Nondeterministic Strategies and their Refinement in Strategy Logic

Giuseppe De Giacomo\textsuperscript{1}, Bastien Maubert\textsuperscript{2} and Aniello Murano\textsuperscript{2}

\textsuperscript{1}Sapienza Università di Roma
\textsuperscript{2}Università degli Studi di Napoli Federico II
degiacomo@diag.uniroma1.it, bastien.maubert@gmail.com, murano@na.infn.it

Abstract

Nondeterministic strategies are strategies (or protocols, or plans) that, given a history in a game, assign a set of possible actions, all of which are winning. An important problem is that of refining such strategies. For instance, given a nondeterministic strategy that allows only safe executions, refine it to, additionally, eventually reach a desired state of affairs. We show that strategic problems involving strategy refinement can be solved elegantly in the framework of Strategy Logic (SL), a very expressive logic to reason about strategic abilities. Specifically, we introduce an extension of SL with nondeterministic strategies and an operator expressing strategy refinement. We show that model checking this logic can be done at no additional computational cost with respect to standard SL, and can be used to solve a variety of problems such as synthesis of maximally permissive strategies or maximally permissive Nash equilibria.

1 Introduction

Program synthesis is a fundamental problem in different areas of computer science such as formal methods, artificial intelligence or control theory. The objects to be synthesised are sometimes called systems, plans, strategies, protocols or controllers, but essentially they are always functions that define which transition to take, or which action to perform, after a given finite run of the system. The basic problem is, in essence, to synthesise (if possible) a finite representation of such a function such that the set of generated runs satisfies a given specification. When the system is closed, i.e., when the evolution of the system depends solely on its actions, a plan is essentially a sequence of actions, and synthesis is usually rather easy. For open systems that interact with an environment in a game-like manner, the synthesised program has to enforce the specification no matter how the environment behaves; it is thus no longer a sequence of actions but a tree-like object (that we will call strategy), and the synthesis problem is more challenging. Variants of this problem include supervisory control (Ramadge and Wonham 1987), reactive synthesis (Pnueli and Rosner 1989) and nondeterministic planning (Geffner and Bonet 2013).

One important difference between the different variants of the problem is that some consider deterministic strategies while others allow for nondeterministic ones, which prescribe for each finite run a set of actions to choose from nondeterministically. In particular this is the case in synthesis with reactive environments (Kupferman et al. 2000), where the environment can use nondeterministic strategies, and in supervisory control (Ramadge and Wonham 1987) where the synthesised controller not only is nondeterministic, but also should allow as many behaviours as possible without breaking the specification. Indeed, in the classic setting that considers only safety objectives, the existence of a maximally permissive controller is always guaranteed (Wonham 2014). Note that it is the only type of objectives for which this holds (Bernet, Janin, and Walukiewicz 2002).

The notion of maximal permissiveness is important also because a more permissive system can more easily be refined latter on to make it satisfy additional requirements. For instance, given a nondeterministic strategy that allows only safe executions, one may want to refine it to, in addition, be sure to eventually reach a desired state of affairs. Maximally permissive strategies have thus been studied in generalisations of supervisory control to specifications that go beyond safety (Pinchinat and Riedweg 2005), but also in computational game theory in the formal methods community (Bernet, Janin, and Walukiewicz 2002; Bouyer et al. 2011) and in the community of reasoning about actions (De Giacomo, Lopespérance, and Muise 2012; Baniashehemi, De Giacomo, and Lesperance 2018; De Giacomo, Patrizi, and Sardiña 2013).

The literature thus contains a plethora of different synthesis problems. To provide a general framework to specify and solve such problems, Strategy Logic was introduced in (Chatterjee, Henzinger, and Piterman 2010), and extended to the multi-agent setting in (Mogavero et al. 2014). This very expressive logic treats strategies as first-order objects, and can express a variety of complex synthesis problems such as distributed synthesis, synthesis of Nash equilibria or rational synthesis (Berthom et al. 2020). However until now it has never been considered with nondeterministic strategies, and thus could not naturally capture problems such as supervisory control or synthesis with reactive environments.

In this work we introduce an extension of Strategy Logic that allows for nondeterministic strategies and contains a refinement operator \( \preceq \). In the resulting logic, called \( SL^\prec \), formula \( x \preceq y \) means that strategy \( x \) refines strategy \( y \), or in other words, that \( y \) is more permissive than \( x \). Because quantification on deterministic strategies and maximal permissiveness can be expressed using quantification on
nondeterministic strategies and the refinement operator, $\mathsf{SL}^\prec$ strictly extends $\mathsf{SL}$ and can capture, in addition, all the synthesis problems mentioned above. It can also express module checking (Kupferman, Vardi, and Wolper 2001), and in the context of multi-agent systems, $\mathsf{SL}^\prec$ can be used to synthesise maximally permissive Nash equilibria or subgame-perfect equilibria.

When the specification is expressed in a branching-time logic such as $\mathsf{CTL}^\omega$, formulas such as $\mathsf{E}\psi_1 \land \mathsf{E}\psi_2$ express the existence of runs satisfying $\psi_1$ and others satisfying $\psi_2$. When both the system and the environment use nondeterministic strategies, it does not say however that the system can choose to enforce $\psi_1$ or $\psi_2$ independently of what the environment does. We show that this property, which we call unilateral forcing, can be expressed in $\mathsf{SL}^\prec$.

We solve the model-checking problem for $\mathsf{SL}^\prec$, and establish that it is no harder than for classic $\mathsf{SL}$. As it is usually the case for this kind of logics, the model-checking algorithm for $\mathsf{SL}^\prec$ can provide finite witness strategies when they exist, such that we obtain a unified procedure to solve all the synthesis problems from the literature mentioned above, with optimal asymptotic complexity, and solve new ones, such as nondeterministic synthesis with unilateral forcing, of synthesis of maximally permissive Nash equilibria.

Plan. We introduce nondeterministic strategies and their refinement in Section 2. In Section 3 we present the logic $\mathsf{SL}^\prec$, and in Section 4 we show how it captures a number of problems, some existing and others new. The model-checking procedure for $\mathsf{SL}^\prec$ is presented in Section 5, and we conclude in Section 6.

2 Non-deterministic strategies

We first recall classic concurrent game structures, non-deterministic strategies, and the notion of strategy refinement.

2.1 Notations

Let $\Sigma$ be an alphabet. A finite (resp. infinite) word over $\Sigma$ is an element of $\Sigma^*$ (resp. $\Sigma^\omega$). The length of a finite word $w = w_0w_1\ldots w_n$ is $|w| := n + 1$, and 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 $w$, $w_{i:j}$ is the prefix of $w$ that ends at position $i$ and $w_{i:i}$ is the suffix that starts at position $i$. We write $w \preceq w'$ if $w$ is a prefix of $w'$, and $\text{pref}(w)$ is the set of finite prefixes of word $w$. The domain of a mapping $f$ is written $\text{dom}(f)$.

2.2 Concurrent game structures

Let $(\Lambda, \Pi, a)$ be a finite non-empty set of atomic propositions, and $\Lambda$ a finite non-empty set of agents or players.

Definition 1. A concurrent game structure (or CGS) is a tuple $\mathcal{G} = (\Lambda, \Pi, a, \ell, v_0)$ where

- $\Lambda$ is a finite non-empty set of actions,
- $\Pi$ is a finite non-empty set of positions,
- $E : \Pi \times \Pi\Lambda \rightarrow \Pi$ is a transition function,
- $\ell : \Pi \rightarrow 2^\Lambda$ a labelling function, and
- $v_0 \in \Pi$ is an initial position.

