# Formal Analysis of Stochastic Cognitive Models: A Translation Framework from Soar to PRISM

Parth Ganeriwala[1], Candice Chambers[1], James I. Lathrop[2], Siddhartha Bhattacharyya[1], Junaid Babar[3], Stephen B. Gilbert[2], Isaac Amundson[3], Michael C. Dorneich[2], and Mohammed Abdul Hafeez Khan[1]

[1] Florida Institute of Technology, Melbourne FL, USA sbhattacharyya@fit.edu
[2] Iowa State University, Ames, IA, USA gilbert@iastate.edu
[3] Collins Aerospace, Cedar Rapids, IA, USA isaac.amundson@collins.com

**Abstract.** Cognitive architectures such as Soar are widely used to model human decision making in human–machine systems, but they are typically evaluated via simulation and therefore provide limited guarantees that do not cover all possible system behaviors. In this paper, we present a translation framework that compiles cognitive models in Soar into discrete-time Markov chains for analysis using the PRISM probabilistic model checker. The translation maps bounded abstractions of Soar's working-memory elements to finite-domain PRISM variables, production rules to guards, and decision-cycle phases to an explicit control-state automaton. We demonstrate the approach on an augmented-reality surveillance case study, in which the translated Soar task model is synchronized with probabilistic modules capturing the evolution of cognitive attacks such as cybersickness, and its task-performance effects. The resulting PRISM models support quantitative verification of properties that depend on cognitive task progression and stochastic human-state dynamics.

**Keywords:** Cognitive Architecture · Model Translation · Probabilistic Reasoning.



## 1 Introduction

Cognitive models are increasingly used to analyze and predict human behavior in systems where operator actions are integral to overall system dynamics [33, 3]. Examples include supervisory control, human–autonomy teaming, and mixed-reality interfaces [19, 32]. Cognitive architectures such as Soar provide a rule-based computational model of human decision making and are widely used to analyze task sequences, attention, and internal cognitive states [20]. However, Soar models are typically evaluated through simulation, which limits their ability to provide guarantees about all possible executions or low-probability events that may have safety-critical implications [13, 11, 25, 6].

Formal verification, and model checking in particular, provides algorithmic techniques for establishing whether all behaviors of a given model satisfy temporal-logic specifications [3]. Probabilistic model checking extends these techniques to stochastic systems, enabling quantitative reasoning about event probabilities, expected times, and performance degradation [17–19]. Applying these techniques to cognitive models requires translating architecture-level constructs into an explicit finite-state probabilistic model. In this paper, we present our workflow and tool chain for formally analyzing stochastic cognitive models. We start from a Soar cognitive architecture [20] and compile a task-oriented subset of its decision-cycle behavior into a PRISM discrete-time Markov chain (DTMC) [19]. DTMCs [28] have long been used to model complex probabilistic systems [31, 15]. Constructing such a translation is challenging because Soar's decision-cycle semantics can be realized by multiple verification-distinct encodings while still appearing reasonable. For instance, if two operators are proposed with indifferent preferences, a DTMC translation must commit to an explicit probabilistic tie-break, and different choices can yield different quantitative guarantees. Likewise, production firing and working-memory updates must be captured in a way that preserves the intended rule-execution structure while yielding a well-formed probabilistic transition system. We have developed a translation framework that maps Soar models to DTMCs for analysis in PRISM. The translation treats a Soar model as a structured state-transition system: working memory elements become PRISM variables, productions become guarded commands, and the Soar decision cycle is represented as a control automaton [8, 5, 27]. The resulting PRISM model supports the specification and verification of probabilistic temporal properties over the behavior described at the cognitive level.

Our translation framework is a key component of the Modeling and Analysis Toolkit for Realizable Intrinsic Cognitive Security (MATRICS) [1], currently being developed on the DARPA Intrinsic Cognitive Security (ICS) program. ICS is exploring the feasibility of applying formal analysis to cognitive models of human behavior to prove that users of mixed-reality systems are protected from cognitive attacks. Such attacks are successful when an adversary manipulates the environment in a manner that affects the cognitive abilities of the user, thereby causing harm to the user or negative impact to the mission. Examples of cognitive attacks include inducing nausea, eye-strain, and dizziness (cybersickness) by increasing latency in display output, shining a laser or bright light to wash out information on a digital display, and flooding an observation zone with decoys to increase visual search demand, resulting in missed identifications. The inherent tight coupling between a mixed-reality system and its user contributes to an expanded attack surface with unique characteristics. The ability to model and formally prove cognitive security properties during the design of these systems will be critical as mixed-reality becomes more commonplace in society.

The keycontributions of this work are:

1. A semantics-driven translation from Soar's decision cycle and production-rule execution to a finite-state DTMC model for analysis in PRISM.

2. An open-source implementation that (i) applies a finite abstraction to selected Soar state variables, (ii) generates the corresponding PRISM DTMC modules, and (iii) supports composition with external stochastic modules.
3. A case study that uses the translated and composed DTMC models to check properties that relate task progression and abstracted cognitive/task-control state to stochastic physiological effects.

