}$. In Section 5.2.3 we focus on automata over infinite words and the notion of history-deterministic automata.

The novel technical contribution of this section is an extension of the classical property of history-deterministic automata: the original result says that given a history-deterministic automaton over infinite words, one can simulate it along every branch of a tree. This is the key argument to handle the **A** operator in PROMPT-SL. In Section 5.2.4 we extend this result by allowing the automaton to skip a bounded number of paths, which will allow us to capture the bounded-outcome operator $\mathbf{A}^{\leq N}$ in BOSL.

5.2.1 Cost games

The semantics of cost automata are given by turn-based two-player games, which are essentially a special case of the general notion of games given in Section 3.2. We give here a slightly modified definition better fitting the technical developments. The definition of such games is parameterised by an *objective* $W \subseteq \Omega^\omega$, where Ω is a finite set of labels.

Definition 12 *A W-*game *is given by $G = (V, V_E, V_A, v_0, E, c)$, where $V = V_E \uplus V_A$ is a set of* vertices *divided into the vertices V_E controlled by Eve and the vertices V_A controlled by Adam, $v_0 \in V$ is an* initial vertex*, $E \subseteq V \times V$ is a left-total* transition relation*, and $c : V \to \Omega$ is a labelling function.*

A *finite* (resp. *infinite*) *play* is a finite (resp. infinite) word $\rho = v_0 \dots v_n$ (resp. $\pi = v_0 v_1 \dots$) such that for every i such that $0 \le i < |\rho| - 1$ (resp. $i \ge 0$), $(v_i, v_{i+1}) \in E$. A *strategy* for Eve (resp. for Adam) is a function $\sigma : V^* \cdot V_E \to V$ (resp. $\sigma : V^* \cdot V_A \to V$) such that for all finite play $\rho \in V^* \cdot V_E$ (resp. $\rho \in V^* \cdot V_A$), we have $(\text{last}(\rho), \sigma(\rho)) \in E$. Given a strategy σ for Eve and σ' for Adam, we let $\text{Outcome}(\sigma, \sigma')$ be the unique infinite play that starts in v_0 and is consistent with σ and σ'. We say that a strategy σ *ensures* $W \subseteq \Omega^\omega$ if for all σ', the infinite word obtained by applying c to each position of the play $\text{Outcome}(\sigma, \sigma')$ is in W. We say that Eve *wins* the W-game G if there exists a strategy for her that ensures W. The same notions apply to Adam.

We now introduce the objectives we will be using.

- Given $d \in \mathbb{N}^*$, the *parity objective* **parity** $\subseteq \{1, \dots, d\}^\omega$ is the set of infinite words in which the maximum label appearing infinitely many times is even.
- The *distance objective* uses the set of labels $\{\varepsilon, \mathtt{i}\}$ acting on a counter taking values in the natural numbers and initialised to 0. The labels ε and $\mathtt{i}$ are seen as actions on the counter: the action ε leaves the counter unchanged and $\mathtt{i}$ increments the counter by 1. For $n \in \mathbb{N}$, the distance objective **distance**$(n) \subseteq \{\varepsilon, \mathtt{i}\}^\omega$ is the set of infinite words such that the counter is bounded by n.

  The *regular distance objective* **fininc** $\subseteq \{\varepsilon, \mathtt{i}\}^\omega$ is the set of infinite words such that the counter is incremented finitely many times.
- The *co-distance objective* uses set of labels $\{\varepsilon, \mathtt{i}\}$, where ε and $\mathtt{i}$ have the same interpretation as in **distance**(n). For $n \in \mathbb{N}$, the objective $\overline{\textbf{distance}}(n) \subseteq \{\varepsilon, \mathtt{i}\}^\omega$ is the set of infinite words such that the counter eventually reaches value n.
- The objectives can be combined: **parity** $\cap$ **distance**$(n) \subseteq (\{1, \dots, d\} \times \{\varepsilon, \mathtt{i}\})^\omega$ is the Cartesian product of the parity and the distance objective (where a pair of infinite words is assimilated with the infinite word formed of the pairs of letters at same position).