In a position $v \in \Pi$, where atomic propositions $\ell(v)$ hold, each player $a$ chooses an action $c_a \in \Lambda$, and the game proceeds to position $E(v, c_a)$, where $c_a \in \Lambda\Lambda$ stands for the joint action $(c_a)_{a \in \Lambda}$.

Given a joint action $c = (c_a)_{a \in \Lambda}$ and $a \in \Lambda$, we let $c_a$ denote $c_a$. A finite (resp. infinite) play is a finite (resp. infinite) word $\rho = v_0\ldots v_n$ (resp. $\pi = v_0v_1\ldots$) such that $v_0 = v$, and for every $i$ such that $0 \leq i < |\rho| - 1$ (resp. $i \geq 0$), there exists a joint action $c$ such that $E(v_i, c) = v_{i+1}$. Given two finite plays $\rho$ and $\rho'$, we say that $\rho'$ is a continuation of $\rho$ if $\rho' \in \rho \cdot \Pi^\omega$, and we write $\text{Cont}(\rho)$ for the set of continuations of $\rho$.

Remark 1. Recall that turn-based game structures can be seen as a special case of concurrent game structures in which the state space is partitioned between players, and in each position, only the actions of the player to whom it belongs have an impact (Alur, Henzinger, and Kupferman 2002).

2.3 Strategy refinement

Given a CGS $\mathcal{G}$, a non-deterministic strategy, or strategy for short, for a player $a$ is a function $\sigma : \text{Cont}(v_0) \rightarrow 2^\Lambda \setminus \emptyset$ that maps each finite play in $\mathcal{G}$ to a nonempty finite set of actions that the player may choose from after this finite play. A strategy $\sigma$ is deterministic if for every finite play $\rho$, $\sigma(\rho)$ is a singleton. We let $\text{Str}$ denote the set of all (nondeterministic) strategies, and $\text{Str}^d \subset \text{Str}$ the set of deterministic ones (note that these sets depend on the CGS under consideration).

Formulas of our logic $\mathsf{SL}^\prec$ will be evaluated at the end of a finite play $\rho$ (which can be simply the initial position of the game), and since $\mathsf{SL}^\prec$ contains only future-time temporal operators, the only relevant part of a strategy $\sigma$ when evaluating a formula after finite play $\rho$ is its definition on continuations of $\rho$. We thus define the restriction of $\sigma$ to $\rho$ as the restriction of $\sigma$ to $\rho \cdot \Pi^\omega$, that we write $\sigma|_{\rho} : \text{Cont}(\rho) \rightarrow 2^\Lambda \setminus \emptyset$.

We will then say that a strategy $\sigma$ refines another strategy $\sigma'$ after a finite play $\rho$ if the first one is more restrictive than the second one on continuations of $\rho$. More formally:

Definition 2. A strategy $\sigma$ refines strategy $\sigma'$ after finite play $\rho$, written $\sigma \preceq_{\rho} \sigma'$, if for every $\rho' \in \text{Cont}(\rho)$, $\sigma|_{\rho'} \subseteq \sigma'|_{\rho'}$. We simply say that $\sigma$ refines $\sigma'$ if it refines it after the initial position $v_0$, and in that case we write $\sigma \preceq \sigma'$.

3 Strategy Logic with refinement

In this section we introduce $\mathsf{SL}^\prec$, which extends $\mathsf{SL}$ with nondeterministic strategies, an outcome quantifier that quantifies over possible outcomes of a strategy profile, and more importantly, a refining operator that expresses that a strategy refines another. We first fix some basic notations.

3.1 Syntax

In addition to the sets of propositions $\Pi$ and agents $\Lambda$, we now fix $\text{Var}$, a finite non-empty set of variables.

Definition 3. The syntax of $\mathsf{SL}^\prec$ is defined by the following grammar:

$$
\varphi ::= p \mid \neg \varphi \mid \varphi \lor \varphi \mid \exists x \varphi \mid x \leq y \mid (a, x)\varphi \mid E\varphi
$$

$$
\psi ::= \varphi \mid \neg \psi \mid \psi \lor \psi \mid X\psi \mid \psi U\psi
$$

where $p \in \Pi$, $x, y \in \text{Var}$ and $a \in \Lambda$.
Formulas of type $\varphi$ are called state formulas, those of type $\psi$ are called path formulas, and $\text{SL}^\omega$ consists of all state formulas.

Temporal operators, $X$ (next) and $U$ (until), have the usual meaning. The refinement operator expresses that the strategy denoted by a variable $x$ is more restrictive than another one or, in other words, that it allows less behaviours: $x \leq y$ reads as “strategy $x$ refines strategy $y$”. The strategy quantifier $\exists x$ has its usual meaning, except that it now quantifies on nondeterministic strategies: $\exists x \varphi$ reads as “there exists a nondeterministic strategy $x$ such that $\varphi$ holds”, where $x$ is a strategy variable. As usual, the binding operator $(a_x)$ assigns a strategy to an agent, and $(a, x)\varphi$ reads as “when agent $a$ plays strategy $x$, $\varphi$ holds”. Finally, the outcome quantifier $E$ quantifies on outcomes of strategies currently in use: $E \psi$ reads as “$\psi$ holds in some outcome of the strategies currently used by the players”.

We use usual abbreviations $\top := p \lor \neg p$, $\bot := \neg \top$, $\varphi \rightarrow \varphi' := \neg \varphi \lor \varphi'$, $\varphi \leftrightarrow \varphi' := \varphi \rightarrow \varphi' \land \varphi' \rightarrow \varphi$, $A \psi := \neg E \neg \psi$, $F \psi := T \psi \land \psi$ and $\forall \varphi := \neg \exists \neg \varphi$.

For every formula $\varphi \in \text{SL}^\omega$, we let $\text{free}(\varphi)$ be the set of variables that appear free in $\varphi$, i.e., that appear out of the scope of a strategy quantifier. A formula $\varphi$ is a sentence if $\text{free}(\varphi)$ is empty. Finally, we let the size $|\varphi|$ of a formula $\varphi$ be the number of symbols in $\varphi$.

### 3.2 Semantics

$\text{SL}^\omega$ formulas are interpreted in a CGS, and the semantics makes use of the following additional notions.

An assignment $\chi : \text{Ag} \cup \text{Var} \rightarrow \text{Str}$ is a partial function that assigns a strategy to each player and strategy variable in its domain. For an assignment $\chi$, player $a$ and strategy $\sigma$, $\chi[a \rightarrow \sigma]$ is the assignment of domain $\text{dom}(\chi) \cup \{a\}$ that maps $a$ to $\sigma$ and is equal to $\chi$ on the rest of its domain, and $\chi[x \rightarrow \sigma]$ is defined similarly, where $x$ is a variable. We write $\text{Ag}(\chi)$ for $\text{dom}(\chi) \cap \text{Ag}$, and $\text{Var}(\chi)$ for $\text{dom}(\chi) \cap \text{Var}$. An assignment is variable-complete for a formula $\varphi \in \text{SL}^\omega$ if $\text{free}(\varphi) \subseteq \text{Var}(\chi)$.

For an assignment $\chi$ and a finite play $\rho$, we define the outcomes of $\chi$ from $\rho$ as the set of infinite plays that start with $\rho$ and are then extended by letting players follow the strategies assigned by $\chi$. Formally, we define $\text{Out}(\chi, \rho)$ as the set of plays of the form $\rho \cdot v_1 v_2 \ldots$ such that for all $i \geq 0$, there exists $c$ such that for all $a \in \text{Ag}(\chi)$, it holds that $c_a \in \chi(a)(\rho \cdot v_1 \ldots v_i)$ and $v_{i+1} = E(v_i, c)$, with $v_0 = \text{last}(\rho)$.

