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-Guarantee 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.
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 $\mathcal{A}$, an agent indexed set of variables $\{V_a\}_{a \in \mathcal{A}}$, and a set of atomic propositions $\Pi$ we can form a SLeRCN formula over variables $\{V_a\}_{a \in \mathcal{A}}$ via the following grammar. All variables are
associated with an owning agent, their index in $\{V_a\}_{a \in \mathcal{A}}$. 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 $\zeta$ is a mapping from agents to appropriate variables (e.g., $a \mapsto x_a$
for $x_a \in V_a$).

$$\Phi ::= p \mid \Psi(\zeta) \mid \Phi \land \Phi \mid \neg \Phi \mid \exists x_a.\Phi \mid x_a \sqsubseteq y_a$$

$$\Psi ::= \Phi \mid \Psi \land \Psi \mid \neg \Psi \mid \bigcirc \Psi \mid \Psi \bigotimes \Psi$$

Formulae generated from the $\Phi$ nonterminal are state formulae and those from the $\Psi$ 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 $\psi$ hold along all the paths specified
by the strategies in $\zeta$?”; $x_a \sqsubseteq y_a$ 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 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 $(\mathcal{A}, Q, \Sigma, e, \Delta, \Pi, \pi)$ s.t.

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

We lift $\Delta$ to $\Delta : Q \times (2^\Sigma)^{\mathcal{A}} \to 2^Q$ 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 \( (\mathcal{A}, Q, \Sigma, e, \Delta, \Pi, \pi) \) and \( \mathcal{A} \subseteq Q \rightarrow 2^\Sigma \) is an \( \mathcal{A} \)-strategy if \( \forall \rho \in Q^+, \mathcal{A}(\rho) \neq \emptyset \land \mathcal{A}(\rho) \subseteq e(\rho_{[\rho]}^{-1}, a) \) (i.e., it makes a selection from the actions allowed by the enabled function). We will denote the set of \( \mathcal{A} \)-strategies by \( \Theta_\mathcal{A} \). A strategy \( \sigma \) is called deterministic if for every history it chooses exactly one action (i.e., for every \( \rho \in Q^+ \), \( \mathcal{A}(\rho) \) is a singleton). A system strategy \( \sigma : a \in \mathcal{A} \rightarrow \Theta_a \) is an \( \mathcal{A} \)-strategy for every agent. We will write \( \sigma_a \) for the \( \mathcal{A} \)-strategy component of the system strategy \( \sigma \). The set of all strategy is \( \Theta \). We lift \( \sigma \in \Theta \) to \( \sigma : Q^+ \rightarrow 2^\Sigma \) 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 \( C = (\mathcal{A}, Q, \Sigma, e, \Delta, \Pi, \pi) \), a history \( \rho \in Q^+ \), and a system strategy \( \sigma \in \Theta \), the set of outcomes of length \( n \) is defined by

\[
\Theta \subseteq Q^+ \land \forall \mathcal{A}(\rho) \subseteq e(\rho_{[\rho]}^{-1}, a) \text{ (i.e., it makes a selection from the actions allowed by the enabled function).}
\]

We will denote the set of \( \mathcal{A} \)-strategies by \( \Theta_\mathcal{A} \). A strategy \( \sigma \) is called deterministic if for every history it chooses exactly one action (i.e., for every \( \rho \in Q^+ \), \( \mathcal{A}(\rho) \) is a singleton). A system strategy \( \sigma : a \in \mathcal{A} \rightarrow \Theta_a \) is an \( \mathcal{A} \)-strategy for every agent. We will write \( \sigma_a \) for the \( \mathcal{A} \)-strategy component of the system strategy \( \sigma \). The set of all strategy is \( \Theta \). We lift \( \sigma \in \Theta \) to \( \sigma : Q^+ \rightarrow 2^\Sigma \) 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 4 (\( \rho \)-Refinement).** Given two \( \mathcal{A} \)-strategies \( \sigma_1 \) and \( \sigma_2 \) and \( \rho \in Q^+ \), \( \sigma_1 \) is a \( \rho \)-refinement of \( \sigma_2 \), written \( \sigma_1 \sqsubseteq_\rho \sigma_2 \), if \( \forall \rho' \in Q^+ \sigma_1(\rho') \subseteq \sigma_2(\rho' \cdot \rho') \).

