Strategy Logic extended with Refinement, CGS, and Nondeterminism

Dennis Griffith Elsa L. Gunter
University of Illinois at Urbana-Champaign
{dgriffi3,egunter}@illinois.edu

In this paper, we introduce SLeRCN, an extension of Strategy Logic (SL). The extensions syntactic and semantic. The main syntactic extensions include lifting the restrictions in Chatterjee et al. [7] that formulae must be hierarchically closed. Semantic extensions include support for arbitrary numbers of player, concurrent game structures, and nondeterministic strategies. We show that the model checking problem decidable via tree automata, but our algorithm is nonelementary. For the restricted case of positional strategies the model checking problem is in EXPTIME. We show that ATL, SL, and Rely-Guarentee Temporal Logic (RGTL) can be embedded in SLeRCN. As a result we show that model checking for RGTL is decidable, a previously unknown result. Additionally, SLeRCN can be used to characterize the existence of Qualitative Nash Equilibria and Secure Equilibria.

1 Introduction

Games have a long history of being used to reason about programs formally. In a concurrent game, some set of players choose a move to make in each turn and by doing so forever generate an infinite sequence of game states. The sequences then may be classified in various ways depending on the goal of the user. For example, in verifying that a concurrent system is correct we may designate some play sequences as "safe" and verify that our system, when its program is viewed as controlling how the players choose their moves, only can generate "safe" sequences. Alternatively, we might designate certain sequences as "winning" for a particular player, then some natural questions to ask might be "Can all the players work together to win?" or "Can one player prevent all the others from winning?"

To assist with reasoning about concurrent games, several logics have been proposed such as Alternating-time Temporal Logic (ATL) [2], Game Logic [2], STIT-ATL [5], RGTL [11], and others [1,4,8,12] Using these, we can compactly specify and reason (in an automated fashion even) about properties of our game and players. Unfortunately, each of these has some limitations as pointed out by Mansky et al. [11]. Of particular interest, RGTL allows us to express some properties that are very hard to state in the other logics while having certain useful reasoning principles (e.g., a rely-guarantee reasoning principle allowing for modular specifications). This expressive power comes at a cost, however, RGTL didn't have a model checking algorithm.

SL [7] is a very general temporal logic with explicit strategy quantification that deals with a different class of games, two-player turn-based games. In this paper (Section 2), we extend SL into a new logic, Strategy Logic extended with Refinement, CGS, and Nondeterminism (SLeRCN),[1] that allows for nondeterministic strategies with refinement over Concurrent Game Structures (CGSs) [2]. Along the way, we lift a syntactic restriction present in SL. We show that we can express ATL, ATL*, SL, and RGTL within SLeRCN (Section 3). Additionally, we show how to express in SLeRCN qualitative Nash equilibria and secure equilibria, as well as how SLeRCN give users fine-grained control over how deterministic

[1] SLeRCN rhymes with gherkin.

To appear in EPTCS.

their players' strategies must be. We show that the model checking problem for SLeRCN is decidable via tree automata and in the case where strategies are positional how a more direct dynamic programming algorithm is possible (Section 4). Coupling this result with the expressibility result for RGTL gives a model checking algorithm for RGTL, a logic that previously had no known model checking algorithm.

2 The Logic

In Section 2.1 we describe the syntax of SLeRCN, in Section 2.2 we describe the semantics of SLeRCN, and in Section 2.3 we discuss basic properties of SLeRCN.

2.1 Syntax

Given a set of agents $\mathscr{A}$, an agent indexed set of variables $\{V_a\}_{a\in\mathscr{A}}$, and a set of atomic propositions Π we can form a SLeRCN formula over variables $\{V_a\}_{a\in\mathscr{A}}$ via the following grammar. All variables are associated with an owning agent, their index in $\{V_a\}_{a\in\mathscr{A}}$. When the set of variables (and the owning agent for each variable) is clear from context or unimportant, we omit their explicit specification. Below x_a and y_a are variables owned by a, $p \in \Pi$, and ζ is a mapping from agents to appropriate variables (e.g., $a \mapsto x_a$ for $x_a \in V_a$).

$$\Phi ::= p \mid \Psi(\zeta) \mid \Phi \wedge \Phi \mid \neg\Phi \mid \exists x_a.\Phi \mid x_a \sqsubseteq y_a \qquad \Psi ::= \Phi \mid \Psi \wedge \Psi \mid \neg\Psi \mid \bigcirc\Psi \mid \Psi \mathscr{U} \Psi$$

Formulae generated from the Φ nonterminal are state formulae and those from the Ψ nonterminal are path formulae. The intended reading for the standard connectives are their normal temporal and boolean interpretation with the following interesting cases: $\psi(\zeta)$ asks "Does ψ hold along all the paths specified by the strategies in ζ?"; $x_a \sqsubseteq y_a$ asks "Does the strategy given by x_a refine that given by y_a at all points in the future? (i.e., x_a allows for a subset of the actions of y_a)".

2.2 Semantics

Before we define define the satisfaction relations for SLeRCN, we need a few basic notions. The underlying structure that SLeRCN operates with is a CGS.To ensure all readers are on the same page, we reproduce the definition of a CGS here.

Definition 1 (CGS)**.** *A Concurrent Game Structure is* $(\mathscr{A}, Q, \Sigma, e, \Delta, \Pi, \pi)$ *s.t.*

- $\mathscr{A}$ *is a finite nonempty set of agents*
- Q *is a finite nonempty set of states*
- Σ *is a finite nonempty set of actions*
- $e : Q \times \mathscr{A} \to 2^{\Sigma} \setminus \emptyset$*, the enabled function, specifies which actions an agent can perform in a state*
- $\Delta : Q \times (\Sigma^{\mathscr{A}}) \to Q$ *a transition function*
- Π *is a set of propositions*
- $\pi : Q \to 2^{\Pi}$ *is a labeling of states by their true propositions*

We lift Δ *to* $\Delta : Q \times \left(2^{\Sigma^{\mathscr{A}}}\right) \to 2^{Q}$ *in the natural way.*

A point to highlight is these the sets of agents and atomic propositions will be those that we use to generate terms from the grammar in Section 2.1. Additionally, CGSs are deadlock free (sometimes called complete) since the enabled function must always be nonempty. Thus, there are no states in which it is impossible to take a transition.

Next, we define nondeterministic history sensitive strategies, the notion of a strategy for each player, and their legality requirements.

Definition 2 (Strategies)**.** *Given a CGS* $(\mathscr{A}, Q, \Sigma, e, \Delta, \Pi, \pi)$ *and* $a \in \mathscr{A}$*,* $\sigma_a : Q^+ \to 2^\Sigma$ *is an a-strategy if* $\forall \rho \in Q^+.\sigma_a(\rho) \neq \emptyset \wedge \sigma_a(\rho) \subseteq e(\rho_{|\rho|-1}, a)$ *(i.e., it makes a selection from the actions allowed by the enabled function). We will denote the set of a-strategies by* Θ_a*. A strategy* σ *is called deterministic if for every history it chooses exactly one action (i.e., for every* $\rho \in Q^+$*,* $\sigma(\rho)$ *is a singleton). A system strategy* $\sigma : a \in \mathscr{A} \to \Theta_a$ *is an a-strategy for every agent. We will write* σ_a *for the a-strategy component of the system strategy* σ*. The set of all system strategies is* Θ*. We lift* $\sigma \in \Theta$ *to* $\sigma : Q^+ \to 2^{\Sigma^{\mathscr{A}}}$ *in the natural way.*

Once we have a CGS and a system strategy, we can define the outcome of playing the game using the system strategy.