While prior work has explored translating cognitive architectures into formal verification models (e.g., nuXmv and Uppaal), most approaches focus on deterministic or timed semantics. In contrast, this work provides a semantics-guided translation from Soar to discrete-time Markov chains (DTMCs) that explicitly captures stochastic behavior and supports probabilistic model checking in PRISM. A key technical contribution is a determinization strategy that eliminates nondeterminism by introducing configuration-driven stochastic input variables, ensuring that each reachable state induces a unique probability distribution over successors. Additionally, our framework enables modular composition of cognitive task models with external stochastic human-state processes, supporting quantitative verification of human-in-the-loop systems under uncertainty.

## 2 Background and Related Work

### 2.1 Cognitive Architectures

Cognitive architectures are computational theories intended to account for a broad range of human cognitive capabilities (e.g., perception, memory, action selection, learning, and reasoning) within a single integrated framework. Well-established architectures that have been utilized for theory development and application include Soar, ACT-R [2], and CLARION [16]. More recently, the research community has proposed a standard model of the mind (e.g., Common Model of Cognition) to capture architectural commonalities across major approaches [22]. Cognitive architectures make different computational assumptions about how cognition is represented and controlled, and each tends to emphasize different mechanisms (e.g., modular perception/action pipelines, distinct memory systems, or learning from implicit vs. explicit knowledge) [24, 16]. For example, ACT-R is commonly described as a modular architecture in which a central production system operates over a small set of buffers that interface perceptual/motor and memory modules [2]. CLARION is organized around interacting implicit and explicit representational layers, with learning mechanisms that relate the two and support skill acquisition and performance [30]. Soar emphasizes operator-based problem solving: productions propose and apply operators, and a recurring decision cycle selects among candidates to advance an explicit symbolic working-memory state [20].

Cognitive architectures are accompanied by software implementations to construct and execute task-specific models. These implementations provide an operational account of architectural mechanisms (e.g., firing of productions, memory

retrieval, and optional learning updates) and support simulation of execution traces and performance measures [26]. Consequently, when assurance questions are posed (e.g., reachability of unsafe states), translation machinery is needed to map architecture-specific constructs into the modeling formalisms accepted by verification tools [5]. In this paper, we use Soar as the cognitive architecture for its operator-centric decision cycle and symbolic working-memory representation. This provides a control structure and state vocabulary that can be systematically translated into a transition-based verification-oriented probabilistic model.

Soar is a cognitive architecture theory and a modeling and execution environment for constructing agents [20]. A Soar agent maintains a symbolic working memory representing the current state and uses production rules to propose operators, express preferences among candidate operators, and apply the selected operator. The agent's decision cycle repeatedly (a) elaborates working-memory inferences, (b) proposes and evaluates operators, (c) selects an operator via a decision cycle, and (d) applies the operator to update working memory and produce external actions [20]. Soar also includes mechanisms for learning (e.g., chunking [21] and reinforcement learning [26]); in this paper we focus on the core decision-cycle and state-update behavior needed to represent task progression.

Several Soar features make formal verification challenging: First, Soar states include relational symbolic structures and dynamic subgoaling, which are not finite-state *a priori*; a verification model therefore requires a finite abstraction over a subset of task-relevant working-memory features. Second, each decision cycle can involve multiple internal firings and intermediate working-memory updates, so the translator must choose an explicit step semantics (e.g., micro-steps within a cycle versus a macro-step to a post-elaboration configuration) while preserving task progression. Third, architectural choices such as tie-breaking among indifferent operator preferences and external inputs interactions must be represented in an explicit transition model; in a DTMC, this typically requires committing to concrete probabilistic resolutions. Finally, learning changes the rule base over time; to keep the model finite and analyzable, verification-oriented translations often begin with a fixed-rule subset and treat learning as out of scope or as modeled separately [20].

### 2.2 Formal Verification Environments

Formal methods comprise a range of techniques for establishing system properties, including deductive methods and model checking. In this work, we use probabilistic model checking, which analyzes an explicit state-transition model against temporal-logic or quantitative specifications [3, 14]. This motivates a backend that provides discrete-time probabilistic semantics, quantitative temporal properties, expected-value queries, and a modular modeling language suitable for composition and code generation.

Several established tools support complementary parts of this space. Uppaal targets timed-automata models and is well suited to dense-time constraints and scheduling properties [4]. nuXmv targets symbolic transition systems and supports CTL/LTL model checking and counterexample generation for functional

properties [9]. Although nuXmv provides scalable SAT/SMT-based logical verification and counterexample generation for functional properties, it does not natively support probabilistic temporal logics or expected-value analysis over stochastic models. PRISM supports probabilistic model checking for Markov models specified using a guarded-command language with interacting modules and probabilistic updates [18]. It can handle multiple probabilistic model classes such as DTMCs, CTMCs, and MDPs. We selected PRISM as our model checker because it directly supports DTMCs, probabilistic temporal logics, and reward structures for expected-value queries [17, 19], enabling end-to-end quantitative assurance arguments over composed cognitive-physiological models.

### 2.3 Bridging the Gap

Formal verification of cognitive architectures requires translating architecture specific constructs like productions, buffers, and decision cycles into verification models with explicit state and transition semantics. In previous work, we developed a workflow for translating Soar models into nuXmv to support verification of purely deterministic requirements and invariants [6]. We have also explored translating aspects of cognitive agents into timed automata for analysis in Uppaal [5], and related work has developed formal operational models and translation schemes from ACT-R to timed automata to support automated defect analysis with model checking [23]. Other work formalizes architecture-level descriptions or reference structures (including those aligned with the Common Model of Cognition) and provides architecture-neutral representations suitable for symbolic model checking [22, 29, 12]. Task-analytic formalisms such as the Enhanced Operator Function Model define human behavior via hierarchical task models with formal semantics and translation to model checkers, including probabilistic model checking in PRISM [7].

