Quantifying Bounds in Strategy Logic

Nathanaël Fijalkow[1]
CNRS, LaBRI, Bordeaux, France
Alan Turing Institute of data science, London, United Kingdom
nfijalkow@turing.ac.uk
0000-0002-6576-4680

Bastien Maubert[2]
University of Naples "Federico II", Naples, Italy
bastien.maubert@gmail.com
0000-0002-9081-2920

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

Abstract

Program synthesis constructs programs from specifications in an automated way. Strategy Logic (SL) is a powerful and versatile 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.

2012 ACM Subject Classification Logic and Verification

Keywords and phrases Prompt LTL, Strategy Logic, Model checking, Automata with counters

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

In order to reason about strategic aspects in distributed systems, temporal logics of programs (such as LTL [35], CTL [5] and CTL* [19]) 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) [13, 33]. Both logics can express the existence of strategies for coalitions that

[1] This project has received funding from the Alan Turing Institute under EPSRC grant EP/N510129/1 and the DeLTA project (ANR-16-CE40-0007).
[2] This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 709188.

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, synthesizing reactive systems from temporal specifications [36, 28, 29] 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 the variables that make the formula true. For instance, Prompt-LTL [2, 30] is an extension of LTL with the operator $\mathbf{F}^{\leq N}$ where N is a (unique) 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 synthesize 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 $\exists s_1(a_1, s_1)\forall s_2(a_2, s_2)\exists N \mathbf{AGF}^{\leq N} p$ 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 (generated by the remaining agents) the atom p holds at least once every N steps.

Another way to parameterise temporal logics is to bound the path quantifiers, expressing, for instance, that at least k different paths satisfy ψ, or all but k paths satisfy ψ [8]. Such operators can bound, 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 $\exists s(a, s)\exists N \mathbf{A}^{\leq N}\mathbf{GF}p$ 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). We do this by applying 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 [14, 16].

Our results allow us to solve a problem left open in [10] 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. The complexity of these problems is no worse than for ordinary LTL, i.e., PSPACE. The technique used in this paper to prove these upper-bounds is a pumping lemma that allows one to reduce/enlarge the parameters.

Parametric-LTL has been studied in the context of open systems and games. For instance, [37] studies the problem of synthesizing 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. The complexity of these problems is, again, no worse than that of solving ordinary LTL games, i.e., 2EXPTIME. The technique used to prove these upper bounds is the alternating-colour technique of [30] that allows one to replace a prompt formula by an LTL formula, and was originally introduced to reason about Prompt-LTL, the fragment of parametric-LTL without $\mathbf{G}^{\leq x}$. We remark that Church's synthesis for Prompt-LTL formulas was shown in [30] to have complexity no worse than that of LTL, i.e., 2EXPTIME.

Promptness was first studied in the context of multi-agent systems in [4]. They study the model-checking problem for the logic Prompt-ATL* and its fragments, for memoryless and memoryful strategies. Again, one finds that the complexity of model checking prompt variations is no worse than the non-prompt ones. That paper also studies the case of systems with imperfect information. 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, the even colors represent requests and odd colors represent grants, and the objective of the player is to ensure that every request is promptly followed by a larger grant [12, 34]. We discuss this in Example 6. Such winning conditions have been generalised to games with costs in [21, 22], leading to the construction of efficient algorithms for synthesizing 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 complexity of decision problems (such as containment) have been studied in [1, 12].

The theory of regular cost functions [14, 16] 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 LTL$^{\leq}$ logic from [26], which extends LTL with a bounded until $\varphi \mathbf{U}^{\leq N} \varphi'$ allowing φ not to hold in at most N (possibly non-consecutive) 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 [20]. 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 [23]. For the Boolean Goal fragment of Strategy Logic in the case of one counter, the problem is still open [9, 23]. 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

In this section we recall Branching-time Strategy Logic [25], a variant of Strategy Logic [33].

For the rest of the paper 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 $w = w_0w_1\dots w_n$ 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 $\mathcal{G} = (V, v_0, \Delta, \ell)$ where V is the set of *vertices*, $v_0 \in V$ is the *initial vertex*, $\Delta : V \times \mathrm{Act}^{\mathrm{Ag}} \to V$ 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. $\pi = v_0v_1\dots$) 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 assignment χ and finite play ρ, $\mathrm{Out}(\chi, \rho)$ is 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 $\rho \cdot v_1 v_2 \ldots$ such that for all $i \geq 0$, there exists $\mathbf{c}$ such that for all $a \in dom(\chi) \cap \mathrm{Ag}$, $\mathbf{c}_a \in \chi(a)(\rho \cdot v_1 \ldots v_i)$ 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 [25], which has the same expressive power as SL but allows to express branching-time properties without resorting to computationally expensive strategy quantifications.

At the syntax level, in addition to usual boolean connectives and temporal operators, we have four 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 following grammar:

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

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

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 $\forall s\varphi = \neg\exists s\neg\varphi$.

A variable s appears *free* in a formula φ if it appears in a binding operator (a, s) that is not in the scope of any strategy quantifier $\langle\langle s\rangle\rangle$.

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 $\mathcal{G}$, an assignment χ that is 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 by mutual induction is as follows:

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

BSL has the same expressivity as SL, and there are linear translations in both directions [25]. 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 [13, 33].

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 at the latest within 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 6).

▶ **Definition 4** (Prompt-SL syntax). The syntax of Prompt-SL formulas is defined by the following grammar:

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

where $p \in \text{AP}$, $s \in \text{Var}$, $a \in \text{Ag}$ and N is a fixed bounding variable. A Prompt-SL *sentence* is a state formula with no free strategy variable, 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, $\mathcal{G}$ 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.

$$\begin{array}{lll}
\mathcal{G}, \chi, \rho, n \models p & \text{if} & p \in \ell(\text{last}(\rho)) \\
\mathcal{G}, \chi, \rho, n \models \neg\varphi & \text{if} & \mathcal{G}, \chi, \rho, n \not\models \varphi \\
\mathcal{G}, \chi, \rho, n \models \varphi \vee \varphi' & \text{if} & \mathcal{G}, \chi, \rho, n \models \varphi \;\text{ or }\; \mathcal{G}, \chi, \rho, n \models \varphi' \\
\mathcal{G}, \chi, \rho, n \models \exists s\varphi & \text{if} & \text{there exists } \sigma \in \text{Strat s.t. } \mathcal{G}, \chi[s \mapsto \sigma], \rho, n \models \varphi \\
\mathcal{G}, \chi, \rho, n \models (a, s)\varphi & \text{if} & \mathcal{G}, \chi[a \mapsto \chi(s)], \rho, n \models \varphi \\
\mathcal{G}, \chi, \rho, n \models (a, ?)\varphi & \text{if} & \mathcal{G}, \chi[a \mapsto ?], \rho, n \models \varphi \\
\mathcal{G}, \chi, \rho, n \models \mathbf{A}\psi & \text{if} & \text{for all } \pi \in \text{Out}(\chi, \rho),\; \mathcal{G}, \chi, \pi, |\rho| - 1 \models \varphi \\
\mathcal{G}, \chi, \rho, n \models \exists N\varphi & \text{if} & \text{there exists } n' \in \mathbb{N} \text{ such that } \mathcal{G}, \chi, \rho, n' \models \varphi \\
 & & \\
\mathcal{G}, \chi, \pi, i, n \models \varphi & \text{if} & \mathcal{G}, \chi, \pi_{\leq i} \models \varphi \\
\mathcal{G}, \chi, \pi, i, n \models \neg\psi & \text{if} & \mathcal{G}, \chi, \pi, i, n \not\models \psi \\
\mathcal{G}, \chi, \pi, i, n \models \psi \vee \psi' & \text{if} & \mathcal{G}, \chi, \pi, i, n \models \psi \;\text{ or }\; \mathcal{G}, \chi, \pi, i, n \models \psi' \\
\mathcal{G}, \chi, \pi, i, n \models \mathbf{X}\psi & \text{if} & \mathcal{G}, \chi, \pi, i + 1, n \models \psi \\
\mathcal{G}, \chi, \pi, i, n \models \psi\mathbf{U}\psi' & \text{if} & \exists j \geq i \text{ s.t. } \mathcal{G}, \chi, \pi, j \models \psi' \\
 & & \quad\text{and } \forall k \text{ s.t. } i \leq k < j,\; \mathcal{G}, \chi, \pi, k \models \psi \\
\mathcal{G}, \chi, \pi, i, n \models \mathbf{F}^{\leq N}\psi & \text{if} & \text{there exists } j \in [i, n] \text{ such that } \mathcal{G}, \chi, \pi, j, n \models \psi.
\end{array}$$

The semantics of a sentence Φ does not depend on the bound n, and we may write $\mathcal{G}, \chi, \rho \models \Phi$ if $\mathcal{G}, \chi, \rho, n \models \Phi$ for some n. In addition a sentence does not require an assignment for its evaluation. Given a game $\mathcal{G}$ with initial vertex v_0 and a sentence Φ, we write $\mathcal{G} \models \Phi$ if $\mathcal{G}, \emptyset, v_0 \models \Phi$, where $\emptyset$ is the empty assignment.