Definition 3 (Outcomes)**.** *For CGS* $C = (\mathscr{A}, Q, \Sigma, e, \Delta, \Pi, \pi)$*, a history* $\rho \in Q^+$*, and a system strategy* $\sigma \in \Theta$*, the set of outcomes of length n is defined by*

$$\frac{}{\varepsilon \in \text{OUT}_0(C, \rho, \sigma)} \qquad \frac{\rho' \in \text{OUT}_{n-1}(C, \rho, \sigma) \qquad q \in \Delta(\rho'_{|\rho'|-1}, \sigma(\rho \cdot \rho'))}{\rho' \cdot q \in \text{OUT}_n(C, \rho, \sigma)}$$

the set of infinite outcomes, $\text{OUT}(C, \rho, \sigma)$*, is defined by*

$$\text{OUT}(C, \rho, \sigma) = \left\{ \gamma \in Q^\omega \mid \forall n.\gamma_{[0,i)} \in \text{OUT}_n(C, \rho, \sigma) \right\}$$

Last, we define a notion of refinement to allow us to capture the informal meaning for $x_a \sqsubseteq y_a$ of Section 2.1.

Definition 4 (ρ-Refinement)**.** *Given two a-strategies* σ_1 *and* σ_2 *and* $\rho \in Q^+$*,* σ_1 *is a* ρ*-refinement of* σ_2*, written* $\sigma_1 \sqsubseteq_\rho \sigma_2$*, if* $\forall \rho' \in Q^*.\sigma_1(\rho \cdot \rho') \subseteq \sigma_2(\rho \cdot \rho')$.

Given an CGS $C = (\mathscr{A}, Q, \Sigma, e, \Delta, \Pi, \pi)$, an environment Γ mapping variables to strategies of the appropriate type (e.g., $\Gamma(x_a) \in \Theta_a$), a history $\rho \in Q^+$, and $\gamma \in Q^\omega$, we define the SLeRCN satisfaction relations ($\vDash_\Phi$ and $\vDash_\Psi$ for state and path formulae respectively) by

$$\begin{array}{lll}
C, \Gamma, \rho \vDash_\Phi p & \text{iff} & p \in \pi(\rho_{|\rho|-1}) \\
C, \Gamma, \rho \vDash_\Phi \psi(\zeta) & \text{iff} & \forall \gamma \in \text{OUT}(C, \rho, \Gamma \circ \zeta).C, \Gamma, \rho, \gamma \vDash_\Psi \psi \\
C, \Gamma, \rho \vDash_\Phi \phi_1 \wedge \phi_2 & \text{iff} & C, \Gamma, \rho \vDash_\Phi \phi_1 \text{ and } C, \Gamma, \rho \vDash_\Phi \phi_2 \\
C, \Gamma, \rho \vDash_\Phi \neg\phi & \text{iff} & \text{not } C, \Gamma, \rho \vDash_\Phi \phi \\
C, \Gamma, \rho \vDash_\Phi \exists x_a.\phi & \text{iff} & \exists \sigma \in \Theta_a.C, \Gamma[x_a \leftarrow \sigma], \rho \vDash_\Phi \phi \\
C, \Gamma, \rho \vDash_\Phi x_a \sqsubseteq y_a & \text{iff} & \Gamma(x_a) \sqsubseteq_\rho \Gamma(y_a) \\
C, \Gamma, \rho, \gamma \vDash_\Psi \phi & \text{iff} & C, \Gamma, \rho \vDash_\Phi \phi \\
C, \Gamma, \rho, \gamma \vDash_\Psi \psi_1 \wedge \psi_2 & \text{iff} & C, \Gamma, \rho, \gamma \vDash_\Psi \phi_1 \text{ and } C, \Gamma, \rho, \gamma \vDash_\Psi \psi_2 \\
C, \Gamma, \rho, \gamma \vDash_\Psi \neg\psi & \text{iff} & \text{not } C, \Gamma, \rho, \gamma \vDash_\Psi \psi \\
C, \Gamma, \rho, \gamma \vDash_\Psi \bigcirc\psi & \text{iff} & C, \Gamma, \rho \cdot \gamma_0, \gamma_{[1,\infty)} \vDash_\Psi \psi \\
C, \Gamma, \rho, \gamma \vDash_\Psi \psi_1 \mathscr{U} \psi_2 & \text{iff} & \exists i. \left(\begin{array}{l} \forall j < i.C, \Gamma, \rho \cdot \gamma_{[0,j)}, \gamma_{[j,\infty)} \vDash_\Psi \psi_1 \\ \wedge C, \Gamma, \rho \cdot \gamma_{[0,i)}, \gamma_{[i,\infty)} \vDash_\Psi \psi_2 \end{array} \right)
\end{array}$$

There is not much surprising in this definition, most connectives have their standard first order or temporal meanings. A few points need further explanation: for the path evolution case ($\psi(\zeta)$) we use the function OUT to generate all the paths that need checking. Additionally, the refinement case ($x_a \sqsubseteq y_a$) is implemented using the previously defined forward looking refinement operator.

2.3 Basic Properties

Once we have a basic semantics we define the usual derived operators (e.g., universal quantification and the eventually temporal operator). There are a few simple results that follow immediately from the semantics.

Lemma 1 (Free Name Congruence)**.** *Let* FN *denote the free names of a SLeRCN formula. For CGS C, environments* Γ_1 *and* Γ_2*, history* $\rho \in Q^+$*, and state formula* ϕ*, if* $\Gamma_1 \restriction_{\text{FN}(\phi)} = \Gamma_2 \restriction_{\text{FN}(\phi)}$ *then* $(C, \Gamma_1, \rho \vDash_\Phi \phi) = (C, \Gamma_2, \rho \vDash_\Phi \phi)$ *and similarly for path formulae.*

This lemma shows that only the environment's behavior on free variables matters and that for closed formulae the only "interesting" environment to consider is the empty environment.

Lemma 2 (Forward Congruence)**.** *For CGS C, environments* Γ_1 *and* Γ_2*, history* $\rho \in Q^+$*, and state formula* ϕ*, if* $\forall v \in V.(\forall \rho'.\Gamma_1(v)(\rho \cdot \rho') = \Gamma_2(v)(\rho \cdot \rho'))$ *then* $(C, \Gamma_1, \rho \vDash_\Phi \phi) = (C, \Gamma_2, \rho \vDash_\Phi \phi)$ *and similarly for path formulae.*

This lemma shows that satisfaction is independent of the past behavior of the environment's strategies.

Lemma 3 (Closed Forgetfulness)**.** *For CGS C, environments* Γ*, history* $\rho \cdot q \in Q^+$*, and closed formula* ϕ*,* $(C, \Gamma, \rho \cdot q \vDash_\Phi \phi) = (C, \Gamma, q \vDash_\Phi \phi)$

This shows that, for a closed formula, satisfaction, in general, can be reduced to satisfaction starting from the most recently visited state. In particular this allows us to cut down the number of different histories that need to be considered from $|Q^+|$ to $|Q|$, which is, in general, a change from countably infinite to finite.