Given an CGS \( C = (\mathcal{A}, Q, \Sigma, e, \Delta, \Pi, \pi) \), an environment \( \Gamma \) mapping variables to strategies of the appropriate type (e.g., \( \Gamma(x_a) \in \Theta_a \)), a history \( \rho \in Q^+ \), and \( \gamma \in Q^0 \), we define the SLRC satisfaction relations (\( \vdash_\phi \) and \( \vdash_\psi \) for state and path formulae respectively) by

\[
\begin{align*}
C, \Gamma, \rho & \vdash_\phi p & \text{ iff } & p \in \pi(\rho_{[\rho]}^{-1}) \\
C, \Gamma, \rho & \vdash_\psi (\zeta) & \text{ iff } & \forall \gamma \in \text{OUT}(C, \rho, \Gamma \circ \zeta), C, \Gamma, \rho, \gamma \vdash_\psi \psi \\
C, \Gamma, \rho & \vdash_\phi (\phi_1 \land \phi_2) & \text{ iff } & C, \Gamma, \rho \vdash_\phi \phi_1 \land C, \Gamma, \rho \vdash_\phi \phi_2 \\
C, \Gamma, \rho & \vdash_\phi \neg \phi & \text{ iff } & \text{ not } C, \Gamma, \rho \vdash_\phi \phi \\
C, \Gamma, \rho & \vdash_\phi \exists x_a \phi & \text{ iff } & \exists \sigma \in \Theta_a, C, \Gamma[x_a \leftarrow \sigma], \rho \vdash_\phi \phi \\
C, \Gamma, \rho & \vdash_\psi (\psi_1 \land \psi_2) & \text{ iff } & \Gamma(y_a) \subseteq \rho \vdash_\psi (\psi_1 \land \psi_2) \\
C, \Gamma, \rho & \vdash_\psi \neg \psi & \text{ iff } & \text{ not } C, \Gamma, \rho \vdash_\psi \psi \\
C, \Gamma, \rho & \vdash_\psi (\psi_1 \lor \psi_2) & \text{ iff } & C, \Gamma, \rho \vdash_\psi \psi_1 \lor C, \Gamma, \rho \vdash_\psi \psi_2 \\
C, \Gamma, \rho & \vdash_\psi \psi_1 \land \psi_2 & \text{ iff } & C, \Gamma, \rho \vdash_\psi \psi_1 \land \psi_2 \\
\end{align*}
\]

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 \( \text{OUT} \) to generate all the paths that need checking. Additionally, the refinement case \( (x_a \subseteq y_a) \) 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 $\Gamma_1$ and $\Gamma_2$, history $\rho \in Q^+$, and state formula $\phi$, if $\Gamma_1 \upharpoonright_{\text{FN}(\phi)} = \Gamma_2 \upharpoonright_{\text{FN}(\phi)}$ then $(C, \Gamma_1, \rho \models_\phi \phi) = (C, \Gamma_2, \rho \models_\phi \phi)$ 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 $\Gamma_1$ and $\Gamma_2$, history $\rho \in Q^+$, and state formula $\phi$, if $\forall v \in V, (\forall \rho^1, \Gamma_1(v)(\rho \cdot \rho^1) = \Gamma_2(v)(\rho \cdot \rho^1))$ then $(C, \Gamma_1, \rho \models_\phi \phi) = (C, \Gamma_2, \rho \models_\phi \phi)$ 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 $\Gamma$, history $\rho \cdot q \in Q^+$, and closed formula $\phi$, $(C, \Gamma, \rho \cdot q \models_\phi \phi) = (C, \Gamma, q \models_\phi \phi)$.