**Definition 4.** The semantics of a state (resp. path) formula is defined on a CGS $G$, an assignment $\chi$ that is variable-complete for $\varphi$, and a finite play $\rho$ (resp. an infinite play $\pi$ and an index $i \in \mathbb{N}$). The definition by mutual induction is as follows (we omit Boolean cases):

- $G, \chi, \rho \models p$ if $p \in \ell(\text{last}(\rho))$
- $G, \chi, \rho \models \exists x \varphi$ if $\exists \sigma \in \text{Str} \text{ s.t. } G, \chi[x \rightarrow \sigma], \rho \models \varphi$
- $G, \chi, \rho \models x \leq y$ if $\chi(x)$ refines $\chi(y)$ after $\rho$
- $G, \chi, \rho \models (a, x)\varphi$ if $G, \chi[a \rightarrow \chi(x)], \rho \models \varphi$
- $G, \chi, \rho \models E \psi$ if $\exists \pi \in \text{Out}(\chi, \rho) \text{ s.t. } G, \chi, \pi, [\rho] \models 1 \models \psi$

### 3.3 Outcomes as strategy refinements

Before stating our main result, we point out that the outcome quantifier $E$ is tightly linked to strategy refinement. More precisely, selecting an individual outcome of an assignment $\chi$ amounts to choosing a deterministic strategy $\sigma_a$ for each player $a$ such that $\sigma_a \preceq \chi(a)$ for each $a \in \text{Ag}(\chi)$. Indeed, fixing a deterministic strategy for each player fixes a unique play, and the refinement constraint ensures that this play follows the nondeterministic strategies assigned by $\chi$.

**Lemma 1.** Let $\psi$ be an LTL formula, $G$ a CGS, $\rho$ a finite play in $G$ and $\chi$ an assignment such that $\chi(a) = \chi(x_a)$ for each $a \in \text{Ag}(\chi)$. It holds that

$$G, \chi, \rho \models E \psi \iff G, \chi, \rho \models \exists a \in \text{Ag} \bigwedge_{a \in \text{Ag}(\chi)} y_a \leq x_a \land (a, y_a)_{a \in \text{Ag}} A \psi$$

Notice that in the last formula, $A \psi$ could be replaced by $E \psi$: indeed, after each agent $a$ has been bound to deterministic strategy $y_a$, there exists a unique outcome.
3.4 Main result

To state precisely the complexity of model checking \( SL^\prec \) we need the notion of simulation depth, introduced in (Berthom et al. 2020), which is meant to count how many nested simulation procedures have to be performed in the automata construction, to change alternating automata into nondeterministic ones. The simulation depth \( \text{sd}(\varphi) \) of a formula \( \varphi \) is a pair \( (k,x) \) where \( k \) is the number of nested simulations needed in the automata construction for \( \varphi \), and \( x \in \{ \text{nd}, \text{alt} \} \) is the type of automaton obtained (nondeterministic or alternating). We write \( \text{sd}_k(\varphi) \) and \( \text{sd}_x(\varphi) \) for, respectively, the first and second component of \( \text{sd}(\varphi) \). The inductive definition is as follows:

\[
\begin{align*}
\text{sd}(p) & := (0, \text{nd}) \\
\text{sd}(\neg \varphi) & := \text{sd}_k(\varphi), \text{alt} \\
\text{sd}(\varphi_1 \lor \varphi_2) & := (\max_{i \in \{1,2\}} \text{sd}_k(\varphi_i), x), \\
\text{sd}(\exists x \varphi) & := (k, \text{nd}), \\
\text{sd}(\forall x \varphi) & := (k, \text{alt}), \\
\text{sd}(E\psi) & := ((0, \text{nd}) \\
& \quad \cup (\max_{\varphi \in \text{max}(\psi)} \text{sd}_y(\varphi), \text{alt}) \text{ otherwise}) \quad \text{if } \psi \in \text{LTL}
\end{align*}
\]

We say that a formula \( \varphi \) has simulation depth \( k \) if \( \text{sd}_k(\varphi) = k \). We now state the following result, which is proved in Section 5.

**Theorem 2.** Model checking \( SL^\prec \) is \( (k + 1) \)-EXPTIME-complete for formulas of simulation depth at most \( k \).

**Remark 2.** Notice that defining \( \exists^d x \varphi \) as \( \exists x (\text{det}(x) \land \varphi) \), where \( \text{det}(x) = \forall y (y \leq x \rightarrow x \leq y) \), introduces a simulation between \( \exists x \) and \( \forall y \). This exponential can be avoided by considering \( \exists^d x \varphi \) as a basic construct in the syntax, whose translation to QCTL* is essentially the same as that of \( \exists x \varphi \) (see Section 5.1), and for which simulation depth is defined as for \( \exists x \varphi \).

In the following sections we show how \( SL^\prec \) captures a number of important problems related to strategy synthesis and nondeterministic strategies.

4 Applications of \( SL^\prec \)

In this section we show how our framework captures generalisations of the classical LTL synthesis problem to the context of nondeterministic strategies.

4.1 Reactive synthesis

We first recall the standard LTL synthesis problem as defined in (Pnueli and Rosner 1989): consider a set of input variables \( I \) controlled by the environment and a set of output variables \( O \) controlled by the system. In each round, first the environment chooses a valuation of the inputs \( i_k \in \{0,1\} \) (called input), and then the system reacts by choosing a valuation on the output variables \( o_k \in \{0,1\} \) (called output); an

infinite word over \( 2^{I \cup O} \) is called an execution. The system has perfect recall, meaning that its choices can depend on all previous choices of the environment, and a strategy for the system is thus a function \( S : (2^I)^+ \rightarrow 2^O \). Given an infinite sequence of inputs \( w = i_0i_1i_2 \ldots \in (2^I)^\omega \), we define the execution \( S(w) = o_0o_1o_2 \ldots \), where, for each \( k \geq 0, o_k \neq S(i_0 \ldots i_k) \).

The \( LTL \) synthesis problem consists in, given \( I, O \) and an LTL formula \( \psi \) over atoms \( I \cup O \), synthesising a (finite representation of a) system \( S : (2^I)^+ \rightarrow 2^O \) such that for all \( w = i_0i_1i_2 \ldots \in (2^I)^\omega \) it holds that \( S(w) = \psi \). This problem can easily be coded in Strategy Logic: one builds a turn-based game arena \( G_{I,O} \) (which can be represented as a CGS, see Remark 1) with two players, E (for Environment) and S (for System) in which first the environment chooses an input \( i \), then the system chooses an output \( o \), reaching a position labelled with atoms \( i \cup o \) and in which it is the environment’s turn to play. The LTL synthesis problem for \( (I, O, \psi) \) can then be solved by model-checking on \( G_{I,O} \) the \( SL^\prec \) formula

\[
\varphi_{\text{synth}}^d(\psi) := \exists^d x. \forall^d y. (S,x)(E,y)A\psi
\]

Note that this really solves the synthesis problem as existing model-checking algorithms for Strategy Logic can synthesise witness strategies (when they exist) for strategy variables existentially quantified at the beginning of the formula. Rewriting \( \varphi_{\text{synth}}^d(\psi) \) as \( \exists^d x. \neg \exists^d y. (S,x)(E,y)E\neg\psi \) we see that it has simulation depth 1 (see Remark 2) and thus can be solved by the model-checking algorithm for \( SL^\prec \) in doubly exponential time (Theorem 2), which is optimal since LTL synthesis is 2EXPTIME-complete (Pnueli and Rosner 1989).