▶ **Example 6.** In bounded parity games [12, 34] 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

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

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

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

Observe that in both these definitions, the bound on the delay between requests and grants can depend on the outcome, i.e. on the opponent's strategy. We can also 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 [14, 16, 15], and their relationship to the non-uniform variants has been investigated in [11]. 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

We now define our second quantitative extension of Strategy Logic, which we call Bounding-outcomes Strategy Logic, or BOSL.

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 7** (BOSL syntax)**.** The syntax of BOSL formulas is given by the following grammar:

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

where $p \in \text{AP}$, $s \in \text{Var}$, $a \in \text{Ag}$ and N is a fixed bounding variable. A BOSL *sentence* is a state formula with no free strategy variable, 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 8** (BOSL semantics)**.** We only give the definition for the new operator $\mathbf{A}^{\leq N}$, the others are as in Definition 5.

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

The full semantics can be found in Appendix A.1. Once again, for a sentence Φ we write $\mathcal{G} \models \Phi$ if $\mathcal{G},\emptyset,v_0,n \models \Phi$ for some $n \in \mathbb{N}$, where $\emptyset$ is the empty assignment.

▶ **Example 9.** As an example we consider the framework of Carayol and Serre [10] that considers 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 whether player 1 has a strategy ensuring to win almost all paths. The paper [10] 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 following 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 [10]. 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 in the Bounding-outcome Strategy Logic formula $\exists s_1(a_1,s_1)\forall s_2(a_2,s_2)\exists N\mathbf{A}^{\leq N}\neg\psi$. Note that the subtlety here is that the value of N is not fixed and may depend on the opponent's strategy. 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 [32] for ATL with strategy context, in [6] for an extension of it with imperfect information and in [7] for Strategy Logic with imperfect information, we go through an adequate extension of QCTL*, which itself extends 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*

In this section 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* [32] 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, an existential quantification on the bound for the bounded eventually and bounded outcome quantifier is also part of the syntax.

5.1.1 Bound-QCTL* syntax

▶ **Definition 10.** 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 $p \in \text{AP}$, 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 $\exists N$. Note that we will be interested in model checking sentences, and every subformula of a sentence is either positive or negative.

5.1.2 Bound-QCTL* semantics

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

▶ **Definition 11.** 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*[3], and $\ell : S \to \Sigma$ is a *labelling function*.

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

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 $\text{Paths}(s) \subseteq S^+$ 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* λ in t is an infinite sequence of nodes such that $\lambda_0 \in \tau$ and for all i, λ_{i+1} is a child of λ_i, and $\text{Branches}(t, u)$ is the set of branches that start in node u. We let $\text{Branches}(t)$ denote the set of branches that start in the root. If S is a singleton, a tree becomes an infinite word.

▶ **Definition 12.** The *tree unfolding of a Kripke structure* $\mathcal{S}$ from state s is the tree $t_{\mathcal{S}}(s) = (\text{Paths}(s), \ell')$, where for every $u \in \text{Paths}(s)$, we have $\ell'(u) = \ell(\text{last}(u))$. We may write $t_{\mathcal{S}}$ for $t_{\mathcal{S}}(s_0)$, the unfolding from the initial state.

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 13.** 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 (the inductive cases for classic CTL* operators can be found in Appendix A.2):

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

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

$$[\![\varphi]\!]_{\mathbf{inf}}(t) = \inf\{n \in \mathbb{N} : t, r, n \models \varphi\} \quad \text{and} \quad [\![\varphi]\!]_{\mathbf{sup}}(t) = \sup\{n \in \mathbb{N} : t, r, n \models \varphi\},$$

with the usual convention that $\inf \emptyset = \infty$ 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 downward (resp. upward) closed if φ is negative (resp. positive). The value of a sentence Φ is always either 0 or ∞ (recall that sentences are necessarily positive formulas and N is always quantified), and given a Kripke structure $\mathcal{S}$, we write $\mathcal{S} \models \Phi$ if $[\![\Phi]\!]_{\mathbf{inf}}(t_{\mathcal{S}}) = 0$.

5.2 Regular cost functions

In this section we develop the theory of regular cost functions over trees for $\overline{\text{distance}}$ automata. To this end we define and study the two dual models of **distance** and $\overline{\textbf{distance}}$-automata for recognising cost functions [14], 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 $\mathbf{A}$ 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.

▶ **Definition 14.** A *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*, $c : V \to \Omega$ 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 $0 \leq i < |\rho| - 1$ (resp. $i \geq 0$), $(v_i, v_{i+1}) \in E$. 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'.

An *objective* is a set $W \subseteq \Omega^\omega$. To make the objective explicit we speak of W-games, which are games with objective W. A strategy σ for Eve *ensures* $W \subseteq \Omega^\omega$ if for all strategy σ' of Adam, the infinite word obtained by applying c to each position of the play $\text{Outcome}(\sigma, \sigma')$ is in W. 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* $\textbf{parity} \subseteq \{1, \dots, d\}^\omega$ 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 $\{\epsilon, \mathtt{i}\}$ 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 $\textbf{distance}(n) \subseteq \{\epsilon, \mathtt{i}\}^\omega$ is the set of infinite words such that the counter is bounded by n.
- The *regular distance objective* $\textbf{fininc} \subseteq \{\epsilon, \mathtt{i}\}^\omega$ is the set of infinite words such that the counter is incremented finitely many times.
- The *co-distance objective* uses set of labels $\{\epsilon, \mathtt{i}\}$, where ϵ and $\mathtt{i}$ have the same interpretation as in $\textbf{distance}(n)$. For $n \in \mathbb{N}$, the objective $\overline{\textbf{distance}}(n) \subseteq \{\epsilon, \mathtt{i}\}^\omega$ is the set of infinite words such that the counter eventually reaches value n.
- The objectives can be combined: $\textbf{parity} \cap \textbf{distance}(n) \subseteq (\{1, \dots, d\} \times \{\epsilon, \mathtt{i}\})^\omega$ 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 [11], relates **distance** and **fininc** in the context of games.

▶ **Lemma 15.** *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 automata over (Σ, S)-trees.

▶ **Definition 16.** A (non-deterministic) *automaton* is a tuple $\mathcal{A} = (Q, q_0, \delta, c)$ 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.

When an automaton is equipped with an objective $W \subseteq \Omega^\omega$ we speak of an W-automaton. To define the semantics of W-automata, we define acceptance games. Given an W-automaton $\mathcal{A}$ and a (Σ, S)-tree $t = (\tau, \ell)$, we define the acceptance W-game $G_{\mathcal{A},t}$ 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 $\mathcal{A}$ if Eve wins the acceptance W-game $G_{\mathcal{A},t}$.

An equivalent point of view is to say that t is accepted by $\mathcal{A}$ if there exists a (Q, S)-tree with same domain as t respecting the transition relation δ with respect to t, such that all branches satisfy W.