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 $\Gamma$, history $\rho \in Q^+$, and formula $\phi$, $C, \Gamma, \rho \models_\phi \exists x_1. \exists y_2. \phi = C, \Gamma, \rho \models_\phi \exists y_2. \exists x_1. \phi$ and $(C, \Gamma, \rho \models_\phi \exists x_1. \exists x_2. \phi) = (C, \Gamma, \rho \models_\phi \exists x_2. \exists x_1. \phi)$.

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 $\forall y_a. (y_a \subseteq x_a \to x_a \subseteq y_a)$. 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 DETER($x_a$).

Lemma 5 (Future Determinism). For CGS C, environment $\Gamma$, history $\rho$, variable $x_a$, and state formula $\phi$, there exists a deterministic strategy $\sigma$ s.t., $C, \Gamma[x_a \leftarrow \sigma], \rho \models_\phi \phi$ if and only if $C, \Gamma, \rho \models_\phi \exists x_a. \left(\text{DETER}(x_a) \land \phi\right)$.

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

(⇒): 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 q₀, an indexed family of closed path formulae \( \{ \psi_a \}_{a \in \mathcal{A}} \) called winning conditions, and a system strategy \( \sigma \in \Theta \) the payoff for an agent \( a \) is

\[
p_a(\sigma) = \begin{cases} 
1 & \text{if } \forall \gamma \in \text{OUT}(C, q_0, \sigma). C, q_0, \gamma \models \psi_a \\
0 & \text{otherwise}
\end{cases}
\]

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

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

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

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

\[
\text{PAYOFF}(P, \xi) \equiv \bigwedge_{a \in P} (\psi_a)(\xi) \land \bigwedge_{a \in (\mathcal{A} \setminus P)} \neg (\psi_a)(\xi)
\]

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 in SLeRCN we take the disjunction across all payoff profiles (\( \bigvee_{P \subseteq \mathcal{A}} \text{NASH}(P) \)). It is also possible to give a slightly different definition of Nash equilibria where, instead of winning only when the winning condition is always 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 \( a \in \mathcal{A} \) by the binary relation on payoff profiles \( \preceq_a \subseteq 2^{\mathcal{A}} \times 2^{\mathcal{A}} \), given by

\[
P_1 \preceq_a P_2 \equiv (a \notin P_1 \land a \in P_2(a)) \lor ((a \in P_1 \iff a \in P_2) \land \forall a' \in \mathcal{A}. (a' \in P_2 \implies a' \in P_1))
\]

\( \preceq_a \) is the strict version of \( \preceq_a \).

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

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

For a given payoff profile \( P \), we can define a SLeRCN state formula \( \text{SECURE}(P) \) 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 \( a \in \mathcal{A} \), \( X = \{ x_a \mid a \in \mathcal{A} \} \), and \( \zeta_X(a) = x_a \). We define \( \text{SECURE}(P) \) as follows:

\[
\text{SECURE}(P) = \exists X. \left( \text{PAYOFF}(P, \zeta_X) \land \bigwedge_{a \in \mathcal{A}} \forall y_a. \bigwedge_{P' \subseteq \mathcal{A}, P \preceq_a P'} \neg \text{PAYOFF}(P', \zeta_X[a \leftarrow y_a]) \right)
\]

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 (\( \bigvee_{P \subseteq \mathcal{A}} \text{SECURE}(P) \)).