Lemma 4 (Quantification Commutivity/Idempotency)**.** *For CGS C, environments* Γ*, history* $\rho \in Q^+$*, and formula* ϕ*,* $C, \Gamma, \rho \vDash_\Phi \exists x_{a_1}.\exists y_{a_2}.\phi = C, \Gamma, \rho \vDash_\Phi \exists y_{a_2}.\exists x_{a_1}.\phi$ *and* $(C, \Gamma, \rho \vDash_\Phi \exists x_a.\exists x_a.\phi) = (C, \Gamma, \rho \vDash_\Phi \exists x_a.\phi)$.

This lemma shows that blocks of quantification could be instead thought of as simultaneous quantification over a set of variables. This justifies a derived notation $\exists X.\phi$ where $X \subseteq V$ is a finite set of variables to quantify over (i.e., $\exists X.\phi \equiv \exists x_1.\exists x_2.\ldots\exists x_n.\phi$ for the $x_i \in X$) and similarly $\forall X.\phi$ for universally quantifying over a set of variables.

3 Expressiveness

Consider the state formula $\forall y_a.(y_a \sqsubseteq x_a \to x_a \sqsubseteq y_a)$. This says that all strategies that refine the strategy specified by x_a are in turn refined by x_a, which by antisymmetry of $\subseteq$ the strategy of x_a is maximally refined. Since a strategy must always select at least one action, being maximally refined means that x_a must select exactly one action at any given state, i.e., the strategy of x_a is deterministic (in the future). We'll denote this expression as $\text{DETER}(x_a)$.

Lemma 5 (Future Determinism)**.** *For CGS C, environment* Γ*, history* ρ*, variable* x_a*, and state formula* ϕ*, there exists a deterministic strategy* σ *s.t.,* $C, \Gamma[x_a \leftarrow \sigma], \rho \vDash_\Phi \phi$ *if and only if* $C, \Gamma, \rho \vDash_\Phi \exists x_a.(\text{DETER}(x_a) \wedge \phi)$.

Sketch. ($\Leftarrow$): Since $C, \Gamma, \rho \vDash_\Phi \exists x_a.(\text{DETER}(x_a) \wedge \phi)$ there is some σ that is deterministic in the future s.t., $C, \Gamma[x_a \leftarrow \sigma], \rho \vDash_\Phi \phi$. Recall that strategies must always present at least one enabled action. A globally deterministic formula that agrees in the future with σ is given by:

$$\sigma'(\rho') = \begin{cases} \sigma(\rho') & \text{if } \exists \rho_1 \in Q^*.(\rho' = \rho \cdot \rho_1) \\ \{z\} & \text{s.t. } z \in \sigma(\rho'_{|\rho'|-1}) \text{ otherwise} \end{cases}$$

Applying Lemma 2 finishes this direction.

($\Rightarrow$): Use σ as the witness. Since σ is deterministic it satisfies DETER. □

From this SLeRCN gets an ability to control where deterministic behavior is required in a reasonably fine grained manner. It allows for embedding logics that normally operate using deterministic strategies into SLeRCN in a fairly natural fashion.

3.1 Qualitative Nash Equilibria

Given a CGS C, an initial state q_0, an indexed family of closed path formulae $\{\psi_a\}_{a\in\mathscr{A}}$ called winning conditions, and a system strategy $\sigma \in \Theta$ the payoff for an agent a is

$$p_a(\sigma) = \begin{cases} 1 & \text{if } \forall \gamma \in \text{OUT}(C, q_0, \sigma).C, \emptyset, q_0, \gamma \vDash_\Psi \psi_a \\ 0 & \text{otherwise} \end{cases}$$

A payoff profile is any subset of $\mathscr{A}$, denoting which agents win in that profile. We say that a system strategy σ achieves a payoff profile P if $\forall a \in \mathscr{A}.a \in P \iff p_a(\sigma) = 1$.

A system strategy σ is said to be a qualitative Nash equilibrium if it the following condition holds (i.e., no agent can unilaterally improve its own payoff):

$$\forall a \in \mathscr{A}.\forall \sigma' \in \Theta_a.p_a(\sigma[a \leftarrow \sigma']) \leq p_a(\sigma)$$

Before defining Nash equilibria [9] in SLeRCN we define a macro $\text{PAYOFF}(P, \zeta)$ that characterizes if, using the strategies specified by ζ, the given payoff profile is achieved:

$$\text{PAYOFF}(P, \zeta) \equiv \bigwedge_{a\in P} (\psi_a)(\zeta) \wedge \bigwedge_{a\in(\mathscr{A}-P)} \neg(\psi_a)(\zeta)$$

This works by checking that all the agents that the profile specifies wins do and similarly that all losers lose.

For a given payoff profile P, we define a SLeRCN state formula $\text{NASH}(P)$ that characterizes when C has a Nash equilibrium with that payoff profile. To doing this will use a pair of variables x_a and y_a for each agent (i.e., $2|\mathscr{A}|$ number of variables). Let $X = \{x_a \mid a \in \mathscr{A}\}$ and $\zeta_X(a) = x_a$. We define $\text{NASH}(P)$ as follows:

$$\text{NASH}(P) = \exists X. \left(\text{PAYOFF}(P, \zeta_X) \wedge \bigwedge_{a\in(\mathscr{A}-P)} \forall y_a.\neg(\psi_a)(\zeta_X[a \leftarrow y_a]) \right)$$

This works by checking if there is a system strategy such that two things hold: the correct payoff profile is achieved and for every agent that loses, it cannot switch strategies in a way that improves its payoff. To characterize the existence of Nash equilibria we take the disjunction across all payoff profiles ($\bigvee_{P\subseteq\mathscr{A}} \text{NASH}(P)$). It is also possible to give a slightly different definition of Nash equilibria where, instead of winning only when the winning condition is alway achieved, each agent has a payoff of 1 if it has the possibility of winning.

3.2 Secure Equilibria

Secure equilibria [6] are like Nash equilibria, but, in addition to maximizing its payoff, each agent would prefer to other agents to lose. This preference is encapsulated for each agent $a \in \mathscr{A}$ by the binary relation on payoff profiles $\preceq_a \subseteq 2^{\mathscr{A}} \times 2^{\mathscr{A}}$ given by

$$P_1 \preceq_a P_2 \equiv (a \notin P_1 \wedge a \in P_2(a)) \vee ((a \in P_1 \iff a \in P_2) \wedge \forall a' \in \mathscr{A}.(a' \in P_2 \implies a' \in P_1))$$

$\prec_a$ is the strict version of $\preceq_a$.