We instantiate this definition for cost automata: the objective **parity** $\cap$ **distance** gives rise to the notion of **distance**-automata. A **distance**-automaton $\mathcal{A}$ *computes* the function $[\![\mathcal{A}]\!]_{\mathbf{d}}$ over trees defined by

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

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

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

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

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

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

To illustrate the definition of **distance**-automata, we now give an example that will be useful later on to capture the bounded path quantifier $\mathbf{A}^{\leq N}$.

▶ **Lemma 17.** *Let $p \in AP$. There exists a* **distance***-automaton recognising the function that counts the number of paths with infinitely many p's.*

Proof. Let us say that a path is bad if it contains infinitely many p. The **distance**-automaton $\mathcal{A}$ has four states:

- $q_{0,\epsilon}$, whose intuitive semantics is "the tree contains one bad path",
- $q_{0,\mathtt{i}}$, meaning "the tree contains at least two bad paths",
- $q_{1,p}$ and $q_{1,\neg p}$, which mean "the tree does not contain any bad path".

All states are initial (note that this is an inconsequential abuse because we defined automata with a single initial state). We use the set of labels $\Omega = \{2,3\} \times \{\epsilon, \mathtt{i}\}$. The transitions are as follows, where $q_0 = \{q_{0,\epsilon}, q_{0,\mathtt{i}}\}$ and $q_1 = \{q_{1,p}, q_{1,\neg p}\}$.

$$\delta = \begin{cases} (q_{0,\epsilon}, a, h) & \text{if } h \text{ contains at most one } q_0 \\ (q_{0,\mathtt{i}}, a, h) & \text{if } h \text{ contains at least two } q_0 \\ (q_{1,\neg p}, a, h) & \text{if } p \notin a \text{ and } h \text{ contains only } q_1 \\ (q_{1,p}, a, h) & \text{if } p \in a \text{ and } h \text{ contains only } q_1 \end{cases}$$

The labelling function is $c(q_{0,\epsilon}) = (2,\epsilon), c(q_{0,\mathtt{i}}) = (2,\mathtt{i}), c(q_{1,\neg p}) = (2,\epsilon)$, and $c(q_{1,p}) = (3,\epsilon)$. We claim that the following two properties hold, which implies Lemma 17.

- if t contains n bad paths, then $[\![\mathcal{A}]\!]_{\mathbf{d}}(t) \leq n-1$,
- if $[\![\mathcal{A}]\!]_{\mathbf{d}}(t) \leq n$, then t contains at most $\text{Card}(S)^n$ bad paths.

Assume that t contains n bad paths, we construct a run for $\mathcal{A}$ (i.e., a labelling of t with states of $\mathcal{A}$) as follows. A node u of the tree is labelled by:

- $q_{0,\epsilon}$ if exactly one $t_{u\cdot s}$ contains a bad path for some direction $s \in S$,
- $q_{0,\mathtt{i}}$ if $t_{u\cdot s}$ contain a bad path for at least two different directions $s \in S$,
- $q_{1,\neg p}$ if t_u does not contain a bad path and $p \notin \ell(u)$,
- $q_{1,p}$ if t_u does not contain a bad path and $p \in \ell(u)$.

This yields a valid run whose branches all satisfy the parity condition. Along a branch the counter is incremented each time there are at least two subtrees with a bad path, which can happen at most $n-1$ times because there are n bad paths. Hence the maximal value of the counter on a branch is $n-1$, implying that $[\![\mathcal{A}]\!]_{\mathbf{d}}(t) \leq n-1$.

We show the second point by induction on n. If $[\![\mathcal{A}]\!]_{\mathbf{d}}(t) = 0$, then t contains at most one bad path. If $[\![\mathcal{A}]\!]_{\mathbf{d}}(t) = n+1$, consider a (Q,S)-tree representing a run of value $n+1$. Because $[\![\mathcal{A}]\!]_{\mathbf{d}}(t) \geq 1$, there is at least one node labelled $q_{0,\mathtt{i}}$. By definition of the transition relation, if there are two nodes on the same level labelled $q_{0,\mathtt{i}}$, then they must descend from another node $q_{0,\mathtt{i}}$ higher in the tree. Thus there is a unique node u labelled $q_{0,\mathtt{i}}$ that is closest to the root (it may be the root itself). Except for u's ancestors, which are labelled with $q_{0,\epsilon}$, all nodes outside of the subtree rooted in u are necessarily labelled with q_1. The subtrees rooted in u's children have a run with value at most n. By induction hypothesis each of these subtrees contains at most $\text{Card}(S)^n$ bad paths, so the tree rooted in u contains at most $\text{Card}(S)^{n+1}$ bad paths. Since nodes labelled by q_1 cannot contain a bad path, this means that t contains at most $\text{Card}(S)^{n+1}$ bad paths. ◀

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

▶ **Lemma 18.** *For every automaton with objective **parity** $\cap$ **fininc** 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 studying ω-automata in [24]. 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.

To give a formal definition we introduce the notation $\mathcal{A}_\sigma$ for $\mathcal{A}$ a W-automaton and a *strategy* $\sigma : \Sigma^* \to \delta$, where δ is the transition relation of $\mathcal{A}$: $\mathcal{A}_\sigma$ 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 $\mathcal{A}_\sigma$ is infinite but deterministic, as for each situation the strategy σ chooses the transition to follow.

▶ **Definition 19** ([14, 17]). We say that a **distance**-automaton $\mathcal{A}$ 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 $[\![\mathcal{A}]\!]_{\mathbf{d}}(w) \leq n \implies [\![\mathcal{A}_\sigma]\!]_{\mathbf{d}}(w) \leq \alpha(n)$.

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 20** ([18]). *Let $\mathcal{A}$ be a history-deterministic **distance**-automaton over infinite words. One can construct a **distance**-automaton recognising the function over trees*

$$t \mapsto \sup \{[\![\mathcal{A}]\!]_{\mathbf{d}}(\lambda) : \lambda \in Branches(t)\}$$

We present an extension of this result where the function can remove a bounded number of paths in the computation. The proof is in Appendix A.3.

▶ **Theorem 21.** *Let $\mathcal{A}$ be a history-deterministic **distance**-automaton over infinite words. One can construct a **distance**-automaton recognising the function over trees*

$$t \mapsto \inf \{\max(Card(B), \sup \{[\![\mathcal{A}]\!]_{\mathbf{d}}(\lambda) : \lambda \notin B\}) : B \subseteq Branches(t)\} .$$

The idea is to combine $\mathcal{A}$ with the automaton defined in the proof of Lemma 17.

5.2.4 Regular cost functions over trees

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

▶ **Definition 22.** A *nested W-automaton* with k slaves over (Σ, S)-trees is given by

- a *master automaton* $\mathcal{A}$, which is a W-automaton over $(2^k, S)$-trees, and
- k *slave automata* $(\mathcal{A}_i)_{i \in [k]}$, which are W-automata over (Σ, S)-trees.

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 $\mathcal{A}_i$. The formal semantics of nested W-automata can be found in Appendix A.4.

The following theorem shows the equivalence between **distance** and $\overline{\textbf{distance}}$-automata over regular trees.

▶ **Theorem 23** ([15]). *Let f be a cost function over regular trees. The following statements are effectively equivalent:*

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

5.3 Model checking Bound-QCTL*

The *model-checking problem* for Bound-QCTL* is the following decision problem: given an instance $(\Phi, \mathcal{S})$ where Φ is a sentence of Bound-QCTL* and $\mathcal{S}$ is a Kripke structure, return 'Yes' if $\mathcal{S} \models \Phi$ 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 24** ([26, 27]). *For every* Prompt-LTL *formula ψ, we can construct a history-deterministic* ***distance****-automaton $\mathcal{A}$ such that $[\![\mathcal{A}]\!]_{\mathbf{d}} \approx [\![\psi]\!]_{\mathbf{inf}}$.*

▶ **Theorem 25.** *Let Φ be a sentence of* Bound-QCTL**. We construct a non-deterministic parity automaton $\mathcal{A}_\Phi$ over (Σ, S)-trees such that for every Kripke structure $\mathcal{S}$ over the set of states S, we have $\mathcal{S} \models \Phi$ if, and only if, $\mathcal{A}_\Phi$ accepts the unfolding $t_{\mathcal{S}}$.*

Proof. Let Φ be a sentence and S a finite set of states. Throughout this proof, by trees we mean regular trees, so in particular $\approx$ is understood over such trees.