Note also that in the case of deterministic strategies, fixing a strategy for each player fixes a unique outcome, and thus \( A\psi \) in the formula above could be replaced by \( E\psi \) without affecting the semantics. Also, once a deterministic strategy \( x \) is fixed for \( S \), each deterministic strategy \( y \) for \( E \) fixes an outcome of strategy \( x \), and each outcome of \( x \) can be obtained by fixing a deterministic strategy \( y \) for \( E \). It then follows by the semantics of the outcome quantifier \( A \) that, when considering only deterministic strategies, \( \varphi_{\text{synth}}^d(\psi) \) is equivalent to \( \exists^d x (S,x)A\psi \).

4.2 Nondeterministic synthesis

Considering nondeterministic strategies, as we do, does not change anything for classical LTL synthesis. Indeed, consider the following formula:

\[
\varphi_{\text{synth}}^nd(\psi) := \exists x. \forall y. (S,x)(E,y)A\psi
\]

which differs from \( \varphi_{\text{synth}}^d(\psi) \) only in that it now allows for nondeterministic strategies. It is easy to check that:

**Proposition 3.** For every LTL formula \( \psi \),

\[
G_{I,O} \models \varphi_{\text{synth}}^d(\psi) \iff G_{I,O} \models \varphi_{\text{synth}}^nd(\psi)
\]

However it makes a difference if, instead of considering only universal satisfaction of an LTL formula on all outcomes, we consider other forms of branching-time specifications, in particular specifications that require existence of
outcomes satisfying different properties, sometimes called possibility requirements (Kupferman and Vardi 1997).

For instance,
\[ \varphi_1 := \exists x y (S, x) \forall y y (E, y) (G p \land EF \neg p) \]
is always false, because a pair of deterministic strategies for the system and the environment determine a unique outcome that cannot satisfy both \( G p \) and \( EF \neg p \). However
\[ \varphi_1' := \exists x y (S, x) \forall y (E, y) (G p \land EF \neg p) \]
can be true: for instance in model \( G_1 \) depicted in Figure 1, where the initial position is marked by an incoming arrow \( G \), the system can use the strategy that allows both \( a \) and \( b \) in the initial position is a witness for the satisfaction of \( \varphi_1' \). We write this strategy \( \sigma_{a,b} \), and we let \( \sigma_a \) and \( \sigma_b \) be the deterministic strategies that allow respectively only \( a \) and only \( b \) in the initial position (which is the only relevant part in this CGS).

If \( \varphi \) is a CTL* formula, we now let
\[ \varphi^{nd}_{\text{synth}}(\varphi) := \exists x. \forall y. (S, x) (E, y) \varphi \]
We can combine the two kinds of specifications (existential and universal) by requiring the existence of behaviours satisfying some properties, while requiring that all behaviours satisfy some other property. For instance, formula \( \varphi^{nd}_{\text{synth}}(EF p \land EF q \land AG (p \lor q)) \) asks that some behaviours reach \( p \), some reach \( q \), and all behaviours go infinitely often through \( p \) or \( q \).

For arbitrary CTL* specifications \( \varphi \), formula \( \varphi^{nd}_{\text{synth}}(\varphi) \) captures the supervisory control problem for CTL* as defined in (Kupferman et al. 2000), where it is proved to be 3EXPTIME-complete. This formula has simulation depth at most 2, and thus the model-checking algorithm for SL\( ^x \) has optimal complexity. The same complexity is proved in (Jiang and Kumar 2006) in a slightly different setting. If one changes \( \exists x \) for \( \exists d \) in \( \varphi^{nd}_{\text{synth}}(\varphi) \), one obtains instead the synthesis problem with reactive environments, also studied in (Kupferman et al. 2000), which has the same complexity.

4.3 Unilateral forcing

We want to clarify what the synthesis problem with nondeterministic strategies as defined by \( \varphi^{nd}_{\text{synth}}(\varphi) \) means when \( \varphi \) involves possibility requirements of the form \( EF \psi \). Take for instance formula \( \varphi^{nd}_{\text{synth}}(E G p \land EF \neg p) \), and consider the CGS \( G_2 \) depicted in Figure 2. It is the case that \( G_2 \models \varphi^{nd}_{\text{synth}}(E G p \land EF \neg p) \), as witnessed by the strategy \( \sigma_{a,b} \) that allows both \( a \) and \( b \) in the initial state. Indeed, if the system follows \( \sigma_{a,b} \), then for any strategy \( \sigma' \) for the environment, there exists an outcome that satisfies \( G p \) and one that satisfies \( F \neg p \). In case strategy \( \sigma' \) is deterministic, the system can choose which kind of outcome to obtain: if \( \sigma' \) plays \( a \) in the initial position, and the system knows it, then the system can enforce \( G p \) by playing \( a \) in the initial position, which is allowed by its strategy \( \sigma \), and to enforce \( F \neg p \) it can play \( b \), also allowed by \( \sigma \). If however the environment uses the nondeterministic strategy \( \sigma_{a,b} \), then the system cannot unilaterally choose which possibility to enforce.

However, the ability to choose unilaterally can also be expressed in SL\( ^x \), once more using the refinement operator. Define the following formula, where \( \psi \) is an LTL formula:
\[
\text{Force}(x, \psi) := \exists x y. y \leq x (S, y) A \psi
\]
Now consider the formula
\[ \varphi_2 := \exists x y (S, x) \forall y (E, y) (\text{Force}(x, G p) \lor \text{Force}(x, F \neg p)) \]
It is easy to see that \( \varphi_2 \) implies \( \varphi^{nd}_{\text{synth}}(E G p \land EF \neg p) \). But when it holds, it means in addition that the system can unilaterally choose which of \( G p \) or \( F \neg p \) should hold, by picking one deterministic refinement of its nondeterministic strategy. For instance \( \varphi_2 \) holds on \( G_1' \) if the system uses \( \sigma_{a,b} \), it can choose to enforce \( G p \) by refining it to \( \sigma_a \), or enforce \( F \neg p \) by refining it to \( \sigma_b \). In both cases, the property will hold no matter how the environment behaves. However, \( \varphi_2 \) does not hold on \( G_2' \): assume first that the system uses one of the deterministic strategies \( \sigma_a \) and \( \sigma_b \), then when the environment also picks a deterministic strategy, it fixes a unique outcome and thus \( \text{Force}(x, G p) \lor \text{Force}(x, F \neg p) \) does not hold. Now assume the system uses \( \sigma_{a,b} \). If the environment also picks \( \sigma_{a,b} \), then \( \text{Force}(x, G p) \lor \text{Force}(x, F \neg p) \) does not hold: indeed, the only deterministic refinements of \( \sigma_{a,b} \) are \( \sigma_a \) and \( \sigma_b \), and for any of these, the outcome \( (G p \lor F \neg p) \) depends on which action the environment picks from its nondeterministic strategy \( \sigma_{a,b} \).

This shows that \( \varphi_2 \) is indeed a stronger requirement than \( \varphi_1 \), which holds on both \( G_1 \) and \( G_2 \).

So in synthesis with nondeterministic strategies, we can require strategies that allow for different types of behaviours, and we have seen that we can also require that the
system be able to choose which kind of behaviour to enforce by refining its strategy. It is then natural to look for strategies that allow for as many behaviours as possible while satisfying the specification. Such strategies are usually called maximally permissive in the literature.