A system strategy σ defines a payoff profile $P_\sigma = \{a \in \mathscr{A} \mid p_a(\sigma) = 1\}$. A system strategy σ then is a secure equilibrium if it satisfies

$$\forall a \in \mathscr{A}.\forall \sigma' \in \Theta_a.P_{\sigma[a \leftarrow \sigma']} \preceq_a P_\sigma$$

For a given payoff profile P, we can define a SLeRCN state formula $\textsc{Secure}(P)$ that characterizes when C has a secure equilibrium with that payoff profile. Like in Section 3.1 we'll use a pair of variables x_a and y_a for each agent $a \in \mathscr{A}$, $X = \{x_a \mid a \in \mathscr{A}\}$, and $\zeta_X(a) = x_a$. We define $\textsc{Secure}(P)$ as follows:

$$\textsc{Secure}(P) = \exists X.\left(\textsc{Payoff}(P, \zeta_X) \wedge \bigwedge_{a \in \mathscr{A}} \forall y_a. \bigwedge_{P' \subseteq \mathscr{A}.P \prec_a P'} \neg \textsc{Payoff}(P', \zeta_X[a \leftarrow y_a])\right)$$

This works by checking if there is a system strategy such that the correct payoff is achieved and that no agent can unilaterally switch to a more desirable payoff. To determine if for any payoff profile a secure equilibrium exists, we again just take the disjunction across all possible payoff profiles ($\bigvee_{P \subseteq \mathscr{A}} \textsc{Secure}(P)$).

3.3 Alternating-time Temporal Logic

ATL is a logic that is standardly interpreted over CGSs where all strategies are deterministic and formulae are given by the following grammar where A is a set of agents:

$$\Phi ::= p \mid \neg\Phi \mid \Phi \wedge \Phi \mid \langle\langle A \rangle\rangle \bigcirc \Phi \mid \langle\langle A \rangle\rangle \Phi \mathscr{U} \Phi$$

Here the connectives $\langle\langle A \rangle\rangle \bigcirc \phi$ means that working together the agents in A can guarantee that (no matter the choices made by the other agents) that $\bigcirc\phi$ holds. Similarly, $\langle\langle A \rangle\rangle \phi_1 \mathscr{U} \phi_2$ means that the agents of A can guarantee that $\phi_1 \mathscr{U} \phi_2$ holds. Equivalently these can be viewed as instead asking "Are there strategies for the agents in A such that for all strategies for the agents not in A, $\bigcirc\phi$ (or $\phi_1 \mathscr{U} \phi_2$) holds?" Using this formulation, we can translate an ATL formula into a SLeRCN state formula over the same CGS. To simplify the quantification case, we will use the set of agents itself as our set of variables. The translation then is given by:[2]

$$[[p]] = p \quad [[\neg\phi]] = \neg[[\phi]] \quad [[\phi_1 \wedge \phi_2]] = [[\phi_1]] \wedge [[\phi_2]]$$
$$[[\langle\langle A \rangle\rangle \bigcirc \phi]] = \exists A.\forall \overline{A}.(\bigcirc[[\phi]])(\mathrm{id}) \wedge \bigwedge_{a \in \mathscr{A}} \mathrm{DETER}(a)$$
$$[[\langle\langle A \rangle\rangle \phi_1 \mathscr{U} \phi_2]] = \exists A.\forall \overline{A}.([[\phi_1]] \mathscr{U} [[\phi_2]])(\mathrm{id}) \wedge \bigwedge_{a \in \mathscr{A}} \mathrm{DETER}(a)$$

[2] id denotes the identity function.

This translation is just an implementation of the insight from above where we existentially quantify over the variables in each ATL quantifier and then universally quantify over the remaining variables and ensure that we only consider deterministic strategies.[3]

ATL* is a generalization of ATL where the quantifiers are not forced to appear with the temporal operators. Like SLeRCN, it has a syntactic bifurcation into state and path formula in it's grammar:

$$\Phi ::= \langle\langle A\rangle\rangle\Psi \mid \neg\Phi \mid \Phi\wedge\Phi \qquad p \mid \Phi \mid \neg\Psi \mid \Psi\wedge\Psi \mid \bigcirc\Psi \mid \Psi\mathscr{U}\Psi$$

Where satisfaction of $\langle\langle A\rangle\rangle\psi$ is informally "The agents in A can guarantee that ψ holds on all paths from the current history," and ϕ in a path formula means "ϕ holds from the current state." We can translate this into SLeRCN by splitting the prior translation for $\langle\langle A\rangle\rangle$ into separate parts:

$$[[\langle\langle A\rangle\rangle\psi]] = \exists A.\forall\overline{A}.[[\psi]](\mathrm{id}) \wedge \bigwedge_{a\in\mathscr{A}} \mathrm{DETER}(a) \quad [[\bigcirc\psi]] = \bigcirc[[\psi]] \quad [[\psi_1\mathscr{U}\psi_2]] = [[\psi_1]]\mathscr{U}[[\psi_2]]$$

3.4 Strategy Logic

SL [7] is a logicthat operates over two-player turn based games where all strategies are deterministic. Formulae in this logic are given by the following grammar, where x and y are strategy variables, and p is an atomic proposition drawn from the set Π:

$$\Phi ::= \Psi(x,y) \mid \Phi\wedge\Phi \mid \neg\Phi \mid \exists x.\Phi \qquad \Psi ::= p \mid \Phi \mid \Psi\wedge\Psi \mid \neg\Psi \mid \bigcirc\Psi \mid \Psi\mathscr{U}\Psi$$

As an additional restriction, any state formula directly contained in a path formula must be closed. We will call formulae that satisfy this property hierarchically closed. Here we have $\Psi(x,y)$ instead of $\Psi(\zeta)$ since, when restricted to two agents, instead of giving a generic agent specifier, giving a variable for agent 1 and a variable for agent 2 suffices. The semantics for this is roughly the same as presented for SLeRCN but with the following major differences. Two-player turn based game graphs are given by $G = (V,E,V_1,V_2,\pi)$ where (V,E) is a finite directed graph, $\{V_1,V_2\}$ is a partition of V with V_1 representing the states where it is agent 1's turn and V_2 those of agent 2, and π labels states with atomic propositions true in that state. The notion of a legal strategy then is a function that takes some history and returns a vertex adjacent to the last vertex in the history (i.e., a strategy σ is legal if $\forall\rho\in V^+.\sigma(\rho)\in\{v \mid (\rho_{|\rho|-1},v)\in E\}$). The notion of outcomes used in defining the behavior of $\psi(x,y)$ changes to account for deterministic strategies. Specifically, $\mathrm{OUT}(G,\rho\cdot q,\sigma_1,\sigma_2)$ is the limit of

$$\mathrm{OUT}_0(G,\rho,\sigma_1,\sigma_2) = \rho \qquad \mathrm{OUT}_{n+1}(G,\rho,\sigma_1,\sigma_2) = \mathrm{OUT}_n(G,\rho,\sigma_1,\sigma_2)\cdot\sigma_i(\mathrm{OUT}_n(G,\rho,\sigma_1,\sigma_2)) \text{ where } \mathrm{OUT}_n(G,\rho,\sigma_1,\sigma_2)_{|\rho|+n-1}\in V_i$$

To translate this into SLeRCN we need to convert the game graph into a CGS in the normal way. That is (V,E,V_1,V_2) goes to $(\{1,2\},V,V\cup\{\tau\},e,\Delta,\Pi,\pi)$ where

$$e(v,i) = \begin{cases}\{v' \mid E(v,v')\} & \text{if } v\in V_i\\ \tau & \text{otherwise}\end{cases} \qquad \Delta(v,f) = \{v'\} \quad \text{s.t. } \mathrm{range}(f)\backslash\{\tau\} = \{v'\}$$

[3] There is a straightforward generalization to a sort of nondeterministic ATL by removing all the occurrences of DETER.