The following result, proven in [10], relates **distance** and **fininc** in the context of games.

Lemma 1 *Let G be a finite game. There exists $n \in \mathbb{N}$ such that Eve wins for* ***parity*** $\cap$ ***distance***(n) *iff Eve wins for* ***parity*** $\cap$ ***fininc***.

5.2.2 Cost automata

We now define non-deterministic automata over (Σ, S)-trees with objective $W \subseteq \Omega^\omega$.

Definition 13 *A W-*automaton *is a tuple $\mathscr{A} = (Q, q_0, \delta, c)$ where Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta \subseteq Q \times \Sigma \times Q^S$ is a transition relation, and $c : Q \to \Omega$ is a labelling function.*

To define the semantics of W-automata, we define acceptance games. Given an W-automaton $\mathscr{A}$ and a (Σ, S)-tree $t = (\tau, \ell)$, we define the acceptance W-game $G_{\mathscr{A},t}$ as follows.

The set of vertices is $(Q \times \tau) \cup (Q \times \tau \times Q^S)$. The vertices of the form $Q \times \tau$ are controlled by Eve, the others by Adam. The initial vertex is (q_0, r), where r is the root of t. The transition relation relates the vertex (q, u) to (q, u, h) if $(q, \ell(u), h) \in \delta$, and (q, u, h) is related to $(h(s), u \cdot s)$ for every $s \in S$. The label of a vertex (q, u) is $c(q)$, and the other vertices are not labelled. We say that t is accepted by $\mathscr{A}$ if Eve wins the acceptance W-game $G_{\mathscr{A},t}$.

We now instantiate this definition. The objective **parity** $\cap$ **distance** gives rise to the notion of **distance**-automata. A **distance**-automaton $\mathscr{A}$ *computes* the following function $[\![\mathscr{A}]\!]_{\mathbf{d}}$ over trees:

$$[\![\mathscr{A}]\!]_{\mathbf{d}}(t) = \inf\{n \in \mathbb{N} : t \text{ is accepted by } \mathscr{A} \text{ with objective } \mathbf{parity} \cap \mathbf{distance}(n)\},$$

and it *recognises* the $\approx$-equivalence class of the function $[\![\mathscr{A}]\!]_{\mathbf{d}}$.

Dually, the objective **parity** $\cap$ $\overline{\textbf{distance}}(n)$ gives rise to $\overline{\textbf{distance}}$-automata. A $\overline{\textbf{distance}}$-automaton $\mathscr{A}$ *computes* the function $[\![\mathscr{A}]\!]_{\overline{\mathbf{d}}}$ over trees defined by

$$[\![\mathscr{A}]\!]_{\overline{\mathbf{d}}}(t) = \sup\left\{n \in \mathbb{N} : t \text{ is accepted by } \mathscr{A} \text{ with objective } \mathbf{parity} \cap \overline{\mathbf{distance}}(n)\right\}$$

and *recognises* the $\approx$-equivalence class of the function $[\![\mathscr{A}]\!]_{\overline{\mathbf{d}}}$.

If $\mathscr{A}$ recognises the $\approx$-equivalence class of the function $f : E \to (\mathbb{N} \cup \{\infty\})$ we abuse notation and say that $\mathscr{A}$ *recognises the function f*.

The objective **parity** gives rise to parity automata. The following lemma follows from the observation that **fininc** is an ω-regular objective.

Lemma 2 *For every **parity** $\cap$**fininc**-automaton one can construct an equivalent parity automaton.*

5.2.3 Regular cost functions over words

The definitions of cost-automata can be applied to infinite words, which is the particular case where S is a singleton. A central notion in the theory of regular cost functions is that of history-deterministic automata over infinite words. Informally, a non-deterministic automaton is history-deterministic if its non-determinism can be resolved by a function considering only the input read so far. This notion has been introduced for ω-automata in [23]. We specialise it here to the case of cost functions, involving a relaxation on the values allowing for a good interplay with the definition of equivalence for cost functions.