4.4 Maximally permissive synthesis

Different definitions of maximally permissive strategies have been used in the literature. In supervisory control theory (Ramadge and Wonham 1987), maximality is expressed in terms of inclusion of sets of behaviours/outcomes, or equivalently by referring to simulation between the unfoldings of the systems where unauthorised transitions have been pruned, as in (Pinchinat and Riedweg 2005). In (Bernet, Janin, and Walukiewicz 2002), strategies are also compared by looking at inclusion of the behaviours/outcomes they allow. However, as it is proved in (Bernet, Janin, and Walukiewicz 2002), the existence of maximally permissive strategies for this notion of maximality is ensured only for simple safety games.

For this reason an alternative notion of strategy permissiveness was introduced in (Bouyer et al. 2009) for reachability games and further studied in (Bouyer et al. 2011) for parity games. In this setting, to each transition in a game is attached a cost that represents the penalty incurred by a strategy that does not allow this transition, and a maximally permissive strategy is one that minimises penalties.

The latter definition ensures that maximally permissive strategies always exist, but the quantitative aspects involved, which are close to mean-payoff games (Ehrenfeucht and Mycielski 1979; Gurvich, Karzanov, and Khachivan 1988), are known to quickly lead to undecidability when introduced in Strategy Logic (Gardy 2017). As for the definition based on inclusion of outcomes, it makes sense in the two-player antagonistic setting; but it is not adapted to our multi-player setting, where the set of outcomes induced by a strategy depends on which agent uses it, and which other agents have a defined strategy.

For this reason we consider the following natural definition of permissiveness based on refinement of strategies:

**Definition 5.** Strategy \( \sigma' \) is more permissive than strategy \( \sigma \) if \( \sigma \preceq \sigma' \).

Given a formula \( \varphi(x) \), we can now express that a strategy \( x \) is maximally permissive with regards to \( \varphi(x) \), i.e., that it satisfies \( \varphi(x) \) and that no more permissive strategy satisfies it. Define formula \( \text{MaxPerm}(x, \varphi) \) as follows:

\[
\text{MaxPerm}(x, \varphi) := \varphi(x) \land (\forall y \ x \prec y \rightarrow \neg \varphi(y))
\]

For instance, coming back to the framework of reactive synthesis, if the specification is a CTL\(^+\) formula \( \varphi \), it holds that

\[
G, X, \nu \models \text{MaxPerm}(x, \forall z(S, x)\langle E, z\rangle \varphi)
\]

if, and only if, \( \chi(x) \) is a maximally permissive system for specification \( \varphi \).

To solve the problem of maximally permissive synthesis for a specification \( \varphi \in \text{CTL}^+ \), we can thus model-check the following SL\(^-\) formula:

\[
\varphi_{\text{sync}}^{\text{max}}(\varphi) := \exists x(a, x)\text{MaxPerm}(x, \forall z(S, x)\langle E, z\rangle \varphi)
\]

When the formula is true, our model-checking algorithm can also produce a maximally permissive witness strategy for \( x \). If \( \varphi \) is a CTL\(^+\) formula, \( \varphi_{\text{sync}}^{\text{max}}(\varphi) \) has simulation depth 3, and thus we obtain a 4EXPTIME upper-bound for this problem. We do not have the lower bounds, but we conjecture that one cannot do better, as the problem is already 3EXPTIME-complete without the constraint of maximal permissiveness (see Section 4.2), which seems to add one exponential to the complexity: However it can be reduced to 3EXPTIME for LTL specifications, i.e., for formulas of the form \( \varphi = A\psi \) with \( \psi \in \text{LTL} \).

4.5 Strategy refinement

One problem that is of interest in AI is that of producing plans that enforce some safety property, and in addition can at any time be refined to reach some secondary goal (Wright, Mattmüller, and Nebel 2018; Percassi and Gerevini 2019).

For instance, consider an electric vehicle transporting rocks from a point A, where they are cut, to a point B, where they are used. The truck must ensure that the stock of rocks at point B never runs out (safety property). We would like to synthesise a strategy for the truck such that this property is satisfied, but also so that at any time, the strategy can be refined to make the truck go through point C, where its battery can be reloaded. The fact that the strategy to reload refines the initial one ensures that the main property remains satisfied, i.e., it remains true that B will not run out even when the truck decides to go reload. We can express this problem in \( \text{SL}^- \) as follows, where "empty" holds when there are no more rocks in point B, and "reload" means that the truck is at point C, reloading its battery.

\[
\varphi_{\text{ref}} := \exists x(\text{truck}, x) \text{AG}(\neg \text{empty} \land (\exists y. y \preceq x \land (\text{truck}, y) \text{AF} \text{Reload}))
\]

The simulation depth for \( \varphi_{\text{ref}} \) is 2, and by Theorem 2 we obtain a 3EXPTIME upper bound for this problem with arbitrary CTL\(^+\) formulas instead of AFReload. However for the particular fixed formula \( \varphi_{\text{ref}} \) we can simplify the procedure to obtain a synthesis algorithm whose running time is a single exponential in the size of the model.

4.6 Module checking

Module checking (Kupferman, Vardi, and Wolper 2001; Jamroga and Murano 2014) is a generalisation of model checking to the setting of open systems, i.e., systems that interact with an environment. In this problem the system’s nondeterministic strategy is fixed, and fixing a nondeterministic strategy for the environment thus yields a computation tree which is a pruning of the full system’s computation tree. The problem then consists in checking that a property, specified for instance in CTL\(^+\), holds in all such computation trees. The module-checking problem for a CTL\(^+\) formula \( \varphi \) can be written as follows in \( \text{SL}^- \):

\[
\varphi_{\text{mod}} := \forall y(E, y) \varphi
\]

Solving the module-checking problem for CTL\(^+\) specifications can thus be done by model checking \( \varphi_{\text{mod}} \). This formula has simulation depth 1, and the procedure thus runs in doubly exponential time, which is asymptotically optimal (Kupferman, Vardi, and Wolper 2001).
4.7 Multi-agent setting

Finally we briefly mention that in the multi-agent setting we can express that a strategy profile (i.e., a strategy for each agent) refines another. In SL\^2 we can thus reason about refinements of game-theoretical solution concepts expressible in Strategy Logic, such as Nash equilibria or subgame-perfect equilibria, and synthesise maximally permissive equilibria. For instance assume that \( A = \{ a_i : i \in [1,n] \} \), each agent \( a_i \) has objective \( \psi_i \in LTL \), and \( \{ a, x \} \) stands for \( (a_1, x_1) \ldots (a_n, x_n) \) where \( a = (a_i)_{i \in [1,n]} \) and \( x = (x_i)_{i \in [1,n]} \). Consider the formula

\[
\varphi_{NE}(x) := (a, x) \bigwedge_{i \in [n]} \left( \exists y_i (a_i, y_i) A \psi_i \right) \rightarrow A \psi_i
\]

Formula \( \varphi_{NE} \) holds with assignment \( \chi \) if, and only if, \( (\chi(x_i))_{i \in [1,n]} \) is a Nash equilibrium. Now generalise formula MaxPerm\( (x, \varphi) \) from Section 4.4 as follows, where \( free(\varphi) = x \) and \( x \leq y \) stands for \( \forall i \in [1,n] x_i \leq y_i \):

\[
\text{MaxPerm}^*(x, \varphi) := \varphi(x) \land (\forall^d y \, y < x \rightarrow \neg \varphi(y))
\]

Synthesis of a maximally permissive Nash equilibrium can now be expressed with the following formula:

\[
\exists^d x \, \text{MaxPerm}^*(x, \varphi_{NE}(x))
\]

This formula having simulation depth 3, we obtain a 4Exp-Time upper bound on the complexity of this problem.