We can then translate a SL formula into a SLeRCN formula:

$$\begin{aligned}
[[p]] &= p & [[\psi(x,y)]] &= [[\psi]](\{1 \mapsto x, 2 \mapsto y\}) \\
[[\neg\phi]] &= \neg[[\phi]] & [[\exists x.\phi]] &= \exists x.(\mathrm{DETER}(x) \wedge [[\phi]]) \\
[[\phi_1 \wedge \phi_2]] &= [[\phi_1]] \wedge [[\phi_2]] & [[\bigcirc\psi]] &= \bigcirc[[\psi]] \\
[[\neg\psi]] &= \neg[[\psi]] & [[\psi_1 \mathscr{U} \psi_2]] &= [[\psi_1]] \mathscr{U} [[\psi_2]] \\
[[\psi_1 \wedge \psi_2]] &= [[\psi_1]] \wedge [[\psi_2]]
\end{aligned}$$

3.5 Rely-Guarentee Temporal Logic

Rely-Guarentee Temporal Logic (RGTL) [11] is a logic designed to support expressing modular properties of compound systems. Syntactically, it may be viewed as extending Computation Tree Logic (CTL*) [3] with a rely-guarantee operator parametrized by sets of agents. The grammar of RGTL is as follows:

$$\Phi ::= p \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mathsf{A}\Psi \mid \Phi \overset{A}{\Rrightarrow} \Phi \qquad\qquad \Psi ::= \Phi \mid \Psi \wedge \Psi \mid \neg\Psi \mid \bigcirc\Psi \mid \Psi \mathscr{U} \Psi$$

As before, p ranges over a set Π of atomic propositions, Φ generates state formulae and Ψ generates path formulae. The state formula $\mathsf{A}\psi$ means along all forward paths starting in the current state, ψ holds (inevitably ψ). The state formula $\phi_1 \overset{A}{\Rrightarrow} \phi_2$ asserts that ϕ_2 holds, provided A may be relied upon to assure that ϕ_1 holds (ϕ_2 A-relies on ϕ_1).

The semantics of RGTL is given over CGSs in the similar manner to SLeRCN, including the use of nondeterministic strategies. We model state formulae by a CGS C, a system strategy σ (instead of a mapping from agent variables to agent strategies), and a history ρ. The incremental definitions of the semantics of the state formula constructs p, $\Phi \wedge \Phi$, and $\neg\Phi$, and all the path formula constructs are essentially the same as for SLeRCN. The state formula construct $\mathsf{A}\Psi$ is essentially the same is the comparable one from CTL*. Formally, we define its semantics by the following:

$$C, \sigma, \rho \vDash_\Phi \mathsf{A}\psi \quad \text{iff} \quad \forall \gamma \in \mathrm{OUT}(C, \rho, \sigma).C, \sigma, \rho, \gamma \vDash_\Psi \psi$$

The semantics of the new construct $\Phi \overset{A}{\Rrightarrow} \Phi$ uses three auxiliary functions on strategies. The first of these is restriction: $\sigma\|_A$, which is σ_a for $a \in A$, and the enabled function on the last state of the history ($e(\rho_{|\rho|-1}, a)$) otherwise. The second is system strategy refinement: $\sigma \sqsubseteq \tau$ iff $\sigma_a \sqsubseteq \tau_a$ for all agents a. The last (partial) function is the intersection of two strategies: $\sigma \sqcap \tau$ where $(\sigma \sqcap \tau)_a(\rho) = (\sigma_a(\rho) \cap \tau_a(\rho))$ for all agents a, and histories ρ, assuming all such intersections are nonempty, and $\sigma \sqcap \tau$ is undefined otherwise. Using these functions, the semantics of the state formula construct $\Phi \overset{A}{\Rrightarrow} \Phi$ is given by the following:

$$\begin{aligned}
C, \sigma, \rho \vDash_\Phi \phi_1 \overset{A}{\Rrightarrow} \phi_2 \quad \text{iff} \quad & \forall \tau.\ ((\tau\|_A \sqsubseteq \sigma\|_A) \wedge (\forall \nu.\ C, (\tau\|_A \sqcap \nu\|_{\overline{A}}), \rho \vDash_\Phi \phi_1)) \\
& \rightarrow C, (\sigma \sqcap (\tau\|_A)), \rho \vDash_\Phi \phi_2
\end{aligned}$$

To help the embedding RGTL into SLeRCN, note that we always have $(\tau\|_A \sqcap \nu\|_{\overline{A}}) = (\tau\restriction_A \cup \nu\restriction_{\overline{A}})$ and $(\tau\|_A \sqsubseteq \sigma\|_A) \rightarrow ((\sigma \sqcap \tau\|_A) = (\tau\restriction_A \cup \sigma\|_{\overline{A}}))$, where $\tau\restriction_A$ is ordinary partial function restriction.

When embedding RGTL in SLeRCN most connectives will be translated unchanged. The only extra complexity comes in the translation of inevitably and the rely-guarantee arrow. We will omit the explicit definition of the translation in all other cases, for space. To account for inevitably, we need to be able to

restrict the paths we consider to those consistent with the current system strategy. To support this, our translation will carry an extra argument ζ giving varaible names for the agent-strategy components of the currently assumed system strategy. To embed the syntax and semantics of RGTL into SLeRCN, we leave the underlying CGS unchanged, and use the operator $[[\_]]_{\_} : \text{RGTL} \times (\mathscr{A} \to V) \to \text{SLeRCN}$ to translate the syntax. In the definition given below, $[[\_]]_{\_}$ will map state formulae to state formulae and path formulae to path formulae.

To account for the rely-guarantee arrow, we need to be able to quantify over system strategies potentially different from the current assumed system strategy (or, at least different from it on subset A of the agents $\mathscr{A}$). SLeRCN uses explicit variables to express properties of strategies. To utilize this capability, we introduce an agent-indexed collection of infinite sets of variables $V = \{V_a\}_{a \in \mathscr{A}}$. We also introduce a mapping of agentes to fresh variables, $\text{FRESH} : \mathscr{P}_f(V) \to (\mathscr{A} \to V)$, which generates a mapping of agents to distinct variables (of the right "type"), where the range of the mapping is distinct from the given set of variables. SLeRCN provides us with an ability to express that one agent strategy refines another. Conjoining over all agents, this gives us a way of stating that one system strategy refines another. The definition of $[[\_]]_{\_}$ for inevitable and the gely-guarantee arrow is as follows:

Definition 5 (RGTL translation)**.**

$$
\begin{aligned}
[[A\psi]]_\zeta =& [[\psi]]_\zeta(\zeta) \\
[[\phi_1 \overset{A}{\Rrightarrow} \psi_2]]_\zeta =& \ \textit{Let } \zeta_1 = \text{FRESH}(\text{RNG}(\zeta)) \textit{ and } \zeta_2 = \text{FRESH}(\text{RNG}(\zeta) \cup \text{RNG}(\zeta_1)) \textit{ in} \\
& \forall(\text{RNG}(\zeta_1)).\ ((\textstyle\bigwedge_{a \in A} \zeta_1(a) \sqsubseteq \zeta(a)) \wedge (\forall(\text{RNG}(\zeta_2)).[[\phi_1]]_{\zeta_1 \restriction_A \cup \zeta_2 \restriction_{\overline{A}}})) \\
& \qquad \to [[\phi_2]]_{\zeta_1 \restriction_A \cup \zeta \restriction_{\overline{A}}}
\end{aligned}
$$