Collectively, these results motivate treating cognitive models as compilable artifacts for verification; however, the dominant targets have been timed automata or non-probabilistic symbolic transition systems [5, 23]. This paper extends this general direction by compiling Soar task models into PRISM DTMCs, so that the translated task dynamics can be composed with an explicit stochastic human-state module and analyzed using probabilistic temporal properties.

## 3 Methodology

In this section, we present our methodology for the formal verification of stochastic cognitive models, such as those developed on the ICS program. Our goal is the verification of human-in-the-loop mission behavior under probabilistic human-state effects. We model task-control logic and cognitive attack effects in Soar, translate it into a PRISM DTMC with discrete-time stochastic modules capturing human-state/performance effects (e.g., cybersickness, response time, etc.), and use PRISM to verify cognitive security properties.

### 3.1 Soar Modeling Pattern

We organize Soar task models using a pattern that separates (i) nominal mission/task progression from (ii) periodic or event-triggered monitoring logic that represents a cognitive-attack effect or human-state assessment. At a high level, the Soar agent maintains a bounded set of task-relevant working-memory features (e.g., current task phase, operator context, timing variables, and any abstracted performance/health indicators). Nominal task execution is represented by a *mission module* that advances these features through a sequence of task phases. Monitoring/attack logic is represented by a separate *cognitive attack* (or *monitor*) *module* that evaluates a condition (often time-indexed and/or context-dependent) and updates a small set of abstract state variables that influence subsequent task behavior. Figure 1 illustrates the pattern.

The mission module proposes and applies task-phase operators, while high-priority switch operators transfer control to the monitor module when a sampling condition holds (e.g., periodic checks or context-triggered events). The monitor module evaluates the relevant condition, applies an outcome operator that commits the corresponding state update, and then returns control to the mission module. This structure yields an explicit and repeatable control vocabulary (operator context and monitor mode) that is preserved by the DTMC translation. To support a DTMC translation, we follow two modeling conventions. First, we

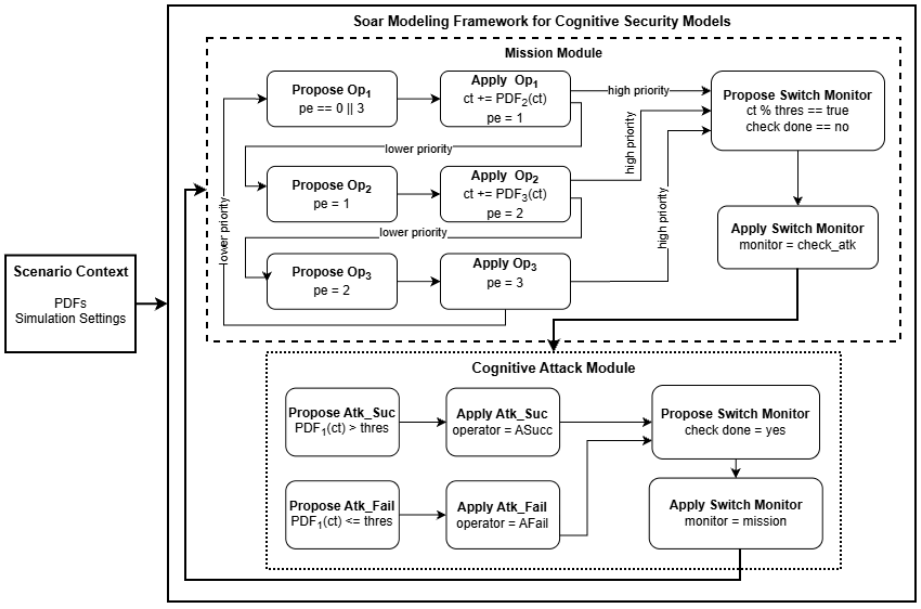


Fig. 1: High-level structure of a Soar modeling pattern for cognitive-security scenarios.

abstract the verification-relevant working-memory features with bounded domains (chosen per scenario), so that the translated state space is finite. Second, any behavior that is underspecified at the architecture level (e.g., ties among indifferent preferences or stochastic effects not represented in the Soar rules) is resolved explicitly via configuration-supplied probability distributions, so that each translated state induces a unique probability distribution over next states.

### 3.2 Translation Framework

To enable formal analysis of cognitive agents in human-in-the-loop systems, we translate Soar task-control logic into a PRISM DTMC and then synchronize it with an external probabilistic human-state module. With DTMCs, each global step corresponds to a fixed-duration time increment and the outgoing transitions from each state form a probability distribution. This provides a semantic basis for quantitative properties (probability of satisfaction within a horizon) and expected-value queries over the composed cognitive-human-state model.