We can also generalise the notion of unilateral forcing by requiring that a subset of agents can together enforce different possibilities by refining their strategies.

5 Model checking SL\(^{\leq} \)

We now turn to establishing that the model-checking problem for SL\(^{\leq} \) is decidable. To do so we extend the classic approach, which is to reduce to QCTL\(^* \), the extension of CTL\(^* \) with (second-order monadic) quantification on atomic propositions. This logic is equivalent to MSO on infinite trees (Laroussinie and Markey 2014), and it is easy to express that a strategy (or the atomic propositions that code for it) refines another one.

**Definition 6.** The syntax of QCTL\(^* \) is defined by the following grammar:

\[
\varphi := p \mid \neg \varphi \mid \varphi \lor \varphi \mid E \varphi \mid \exists p \varphi \mid \psi \mid \neg \psi \mid \psi \lor \psi \mid X \psi \mid \psi U \psi
\]

where \( p \in AP \).

Again, formulas of type \( \varphi \) are called state formulas, those of type \( \psi \) are called path formulas, and QCTL\(^* \) consists of all the state formulas defined by the grammar, and we use standard abbreviation \( A \psi := E \neg \neg \psi \).

The models of QCTL\(^* \) are classic Kripke structures:

**Definition 7.** A Kripke structure, or KS, over AP is a tuple \( S = (S, R, \ell, s_0) \) where \( S \) is a set of states, \( R \subseteq S \times S \) is a left-total\(^1 \) transition relation, \( \ell : S \rightarrow 2^{AP} \) is a labelling function and \( s_0 \in S \) is an initial state.

A path in \( S \) is an infinite sequence of states \( \lambda = s_0 s_1 \ldots \) such that for all \( i \in \mathbb{N}, (s_i, s_{i+1}) \in R \). A finite path is a finite non-empty prefix of a path. Similar to continuations of finite plays, given a finite path \( \lambda \) we write Cont(\( \lambda \)) for the set of finite paths that start with \( \lambda \). We may write \( s \in S \) for \( s \in S \), and we define the size \( |S| \) of a KS \( S = (S, R, \ell, s_0) \) as its number of states: \( |S| := |S| \).

Since we will interpret QCTL\(^* \) on unfoldings of KS, we now define infinite trees.

**Trees.** Let \( X \) be a finite set of directions (typically a set of states). An X-tree \( \tau \) is a nonempty set of words \( \tau \subseteq X^+ \) such that (1) there exists \( r \in X \), called the root of \( \tau \), such that each \( u \in \tau \) starts with \( r (r \preceq u) \); (2) if \( u \cdot x \in \tau \) and \( u \cdot x \neq r \), then \( u \in \tau \); (3) if \( u \in \tau \) then there exists \( x \in X \) such that \( u \cdot x \in \tau \).

The elements of a tree \( \tau \) are called nodes. If \( u \cdot x \in \tau \), we say that \( u \cdot x \) is a child of \( u \). An X-tree \( \tau \) is complete if for every \( u \in \tau \) and \( x \in X \), \( u \cdot x \in \tau \). A path in \( \tau \) is an infinite sequence of nodes \( \lambda = u_0 u_1 \ldots \) such that for all \( i \in \mathbb{N}, u_{i+1} \) is a child of \( u_i \), and Paths\( (u) \) is the set of paths that start in node \( u \).

An AP-labelled X-tree, or (AP, X)-tree for short, is a pair \( t = (\tau, \ell) \), where \( \tau \) is an X-tree called the domain of \( \ell \) and \( \ell : \tau \rightarrow 2^{AP} \) is a labelling, which maps each node to the set of propositions that hold there. For \( p \in AP \), a \( p \)-labelling for a tree is a mapping \( \ell_p : \tau \rightarrow \{0,1\} \) that indicates in which nodes \( p \) holds, and for a labelled tree \( t = (\tau, \ell) \), the \( p \)-labelling of \( t \) is the \( p \)-labelling \( u \mapsto 1 \) if \( p \in \ell(u) \), 0 otherwise. The composition of a labelled tree \( t = (\tau, \ell) \) with a \( p \)-labelling \( \ell_p \) for \( p \) is defined as \( t \otimes \ell_p := (\tau, \ell') \), where \( \ell'(u) = \ell(u) \cup \{ p \} \) if \( \ell_p(u) = 1 \), and \( \ell(u) \setminus \{ p \} \) otherwise. A \( p \)-labelling for a labelled tree \( t = (\tau, \ell) \) is a \( p \)-labelling for its domain \( \tau \). A pointed labelled tree is a pair \( (t, u) \) where \( u \) is a node of \( t \).

Let \( S = (S, R, \ell, s_0) \) be a Kripke structure over AP. The tree-unfolding of \( S \) is the (AP, S)-tree \( t_S := (\tau, \ell_S) \), where \( \tau \) is the set of all finite paths that start in \( s_0 \), and for every \( u \in \tau, \ell_S(u) := \ell(\text{last}(u)) \).

**Definition 8.** We define by induction the satisfaction relation \( \models \) of QCTL\(^* \). Let \( t = (\tau, \ell) \) be an AP-labelled tree, \( u \) a node and \( \lambda \) a path in \( t \) (we omit Boolean cases):

\[
\begin{align*}
&\text{if } p \in \ell(u) \quad t, u \models p \\
&\text{if } \exists \lambda \in \text{Paths}(u) \text{ s.t. } t, \lambda \models \psi \\
&\text{if } \exists p \text{ a } p \text{-labelling for } t \text{ s.t. } t \otimes \ell_p, u \models \varphi \\
&\text{if } t, \lambda \models \varphi \\
&\text{if } t, \lambda \models X \psi \text{ if } t, \lambda_{>0} \models \psi \\
&\text{if } t, \lambda \models \psi U \psi' \text{ if } \exists \bar{\psi} \text{ s.t. } 0 \leq j < \ell(t), t, \lambda_{>j} \models \bar{\psi} \\
\end{align*}
\]

We write \( t \models \varphi \) for \( t, r \models \varphi \), where \( r \) is the root of \( t \). Given a KS \( S \) and a QCTL\(^* \) formula \( \varphi \), we also write \( S \models \varphi \) if \( t_S \models \varphi \). The simulation depth for QCTL\(^* \) is defined exactly as for SL\(^{\leq} \), with the case for \( \exists p \varphi \) corresponding to \( \exists \varphi \), and we have:

**Theorem 4.** The model-checking problem for QCTL\(^* \) is \((k+1)\)-ExpTime-complete for formulas of simulation depth at most \( k \).
5.1 Reduction to QCTL$	extsuperscript{*}$

We use a variant of the reductions presented in (Laroussinie and Markey 2015; Fijalkow et al. 2018; Berthon et al. 2017; Maubert and Murano 2018; Bouyer et al. 2019), which transform instances of the model-checking problem for various strategic logics to (extensions of) QCTL$	extsuperscript{*}$.

Let $(G, \Phi)$ be an instance of the SL model-checking problem, and assume without loss of generality that each strategy variable is quantified at most once in $\Phi$. We define an equivalent instance of the model-checking problem for QCTL$	extsuperscript{*}$.