We first introduce the notation $\mathscr{A}_\sigma$ for $\mathscr{A}$ a W-automaton and a *strategy* $\sigma : \Sigma^* \to \delta$, where δ is the transition relation of $\mathscr{A}$: $\mathscr{A}_\sigma$ is a (potentially infinite) deterministic W-automaton $(Q \times \Sigma^*, (q_0, \varepsilon), \delta_\sigma, c_\sigma)$ where $((q,w),a,(q',wa)) \in \delta_\sigma$ just if $\sigma(w) = (q,a,q')$, and $c_\sigma(q,w) = c(q)$. The automaton $\mathscr{A}_\sigma$ is infinite but deterministic, as for each situation the strategy σ chooses the transition to follow.

Definition 14 ([13, 16]) *We say that a **distance**-automaton $\mathscr{A}$ over infinite words is* history-deterministic *if there exists a function $\alpha : \mathbb{N} \to \mathbb{N}$ such that for every n there exists a strategy σ such that for all words w we have $[\![\mathscr{A}]\!]_d(w) \le n \implies [\![\mathscr{A}_\sigma]\!]_d(w) \le \alpha(n)$.*

We now explain the usefulness of the notion of history-deterministic automata. The situation is the following: we consider a language L over infinite words, and we want to construct an automaton for the language of trees "all branches are in L". Given a deterministic automaton for L one can easily solve this problem by constructing an automaton running the deterministic automaton on all branches.

In the quantitative setting we consider here, we have a function $f : \Sigma^\omega \to \mathbb{N} \cup \{\infty\}$ instead of L, and we wish to construct an automaton computing the function over trees $t \mapsto \sup\{f(\lambda) : \lambda \in \text{Branches}(t)\}$. Unfortunately, **distance**-automata do not determinise, so the previous approach needs to be refined. The construction fails for non-deterministic automata, because two branches may have very different accepting runs even on their shared prefix. The notion of history-deterministic automata yields a solution to this problem, as stated in the following theorem.

Theorem 3 ([17]) *Let $\mathscr{A}$ be a history-deterministic **distance**-automaton over infinite words. One can construct a **distance**-automaton recognising the function over trees*

$$t \mapsto \sup\{[\![\mathscr{A}]\!]_d(\lambda) : \lambda \in Branches(t)\}$$

We present an extension of this result where the function can remove a bounded number of paths in the computation.

Theorem 4 *Let $\mathscr{A}$ be a history-deterministic **distance**-automaton over infinite words. One can construct a **distance**-automaton recognising the function over trees*

$$t \mapsto \inf\{\max(n, \sup\{[\![\mathscr{A}]\!]_d(\lambda) : \lambda \notin B\}) : B \subseteq Branches(t), Card(B) \leq n\}.$$

5.2.4 Regular cost functions over trees

We introduce the notion of nested automata, parameterised by an objective $W \subseteq \Omega^\omega$. They can be seen as a special form of alternating automata which will be convenient in the technical developments.

Definition 15 *A* nested W-automaton *with k slaves over (Σ, S)-trees consists of a* master W*-automaton over $(2^k, S)$-trees, and k* slave W*-automata over (Σ, S)-trees $(\mathscr{A}_i)_{i\in[k]}$.*

The transition relation of the master is $\delta \subseteq Q \times 2^k \times Q^S$. We describe the modus operandi of a nested automaton informally. Let t be a tree and u a node in t, labelled with state q. To take the next transition the master automaton interrogates its slaves: the transition $(q, v, h) \in \delta$ is allowed if for all $i \in v$, the subtree t_u is accepted by $\mathscr{A}_i$. The formal semantics of nested W-automata can be found in Appendix **??**.

Theorem 5 ([14]) *Let f be a cost function over regular trees. The following are effectively equivalent:*

- *there exists a **distance**-automaton recognising f,*
- *there exists a nested **distance**-automaton recognising f,*
- *there exists a $\overline{\textbf{distance}}$-automaton recognising f,*
- *there exists a nested $\overline{\textbf{distance}}$-automaton recognising f.*

5.3 Model checking BOUND-QCTL*

The *model-checking problem* for BOUND-QCTL* is the following: given an instance $(\Phi, \mathscr{S})$ where Φ is a sentence of BOUND-QCTL* and $\mathscr{S}$ is a Kripke structure, return 'Yes' if $\mathscr{S} \models \Phi$ and 'No' otherwise. In this section we prove that this problem is decidable by reducing it to the emptiness problem of parity automata. We will use the following result about **distance**-automata over infinite words.