The translated model state consists of a finite vector of PRISM variables representing abstracted working-memory elements together with derived control information. Each relevant working-memory element (e.g., Soar variable, operator, etc.) is mapped to one or more PRISM variable with finite domains chosen to preserve distinctions required for verification. Initial values are derived from the Soar model's initial working memory. To preserve Soar's execution semantics, the translation introduces a control-state abstraction that enforces the ordering of Soar's decision cycle. Guards on PRISM transitions reference this control state so that production rules are enabled only in their appropriate context, preventing interleavings that violate the Soar architecture.

Each Soar production rule is translated into one or more guarded PRISM commands. Rule conditions become guards over PRISM state variables, and rule actions become updates to those variables. Elaboration rules update states without selecting an operator, while proposal and application rules are conditioned on the control state and operator context. When multiple operator outcomes are possible under uncertainty, the Soar semantics may leave the choice underspecified, resulting in nondeterminism from a verification perspective. To obtain a DTMC, we eliminate such nondeterminism by introducing stochastic input variables whose distributions are specified in the configuration (e.g., derived from empirical data or design assumptions). These variables are sampled at each decision point and used to define mutually exclusive threshold guards over competing operator proposals. As a result, exactly one operator is selected in each decision context, and every reachable state induces a unique probability distribution over successor states. This transformation converts underspecified architectural choices into explicit probabilistic behavior, enabling quantitative reasoning in PRISM.

The translator is parameterized by a configuration file that supplies information not represented in the Soar model itself and that governs how the translated controller is coupled to its external context in PRISM. In particular, the configuration defines the abstraction boundary by selecting which working-memory

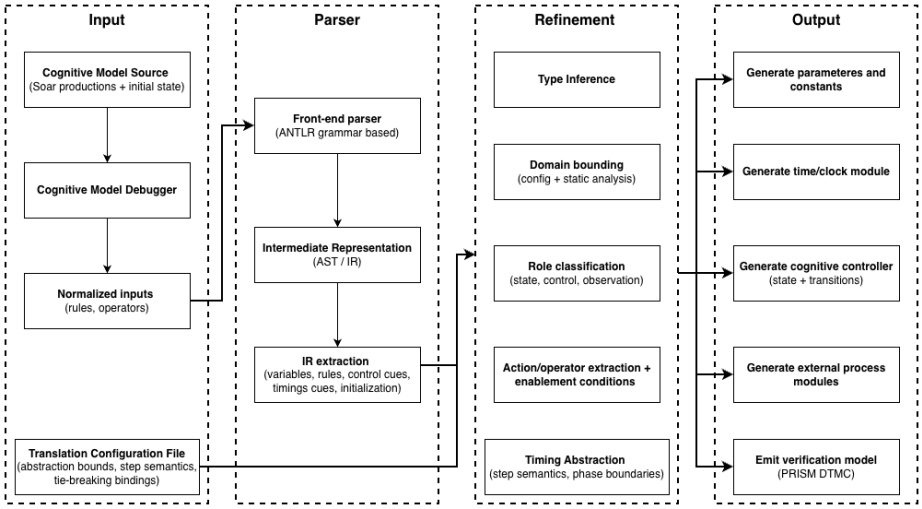


Fig. 2: Generalized Soar-to-PRISM translation workflow.

features are exposed as finite-domain PRISM variables and by providing the corresponding domain bounds or enumerations. It also specifies time-stepping assumptions (e.g., the clock domain and update function) required to interpret the model as a discrete-time system. Beyond state abstraction and timing, the configuration encodes extrinsic dynamics and interface assumptions that are external to the cognitive architecture but interact with the agent during execution. Examples include probability tables or parameters for response-time distributions, error models, and stochastic state processes representing physiological, sensor, or environmental evolution, potentially conditioned on time windows or control context. The configuration may also provide constants, enabling reproducible model generation and systematic variation of scenario parameters without modifying the underlying Soar knowledge base.

Our translation workflow is shown in Figure 2. The front-end parses the Soar model and builds a rule-tuple for each production; the refinement stage performs bounded abstraction, timing and control/context synthesis needed for a DTMC interpretation (including a conflict-resolution compilation that ensures only one synchronized rule-class is enabled per step); and the backend emits PRISM modules whose synchronous composition yields the final DTMC.

Algorithm 1 provides the formal construction used to translate a Soar model into a PRISM DTMC. We treat the input Soar model as a finite collection of production rules, where each production is represented as a tuple, $rname(V, (pre\{\varphi_1, \ldots, \varphi_m\}\ ,\ post\{\psi_1, \ldots, \psi_r\}))$. Here $V$ is a finite set of state variables whose valuation val$(V)$ represents the system state, $pre = \{\varphi_1, \varphi_2, \ldots, \varphi_m\}$ is a finite set of well-formed formulae (WFF) encoding the left-hand-side preconditions, and $post = \{\psi_1, \psi_2, \ldots, \psi_r\}$ is a finite set of WFF encoding the right-hand-side actions [6]. Because PRISM requires a finite-state model, we assume a finite