Define the KS $S_G := \langle S, R, s_i, t^f \rangle$ where

- $S := \{ s_v \mid v \in V \}$,
- $R := \{ (s_v, s_v') \mid \exists c \in ACG \text{ s.t. } E(v, c) = 0' \} \subseteq S^2$, 
- $s_i := s_v$, and
- $t^f(s_v) := t(v) \cup \{ p_0 \} \subseteq AP \cup AP_v$.

For every finite play $\rho = v_0 \ldots v_k$, define the node $u_\rho := s_{v_0} \ldots s_{v_k}$ in $tS_G$. The mapping $\rho \mapsto u_\rho$ defines a bijection between the set of finite plays and the sets of nodes in $tS_G$.

We now describe how to transform an SL$^\infty$ formula $\varphi$ and a partial function $f : Ag \rightarrow Var$ into a QCTL$^\infty$ formula $(\varphi)^f$ (which will also depend on $G$). Suppose that $Ac = \{ c_1, \ldots, c_l \}$, and define $(\varphi)^f$ and $(\varphi)^f_p$ by mutual induction on state and path formulas. The base cases are as follows: $(\varphi)^f := \varphi$ and $(\varphi)^f_p := \varphi$ for atomic propositions and induction hypothesis between the finite play, and thus that atomic propositions instead ensures that $p$ follows the strategy coded by the $p^f_{c(a)}$.

To prove the correctness of the translation we need some additional definitions. First, given a strategy $\sigma$ and a strategy variable $x$ we let $\ell_{\sigma}^x := \{ f_{c(a)} \mid c \in Ac \}$ be the family of $p^f_{c(a)}$-labelings for tree $tS_G$ defined as follows: for each finite play $\rho$ and $c \in Ac$, we let $\ell_{\sigma}^x(\rho) := 1$ if $c \in \sigma(\rho)$, 0 otherwise. For a labelled tree $t$ with same domain as $tS_G$ we write $t \otimes \ell_{\sigma}^x, \ell_{\sigma}^y$ for $t \otimes \ell_{\sigma}^x \otimes \ldots \otimes \ell_{\sigma}^y$.

Second, given an infinite play $\pi$ and a point $i \in N$, let $\lambda_{\pi,i}$ be the infinite path in $tS_G$ that starts in node $u_{\pi,i}$, and is defined as $\lambda_{\pi,i} := u_{\pi,i}u_{\pi,i+1}u_{\pi,i+2} \ldots$.

Finally, we say that a partial function $f : Ag \rightarrow Var$ is compatible with an assignment $\chi$, if $dom(\chi) \cap Ag = dom(f)$ and for all $a \in dom(f)$, $\chi(a) = f(a)$.

**Proposition 5.** For every state subformula $\varphi$ and path subformula $\psi \in \Phi$, finite play $\rho$, infinite play $\pi$, point $i \in N$, for every assignment $\chi$ variable-complete for $\varphi$ (resp. $\psi$) and partial function $f : Ag \rightarrow Var$ compatible with $\chi$, assuming also that no $x$ in $dom(\chi) \cap Var = \{ x_1, \ldots, x_k \}$ is quantified in $\varphi$ or $\psi$, we have
\[
G, \chi, \rho \models \varphi \iff tS_G \otimes \ell_{\chi(x_1)}^x \ldots \otimes \ell_{\chi(x_k)}^x u_{\rho} \models (\varphi)^f
\]
\[
G, \chi, \pi, i \models \psi \iff tS_G \otimes \ell_{\chi(x_1)}^x \ldots \otimes \ell_{\chi(x_k)}^x \chi_{\pi,i} \models (\psi)^f
\]
In addition, $S_G$ is of size linear in $|G|$, and $(\varphi)^f$ and $(\varphi)^f_p$ are of size linear in $|G|^2 + |\varphi|$.

**Proof.** The proof is by induction on $\varphi$. We detail the case for binding, strategy quantification, strategy refinement and outcome quantification, the others follow simply by definition of $S_G$ for atomic propositions and induction hypothesis for remaining cases.

For $\varphi = x \leq y$, assume that $G, \chi, \rho \models x \leq y$. First, observe that since $\chi$ is variable-complete for $\varphi$, $x$ and $y$ are in $dom(\chi)$. Now we have that $\chi(x)_{\rho} \subseteq \chi(y)_{\rho}$ for every $\rho \in Cont(\rho)$. By definition of $\ell_{\chi(x)}^x = \{ \ell_{\rho}^x \mid c \in Ac \}$ and $\ell_{\chi(y)}^y = \{ \ell_{\rho}^y \mid c \in Ac \}$, it follows that for each $c \in Ac$ and $\rho \in Cont(\rho)$, if $\ell_{\rho}^y(\rho') = 1$, then $\ell_{\rho}^x(\rho') = 1$, and thus $tS_G \otimes \ell_{\chi(x)}^x \otimes \ell_{\chi(y)}^y \models AG \bigwedge_{c \in Ac} p^c_{\rho} \rightarrow p^c_{\rho'}$

The result then holds since the labelings $\ell_{\chi(x)}^x$ touch distinct sets of atomic propositions for each variable $x$ in $Var(\chi)$.

For the other direction let $t = tS_G \otimes \ell_{\chi(x_1)}^x \ldots \otimes \ell_{\chi(x_k)}^x$ and assume that $t, u_{\rho} \models AG \bigwedge_{c \in Ac} p^c_{\rho} \rightarrow p^c_{\rho'}$

This implies that for every $\rho' \in Cont(\rho)$,
\[
t, u_{\rho'} \models \bigwedge_{c \in Ac} p^c_{\rho} \rightarrow p^c_{\rho'}
\]
and thus $\chi(x)_{\rho'}$ refines $\chi(y)_{\rho'}$.

For $\varphi = (a, x)\varphi'$, we have $G, \chi, \rho \models (a, x)\varphi'$ if and only if $G, \chi[a \mapsto \chi(x)], \rho \models \varphi'$. The result follows by using the
induction hypothesis with assignment $\chi[a \mapsto x]$ and function $f[a \mapsto x]$. This is possible because $f[a \mapsto x]$ is compatible with $\chi[a \mapsto x]$: indeed $dom(\chi[a \mapsto x]) \cap Ag$ is equal to $dom(\chi) \cap Ag \cup \{a\}$, which, by assumption, is equal to $dom(f) \cup \{a\} = dom(f[a \mapsto x])$. Also by assumption, for all $a^i \in dom(f)$, $\chi(a^i) = \chi(f(a^i))$, and by definition $\chi[a \mapsto x](a^i) = \chi(x)(a^i) = \chi(f[a \mapsto x](a^i))$. For $\varphi = \exists x \varphi'$, assume first that $G, \chi, \rho \models \exists x \varphi'$. There exists a nondeterministic strategy $\sigma$ such that $G, \chi[x \mapsto \sigma], \rho \models \varphi'$. Since $f$ is compatible with $\chi$, it is also compatible with assignment $\chi' = \chi[x \mapsto \sigma]$. By assumption, no variable in \{ $x_1, \ldots, x_k$ \} is quantified in $\varphi$, so that $x \neq x_i$ for all $i$, and thus $\chi'(x_i) = \chi(x_i)$ for all $i$; and because no strategy variable is quantified twice in a same formula, $x$ is not quantified in $\varphi'$, so that no variable in \{ $x_1, \ldots, x_k, x$ \} is quantified in $\varphi'$. By induction hypothesis

$$t_{S_0} \otimes \ell_{x}(\chi(x_1)) \otimes \cdots \otimes \ell_{x}(\chi(x_k)) \otimes \ell_{x}(\chi(x)), u_{\rho} \models (\varphi')^f_s.$$ 

It follows that $t_{S_0} \otimes \ell_{x}(\chi(x_1)) \otimes \cdots \otimes \ell_{x}(\chi(x_k)) \otimes \ell_{x}(\chi(x)), u_{\rho} \models \exists x \varphi'$. Finally, since $\chi'(x_i) = \chi(x_i)$ for all $i$, we conclude that $t_{S_0} \otimes \ell_{x}(\chi(x_1)) \otimes \cdots \otimes \ell_{x}(\chi(x_k)) \otimes \ell_{x}(\chi(x)), u_{\rho} \models (\exists x \varphi')^f_s$.