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:

\[
\Phi ::= p \mid \neg \Phi \mid \Phi \land \Phi \mid \langle\langle A\rangle\rangle \text{O} \Phi \mid \langle\langle A\rangle\rangle \text{U} \Phi \mid \Phi
\]

Here the connectives \( \langle\langle A\rangle\rangle \text{O} \phi \) means that working together the agents in \( A \) can guarantee that (no matter the choices made by the other agents) that \( \text{O} \phi \) holds. Similarly, \( \langle\langle A\rangle\rangle \phi_1 \text{U} \phi_2 \) means that the agents of \( A \) can guarantee that \( \phi_1 \text{U} \phi_2 \) 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 \( A \), \( \text{O} \phi \) (or \( \phi_1 \text{U} \phi_2 \)) 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 by²

\[
[[p]] = p \quad [[\neg \phi]] = \neg [[\phi]] \quad [[\phi_1 \land \phi_2]] = [[\phi_1]] \land [[\phi_2]]
\]

\[
[[\langle\langle A\rangle\rangle \text{O} \phi]] = \exists A. \forall \mathcal{A}. (\langle\langle \phi \rangle\rangle(id) \land \bigwedge_{a \in \mathcal{A}} \text{DETER}(a))
\]

\[
[[\langle\langle A\rangle\rangle \phi_1 \text{U} \phi_2]] = \exists A. \forall \mathcal{A}. (\langle\langle \phi_1 \rangle\rangle \text{U} \langle\langle \phi_2 \rangle\rangle(id) \land \bigwedge_{a \in \mathcal{A}} \text{DETER}(a))
\]

²id 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.\footnote{There is a straightforward generalization to a sort of nondeterministic ATL by removing all the occurrences of DETER.}

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 its grammar:

\[
\Phi ::= \langle\langle A\rangle\rangle \Psi | \neg \Phi | \Phi \land \Phi | \emptyset \Phi | \Psi | \Psi \land \Psi
\]

Where satisfaction of \(\langle\langle A\rangle\rangle \Psi\) is informally “The agents in A can guarantee that \(\psi\) holds on all paths from the current history,” and \(\emptyset \Phi\) in a path formula means “\(\phi\) 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:

\[
[[\langle\langle A\rangle\rangle \Psi]] = \exists A. \forall A. ([[\psi]](id) \land \bigwedge_{a \in \mathcal{A}} \text{DETER}(a)) \quad [[\neg \psi]] = \bigcirc [[\psi]] \quad [[\psi_1 \lor \psi_2]] = [[\psi_1]] \lor [[\psi_2]]
\]

### 3.4 Strategy Logic

SL \([7]\) is a logic that 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 \(\Pi\):

\[
\Phi ::= \Psi(x, y) | \Phi \land \Phi | \neg \Phi | \exists x. \Phi \quad \Psi ::= p | \Phi | \Psi \land \Psi | \neg \Psi | \bigcirc \Psi | \Psi \lor \Psi
\]

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 \(\pi\) 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 \(\sigma\) is legal if \(\forall \rho \in V^+. \sigma(\rho) \in \{v \mid (\rho|_{[v]} \in V_1)\}\)).

The notion of outcomes used in defining the behavior of \(\psi(x, y)\) changes to account for deterministic strategies. Specifically, \(\text{OUT}(G, \rho \cdot q, \sigma_1, \sigma_2)\) is the limit of

\[
\text{OUT}_0(G, \rho, \sigma_1, \sigma_2) = \rho \quad \text{OUT}_{n+1}(G, \rho, \sigma_1, \sigma_2) = \text{OUT}_n(G, \rho, \sigma_1, \sigma_2) \cdot \sigma_1(\text{OUT}_n(G, \rho, \sigma_1, \sigma_2))
\]

where \(\text{OUT}_n(G, \rho, \sigma_1, \sigma_2)|_{\rho|_{[\rho]}} \in V_i\)

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

\[
e(v, i) = \begin{cases} 
\{v' \mid E(v, v')\} & \text{if } v \in V_i \\
\tau & \text{otherwise}
\end{cases}
\]