abstraction $\alpha$ that maps concrete valuations val($V$) into a bounded abstract valuation over a finite set of abstract variables $\widehat{V}$. Each $\widehat{v} \in \widehat{V}$ is represented in PRISM by a finite-domain state variable $x_{\widehat{v}}$, and we write $x = \langle x_{\widehat{v}} \rangle_{\widehat{v} \in \widehat{V}}$ for the resulting abstract state vector. Under this encoding PRISM guards are obtained by abstracting the rule preconditions (the $pre\{\cdot\}$ WFF), and PRISM updates are obtained by abstracting the rule actions (the $post\{\cdot\}$ WFF) for the enabled production(s). To maintain DTMC semantics, Productions are translated into rule-classes indexed by the synthesized control context $c$ apply). Within each context, competing proposal rules are compiled with mutually exclusive threshold guards over sampled stochastic inputs, ensuring that at most one production is enabled and thus at most one synchronized class-transition can fire in any reachable state. This prevents unintended nondeterminism by ensuring each reachable state induces a single probability distribution over successor states. If determinization fails (i.e., multiple productions remain enabled), the translator reports a configuration error rather than emitting a nondeterministic model. All probabilistic commands generated by the translator are required to define valid discrete distributions; configuration-supplied probabilities are checked for normalization (and normalized when appropriate), and invalid configurations are rejected during translation.

In addition to the abstract state $x$, the translated DTMC contains (i) a discrete time index governed by a step semantics $(D_T, \delta_T)$ and (ii) an explicit control variable $c \in C$ that records the current task context needed to preserve Soar's intended rule-enabling conditions. Algorithm 1 defines how these components are combined into the DTMC $\mathcal{P} = (S, s'_0, P)$ and how external stochastic variables $y$ (with configuration-specified distributions) are sampled and composed into the global state. The construction is designed to produce a well-formed DTMC: for every reachable global state $s = (x, c, y, t) \in S$, the enabled synchronized transition(s) induce a single probability distribution over successor states, so that $\sum_{s' \in S} P(s, s') = 1$.

We implemented our Soar2PRISM translator as an open source[4] Java tool that parses Soar production rules using an ANTLR4 grammar and constructs an intermediate representation containing rule names, guards, actions, and extracted working-memory features. The tool supports a configuration-driven workflow: a JSON configuration file supplies translation parameters, finite-domain bounds for selected variables, and external probability tables/distributions used to model stochastic sampling and tie-breaking. The backend emits a PRISM model (`.pm`) consisting of a clock/time module, a Soar-derived controller module that encodes decision-cycle context, and external stochastic modules composed under a shared synchronization label.

### 3.3 Probabilistic Reasoning

Our framework enables probabilistic verification by compiling an executable cognitive model (Soar) into a discrete-time probabilistic transition system (PRISM

[4] https://github.com/ParthGaneriwala/SoarToPrismTranslator

**Algorithm 1** General Time-Stepped Translation of a Soar Model to a PRISM DTMC

**Require:** Soar production rule set $R$ (each $r \in R$ is $rname(V, (pre\{\cdot\}, post\{\cdot\}))$), initial valuation $\mathrm{val}_0(V)$, finite abstraction $\alpha$, step semantics $(D_T, \delta_T)$, time horizon $T$, and a finite set of stochastic input variables $y = \langle y_j \rangle$ with configuration-specified discrete distributions $\{(h_j, \{(v_{jk}, p_{jk})\}_k)\}_j$

**Ensure:** PRISM DTMC $\mathcal{P} = (S, s'_0, P)$

Initialize DTMC and introduce shared synchronization label sync

Introduce a discrete-time clock variable $t \in D_T$ with initial value $t_0$

Define a total clock update function $\delta_T : D_T \to D_T$

Add synchronized clock transition:

$$[\mathsf{sync}] \ \top \ \to \ 1 : (t' = \delta_T(t))$$

Derive a finite control-state set $C$ (task context) from $\{pre_r \mid r \in R\}$ and the time-step semantics

Introduce control variable $c \in C$ with initial value $c_0$

Let $\widehat{V} = \alpha(V)$ be the finite abstract variable set induced by $\alpha$

**for each** $\widehat{v} \in \widehat{V}$ **do**

  Introduce PRISM state variable $x_{\widehat{v}}$ with finite domain $D_{\widehat{v}}$

  Initialize $x_{\widehat{v}} \leftarrow \alpha(\mathrm{val}_0(V))[\widehat{v}]$

**end for**

**for each** stochastic component $E_j$ **do**

  Introduce variables $y_j$ with finite domains

  Add transitions of the form:

$$[\mathsf{sync}] \ h_j(x, c, t) \to \sum_k p_{jk} : y'_j = v_{jk}$$

**end for**

Partition $R$ into rule-classes $\{R_i\}$ by the control-context predicates extracted from $pre_r$ (indexed by $c \in C$)

Synthesize threshold guards over sampled inputs to enforce: for all reachable states $s$, $|\{r \in R_i \mid \alpha(pre_r) \text{ holds in } s\}| \leq 1$

**for each** rule class $R_i \subseteq R$ **do**

  Derive guard predicate from preconditions:

$$g_i(x, c, y) = \big( \vee_{r \in R_i} \ \alpha(pre_r) \big)$$

  Let $r^*$ be the unique enabled rule in $R_i$ (by threshold-guard determinization); otherwise report a configuration error

  Set $u_i(x, c, y) = \alpha(post_{r^*})$

  Add synchronized transition:

$$[\mathsf{sync}] \ g_i(x, c, y) \to (x', c') = u_i(x, c, y)$$

**end for**

Ensure that for every state $s \in S$:

$$\sum_{s' \in S} P(s, s') = 1$$

**return** $\mathcal{P}$

Table 1: Representative properties supported by the framework over the translated PRISM DTMC. $T$ represents user-configured finite horizon.