For each subformula φ of Φ, we construct by induction on φ the following automata:

1. if φ is positive, a **distance**-automaton $\mathcal{A}_\varphi$ such that $[\![\mathcal{A}_\varphi]\!]_{\mathbf{d}} \approx [\![\varphi]\!]_{\mathbf{inf}}$,
2. if φ is negative, a $\overline{\textbf{distance}}$-automaton $\mathcal{A}_\varphi$ such that $[\![\mathcal{A}_\varphi]\!]_{\overline{\mathbf{d}}} \approx [\![\varphi]\!]_{\mathbf{sup}}$.

We give the most interesting inductive cases, the remaining ones can be found in Appendix A.5.

- $\varphi = \mathbf{A}\psi$: The idea is similar to the automata construction for branching-time logic [31]: intuitively, treat ψ as an LTL formula over maximal state subformulas, run a deterministic automaton for ψ on all branches of the tree, and launch automata for the maximal state subformulas of ψ when needed. In our case, we will construct a nested automaton to do this, and in place of a deterministic parity automaton for ψ we will use a history-deterministic **distance**-automaton. Finally, we will convert the nested **distance**-automaton into a **distance**-automaton.
  Suppose that φ is positive (the case that φ is negative is treated dually). Then also ψ is positive. We will construct a nested **distance**-automaton $\mathcal{B}$ such that $[\![\mathcal{B}]\!]_{\mathbf{d}} \approx [\![\varphi]\!]_{\mathbf{inf}}$.
  Let $\varphi_1, \ldots, \varphi_k$ be the maximal state subformulas of the path formula ψ. We see these formulas as atomic propositions, so that the formula ψ can be seen as a Prompt-LTL formula on infinite words over the alphabet 2^k. Apply Theorem 24 to ψ to get a history-deterministic **distance**-automaton $\mathcal{A}_\psi$ over infinite words such that $[\![\mathcal{A}_\psi]\!]_{\mathbf{d}} \approx [\![\psi]\!]_{\mathbf{inf}}$. Then, apply Theorem 20 to $\mathcal{A}_\psi$ to get a **distance**-automaton $\mathcal{A}$ such that $[\![\mathcal{A}]\!]_{\mathbf{d}}(t) = \sup\{[\![\mathcal{A}_\psi]\!]_{\mathbf{d}}(\lambda) : \lambda \in \text{Branches}(t)\}$. The master of $\mathcal{B}$ is $\mathcal{A}$.
  Since ψ is positive, the formulas $\varphi_1, \ldots, \varphi_k$ are either positive or negative. By the induction hypothesis, for every i, if φ_i is positive we construct a **distance**-automaton $\mathcal{A}_i$ such that $[\![\mathcal{A}_i]\!]_{\mathbf{d}} \approx [\![\varphi_i]\!]_{\mathbf{inf}}$; and if φ_i is negative, we construct a $\overline{\textbf{distance}}$-automaton $\mathcal{A}'_i$ such that $[\![\mathcal{A}'_i]\!]_{\overline{\mathbf{d}}} \approx [\![\varphi_i]\!]_{\mathbf{sup}}$. In the latter case, thanks to Theorem 23 we construct a **distance**-automaton $\mathcal{A}_i$ such that $[\![\mathcal{A}_i]\!]_{\mathbf{d}} \approx [\![\varphi_i]\!]_{\mathbf{sup}}$. The slaves of $\mathcal{B}$ are $\mathcal{A}_1, \ldots, \mathcal{A}_k$.
  This completes the construction of $\mathcal{B}$, see Appendix A.5 for its correctness.

- $\varphi = \mathbf{A}^{\leq \mathbf{N}}\psi$: The construction is the same as for $\mathbf{A}\psi$, except for the construction of the master $\mathcal{A}$, in which we replace Theorem 20 by Theorem 21 to account for the possibility of removing a bounded number of paths.
- $\varphi = \exists \mathbf{N}\, \varphi'$: Note that φ cannot be negative. Since φ is positive, also φ' is positive. By the induction hypothesis, there exists a **distance**-automaton $\mathcal{A}_{\varphi'}$ such that $[\![\mathcal{A}_{\varphi'}]\!]_{\mathbf{d}} \approx [\![\varphi']\!]_{\mathbf{inf}}$. Since φ is a positive sentence, we have $[\![\varphi]\!]_{\mathbf{inf}}(t) \in \{0, \omega\}$ for every t. Now,

$$\begin{aligned}[\![\varphi]\!]_{\mathbf{inf}}(t) = 0 &\iff \exists n \in \mathbb{N}, [\![\varphi']\!]_{\mathbf{inf}}(t) \leq n \\ &\iff \exists n \in \mathbb{N}, \text{Eve wins } G_{\mathcal{A}_{\varphi'},t} \text{ for the objective } \mathbf{parity} \cap \mathbf{distance}(n) \\ &\iff \text{Eve wins } G_{\mathcal{A}_{\varphi'},t} \text{ for the objective } \mathbf{parity} \cap \mathbf{fininc}\end{aligned}$$

  The third equivalence follows from Lemma 15. We can now apply Lemma 18 to the **parity** $\cap$ **fininc**-automaton $\mathcal{A}_{\varphi'}$ to get an equivalent parity automaton $\mathcal{A}_{\varphi}$. Then the last item is equivalent to Eve winning the parity game $G_{\mathcal{A}_{\varphi},t}$, which is equivalent to $[\![\mathcal{A}_{\varphi}]\!]_{\mathbf{d}}(t) = 0$ (since $[\![\mathcal{A}_{\varphi}]\!]_{\mathbf{d}}(t) \in \{0, \omega\}$ because $\mathcal{A}_{\varphi}$ has no counter).

This completes the proof of the inductive hypothesis. Finally, since Φ is a sentence, $\mathcal{A}_{\Phi}$ is a parity automaton. Indeed, in the inductive steps, the boundedness operators introduces a counter (if there was not one already), the $\exists N$ step removes the counter, and other operators applied to arguments that do not have a counter produce automata with no counters. ◀

5.4 Model checking Prompt-SL and BOSL

The model-checking problem for Prompt-SL (resp. BOSL) is the following: given a game $\mathcal{G}$ and a sentence Φ of Prompt-SL (resp. BOSL), decide whether $\mathcal{G} \models \Phi$.

As for ATL* with strategy context [32] and Strategy Logic with imperfect information [7], the model-checking problems for both Prompt-SL and BOSL (as well as their combination) can be easily reduced to that of Bound-QCTL* (see Appendix A.6). As a consequence of these reductions and of Theorem 25, we get:

▶ **Theorem 26.** *The model-checking problem is decidable for* Prompt-SL *and* BOSL.

The model-checking procedure is nonelementary, but because Prompt-SL and BOSL subsume SL we know from [33] that no elementary procedure exists. We leave precise complexity analysis for future work.

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 quantitative extensions of SL with decidable model-checking problem.

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, and Orna Kupferman. Promptness in ω-regular automata. In *ATVA*, LNCS 6252, pages 22–36. Springer, 2010.

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

3 Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

4 Benjamin Aminof, Aniello Murano, Sasha Rubin, and Florian Zuleger. Prompt alternating-time epistemic logics. In *KR*, pages 258–267. AAAI Press, 2016. URL: `http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12890`.

5 Mordechai Ben-Ari, Zohar Manna, and Amir Pnueli. The temporal logic of branching time. In *POPL*, pages 164–176, 1981.

6 Raphaël Berthon, Bastien Maubert, and Aniello Murano. Decidability results for ATL* with imperfect information and perfect recall. In *AAMAS*, 2017.

7 Raphaël Berthon, Bastien Maubert, Aniello Murano, Sasha Rubin, and Moshe Y. Vardi. Strategy logic with imperfect information. In *LICS*, 2017.