For the other direction, assume that

$$t_{S_0} \otimes \ell_{x}(\chi(x_1)) \otimes \cdots \otimes \ell_{x}(\chi(x_k)) \otimes \ell_{x}(\chi(x)), u_{\rho} \models (\varphi')^f_s.$$ 

and recall that $(\varphi')^f_s = \exists x \varphi'_1 \cdots \exists x \varphi'_k \varphi_{str}(x) \land (\varphi')^f_s$. Write $t = t_{S_0} \otimes \ell_{x}(\chi(x_1)) \otimes \cdots \otimes \ell_{x}(\chi(x_k))$. There exist $\ell_{x_1} \otimes \cdots \otimes \ell_{x_k}$-labellings such that $t \otimes \ell_{x_1} \otimes \cdots \otimes \ell_{x_k} \models \varphi_{str}(x) \land (\varphi')^f_s$.

By $\varphi_{str}(x)$, these labellings code for a strategy $\sigma$. Let $\chi' = \chi[x \mapsto \sigma]$. For all $1 \leq i \leq k$, by assumption $x \neq x_i$, and thus $\chi'(x_i) = \chi(x_i)$. The above can thus be rewritten as

$$t_{S_0} \otimes \ell_{x}(\chi(x_1)) \otimes \cdots \otimes \ell_{x}(\chi(x_k)) \otimes \ell_{x}(\chi(x)), u_{\rho} \models \varphi_{str}(x) \land (\varphi')^f_s.$$ 

By induction hypothesis we have $G, \chi[x \mapsto \sigma], \rho \models \varphi'$, hence $G, \chi, \rho \models \exists x \varphi'$. For $\varphi = \exists x \varphi$, the proof is similar, using $\varphi_{str}(x)$ instead of $\varphi_{str}(x)$.

For $\varphi = \psi$, assume that $G, \chi, \rho \models \psi$. There exists a play $\pi \in Out(\chi, \rho)$ s.t. $G, \chi, \pi, |\rho| - 1 \models \psi$. By induction hypothesis, $t_{S_0} \otimes \ell_{x}(\chi(x_1)) \otimes \cdots \otimes \ell_{x}(\chi(x_k)), \chi, \pi, |\rho| - 1 \models (\psi)^f_p$.

Since $\pi$ is an outcome of $\chi$, each agent $a \in dom(\chi) \cap Ag$ follows strategy $\chi(a)$ in $\pi$. Because $dom(\chi) \cap Ag = dom(f)$ and for all $a \in dom(f)$, $\chi(a) = \chi(f(a))$, each agent $a \in dom(f)$ follows the strategy $\chi(f(a))$, which is coded by atoms $\ell_{\chi(f(a))}$ in the translation of $\Phi$. Therefore $\lambda_{\pi, |\rho| - 1}$ also satisfies $\psi_{out}'$, hence $t_{S_0} \otimes \ell_{x}(\chi(x_1)) \otimes \cdots \otimes \ell_{x}(\chi(x_k)), \chi, \pi, |\rho| - 1 \models \psi_{out}' \land (\psi)^f_p$.

and we are done.

For the other direction, assume that $t_{S_0} \otimes \ell_{x}(\chi(x_1)) \otimes \cdots \otimes \ell_{x}(\chi(x_k)), u_{\rho} \models (\psi_{out}^f \land (\psi)^f_p)$. There exists a path $\lambda$ in $t_{S_0} \otimes \ell_{x}(\chi(x_1)) \otimes \cdots \otimes \ell_{x}(\chi(x_k))$, starting in node $u_{\rho}$, that satisfies both $\psi_{out}^f$ and $(\psi)^f_p$. By construction of $S_0$ there exists an infinite play $\pi$ such that $\pi_{|\rho| - 1} = \rho$ and $\lambda = \lambda_{\pi, |\rho| - 1}$. By induction hypothesis, $G, \chi, \pi, |\rho| - 1 \models \psi$. Because $\lambda_{\pi, |\rho| - 1}$ satisfies $\psi_{out}^f$, $dom(\chi) \cap Ag = dom(f)$, and for all $a \in dom(f)$, $\chi(a) = \chi(f(a))$, it is also the case that $\pi \in Out(\chi, \rho)$, hence $G, \chi, \rho \models \psi$.

Applying Proposition 5 to the sentence $\Phi$, $\rho = \nu$, any assignment $\chi$, and the empty function $\emptyset$, we get:

$$G \models \Phi \quad \text{if and only if} \quad t_{S_0} \models (\Phi)^g_0.$$ 

To obtain Theorem 2 as a consequence of Theorem 4, it only remains to observe that the translation $(\cdot)^g_0$ preserves simulation depth, modulo a detail that needs a particular treatment in the simulation of $\exists x \varphi$ (the case of $\exists^2 x \varphi$ is similar). We have $(\exists x \varphi)^g_0 = \exists^2 x \varphi'_1 \cdots \exists^2 x \varphi'_k \varphi_{str}(x) \land (\varphi')^f_s$, and even when $sd_k(\varphi) = nd$, we have $sd_k(\varphi_{str}(x) \land (\varphi')^f_s) = alt$, and in that case it follows that $sd_k((\exists x \varphi)^g_0)$ is one more than $sd_k(\exists x \varphi)$. But this additional exponential can be avoided by noting that $\varphi_{str}(x)$ can be recognised by a very simple deterministic tree automaton, which can be put in product with a nondeterministic automaton for $(\varphi')^f_s$, yielding a nondeterministic automaton of polynomial size. In other words, in this precise case conjunction does not introduce alternation, and if the automaton for $(\varphi')^f_s$ is nondeterministic we can obtain a nondeterministic automaton for $\varphi_{str}(x) \land (\varphi')^f_s$ without incurring an exponential blowup.

6 Conclusion

In this work we extended Strategy Logic with nondeterministic strategies and a refinement operator that expresses that a strategy is more permissive than another. We showed how the resulting logic $SL^\infty$ captures in a natural manner a variety of problems previously not expressible in Strategy Logic, such as module checking, synthesis with reactive environments or synthesis of maximally permissive strategies. We also showed how the refinement operator allows us to specify meaningful requirements for nondeterministic synthesis that are not expressible in CTL*, such as uniliteral forcing. We solved the model-checking problem for $SL^\infty$ by reduction to $QCTL^*$, and we established its complexity in terms of the simulation depth of the formulas. This precise measure shows that, for the problems from the literature whose precise complexity is known, the synthesis procedures that we obtain via $SL^\infty$ have optimal complexity. The model-checking algorithm for $SL^\infty$ also provides synthesis procedures for problems that, up to our knowledge, have not been solved before, such as synthesis with uniliteral forcing specifications or synthesis of maximally permissive Nash equilibria. As future work we plan to establish the precise complexity of these problems to see if our procedure is optimal.
References


Banihashemi, B.; De Giacomo, G.; and Lespérance, Y. 2018. Hierarchical agent supervision. In AAMAS.