| Class | Property | What it answers |
|---|---|---|
| Liveness | $P_{\geq 1}[\, F$ `"terminal"` $]$ | Does the model eventually reach a terminal condition (i.e., no deadlock before termination)? |
| Liveness | $P_{\geq 1}[\, F^{\leq T}$ `"milestone"` $]$ | Is a key task always reachable within the horizon? |
| Safety | $P_{\geq 1}[\, G\ (t \leq T)\,]$ | Is execution constrained to bounded horizon semantics? |
| Safety | $P_{\geq 1}[\, G\ (v \leq V_{\max})\,]$ | Are abstract state variables guaranteed to stay within limits? |
| Safety | $P_{\geq 1}[\, G($`"terminal"` $\Rightarrow$ $P_{\geq 1}[X$ `"terminal"`$])\,]$ | Is the terminal region absorbing (once terminated, cannot transition to non-terminal state)? |
| Reachability | $P_{=?}[\, F^{\leq T}$ `"hazard"` $]$ | What is the probability of entering a hazard during the mission? |
| Reachability | $P_{=?}[\, F^{\leq T}$ `"adverse_event"` $]$ | What is the probability of at least one adverse event occurring within the horizon? |

DTMC). This enables (i) liveness queries to establish progress to required milestones within a horizon, (ii) safety queries to establish probability-1 invariants such as boundedness and absorption at the horizon, and (iii) quantitative reachability queries to compute the probability of entering safety-critical regions within a mission window; all over PRISM labels that denote semantically meaningful regions of the translated state space (Table 1). Labels such as `"milestone"`, `"hazard"`, and `"adverse_event"` are defined as atomic propositions over the translated state variables, allowing the same query templates to be instantiated for different tasks and domains. After translation, the model state is finite (by construction via bounded abstraction) and each synchronized step induces a unique probability distribution over successors, so standard probabilistic model checking applies.

When performance measures are needed in addition to risk measures, we use PRISM reward structures to compute expected quantities such as expected time-to-completion, expected number of adverse events, or expected accumulated cost over a bounded horizon. As the translation is configuration-driven, the same property suite can be re-evaluated under different timing assumptions and probability parameters without modifying the underlying Soar model, supporting systematic sensitivity analysis.

## 4 Use Case

In this section, we apply our methodology presented in Section 3 to an augmented-reality (AR) surveillance mission in which an operator, wearing a helmet-mounted display (HMD), monitors a secure area and must identify suspicious individuals in the observation zone with the aid of a visual cue. For each observation, the operator performs a simple decision workflow: (i) scan/select the individual of interest, (ii) decide whether the individual is a threat based on the available visual cues, and (iii) report the decision (threat / non-threat). This sequence repeats across events over a bounded mission horizon, and mission performance depends on both decision correctness and timeliness.

The HMD surveillance application will be subject to network- and system-induced latency, possibly due to an adversarial attack. In the context of AR environments, latency refers to the delay between an operator's action (such as head movement or interaction with the system) and the corresponding system response, when the environment is updated (e.g., through visual or sensory feedback) to reflect the operator's input. Latency affects when information becomes available on the HMD display and when an operator action takes effect, influencing the time required to complete each event and the likelihood of missed or incorrect responses. Furthermore, latency has been shown to cause operator cybersickness, with symptoms including dizziness and nausea [10].

Having developed cognitive models[5] of the surveillance task and cybersickness onset (e.g., from pilot studies or the cognitive science literature), we apply our verification workflow to establish guarantees on the likelihood of mission success under specific operating conditions. We treat mission success as completing the required sequence of threat classifications within a bounded horizon while maintaining acceptable performance (e.g., timely responses and correct classifications). Failure corresponds to outcomes such as missed reports, misclassification under latency, exceeding a time bound, or reaching an unacceptable human-state condition (cybersickness) before mission completion. In our models, cybersickness evolves over time and modulates performance-related quantities such as response-time distributions and classification accuracy. This enables quantitative verification queries that jointly depend on task logic, timing/latency effects, and stochastic human-state evolution.

### 4.1 Soar Model

We model the operator's task-control logic in Soar following the modeling pattern of Section 3.1. The mission module encodes the nominal *scan/select*, *decide*, *decided* progression as a sequence of operator proposals and applications. Its working-memory state records the current task phase, timing information, and any abstract variables needed to capture task context relevant to verification (e.g., whether an event is in progress, whether a decision has been issued). The

[5] All use case artifacts discussed in this section are included in our Soar2PRISM translator repository.


Fig. 3: Soar2PRISM translation of the cybersickness-monitor production fragment and the corresponding synchronized PRISM DTMC commands.

time it takes the operator to reach a decision is governed by a probability density function (PDF). In this case study we use a placeholder distribution (Gaussian) to parameterize decision time as a function of elapsed time in the environment. This can be replaced by empirically-estimated distributions when available. The mission module is responsible for advancing the task from one event to the next until the horizon is reached.