8 Alessandro Bianco, Fabio Mogavero, and Aniello Murano. Graded computation tree logic. *ACM Transactions on Computational Logic*, 13(3):25:1–25:53, 2012. URL: `http://doi.acm.org/10.1145/2287718.2287725`, `doi:10.1145/2287718.2287725`.

9 Patricia Bouyer, Patrick Gardy, and Nicolas Markey. Weighted strategy logic with boolean goals over one-counter games. In *FSTTCS 2015*, pages 69–83, 2015. URL: `https://doi.org/10.4230/LIPIcs.FSTTCS.2015.69`, `doi:10.4230/LIPIcs.FSTTCS.2015.69`.

10 Arnaud Carayol and Olivier Serre. How good is a strategy in a game with nature? In *LICS*, pages 609–620. IEEE Computer Society, 2015.

11 Krishnendu Chatterjee and Nathanaël Fijalkow. Infinite-state games with finitary conditions. In *CSL*, pages 181–196, 2013. URL: `https://doi.org/10.4230/LIPIcs.CSL.2013.181`, `doi:10.4230/LIPIcs.CSL.2013.181`.

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

13 Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Strategy Logic. *Information and Computation*, 208(6):677–693, 2010. URL: `http://dx.doi.org/10.1016/j.ic.2009.07.004`, `doi:10.1016/j.ic.2009.07.004`.

14 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *ICALP*, 2009.

15 Thomas Colcombet. Fonctions régulières de coût. Habilitation Thesis, 2013.

16 Thomas Colcombet. Regular cost functions, part I: logic and algebra over words. *Logical Methods in Computer Science*, 9(3), 2013.

17 Thomas Colcombet and Nathanaël Fijalkow. The bridge between regular cost functions and ω-regular languages. In *ICALP*, pages 126:1–126:13, 2016. URL: `https://doi.org/10.4230/LIPIcs.ICALP.2016.126`, `doi:10.4230/LIPIcs.ICALP.2016.126`.

18 Thomas Colcombet and Christof Löding. Regular cost functions over finite trees. In *LICS*, pages 70–79, 2010. URL: `https://doi.org/10.1109/LICS.2010.36`, `doi:10.1109/LICS.2010.36`.

19 E. Allen Emerson and Joseph Y. Halpern. "Sometimes" and "Not Never" revisited: On branching versus linear time. In *POPL*, pages 127–140, 1983.

20 Nathanaël Fijalkow, Florian Horn, Denis Kuperberg, and Michał Skrzypczak. Trading bounds for memory in games with counters. In *ICALP*, pages 197–208, 2015. URL: `https://doi.org/10.1007/978-3-662-47666-6_16`, `doi:10.1007/978-3-662-47666-6_16`.

21 Nathanaël Fijalkow and Martin Zimmermann. Cost-Parity and Cost-Street Games. In *FSTTCS*, volume LIPIcs 18, pages 124–135, 2012.

22 Nathanaël Fijalkow and Martin Zimmermann. Parity and Streett games with costs. *Logical Methods in Computer Science*, 10(2), 2014. URL: `https://doi.org/10.2168/LMCS-10(2:14)2014`, `doi:10.2168/LMCS-10(2:14)2014`.

23 Patrick Gardy. *Semantics of Strategy Logic*. Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, June 2017. URL: `https://tel.archives-ouvertes.fr/tel-01561802`.

24 Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In *CSL*, pages 395–410, 2006.

25 Sophia Knight and Bastien Maubert. Dealing with imperfect information in strategy logic. In *SR*, 2015.

26 Denis Kuperberg. Linear temporal logic for regular cost functions. *Logical Methods in Computer Science*, 10(1), 2014.

27 Denis Kuperberg and Michael Vanden Boom. On the expressive power of cost logics over infinite words. In *ICALP*, pages 287–298, 2012.

28 O. Kupferman, P. Madhusudan, P. S. Thiagarajan, and M. Y. Vardi. Open systems in reactive environments: Control and synthesis. In *CONCUR*, LNCS 1877, pages 92–107. Springer, 2000.

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

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

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

32 François Laroussinie and Nicolas Markey. Augmenting ATL with strategy contexts. *Information and Computation*, 245:98–123, 2015.

33 Fabio Mogavero, Aniello Murano, Giuseppe Perelli, and Moshe Y. Vardi. Reasoning about strategies: On the model-checking problem. *ACM Transactions on Computational Logic*, 15(4):34:1–34:47, 2014. URL: `http://doi.acm.org/10.1145/2631917`, `doi:10.1145/2631917`.

34 Fabio Mogavero, Aniello Murano, and Loredana Sorrentino. On promptness in parity games. *Fundamenta Informaticae*, 139(3):277–305, 2015.

35 Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.

36 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989.

37 Martin Zimmermann. Optimal bounds in parametric LTL games. *Theoretical Computer Science*, 493:30–45, 2013. URL: `http://dx.doi.org/10.1016/j.tcs.2012.07.039`, `doi:10.1016/j.tcs.2012.07.039`.

A Appendix

A.1 BOSL semantics

▶ **Definition 4.** The semantics is defined inductively as follows, where φ (resp. ψ) is a cost-SL state (resp. path) formula, $\mathcal{G}$ 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.

$\mathcal{G}, \chi, \rho, n \models p$	if	$p \in \ell(\text{last}(\rho))$
$\mathcal{G}, \chi, \rho, n \models \neg\varphi$	if	$\mathcal{G}, \chi, \rho, n \not\models \varphi$
$\mathcal{G}, \chi, \rho, n \models \varphi \vee \varphi'$	if	$\mathcal{G}, \chi, \rho, n \models \varphi$ or $\mathcal{G}, \chi, \rho, n \models \varphi'$
$\mathcal{G}, \chi, \rho, n \models \exists s\varphi$	if	there exists $\sigma \in \text{Strat}$ s.t. $\mathcal{G}, \chi[s \mapsto \sigma], \rho, n \models \varphi$
$\mathcal{G}, \chi, \rho, n \models (a, s)\varphi$	if	$\mathcal{G}, \chi[a \mapsto \chi(s)], \rho, n \models \varphi$
$\mathcal{G}, \chi, \rho, n \models (a, ?)\varphi$	if	$\mathcal{G}, \chi[a \mapsto ?], \rho, n \models \varphi$
$\mathcal{G}, \chi, \rho, n \models \mathbf{A}\psi$	if	for all $\pi \in \text{Out}(\chi, \rho)$, $\mathcal{G}, \chi, \pi, \lvert\rho\rvert - 1 \models \varphi$
$\mathcal{G}, \chi, \rho, n \models \mathbf{A}^{\leq N}\psi$	if	$\lvert\{\pi \in \text{Out}(\rho, \chi) \mid \mathcal{G}, \chi, \pi, \lvert\rho\rvert - 1, n \not\models \psi\}\rvert \leq n$
$\mathcal{G}, \chi, \rho, n \models \exists N\varphi$	if	there exists $n' \in \mathbb{N}$ such that $\mathcal{G}, \chi, \rho, n' \models \varphi$
$\mathcal{G}, \chi, \pi, i, n \models \varphi$	if	$\mathcal{G}, \chi, \pi_{\leq i} \models \varphi$
$\mathcal{G}, \chi, \pi, i, n \models \neg\psi$	if	$\mathcal{G}, \chi, \pi, i, n \not\models \psi$
$\mathcal{G}, \chi, \pi, i, n \models \psi \vee \psi'$	if	$\mathcal{G}, \chi, \pi, i, n \models \psi$ or $\mathcal{G}, \chi, \pi, i, n \models \psi'$
$\mathcal{G}, \chi, \pi, i, n \models \mathbf{X}\psi$	if	$\mathcal{G}, \chi, \pi, i + 1, n \models \psi$
$\mathcal{G}, \chi, \pi, i, n \models \psi\mathbf{U}\psi'$	if	$\exists j \geq i$ s.t. $\mathcal{G}, \chi, \pi, j \models \psi'$
		and $\forall k$ s.t. $i \leq k < j$, $\mathcal{G}, \chi, \pi, k \models \psi$