\(\Delta(v, f) = \{v'\}\) s.t. \(\text{range}(f) \setminus \{\tau\} = \{v'\}\)
We can then translate a SL formula into a SLeRCN formula:

\[
\begin{align*}
[[p]] &= p \\
[[\neg \phi]] &= \neg [[\phi]] \\
[[\phi_1 \land \phi_2]] &= [[\phi_1]] \cap [[\phi_2]] \\
[[\neg \psi]] &= \neg [[\psi]] \\
[[\psi_1 \lor \psi_2]] &= [[\psi_1]] \lor [[\psi_2]]
\end{align*}
\]

\[
[[\psi(x,y)]] = [[\psi]]([1 \mapsto x, 2 \mapsto y])
\]

3.5 Rely-Guarantee Temporal Logic

Rely-Guarantee 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:

\[
\Phi ::= p \mid \Phi \land \Phi \mid \neg \Phi \mid A \Psi \mid \Phi \supseteq \Phi \\
A ::= \Phi \\
\Psi ::= \Phi \mid \Psi \land \Phi \mid \neg \Psi \mid \circ \Psi \mid \Psi \land \Psi
\]

As before, \( p \) ranges over a set \( \Pi \) of atomic propositions, \( \Phi \) generates state formulae and \( \Psi \) generates path formulae. The state formula \( A \Psi \) means along all forward paths starting in the current state, \( \psi \) holds (inevitably \( \psi \)). The state formula \( \phi_1 \supseteq \phi_2 \) asserts that \( \phi_2 \) holds, provided \( A \) may be relied upon to assure that \( \phi_1 \) holds (\( \phi_2 \) \( A \)-relies on \( \phi_1 \)).

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 \( \sigma \) (instead of a mapping from agent variables to agent strategies), and a history \( p \). The incremental definitions of the semantics of the state formula constructs \( p \), \( \Phi \land \Phi \), and \( \neg \Phi \), and all the path formula constructs are essentially the same as for SLeRCN. The state formula construct \( A \Psi \) is essentially the same is the comparable one from CTL*. Formally, we define its semantics by the following:

\[
C, \sigma, p \models_\Phi A \psi \iff \forall \gamma \in \text{OUT}(C, p, \sigma).C, \sigma, \rho, \gamma \models_\Phi \psi
\]

The semantics of the new construct \( \Phi \supseteq \Phi \) uses three auxiliary functions on strategies. The first of these is restriction: \( \sigma|_A \), which is \( \sigma_a \) for \( a \in A \), and the enabled function on the last state of the history \( (e(|p|_{\Pi} - 1, a)) \) otherwise. The second is system strategy refinement: \( \sigma \sqsubseteq \tau \) if \( \sigma_0 \subseteq \tau_0 \) 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 \( \rho \), assuming all such intersections are nonempty, and \( \sigma \sqcap \tau \) is undefined otherwise. Using these functions, the semantics of the state formula construct \( \Phi \supseteq \Phi \) is given by the following:

\[
C, \sigma, p \models_\Phi \phi_1 \supseteq \phi_2 \iff \forall \tau. ((\tau|_A \subseteq \sigma|_A) \land (\forall \nu. C, (\tau|_A \cap \nu|_A), \rho \models_\Phi \phi_1)) \\
\rightarrow C, (\sigma \sqcap (\tau|_A)), \rho \models_\Phi \phi_2
\]

To help the embedding RGTL into SLeRCN, note that we always have \( (\tau|_A \cap \nu|_A) = (\tau|_A \cup \nu|_A) \) and \( (\tau|_A \subseteq \sigma|_A) \rightarrow ((\sigma \sqcap \tau|_A) = (\tau|_A \cup \sigma|_A)) \), where \( \tau|_A \) 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 \( \zeta \) giving variable 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 \([\_]\) : \( \text{RGTL} \times (\mathcal{A} \rightarrow V) \rightarrow \text{SLeRCN} \) 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 \( \mathcal{A} \)). SLeRCN uses explicit variables to express properties of strategies. To utilize this capability, we introduce an agent-indexed collection of infinite sets of variables \( V = \{ V_a \}_{a \in \mathcal{A}} \). We also introduce a mapping of agents to fresh variables, \( \text{FRESH} : \mathcal{P}_f(V) \rightarrow (\mathcal{A} \rightarrow V) \), 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).