Theorem 6 ([25, 26]) *For every PROMPT-LTL formula ψ, we can construct a history-deterministic **distance**-automaton $\mathscr{A}$ such that $[\![\mathscr{A}]\!]_d \approx [\![\psi]\!]_{inf}$.*

Theorem 7 *Let Φ be a sentence of* BOUND-QCTL*. *We construct a non-deterministic parity automaton $\mathscr{A}_\Phi$ over (Σ, S)-trees such that for every Kripke structure $\mathscr{S}$ over the set of states S, we have $\mathscr{S} \models \Phi$ if, and only if, $\mathscr{A}_\Phi$ accepts the unfolding $t_{\mathscr{S}}$.*

Proof sketch The idea of the proof is to build for each subformula φ of Φ, a **distance**-automaton (resp. $\overline{\textbf{distance}}$-automaton) $\mathscr{A}_\varphi$ such that $[\![\mathscr{A}_\varphi]\!]_{\mathbf{d}} \approx [\![\varphi]\!]_{\mathbf{inf}}$ (resp. $[\![\mathscr{A}_\varphi]\!]_{\overline{\mathbf{d}}} \approx [\![\varphi]\!]_{\mathbf{sup}}$) if φ is positive (resp. negative).

For formulas of the form $\varphi = \mathbf{A}\psi$, one treats ψ as a PROMPT-LTL formula on infinite words with maximal state subformulas as atoms. By applying Theorem 6 on ψ we get a history-deterministic **distance**-automaton $\mathscr{A}_\psi$ over infinite words such that $[\![\mathscr{A}_\psi]\!]_{\mathbf{d}} \approx [\![\psi]\!]_{\mathbf{inf}}$. One then applies Theorem 3 to $\mathscr{A}_\psi$ to get a **distance**-automaton $\mathscr{A}$ such that $[\![\mathscr{A}]\!]_{\mathbf{d}}(t) = \sup\{[\![\mathscr{A}_\psi]\!]_{\mathbf{d}}(\lambda) : \lambda \in \text{Branches}(t)\}$. $\mathscr{A}$ is the master of the nested automaton we build; by induction hypothesis one builds a slave **distance**-automaton $\mathscr{A}'_\varphi$ for each maximal state subformula φ' of φ, using Theorem 5 when necessary.

For $\varphi = \mathbf{A}^{\leq N}\psi$, the construction is the same as for $\mathbf{A}\psi$, except for the construction of the master $\mathscr{A}$, in which we replace Theorem 3 by Theorem 4.

For $\varphi = \exists N\,\varphi'$, one builds a **distance**-automaton for φ' and uses Lemma 1 together with Lemma 2 to remove the counter and obtain a parity automaton. □

The model-checking problems for both PROMPT-SL and BOSL (as well as their combination) can be easily reduced to that of BOUND-QCTL*. As a consequence, by Theorem 7 we get:

Theorem 8 *The model-checking problem is decidable for* PROMPT-SL *and* BOSL.

6 Conclusion

We introduced two quantitative extensions of Branching-time Strategy Logic (BSL), i.e., PROMPT-SL that extends BSL with $\mathbf{F}^{\leq N}$ that limits the range of the eventuality, and BOSL that extends BSL with $\mathbf{A}^{\leq N}$ that limits the range of the outcome quantifier. We proved that model checking both these logics is decidable. To the best of our knowledge these are the first decidable quantitative extensions of SL.

In order to prove our results we used notions from the theory of regular cost functions to develop new technical insights necessary to address PROMPT-SL and BOSL. Moreover, as an intermediate formalism between cost automata and logics for strategic reasoning we introduced BOUND-QCTL*, a quantitative extension of QCTL*, and proved its model checking decidable. Using this, it is easy to see that also the extension of BSL with $\exists N$ and both $\mathbf{F}^{\leq N}$ and $\mathbf{A}^{\leq N}$ has a decidable model-checking problem.