A.2 Bound-QCTL* semantics

Given two trees t, t' and an atomic proposition p, we write $t \equiv_p t'$ if they have the same domain τ and for all p' in AP such that $p' \neq p$, for all u in τ, we have $p' \in \ell(u)$ iff $p' \in \ell'(u)$;

▶ **Definition 7.** 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:

$t, u, n \models p$	if	$p \in \ell(u)$
$t, u, n \models \neg\varphi$	if	$t, u, n \not\models \varphi$
$t, u, n \models \varphi \vee \varphi'$	if	$t, u, n \models \varphi$ or $t, u, n \models \varphi'$
$t, u, n \models \mathbf{A}\psi$	if	$\forall \lambda \in \text{Branches}(t, u)$ we have $t, \lambda, n \models \psi$
$t, u, n \models \mathbf{A}^{\leq N}\psi$	if	$\text{Card}(\{\lambda \in \text{Branches}(t, u) : t, \lambda, n \not\models \psi\}) \leq n$
$t, u, n \models \exists p\, \varphi$	if	$\exists t' \equiv_p t$ such that $t', u, n \models \varphi$
$t, u, n \models \exists N\varphi$	if	$\exists n' \in \mathbb{N}$ such that $t, u, n' \models \varphi$,
$t, \lambda, n \models \varphi$	if	$t, \lambda_0, n \models \varphi$
$t, \lambda, n \models \neg\psi$	if	$t, \lambda, n \not\models \psi$
$t, \lambda, n \models \psi \vee \psi'$	if	$t, \lambda, n \models \psi$ or $t, \lambda, n \models \psi'$
$t, \lambda, n \models \mathbf{X}\psi$	if	$t, \lambda_{\geq 1}, n \models \psi$
$t, \lambda, n \models \psi\mathbf{U}\psi'$	if	$\exists j \geq 0$ such that $t, \lambda_{\geq j}, n \models \psi'$
		and $\forall k$ such that $0 \leq k < j$, $t, \lambda_{\geq k}, n \models \psi$
$t, \lambda, n \models \mathbf{F}^{\leq N}\psi$	if	$\exists j$ such that $0 \leq j \leq n$ and $t, \lambda_{\geq j}, n \models \psi$

A.3 Proof of Theorem 21

▶ **Theorem 21.** *Let $\mathcal{A}$ be a history-deterministic* ***distance****-automaton over infinite words. One can construct a* ***distance****-automaton recognising the function over trees*

$$f : t \mapsto \inf \{\max(n, \sup \{[\![\mathcal{A}]\!]_{\boldsymbol{d}}(\lambda) : \lambda \notin B\}) : n \in \mathbb{N}, B \subseteq \mathit{Branches}(t), \mathit{Card}(B) \leq n\}.$$

To prove Theorem 21 we combine $\mathcal{A}$ with the automaton defined in the proof of Lemma 17.

Proof. We write $\mathcal{A} = (Q, q_0, \delta, c)$ for the history-deterministic **distance**-automaton over infinite words, and let us say that the set of labels is $\{1, \ldots, d\} \times \{\epsilon, \mathtt{i}\}$ with d even.

We construct a **distance**-automaton $\mathcal{B}$ for f as follows. The set of states is $Q \times \{p_{0,\epsilon}, p_{0,\mathtt{i}}, p_1\}$, where the semantics of $p_{0,\epsilon}$ and $p_{0,\mathtt{i}}$ is "some path will be skipped" and p_1 means "no path will be skipped". The initial state is $(q_0, p_{0,\epsilon})$. The first component simulates the automaton $\mathcal{A}$ on all branches, while the second acts as follows, with $p_0 = \{p_{0,\epsilon}, p_{0,\mathtt{i}}\}$.

$$\delta = \begin{cases} (p_{0,\epsilon}, a, h) & \text{if } h \text{ contains at most one } p_0 \\ (p_{0,\mathtt{i}}, a, h) & \text{if } h \text{ contains at least two } p_0 \\ (p_1, a, h) & \text{if } h \text{ contains only } p_1 \end{cases}$$

The labelling function c' is

$$\begin{array}{lcll} c'(q, p_{0,\epsilon}) & = & (d, a) & \text{where } c(q) = (o, a) \\ c'(q, p_{0,\mathtt{i}}) & = & (d, \mathtt{i}) & \text{where } c(q) = (o, a) \\ c'(q, p_1) & = & c(q) & \end{array}$$

The proof of correctness is the same as for Lemma 17, substantiating the following claims:

- if $f(t) \leq n$, then $[\![\mathcal{B}]\!]_{\mathbf{d}}(t) \leq n$,
- if $[\![\mathcal{B}]\!]_{\mathbf{d}}(t) \leq n$, then $f(t) \leq \mathrm{Card}(S)^n$.

◀

A.4 Semantics of nested W-automata

▶ **Definition 22.** A *nested W-automaton* with k slaves over (Σ, S)-trees is given by

- a *master automaton* $\mathcal{A}$, which is a W-automaton over $(2^k, S)$-trees, and
- k *slave automata* $(\mathcal{A}_i)_{i \in [k]}$, which are W-automata over (Σ, S)-trees.

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 $\mathcal{A}_i$.

To define the semantics of nested W-automata, we define the corresponding acceptance games. Given a nested W-automaton $\mathcal{B} = (\mathcal{A}, (\mathcal{A}_i)_{i \in [k]})$ and a tree t, we define the acceptance W-game $G_{\mathcal{B},t}$ as follows. Let $\mathcal{A} = (Q, q_0, \delta, c)$.

- The set of vertices is $(Q \times t) \cup (Q \times t \times Q^S)$. The vertices of the form (q, u) are controlled by Eve, those of the form (q, u, h) by Adam.
- The initial vertex is (q_0, r), where r is the root of t.
- The transition relation E is defined as follows

$$\begin{cases} (q, u) \; E \; (q, u, h) & \text{if } (q, \ell(u), h) \in \delta \text{ and } \forall i \in v,\ t_u \text{ is accepted by } \mathcal{A}_i, \\ (q, u, h) \; E \; (h(s), u \cdot s). & \end{cases}.$$

- The labelling function maps (q, u) to $c(q)$, the other vertices are not labelled.

We say that t is accepted by $\mathcal{B}$ if Eve wins the acceptance W-game $G_{\mathcal{B},t}$.

A.5 Proof of Theorem 25

▶ **Theorem 25.** *Let Φ be a sentence of* Bound-QCTL**. We construct a non-deterministic parity automaton $\mathcal{A}_\Phi$ over (Σ, S)-trees such that for every Kripke structure $\mathcal{S}$ over the set of states S, we have $\mathcal{S} \models \Phi$ if, and only if, $\mathcal{A}_\Phi$ accepts the unfolding $t_{\mathcal{S}}$.*

Proof. Let Φ be a sentence and S a finite set of states. Throughout this proof, by trees we mean regular trees, so in particular $\approx$ is understood over such trees.

For each subformula φ of Φ, we construct by induction on φ the following automata:

1. if φ is positive, a **distance**-automaton $\mathcal{A}_\varphi$ such that $[\![\mathcal{A}_\varphi]\!]_{\mathbf{d}} \approx [\![\varphi]\!]_{\mathbf{inf}}$,
2. if φ is negative, a $\overline{\textbf{distance}}$-automaton $\mathcal{A}_\varphi$ such that $[\![\mathcal{A}_\varphi]\!]_{\overline{\mathbf{d}}} \approx [\![\varphi]\!]_{\mathbf{sup}}$.

Here are the constructions or proofs of correctness not present in the body of the paper.