\[
\begin{align*}
[A \psi]_{\zeta} &= \lbrack \psi \rbrack_{\zeta}(\zeta) \\
[\phi_1 \Rightarrow A \psi_2]_{\zeta} &= \text{Let } \zeta_1 = \text{FRESH}(\text{RNG}(\zeta)) \text{ and } \zeta_2 = \text{FRESH}(\text{RNG}(\zeta) \cup \text{RNG}(\zeta_1)) \text{ in } \\
&\forall (\text{RNG}(\zeta_1)). ((\bigwedge_{a \in A} \zeta_1(a) \sqsubseteq \zeta(a)) \land (\forall (\text{RNG}(\zeta_2)). \lbrack \phi_1 \rbrack_{\zeta_1 \cup \zeta_2})) \\
&\Rightarrow [\lbrack \phi_2 \rbrack]_{\zeta_1 \cup \zeta_2}
\end{align*}
\]

With the definition, we can show:

**Lemma 6.** \( C \) be a CGS with agent set \( \mathcal{A} \) and atomic propositions \( \Pi \). Let \( \phi \) and \( \psi \) be state and path RGTL formulae over the agent set \( \mathcal{A} \) and atomic propositions \( \Pi \). Let \( \{x_a | a \in \mathcal{A}\} \) be a set of variables indexed by \( \mathcal{A} \). Let \( \zeta(a) = x_a \). Let \( \sigma \) be a system strategy and \( \Gamma \) be the environment such that \( \Gamma(x_a) = \sigma_a \). Let \( \rho \) be a history of \( C \), and \( \gamma \in \text{OUT}(C, \rho, \sigma) \). Then

\[
C, \sigma, \rho \models_{\phi} \phi \text{ iff } C, \Gamma, \rho \models_{\text{SLeRCN}} [\phi]_{\zeta} \\
C, \sigma, \rho \gamma \models_{\psi} \psi \text{ iff } C, \Gamma, \rho \gamma \models_{\text{SLeRCN}} [\psi]_{\zeta}
\]

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 \( \phi \), does \( C, \emptyset, q_0 \models_{\phi} \phi \) 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 \( \sigma \) is positional if \( \forall q \in Q. \forall \rho \in Q^{+}. \sigma(\rho \cdot q) = \sigma(\rho \cdot q) \). This definition has two immediate and
Figure 1: Model Checking Positionally

important consequences: the set of positional strategies for an agent is finite (and is a subset of \(2^Q\)); satisfaction is sensitive only to the most recent state visited (i.e., \(\Gamma, \rho \cdot q \models \phi \iff C, \Gamma, q \models \Phi\)). 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 \textsc{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 Temporal Logic (LTL) \cite{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 \textsc{Convert}.

The last complication is that a path formula \(\psi\) 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 \textsc{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 \textsc{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 \(\psi\) into an LTL formula. As an example, suppose \(\psi\) contains a state formula \(\phi\). Then we would replace \(\phi\) with \(p_{C, \Gamma, \phi}\) (i.e., \(\psi[p_{C, \Gamma, \phi}/\phi]\)), where \(p_{C, \Gamma, \phi} \in \pi(q)\) iff \textsc{Positional} \((C, \Gamma, q, \phi)\). This process is represented by the function \textsc{Simplify} in Figure 4.1.