The cognitive attack monitor module represents a periodic health assessment that may affect subsequent task performance. At configurable sampling times (every 2, 5, or 10 minutes for this use case), a high-priority switch transfers control from the mission module to the monitor. The monitor evaluates whether cybersickness has occurred for the current time window and commits the corresponding abstract health-state update before returning control to the mission module. This organization makes the monitor transitions explicit in the control flow, which is important for generating a time-stepped verification model.

We apply the Soar2PRISM translator to this Soar agent to generate the PRISM DTMC controller module (Figure 3). The generated PRISM module is synchronized (via `[sync]`) with the external stochastic modules that encode cybersickness and performance effects.

### 4.2 PRISM Model Composition

The verification model analyzed in PRISM is the synchronous composition of: (i) the translated DTMC controller derived from the Soar mission/monitor logic, and (ii) external stochastic modules that encode human-state and performance effects. Intuitively, the translated controller captures what the operator does next (task phase progression and when monitoring occurs), while the external modules capture uncertain effects that influence outcomes.

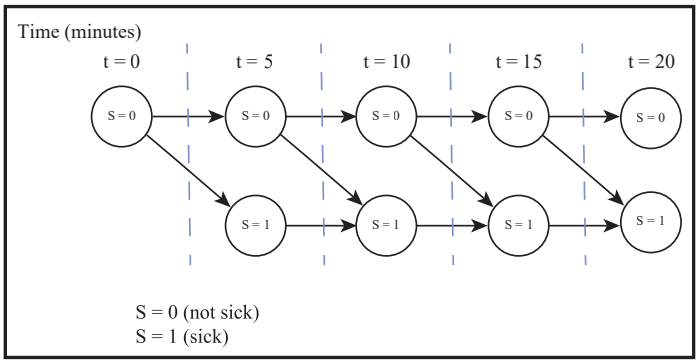


Fig. 4: DTMC for operator sickness model.

Conceptually, the composed DTMC can be viewed as two coupled components: a cybersickness process and a task-performance process whose transition probabilities are conditioned on the current sickness state. The cybersickness Markov chain models the operator's sickness level over time. Although the sickness level can vary over a range of integer values, for this use case, it is constrained to 0 (not sick) and 1 (sick). It is simple to generalize the model and the translation to more than two sickness levels. Figure 4 shows the Markov model for two sickness levels over a duration of 20 minutes. Note that time can be easily generalized to arbitrary durations and resolutions; however, large durations or fine resolutions will result in large state spaces in the PRISM model. The cybersickness model also assumes that once an operator becomes sick, they will remain sick for the model's duration (i.e., transitions only move the operator from not sick to sick). At any given time, the operator may become sick (or remain not sick), determined by the probabilities of the transitions between states. For this use case, these probabilities are derived from experimental values.

Synchronized in time with the sickness DTMC, the task DTMC uses the current state of sickness in the sickness DTMC to model the operator's decisions and response times. At any given time and sickness level, the operator may be in one of three possible states: Scanning and Selecting, Deciding, and Decided as described in Section 4.1. This corresponds to the conceptual state diagram shown in Figure 5. Note that the *Decided* state requires no time; it simply marks the time that the operator has decided on the entity. In the final translation, this state is merged with other states so that no time passes in the DTMC when the operator makes a decision.

The response time between Scanning/Selecting and Deciding, and Deciding and Decided is an important metric in this mission scenario. These response times are probabilistic and modeled as a discrete-time stochastic process. In this use case, response time is modeled as a sequence of binary random variables, with probabilities determined by a time-dependent function. For example, the probability at time $t$ is determined by an exponential function with parameter $\lambda$. A finite sequence of states in the DTMC model is used to approximate the

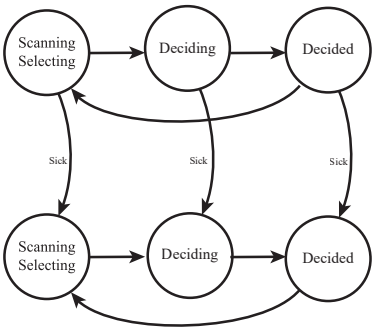


Fig. 5: Overall task state model.

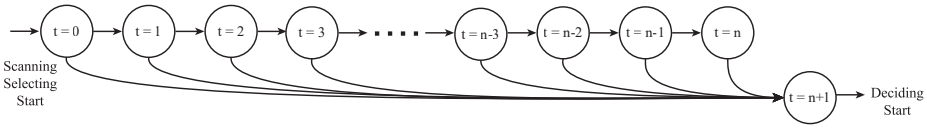


Fig. 6: Modeling response time with a DTMC.

stochastic process. Since the stochastic process is potentially infinite, the tail probabilities are accumulated in the final state. An example using the exponential function is shown in Figure 6. This figure shows how to model the operator's response time for scanning and performing a selection. The probability of making a selection at any given time $t$ is given by the transitions to the Deciding Start state.

In this use case, response times are in seconds, sampled with 0.5-second resolution, but the sickness model is only sampled every 5 minutes. The two models must be combined to a single DTMC by inserting probability 1 transitions in the sickness model so that it matches the sample rate of the task model. The transition from the Decided state back to the Scanning/Selecting state can be split into two probabilities, one to model correct entity identification and one for incorrect entity identification. PRISM rewards are used to track the number of correct and incorrect user decisions, and PRISM properties are used to guarantee probabilistic performance criteria.