- $\varphi = \mathbf{p}$:
  The formula φ is both positive and negative. Seeing it a positive formula, we define a **distance**-automaton $\mathcal{A}_p$ with one state q_0 and transition function defined as follows:

  $$\delta(q_0, a) = \begin{cases} \top & \text{if } p \in a \\ \bot & \text{otherwise.} \end{cases}$$

  Seeing φ as a negative formula, we define a $\overline{\textbf{distance}}$-automaton $\mathcal{A}_p$ in exactly the same way.
- $\varphi = \neg\varphi'$:
  If φ is negative, then φ' is positive. By definition,

  $$[\![\varphi]\!]_{\mathbf{sup}}(t) = \sup\{n \in \mathbb{N} : t, r, n \models \varphi\} = \inf\{n \in \mathbb{N} : t, r, n \models \varphi'\} - 1 = [\![\varphi']\!]_{\mathbf{inf}}(t) - 1.$$

  In particular, $[\![\varphi]\!]_{\mathbf{sup}} \approx [\![\varphi']\!]_{\mathbf{inf}}$. By induction hypothesis, there exists a **distance**-automaton $\mathcal{A}_{\varphi'}$ such that $[\![\mathcal{A}_{\varphi'}]\!]_{\mathbf{d}} \approx [\![\varphi']\!]_{\mathbf{inf}}$. Thanks to Theorem 23, there exists a $\overline{\textbf{distance}}$-automaton $\mathcal{A}_\varphi$ such that $[\![\mathcal{A}_\varphi]\!]_{\overline{\mathbf{d}}} \approx [\![\mathcal{A}_{\varphi'}]\!]_{\mathbf{d}}$. It follows that $[\![\mathcal{A}_\varphi]\!]_{\overline{\mathbf{d}}} \approx [\![\varphi]\!]_{\mathbf{sup}}$.
  If φ is positive, then φ' is negative, and a similar reasoning applies, using Theorem 23 to turn a **distance**-automaton into an equivalent $\overline{\textbf{distance}}$-automaton.
- $\varphi = \varphi_1 \vee \varphi_2$:
  If φ is positive, then both φ_1 and φ_2 are positive. By induction hypothesis, there exist two **distance**-automata $\mathcal{A}_{\varphi_1}$ and $\mathcal{A}_{\varphi_2}$ such that $[\![\mathcal{A}_{\varphi_1}]\!]_{\mathbf{d}} \approx [\![\varphi_1]\!]_{\mathbf{inf}}$ and $[\![\mathcal{A}_{\varphi_2}]\!]_{\mathbf{d}} \approx [\![\varphi_2]\!]_{\mathbf{inf}}$. We construct $\mathcal{A}_\varphi$ by taking the disjoint union of $\mathcal{A}_{\varphi_1}$ and $\mathcal{A}_{\varphi_2}$ and adding a new initial state that nondeterministically chooses which of $\mathcal{A}_{\varphi_1}$ or $\mathcal{A}_{\varphi_2}$ to execute on the input tree, so that $[\![\mathcal{A}_\varphi]\!]_{\mathbf{d}} = \min\{[\![\mathcal{A}_{\varphi_1}]\!]_{\mathbf{d}}, [\![\mathcal{A}_{\varphi_2}]\!]_{\mathbf{d}}\} \approx \min\{[\![\varphi_1]\!]_{\mathbf{inf}}, [\![\varphi_2]\!]_{\mathbf{inf}}\} = [\![\varphi]\!]_{\mathbf{inf}}$.
  If φ is negative, both φ_1 and φ_2 are negative. The same construction yields an automaton $\mathcal{A}_\varphi$ such that $[\![\mathcal{A}_\varphi]\!]_{\overline{\mathbf{d}}} = \max\left\{[\![\mathcal{A}_{\varphi_1}]\!]_{\overline{\mathbf{d}}}, [\![\mathcal{A}_{\varphi_2}]\!]_{\overline{\mathbf{d}}}\right\} \approx \max\{[\![\varphi_1]\!]_{\mathbf{sup}}, [\![\varphi_2]\!]_{\mathbf{sup}}\} = [\![\varphi]\!]_{\mathbf{sup}}$.
- $\varphi = \mathbf{A}\psi$: The idea is similar to the automata construction for branching-time logic [31]: intuitively, treat ψ as an LTL formula over maximal state subformulas, run a deterministic automaton for ψ on all branches of the tree, and launch automata for the maximal state subformulas of ψ when needed. In our case, we will construct a nested automaton to do this, and in place of a deterministic parity automaton for ψ we will use a history-deterministic **distance**-automaton. Finally, we will convert the nested **distance**-automaton into a **distance**-automaton.

  So, suppose that φ is positive (the case that φ is negative is treated dually). Then also ψ is positive. We will construct a nested **distance**-automaton $\mathcal{B}$ such that $[\![\mathcal{B}]\!]_{\mathbf{d}} \approx [\![\varphi]\!]_{\mathbf{inf}}$. Let $\varphi_1, \ldots, \varphi_k$ be the maximal state subformulas of the path formula ψ. We see these formulas as atomic propositions, so that the formula ψ can be seen as a Prompt-LTL

formula on infinite words over the alphabet 2^k. Apply Theorem 24 to ψ to get a history-deterministic **distance**-automaton $\mathcal{A}_\psi$ over infinite words such that $[\![\mathcal{A}_\psi]\!]_{\mathbf{d}} \approx [\![\psi]\!]_{\mathbf{inf}}$. Then, apply Theorem 20 to $\mathcal{A}_\psi$ to get a **distance**-automaton $\mathcal{A}$ such that $[\![\mathcal{A}]\!]_{\mathbf{d}}(t) = \sup\{[\![\mathcal{A}_\psi]\!]_{\mathbf{d}}(\lambda) : \lambda \in \mathrm{Branches}(t)\}$. The master of $\mathcal{B}$ is $\mathcal{A}$.
Since ψ is positive, the formulas $\varphi_1, \dots, \varphi_k$ are either positive or negative. By the induction hypothesis, for every i, if φ_i is positive we construct a **distance**-automaton $\mathcal{A}_i$ such that $[\![\mathcal{A}_i]\!]_{\mathbf{d}} \approx [\![\varphi_i]\!]_{\mathbf{inf}}$; and if φ_i is negative, we construct a $\overline{\textbf{distance}}$-automaton $\mathcal{A}'_i$ such that $[\![\mathcal{A}'_i]\!]_{\overline{\mathbf{d}}} \approx [\![\varphi_i]\!]_{\mathbf{sup}}$. In the latter case, thanks to Theorem 23 we construct a **distance**-automaton $\mathcal{A}_i$ such that $[\![\mathcal{A}_i]\!]_{\mathbf{d}} \approx [\![\varphi_i]\!]_{\mathbf{sup}}$. The slaves of $\mathcal{B}$ are $\mathcal{A}_1, \dots, \mathcal{A}_k$.
This completes the construction of $\mathcal{B}$. We now prove that $[\![\mathcal{B}]\!]_{\mathbf{d}} \approx [\![\varphi]\!]_{\mathbf{inf}}$. For the sake of simplicity, we assume that $[\![\mathcal{A}_\psi]\!]_{\mathbf{d}} = [\![\psi]\!]_{\mathbf{d}}$ and $[\![\mathcal{A}_i]\!]_{\mathbf{d}} = [\![\varphi_i]\!]$ for every i, i.e. we replace $\approx$ by equality. This simplification does not affect the arguments and makes the proof easier to read.

- We prove that $[\![\varphi]\!]_{\mathbf{inf}} \leq [\![\mathcal{B}]\!]_{\mathbf{d}}$. It is sufficient to show that $[\![\mathcal{B}]\!]_{\mathbf{d}}(t) \leq n$ implies $[\![\varphi]\!]_{\mathbf{inf}}(t) \leq n$. A run of $\mathcal{B}$ on t witnessing that $[\![\mathcal{B}]\!]_{\mathbf{d}}(t) \leq n$ yields for each branch λ a run of $\mathcal{A}_\psi$ such that $[\![\mathcal{A}_\psi]\!]_{\mathbf{d}}(\lambda) \leq n$. The slave automata diligently check that the atomic propositions $\varphi_1, \dots, \varphi_k$ have been correctly used, so indeed $t, \lambda, n \models \psi$, thus $[\![\varphi]\!]_{\mathbf{inf}}(t) \leq n$.
- We prove that $[\![\mathcal{B}]\!]_{\mathbf{d}} \leq [\![\varphi]\!]_{\mathbf{inf}}$. It is sufficient to show that $[\![\varphi]\!]_{\mathbf{inf}}(t) \leq n$ implies $[\![\mathcal{B}]\!]_{\mathbf{d}}(t) \leq n$. By the semantics of φ for all branches λ of t we have $t, \lambda, n \models \psi$. This yields a run of $\mathcal{B}$ on t witnessing that $[\![\mathcal{B}]\!]_{\mathbf{d}}(t) \leq n$.