After solving our complications, the above algorithm has a substantial amount of recomputation of calls to \textsc{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 $\Upsilon$-labeled $D$-trees which are given by a tuple $(D, \Upsilon, \upsilon)$ where $D$ is a (usually finite) set of directions, $\Upsilon$ 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 $\phi$ 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 $(\mathcal{S}, D, \Upsilon, \delta, q_0, \chi)$ where

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

The size of an APT $A$, denoted $|A|$, is $|\mathcal{S}|$. 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 $\psi$ with $O(|\psi|)$ states. [16]
2. Given an APT $A$ over $\Upsilon$-labeled $D$-trees we can construct an APT $\bar{A}$ over $\Upsilon$-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 $\Upsilon$-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 $\Upsilon_1$-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 $p \in 2^\Gamma$ we can construct an APT $\text{Pred}(p)$ over $\mathcal{L}$-labeled $D$-trees where $(D, \mathcal{L}, \nu) \in L(\text{Pred}(p)) \iff \forall n. \nu(n) \in p$. 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 $2^\Omega$-labeled $Q$-tree by (for a given strategy $\sigma$) having each node $\rho$ labeled by $\sigma(\rho)$. This approach can be generalized give what is essentially a pointwise description of an environment $\Gamma$ as a $(2^\Omega)^V$-labeled $Q$-tree with node $\rho$ 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 $\epsilon$ 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 $\{(q, \Gamma) \in Q \times (2^\Omega)^V | \forall x_\alpha \in \text{dom}(\Gamma). \Gamma(x_\alpha) \neq \emptyset \land \Gamma(x_\alpha) \subseteq e(q, \alpha)\}$, 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 $\alpha$-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 $\phi$ over environments $\Gamma$ with $\text{dom}(\Gamma) = X$.

- 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 $\zeta$ and the local environment label. Secondly, whenever the algorithm would need to build an automata to check some state formula $\phi$ (which it, of course, doesn’t know how to do), use instead the APT $[(\phi)]_X$.

- For $[(\nu)]_X$, 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 $[(\phi_1 \land \phi_2)]_X$ or $[\neg \phi]|_X$ return either $[(\phi_1)] \cap [(\phi_2)]$ or $[(\phi)]_X$ as appropriate.

- For $[(x_\alpha \subseteq y_\alpha)]_X$ then return the automata that does a pointwise verification of this property. That is, at every node (each labeled by some $(q, \Gamma)$) check that $\Gamma(x_\alpha) \subseteq \Gamma(y_\alpha)$ and then recursively check this on all the node’s children.

- For $[(\exists x_\alpha. \phi)]_X$, find $[(\phi)]_{X \cup \{x_\alpha\}}$ and then use closure under the function $(q, \Gamma) \mapsto (q, \Gamma|_X)$. 


For a closed state formula $\phi$, $[[\phi]]_{\emptyset}$ 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 $\rho$, $(p_{\rho|-1}, \emptyset)$). Let $\text{DIR}^Q_{q_0}$ 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 $\text{DIR}^Q_{q_0} \cap [[\phi]]_{\emptyset} = \emptyset$.

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

### 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 $\phi$ has a number states that is in the worst case a tower of exponentials of height $|\phi|$ (or more precisely, $QR(\phi)$, where $QR$ is the quantifier rank of $\phi$). Applying APT language emptiness gives us that model checking for closed state formulae $\phi$ is in $(1 + qr(\phi))$-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 quantifiers. 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 $Q \times (2^Q)_{\text{nd}} \times Q^\text{nd}$ where $V_{\text{nd}}$ 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 formulae are closed only their satisfaction on empty environments matters (i.e., $C, \Gamma, \rho \models \phi \iff C, \emptyset, \rho \models \phi$); only the ongoing behavior of strategies matter (i.e., $(\forall \rho' \in Q'. \sigma(\rho \cdot \rho') = \sigma'(\rho \cdot \rho')) \implies (C, \Gamma|x_\rho \leftarrow \sigma, \rho \models \phi \iff C, \Gamma|x_\rho \leftarrow \sigma', \rho \models \phi)$); and we can “bake-in” histories to certain strategies (i.e., $(\forall \sigma. \exists \sigma'. \sigma(\rho \cdot \rho') = \sigma(\rho_{\rho|-1} \cdot \rho')$) 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 $p_\rho$ for a closed state formula $\phi$ contained inside a larger formula. In profitable cases, this allows us to exchange one very expensive problem for $|Q| (\text{qr}(\phi) + 1)$-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