With the definition, we can show:

Lemma 6. *C be a CGS with agent set $\mathscr{A}$ and atomic propostions Π. Let ϕ and ψ be state and path RGTL formulae over the agent set $\mathscr{A}$ and atomic propositions Π. Let $\{x_a | a \in \mathscr{A}\}$ be a set of variables indexed by $\mathscr{A}$. Let $\zeta(a) = x_a$. Let σ be a system strategy and Γ be the environment such that $\Gamma(x_a) = \sigma_a$. Let ρ be a history of C, and $\gamma \in \text{OUT}(C, \rho, \sigma)$. Then*

$$
\begin{aligned}
C, \sigma, \rho \vDash^{RGTL}_{\Phi} \phi \quad & \textit{iff} \quad C, \Gamma, \rho \vDash^{SLeRCN}_{\Phi} [[\phi]]_\zeta \\
C, \sigma, \rho\gamma \vDash^{RGTL}_{\Psi} \psi \quad & \textit{iff} \quad C, \Gamma, \rho\gamma \vDash^{SLeRCN}_{\Psi} [[\psi]]_\zeta
\end{aligned}
$$

This translation and the result of Section 4.2 will yield that model checking RGTL is decidable.

4 Model Checking

Knowing that SLeRCN allows us to express interesting properties isn't everything we'd like to know; we haven't yet seen if the model checking problem is decidable for SLeRCN. Specifically, we would like to answer the question "Given CGS C, initial state q_0, and closed state formula ϕ, does $C, \emptyset, q_0 \vDash_\Phi \phi$ hold?" Before considering the general case (Section 4.2), we examine the case where strategies are positional (Section 4.1).

4.1 Positional Model Checking

A strategy is called positional if it only considers its current position when making decisions. Formally, a strategy σ is positional if $\forall q \in Q. \forall \rho\ \rho' \in Q^*. \sigma(\rho \cdot q) = \sigma(\rho' \cdot q)$. This definition has two immediate and

$\underline{\text{POSITIONAL}(C,\Gamma,q,\phi):}$

case ϕ of

$p \rightarrow p \in \pi(q)$

$\phi_1 \wedge \phi_2 \rightarrow$ Positional(C,Γ,q,ϕ_1) and Positional(C,Γ,q,ϕ_2)

$\neg\phi \rightarrow$ not Positional(C,Γ,q,ϕ)

$\exists x_a.\phi \rightarrow$ any $\sigma \in \Theta_a$ s.t. Positional$(C,\Gamma[x_a \leftarrow \sigma],q,\phi)$

$x_a \sqsubseteq y_a \rightarrow$ for all $q \in Q$, $\Gamma(x_a) \subseteq \Gamma(y_a)$ holds

$\psi(\zeta) \rightarrow \text{LTL}(\text{CONVERT}(C,\Gamma,q,\zeta), \text{SIMPLIFY}(C,\Gamma,\psi))$

Figure 1: Model Checking Positionally

important consequences: the set of positional strategies for an agent is finite (and is a subset of $\left(2^Q\right)^Q$); satisfaction is sensitive only to the most recent state visited (i.e., $C,\Gamma,\rho\cdot q \vDash \phi \iff C,\Gamma,q \vDash \Phi$). Another way to think about positional strategies is as restrictions of the enabled function of their CGS.

Figure 4.1 outlines how to perform model checking of SLeRCN formulae over positional strategies. Most of this is just a straightforward implementation of our semantics in terms of POSITIONAL. Worth noting is, in the existential quantification case, since the set of legal strategies is finite, we can examine all the possibilities directly. A more complicated case is needed to handle path formulae. Our use of "off-the-shelf" Linear Temoral Logic (LTL) [15] model checking is encapsulated by the LTL function. Unfortunately, we need to do a bit of preprocessing first to account for a few complications. First, CGSs are not the traditional basis for LTL satisfaction. To handle this, we can just check our LTL formula against the function graph of the CGS transition function and some distinguished initial state. Secondly, we need to account for the fact that not all paths through the CGS are actually results allowed by the strategies of $\Gamma\circ\zeta$. Note that positional strategies are essentially just redefinitions of the enabled function of the CGS. As a result, we can just specialize the CGS before handing it off to our LTL model checker. We encapsulate both of these observations in the function CONVERT.

The last complication is that a path formula ψ isn't always an LTL formula, but may have complicated state formulae contained inside of it. Note that for a fixed CGS, state formula, and environment there are only a finite number of potentially "interesting" calls to POSITIONAL (i.e., one per state in the CGS). As a result of this along with the earlier observation that satisfaction only depends on the most recent state, these calls to POSITIONAL, where only the state varies, are essentially atomic predicates. We can utilize this by replacing complex state formulae with (new) atomic predicates and thus simplify ψ into an LTL formula. As an example, suppose ψ contains a state formula ϕ. Then we would replace ϕ with $p_{C,\Gamma,\phi}$ (i.e., $\psi[p_{C,\Gamma,\phi}/\phi]$), where $p_{C,\Gamma,\phi} \in \pi(q)$ iff POSITIONAL(C,Γ,q,ϕ). This process is represented by the function SIMPLIFY in Figure 4.1.

After solving our complications, the above algorithm has a substantial amount of recomputation of calls to POSITIONAL so memoization can be profitably used. This gives a algorithm that is in EXPTIME, with the cost being mostly a factor of the large number of entries in the memoization table and the quantification case.

4.2 General Model Checking

The general case, where strategies are fully history sensitive, unfortunately will not work out as nicely. To handle this more complicated case, we first review the definition of Alternating Parity Tree Automatas (APTs) and some classic results we'll need to describe our model checking algorithm (Section 4.2.1). With

these we describe our model checking algorithm (Section 4.2.2) and then describe some opportunities for optimizations (Section 4.2.3).

4.2.1 Trees and Automata

Trees can be thought of as prefix closed sets of strings over a fixed set of directions. Of interest here are full infinite Υ-labeled D-trees which are given by a tuple (D, Υ, υ) where D is a (usually finite) set of directions, Υ is a (usually finite) set of labels that can label nodes in the tree, and $\upsilon : D^* \to \Upsilon$ is a labeling function. Each node n in the tree is a string in D^* (i.e., a node is the path taken from the root to reach it) and its associated label is $\upsilon(n)$.

Since the set of full infinite trees is uncountable, if $|D| \geq 2$ and $|\Upsilon| \geq 2$, working with them directly is hard. One approach is to use tree automata to reason about sets of trees. This is a generalization of the way that (infinite) word automata allow for reasoning about sets of (infinite) words. In this paper we use APTs, which have a more thorough treatment elsewhere [13], but we summarize the relevant results here.

Informally, APTs allow us to specify programs that have a state at a position in the input tree and then can perform certain actions: they can read the label of the current node and nondeterministically split to send any number of copies of themselves (possibly) in different states to any of the current node's children that continue processing in parallel. As an example, suppose we wanted to check that the ATL formula $\langle\langle\emptyset\rangle\rangle X\phi$ starting from the root of the tree holds, our program might contain a step like "For all children of the current node, send a copy of the program to that child and call the function that checks if ϕ holds starting from that child."