References

[1] Shaull Almagor, Yoram Hirshfeld & Orna Kupferman (2010): *Promptness in ω-regular automata*. In: *ATVA*, LNCS 6252, Springer, pp. 22–36.

[2] Rajeev Alur, Kousha Etessami, Salvatore La Torre & Doron Peled (2001): *Parametric temporal logic for model measuring*. *ACM Transactions on Computational Logic* 2(3), pp. 388–407.

[3] Rajeev Alur, Thomas A. Henzinger & Orna Kupferman (2002): *Alternating-time temporal logic*. *Journal of the ACM* 49(5), pp. 672–713. Available at `http://doi.acm.org/10.1145/585265.585270`.

[4] Benjamin Aminof, Aniello Murano, Sasha Rubin & Florian Zuleger (2016): *Prompt Alternating-Time Epistemic Logics*. In: *KR*, AAAI Press, pp. 258–267. Available at `http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12890`.

[5] Mordechai Ben-Ari, Zohar Manna & Amir Pnueli (1981): *The Temporal Logic of Branching Time*. In: *POPL*, pp. 164–176. Available at `http://doi.acm.org/10.1145/567532.567551`.

[6] Raphaël Berthon, Bastien Maubert & Aniello Murano (2017): *Decidability results for ATL* with imperfect information and perfect recall*. In: *AAMAS*.

[7] Raphaël Berthon, Bastien Maubert, Aniello Murano, Sasha Rubin & Moshe Y. Vardi (2017): *Strategy Logic with imperfect information*. In: *LICS*.

[8] Patricia Bouyer, Patrick Gardy & Nicolas Markey (2015): *Weighted Strategy Logic with Boolean Goals Over One-Counter Games*. In: *FSTTCS 2015*, pp. 69–83, doi:10.4230/LIPIcs.FSTTCS.2015.69. Available at `https://doi.org/10.4230/LIPIcs.FSTTCS.2015.69`.

[9] Arnaud Carayol & Olivier Serre (2015): *How Good Is a Strategy in a Game with Nature?* In: *LICS*, IEEE Computer Society, pp. 609–620.

[10] Krishnendu Chatterjee & Nathanaël Fijalkow (2013): *Infinite-state games with finitary conditions*. In: *CSL*, pp. 181–196, doi:10.4230/LIPIcs.CSL.2013.181. Available at `https://doi.org/10.4230/LIPIcs.CSL.2013.181`.

[11] Krishnendu Chatterjee, Thomas A Henzinger & Florian Horn (2009): *Finitary winning in ω-regular games*. *ACM Transactions on Computational Logic* 11(1), p. 1.

[12] Krishnendu Chatterjee, Thomas A. Henzinger & Nir Piterman (2010): *Strategy Logic*. *Information and Computation* 208(6), pp. 677–693, doi:10.1016/j.ic.2009.07.004. Available at `http://dx.doi.org/10.1016/j.ic.2009.07.004`.

[13] Thomas Colcombet (2009): *The Theory of Stabilisation Monoids and Regular Cost Functions*. In: *ICALP*.

[14] Thomas Colcombet (2013): *Fonctions Régulières de Coût*. Habilitation Thesis.

[15] Thomas Colcombet (2013): *Regular Cost Functions, Part I: Logic and Algebra over Words*. *Logical Methods in Computer Science* 9(3).

[16] Thomas Colcombet & Nathanaël Fijalkow (2016): *The Bridge Between Regular Cost Functions and ω-Regular Languages*. In: *ICALP*, pp. 126:1–126:13, doi:10.4230/LIPIcs.ICALP.2016.126. Available at `https://doi.org/10.4230/LIPIcs.ICALP.2016.126`.

[17] Thomas Colcombet & Christof Löding (2010): *Regular Cost Functions over Finite Trees*. In: *LICS*, pp. 70–79, doi:10.1109/LICS.2010.36. Available at `https://doi.org/10.1109/LICS.2010.36`.