Finally, applying Theorem 23 to the nested **distance**-automaton $\mathcal{B}$ we get a **distance**-automaton $\mathcal{A}_\varphi$ such that $[\![\mathcal{A}_\varphi]\!]_{\mathbf{d}} \approx [\![\mathcal{B}]\!]_{\mathbf{d}}$.

- $\varphi = \exists \mathbf{p}\, \varphi'$:
If φ is positive, then φ' is positive. In this case unravelling the definitions we have

$$[\![\varphi]\!]_{\mathbf{inf}}(t) = \inf\{[\![\varphi']\!]_{\mathbf{inf}}(t') : t' \equiv_p t\}.$$

By the induction hypothesis, there exists a **distance**-automaton $\mathcal{A}_{\varphi'}$ such that $[\![\mathcal{A}_{\varphi'}]\!]_{\mathbf{d}} \approx [\![\varphi']\!]_{\mathbf{inf}}$. We obtain a **distance**-automaton $\mathcal{A}_\varphi$ by performing the usual projection operation. Everything remains the same, but the transition relation: (q, a, h) is in the new transition relation if there exists a' such that $a' \equiv_p a$ and (q, a', h) is in δ, where $a' \equiv_p a$ if for all p' in AP such that $p' \neq p$, we have $p' \in a'$ if, and only if, $p \in a$.
If φ is negative, then φ' is negative. The same reasoning and construction applies in this case, with

$$[\![\varphi]\!]_{\mathbf{sup}}(t) = \sup\{[\![\varphi']\!]_{\mathbf{sup}}(t') : t' \equiv_p t\}.$$

This completes the proof of the inductive hypothesis. Finally, since Φ is a sentence, $\mathcal{A}_\Phi$ is a parity automaton. Indeed, in the inductive steps, the boundedness operators introduces a counter (if there was not one already), the $\exists N$ step removes the counter, and every other operator applied to arguments that do not have a counter produces an automaton with no counters. ◀

A.6 Reductions for Prompt-SL and BOSL

Models transformation. We first define for every game $\mathcal{G}$ a Kripke structure $\mathcal{S}_\mathcal{G}$ and a bijection $\rho \mapsto u_\rho$ between the set of finite plays starting in the initial vertex and the set of nodes in $t_{\mathcal{S}_\mathcal{G}}$. We consider propositions $\mathrm{AP}_v = \{p_v \mid v \in V\}$, that we assume to be disjoint from AP. Define the Kripke structure $\mathcal{S}_\mathcal{G} = (S, R, s_0, \ell')$ where

- $S = \{s_v \mid v \in V\}$,
- $R = \{(s_v, s_{v'}) \mid \exists \mathbf{c} \in \mathrm{Act}^{\mathrm{Ag}} \text{ s.t. } \Delta(v, \mathbf{c}) = v'\} \subseteq S^2$,
- $s_0 = s_{v_0}$, and
- $\ell'(s_v) = \ell(v) \cup \{p_v\} \subseteq \mathrm{AP} \cup \mathrm{AP}_v$.

For every finite play $\rho = v_0 \dots v_k$, define the node $u_\rho = s_{v_0} \dots s_{v_k}$ in $t_{\mathcal{S}_\mathcal{G}}$ (which exists, by definition of $\mathcal{S}_\mathcal{G}$ and of tree unfoldings). Note that the mapping $\rho \mapsto u_\rho$ defines a bijection between the set of paths from v_0 and the set of nodes in $t_{\mathcal{S}_\mathcal{G}}$.

Formulas translation. Given a game $\mathcal{G}$ and a formula φ of Prompt-SL or BOSL, we define a Bound-QCTL* formula (φ) such that $\mathcal{G} \models \varphi$ if and only if $\mathcal{S}_\mathcal{G} \models (\varphi)$. More precisely, this translation is parameterised with a partial function $f : \mathrm{Ag} \rightharpoonup \mathrm{Var}$ which records bindings of agents to strategy variables. Suppose that $\mathrm{Act} = \{c_1, \dots, c_l\}$. We define the two functions $(\cdot)^f_s$ and $(\cdot)^f_p$ by mutual induction on, respectively, state formulas φ and path formulas ψ.

Here is the definition of $(\cdot)^f_s$ for state formulas:

$$
\begin{aligned}
(p)^f_s &= p & (\neg\varphi)^f_s &= \neg(\varphi)^f_s \\
(\varphi_1 \vee \varphi_2)^f_s &= (\varphi_1)^f_s \vee (\varphi_2)^f_s & (\exists N \varphi)^f_s &= \exists N (\varphi)^f_s \\
((a, s)\varphi)^f_s &= (\varphi)^{f[a \mapsto s]}_s & ((a, ?)\varphi)^f_s &= (\varphi)^{f[a \mapsto ?]}_s \\
(\exists s \varphi)^f_s &= \exists p^s_{c_1} \dots \exists p^s_{c_l} . \varphi_{\mathrm{str}}(s) \wedge (\varphi)^f_s, & \text{where } \varphi_{\mathrm{str}}(s) &= \mathbf{AG} \bigvee_{c \in \mathrm{Act}} (p^s_c \wedge \bigwedge_{c' \neq c} \neg p^s_{c'}) \\
(\mathbf{A}\psi)^f_s &= \mathbf{A}(\psi_{\mathrm{out}}(f) \to (\psi)^f_p) & (\mathbf{A}^{\leq N}\psi)^f_s &= \mathbf{A}^{\leq N}(\psi_{\mathrm{out}}(f) \to (\psi)^f_p)
\end{aligned}
$$

where

$$
\psi_{\mathrm{out}}(f) = \mathbf{G} \bigwedge_{v \in V} \left(p_v \to \bigvee_{\mathbf{c} \in \mathrm{Act}^{\mathrm{Ag}}} \left(\bigwedge_{a \in dom(f)} p^{f(a)}_{\mathbf{c}_a} \wedge \mathbf{X} p_{\Delta(v, \mathbf{c})} \right) \right),
$$

and for path formulas:

$$
\begin{aligned}
(\varphi)^f_p &= (\varphi)^f_s & (\neg\psi)^f_p &= \neg(\psi)^f_p \\
(\varphi_1 \vee \varphi_2)^f_p &= (\varphi_1)^f_p \vee (\varphi_2)^f_p & (\mathbf{X}\psi)^f_p &= \mathbf{X}(\psi)^f_p \\
(\psi \mathbf{U} \psi')^f_p &= (\psi)^f_p \mathbf{U} (\psi')^f_p & (\mathbf{F}^{\leq N}\psi)^f_p &= \mathbf{F}^{\leq N}(\psi)^f_p
\end{aligned}
$$

One can prove the following lemma, where φ is either a Prompt-SL or a BOSL formula. The translation is essentially the same as in [32] and [7], and the cases for the new operators should be clear from their semantics.

▶ **Lemma 26.** *Suppose that $dom(f) = dom(\chi) \cap Ag$ and for all $a \in dom(f)$, $f(a) = x$ implies $\chi(a) = \chi(x)$. Then*

$$\mathcal{G}, \chi, \rho, n \models \varphi \quad \textit{if and only if} \quad t_{\mathcal{S}_\mathcal{G}}, u_\rho, n \models (\varphi)^f.$$

Applying this to a sentence Φ, any assignment χ, the initial vertex v_0 of $\mathcal{G}$, any bound n and the empty function $\emptyset$, we get that

$$\mathcal{G} \models \varphi \quad \text{if and only if} \quad t_{\mathcal{S}_\mathcal{G}} \models (\varphi)^\emptyset.$$