Positive boolean formulae over a set P, denoted $\mathbb{B}^+(P)$, are propositional formulae that are formed out of atomic predicates in P, conjunction, disjunction, true, and false. As usual for a boolean formula this specifies a subset of 2^P that satisfies the formula, denoted $\models_{\mathbb{B}^+}$.

Definition 6 (Alternating Parity Tree Automata [13])**.** *An APT is a tuple* $(\mathfrak{S}, D, \Upsilon, \delta, q_0, \chi)$ *where*

- $\mathfrak{S}$ *is a finite set of states.*
- D *is a finite set of directions.*
- Υ *is a finite input alphabet.*
- $\delta : \mathfrak{S} \times \Upsilon \to \mathbb{B}^+(D \times \mathfrak{S})$ *is a transition function.*
- $\chi : \mathfrak{S} \to \mathbb{N}$ *is a coloring.*

The size of an APT A, denoted $|A|$*, is* $|\mathfrak{S}|$*. The set of trees accepted by A, the language of A, is denoted* $L(A)$.

Lastly, we need a handful of basic results to build our algorithm out of:

1. We can construct an APT that accepts a tree if and only if every infinite path of the tree satisfies an LTL formula ψ with $O(|\psi|)$ states. [16]
2. Given an APT A over Υ-labeled D-trees we can construct an APT $\bar{A}$ over Υ-labeled D-trees with $L(\bar{A}) = \overline{L(A)}$ and the same number of states. [13]
3. Given two APTs A and B over Υ-labeled D-trees we can construct an APT $A \cap B$ with $L(A \cap B) = L(A) \cap L(B)$ and $|A| + |B| + 1$ states. [13]
4. Given an APT A over Υ_1-labeled D-trees and a surjection $f : \Upsilon_1 \to \Upsilon_2$ we can construct an APT $f(A)$ s.t., $(D, \Upsilon, \upsilon) \in L(f(A))$ if and only if some tree in $\{(D, \Upsilon, \upsilon') \mid \forall n \in D^*.\upsilon'(n) \in f^{-1}(\upsilon(n))\}$ is also in $L(A)$. We can also perform this construction in a universal fashion. The resulting APT has a number of states exponential in the size of A. [13, 14]

5. Given a $p \in 2^{\Upsilon}$ we can construct an APT $\textsc{Pred}(p)$ over Υ-labeled D-trees where $(D, \Upsilon, \upsilon) \in L(\textsc{Pred}(p)) \iff \forall n.\upsilon(n) \in p$. This construction has 2 states.
6. Given an APT A we can decide $L(A) = \emptyset$ in time exponential in $|A|$. [10, 14]

4.2.2 The Algorithm

Unfortunately, the approach that worked in the positional case will fail in the general case because there are an infinite number of strategies to consider when handling quantification, and hence, an infinite set of "interesting" environments even after using Lemma 1. Tree automata will allow for the representation of the set of satisfying environments finitely and for reducing model checking to checking emptiness of a suitably constructed automata. To do this, environments are encoded as trees in the following manner. Histories are viewed as paths in a Q-tree. This gives a natural encoding of a strategy as a 2^Q-labeled Q-tree by (for a given strategy σ) having each node ρ labeled by $\sigma(\rho)$. This approach can be generalized give what is essentially a pointwise description of an environment Γ as a $\left(2^Q\right)^V$-labeled Q-tree with node ρ labeled by $(\lambda x.\Gamma(x)(\rho))$. In the above two definitions there is a slight problem: since strategies are only defined on Q^+, the label of the root is undefined. To cope with this, the root's label can be fixed with either a special root label ε or by assuming that our trees only records the behavior of an environment starting from some initial state q_0 (i.e., our labels would be given by $\lambda x.\Gamma(x)(q_0 \cdot \rho)$). From Lemma 2, we know that the second option will be sufficient. To simplify the following constructions we'll also alter the labeling slightly to be $\{(q,\Gamma) \in Q \times \left(2^Q\right)^V \mid \forall x_a \in \mathrm{dom}(\Gamma).\Gamma(x_a) \neq \emptyset \wedge \Gamma(x_a) \subseteq e(q,a)\}$, that is, each label encodes a state and only legal strategies for that state. This shift will allow us to quantify over all strategies easily rather than over all "strategies" (including illegal ones) as would naturally arise with the first encoding. The same goal could be reached if the actions allowed by each state were distinct (i.e., we could figure out which state we're looking at just by examining a nonempty set of actions). The last simplifying assumption that we'll want to make is to assume that there are no shadowing bindings (which can always be guaranteed by α-conversion).

Throughout this construction, we want to generate an APT that accepts environments satisfying a given SLeRCN formula (encoded as trees). We can then use emptiness to decide if there are any satisfying environments, or for closed state formulae, the model checking problem. To ensure that our label set is finite, we need to take a bit of care in ensuring that our environments defined exactly for some known finite set of variables at each step of the construction. The following is a sketch of the inductively built APT $[[\phi]]_X$ for ϕ over environments Γ with $\textsc{dom}(\Gamma) = X$.

- For $[[\psi(\zeta)]]_X$, return the standard LTL construction [16] to check trees with two minor changes. First, whenever the standard construction would move to new nodes in the tree, only move to those specified by the strategies selected by ζ and the local environment label. Secondly, whenever the algorithm would need to build an automata to check some state formula ϕ (which it, of course, it doesn't know how to do), use instead the APT $[[\phi]]_X$.
- For $[[p]]_X$, return the automata that checks the current state q (given by the label) and either succeeds or fails depending on $p \in \pi(q)$.
- For $[[\phi_1 \wedge \phi_2]]_X$ or $[[\neg\phi]]_X$ return either $[[\phi_1]] \cap [[\phi_2]]$ or $\overline{[[\phi]]_X}$ as appropriate.
- For $[[x_a \sqsubseteq y_a]]_X$ then return the automata that does a pointwise verification of this property. That is, at every node (each labeled by some (q,Γ)) check that $\Gamma(x_a) \subseteq \Gamma(y_a)$ and then recursively check this on all the node's children.
- For $[[\exists x_a.\phi]]_X$, find $[[\phi]]_{X \cup \{x_a\}}$ and then use closure under the function $(q,\Gamma) \mapsto (q, \Gamma\restriction_X)$.

For a closed state formula ϕ, $[[\phi]]_\emptyset$ is an APT that accepts $Q \times \{\emptyset\}$-labeled Q-trees. A slight complication is that while the set of legal environments with no variables is very simple, we haven't enforced that our initial tree is labeled with the correct states (i.e., a designated $(q_0, \emptyset)$ for the root and for node ρ, $(\rho_{|\rho|-1}, \emptyset)$). Let $\mathrm{DIR}^Q_{q_0}$ be an APT that ensures that all the entries in the input tree meet the above requirements. The last step is then to check if $\mathrm{DIR}^Q_{q_0} \cap [[\phi]]_\emptyset = \emptyset$.

Lemma 7 (Model Checking Decidability)**.** *For a CGS $C = (\mathscr{A}, Q, \Sigma, e, \Delta, \Pi, \pi)$, initial state q_0, and closed state formula ϕ, $\mathrm{DIR}^Q_{q_0} \cap [[\phi]]_\emptyset = \emptyset$ iff $C, \emptyset, q_0 \vDash_\Phi \phi$.*

4.2.3 Running Time

Handling quantification is incredibly expensive in comparison to the other cases the construction handles. As a result the automata built for a formula ϕ has a number states that is in the worst case a tower of exponentials of height $|\phi|$ (or more precisely, $\mathrm{QR}(\phi)$, where QR is the quantifier rank of ϕ). Applying APT language emptiness gives us that model checking for closed state formulae ϕ is in $(1 + \mathrm{qr}(\phi))$-EXPTIME.