[18] E. Allen Emerson & Joseph Y. Halpern (1983): *"Sometimes" and "Not Never" Revisited: On Branching Versus Linear Time*. In: *POPL*, pp. 127–140. Available at `http://doi.acm.org/10.1145/567067.567081`.

[19] Nathanaël Fijalkow, Florian Horn, Denis Kuperberg & Michał Skrzypczak (2015): *Trading Bounds for Memory in Games with Counters*. In: *ICALP*, pp. 197–208, doi:10.1007/978-3-662-47666-6˙16. Available at `https://doi.org/10.1007/978-3-662-47666-6_16`.

[20] Nathanaël Fijalkow & Martin Zimmermann (2012): *Cost-Parity and Cost-Street Games*. In: *FSTTCS*, LIPIcs 18, pp. 124–135.

[21] Nathanaël Fijalkow & Martin Zimmermann (2014): *Parity and Streett Games with Costs*. *Logical Methods in Computer Science* 10(2), doi:10.2168/LMCS-10(2:14)2014. Available at `https://doi.org/10.2168/LMCS-10(2:14)2014`.

[22] Patrick Gardy (2017): *Semantics of Strategy Logic*. Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France. Available at `https://tel.archives-ouvertes.fr/tel-01561802`.

[23] Thomas A. Henzinger & Nir Piterman (2006): *Solving Games Without Determinization*. In: *CSL*, pp. 395–410.

[24] Sophia Knight & Bastien Maubert (2015): *Dealing with imperfect information in Strategy Logic*. In: *SR*.

[25] Denis Kuperberg (2014): *Linear Temporal Logic for Regular Cost Functions*. *Logical Methods in Computer Science* 10(1).

[26] Denis Kuperberg & Michael Vanden Boom (2012): *On the Expressive Power of Cost Logics over Infinite Words*. In: *ICALP*, pp. 287–298.

[27] O. Kupferman, P. Madhusudan, P. S. Thiagarajan & M. Y. Vardi (2000): *Open Systems in Reactive Environments: Control and Synthesis*. In: *CONCUR*, LNCS 1877, Springer, pp. 92–107.

[28] Orna Kupferman, Giuseppe Perelli & Moshe Y. Vardi (2016): *Synthesis with rational environments*. *Annals of Mathematics and Artificial Intelligence* 78(1), pp. 3–20, doi:10.1007/s10472-016-9508-8. Available at `https://doi.org/10.1007/s10472-016-9508-8`.

[29] Orna Kupferman, Nir Piterman & Moshe Y Vardi (2009): *From liveness to promptness*. *Formal Methods in System Design* 34(2), pp. 83–103.

[30] Orna Kupferman, Moshe Y. Vardi & Pierre Wolper (2000): *An automata-theoretic approach to branching-time model checking*. *Journal of the ACM* 47(2), pp. 312–360, doi:10.1145/333979.333987. Available at `http://doi.acm.org/10.1145/333979.333987`.

[31] François Laroussinie & Nicolas Markey (2015): *Augmenting ATL with strategy contexts*. Information and Computation 245, pp. 98–123.

[32] Fabio Mogavero, Aniello Murano, Giuseppe Perelli & Moshe Y. Vardi (2014): *Reasoning About Strategies: On the Model-Checking Problem*. ACM Transactions on Computational Logic 15(4), pp. 34:1–34:47, doi:10.1145/2631917. Available at `http://doi.acm.org/10.1145/2631917`.

[33] Fabio Mogavero, Aniello Murano & Loredana Sorrentino (2015): *On Promptness in Parity Games*. Fundamenta Informaticae 139(3), pp. 277–305.

[34] Amir Pnueli (1977): *The Temporal Logic of Programs*. In: FOCS, pp. 46–57. Available at `http://doi.ieeecomputersociety.org/10.1109/SFCS.1977.32`.

[35] Amir Pnueli & Roni Rosner (1989): *On the synthesis of a reactive module*. In: POPL, pp. 179–190.

[36] Martin Zimmermann (2013): *Optimal bounds in parametric LTL games*. Theoretical Computer Science 493, pp. 30–45, doi:10.1016/j.tcs.2012.07.039. Available at `http://dx.doi.org/10.1016/j.tcs.2012.07.039`.