In practice, there are a few optimizations that are both simple and worth considering. First, the bulk of the cost in handling quantification lies in performing a nondeterminizing of the recursively constructed APT. One way to make use of this observation is to try to ensure that our formula uses bunched quantification (Lemma 4) as much as possible (i.e., for a set of variables V, handle $\exists V.\phi$ analogously to $\exists x_a.\phi$). This change can reduce a formula's quantifier rank in some cases. Second, bounded quantification with the DETER macro occurs fairly often (Sections 3.3 and 3.4) and can be recognized and handled specially. Essentially, this change involves swapping the component of that label from a nondeterministic choice of states to a deterministic one, i.e., the label set would change to $Q \times (2^Q)^{V_{\mathrm{nd}}} \times Q^{V_{\mathrm{d}}}$ where V_{nd} are those variables that may have nondeterministic strategies assigned to them and V_{d} are the known deterministic variables. This change removes a quantifier from formulae that it affects, giving us an opportunity to save a quantifier rank in those formulae.

The last optimization discussed here involves identifying pieces of work that can be performed separately and then compactly summarized for use in the general model checking algorithm. Closed state formula have three properties of interest that will allow them to be handled in isolation: since these state formulae are closed only their satisfaction on empty environments matters (i.e., $C, \Gamma, \rho \vDash \phi \iff C, \emptyset, \rho \vDash \Phi$); only the ongoing behavior of strategies matter (i.e., $(\forall \rho' \in Q^*.\sigma(\rho \cdot \rho') = \sigma'(\rho \cdot \rho')) \implies (C, \Gamma[x_a \leftarrow \sigma], \rho \vDash \phi \iff C, \Gamma[x_a \leftarrow \sigma'], \rho \vDash \phi)$); and we can "bake-in" histories to certain strategies (i.e., $\forall \sigma.\exists \sigma'.\forall \rho'.\sigma(\rho \cdot \rho') = \sigma(\rho_{|\rho|-1} \cdot \rho')$) and, similarly, we can make strategies forget some initial amount of history. Akin to how we handled state formulae in the positional case (Section 4.1), these observations allow us to substitute a predicate p_ϕ for a closed state formula ϕ contained inside a larger formula. In profitable cases, this allows us to exchange one very expensive problem for $|Q|$ $(\mathrm{qr}(\phi) + 1)$-EXPTIME problems plus one problem for the residual formula. Hopefully, this has a reduced quantifier rank and so the split will save some time. At the very least, this exposes an opportunity for parallelism.

5 Acknowledgments

This material is based upon work supported in part by NASA Contract NNA10DE79C and NSF Grant 0917218. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NASA or NSF.

References

[1] Thomas Ågotnes, Valentin Goranko & Wojciech Jamroga (2007): *Alternating-time temporal logics with irrevocable strategies*. In Dov Samet, editor: *TARK*, pp. 15–24. Available at http://doi.acm.org/10.1145/1324249.1324256.

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

[3] Mordechai Ben-Ari, Zohar Manna & Amir Pnueli (1981): *The Temporal Logic of Branching Time*. In John White, Richard J. Lipton & Patricia C. Goldberg, editors: *POPL*, ACM Press, pp. 164–176. Available at http://doi.acm.org/10.1145/567532.567551.

[4] Thomas Brihaye, Arnaud Da Costa Lopes, François Laroussinie & Nicolas Markey (2009): *ATL with Strategy Contexts and Bounded Memory*. In Sergei N. Artëmov & Anil Nerode, editors: *LFCS*, *Lecture Notes in Computer Science* 5407, Springer, pp. 92–106. Available at http://dx.doi.org/10.1007/978-3-540-92687-0_7.

[5] Jan Broersen, Andreas Herzig & Nicolas Troquard (2006): *A STIT-Extension of ATL*. In Michael Fisher, Wiebe van der Hoek, Boris Konev & Alexei Lisitsa, editors: *JELIA*, *Lecture Notes in Computer Science* 4160, Springer, pp. 69–81. Available at http://dx.doi.org/10.1007/11853886_8.

[6] Krishnendu Chatterjee, Thomas A. Henzinger & Marcin Jurdzinski (2006): *Games with secure equilibria*. *Theor. Comput. Sci.* 365(1-2), pp. 67–82. Available at http://dx.doi.org/10.1016/j.tcs.2006.07.032.

[7] Krishnendu Chatterjee, Thomas A. Henzinger & Nir Piterman (2010): *Strategy logic*. *Inf. Comput.* 208(6), pp. 677–693. Available at http://dx.doi.org/10.1016/j.ic.2009.07.004.

[8] Catalin Dima, Constantin Enea & Dimitar P. Guelev (2010): *Model-Checking an Alternating-time Temporal Logic with Knowledge, Imperfect Information, Perfect Recall and Communicating Coalitions*. In Angelo Montanari, Margherita Napoli & Mimmo Parente, editors: *GANDALF*, *EPTCS* 25, pp. 103–117. Available at http://dx.doi.org/10.4204/EPTCS.25.12.

[9] Jr. John Forbes Nash (1950): *Equilibrium points in n-person games*. *Proceedings of the National Academy of Sciences USA* (36), pp. 48–49.

[10] Orna Kupferman & Moshe Y. Vardi (1998): *Weak Alternating Automata and Tree Automata Emptiness*. In: *STOC*, pp. 224–233. Available at http://doi.acm.org/10.1145/276698.276748.

[11] William Mansky & Elsa Gunter (2012): *Using Locales to Define a Rely-Guarantee Temporal Logic*. In Lennart Beringer & Amy Felty, editors: *Interactive Theorem Proving, (ITP 2012)*, LNCS, Springer Verlag, Princeton, New Jersey. In press.

[12] Fabio Mogavero, Aniello Murano & Moshe Y. Vardi (2010): *Relentful Strategic Reasoning in Alternating-Time Temporal Logic*. In Edmund M. Clarke & Andrei Voronkov, editors: *LPAR (Dakar)*, *Lecture Notes in Computer Science* 6355, Springer, pp. 371–386. Available at http://dx.doi.org/10.1007/978-3-642-17511-4_21.

[13] David E. Muller & Paul E. Schupp (1987): *Alternating Automata on Infinite Trees*. *Theor. Comput. Sci.* 54, pp. 267–276. Available at http://dx.doi.org/10.1016/0304-3975(87)90133-2.

[14] David E. Muller & Paul E. Schupp (1995): *Simulating Alternating Tree Automata by Nondeterministic Automata: New Results and New Proofs of the Theorems of Rabin, McNaughton and Safra*. *Theor. Comput. Sci.* 141(1&2), pp. 69–107. Available at http://dx.doi.org/10.1016/0304-3975(94)00214-4.

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

[16] Moshe Y. Vardi (1995): *Alternating Automata and Program Verification*. In Jan van Leeuwen, editor: *Computer Science Today*, *Lecture Notes in Computer Science* 1000, Springer, pp. 471–485. Available at http://dx.doi.org/10.1007/BFb0015261.