### 4.3 Property Verification

We performed probabilistic property verification in PRISM on the translated DTMC by evaluating a suite of probabilistic computation tree logic (PCTL) and reward-based queries over the generated model. For the bounded abstraction used in this study, Soar2PRISM successfully constructs the translated DTMC with 3,605,682 reachable states and 4,176,825 transitions, demonstrating that the translation produces a finite-state model at the selected abstraction boundary. To support systematic validation, we introduce a small set of atomic propositions

over the translated state space using PRISM labels. These labels denote semantically meaningful regions of the DTMC state space, including the finite-horizon endpoint (`"end"`), sickness status (`"not sick"`, `"sick"`), controller phases ( `"select"`, `"decide"`, `"done"`), and the presence of any recorded decision error (`"any_error"`). The labels provide a verification-oriented interface for stating requirements and translation properties without exposing use case-specific variable names in the properties themselves. We evaluated the property suite shown in Figure 7 to quantify mission risk and performance under cybersickness dynamics. Progress and structural-invariant checks confirm that the mission controller reaches the horizon and that time and counter variables remain within bounds, while the cybersickness and error-related queries provide quantitative evidence about the likelihood of adverse outcomes (e.g., being sick at the horizon and observing at least one decision error within the mission window). For example, in our configuration, the probability of encountering at least one adverse event within the mission horizon was computed as 0.84, while the probability of the operator being in a sick state at the horizon was 0.66, illustrating how the framework enables quantitative assessment of mission risk under stochastic human-state dynamics.

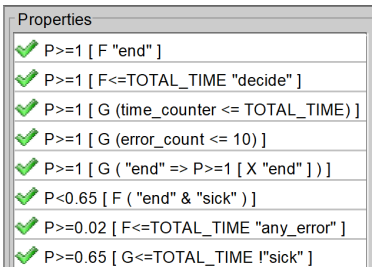


Fig. 7: Representative PRISM property suite evaluated for the use case.

## 5 Discussion

### 5.1 Translator Validation

We evaluated the Soar-to-PRISM translator on the use case by focusing on translation fidelity, i.e., whether the generated PRISM model preserves the intended Soar control structure and yields a well-formed DTMC suitable for probabilistic model checking. To reduce the risk of implementation-specific errors in the generated PRISM code, we also constructed a reference PRISM model using an independent C#-based generator. This generator takes the same probability distributions and a structured description of the intended DTMC components and emits PRISM code. We used this reference model as a validation check by (i) confirming successful PRISM model construction under the same bounds and (ii)

evaluating the same validation property suite on both models. The Soar2PRISM translator is needed because the Soar agent is the executable cognitive/task specification in our workflow; generating PRISM directly would require re-encoding task-control logic and decision-cycle structure by hand, which is error-prone and can diverge from the Soar model as it evolves. Compiling from Soar preserves traceability from production rules and decision-cycle context to DTMC states and transitions, and supports regeneration of verification models when the cognitive model changes.

To further validate the translation, we performed basic behavioral consistency checks between the Soar model and the generated PRISM DTMC. In particular, we verified that key task-phase progressions occur in the expected order and that control transitions between mission and monitoring modules are preserved. Additionally, we conducted mutation-style checks by modifying selected probability parameters (e.g., increasing sickness transition likelihood) and observing corresponding changes in quantitative verification results (e.g., increased probability of adverse events). These checks provide confidence that the translated model responds consistently to parameter changes and reflects the intended system behavior.

**Structural correctness and DTMC well-formedness.** A primary validation objective is that the generated model is a DTMC with a total step relation: in every reachable global state there is at least one enabled transition and the enabled commands induce a unique probability distribution over successor states. The translator enforces this objective via a global synchronization label (`[sync]`) shared by all modules, which yields a lockstep, time-stepped execution semantics consistent with the DTMC interpretation. To avoid deadlocks under synchronization, each generated module includes explicit default commands that preserve local variables whenever no specialized rule is enabled. For stochastic sampling steps, the translator emits normalized probabilistic commands so that outgoing branch probabilities form valid distributions (up to floating-point tolerance). Together, these construction rules ensure that the composed model is accepted by PRISM as a DTMC and that model construction completes without deadlocks or unintended nondeterminism.

**Semantic alignment with the use case abstractions.** We next validated that key semantic patterns from the Soar model and translation configuration are preserved in the PRISM output. First, the translator correctly realizes the finite-horizon, discrete-time mission semantics: the clock variable advances from 0 to `TOTAL_TIME` and then remains constant, corresponding to the label `"end"`. Second, the translator preserves the monitoring abstraction for health state by generating windowed sampling and commit-point updates: sampling occurs at the configured window boundaries and the sampled value is committed at the end of each window (with the sampling flag reset), matching the intended interpretation of periodic checks. Third, response-time and decision-error sampling are guarded by the Soar-derived controller context: response-time distributions are triggered by controller phase labels (e.g., `"select"` or `"decide"`) and conditioned on the current sickness status label, and decision correctness is sampled

when the decision phase completes. These guard structures confirm that the translation preserves the intended coupling between cognitive phase, time progression, and stochastic performance effects.

**Translator validation queries.** We evaluated the queries from Section 4.3 on the translated PRISM DTMC as translation-validation checks, focusing on mission progress, invariants and integration of the controller and stochastic modules. All liveness and safety queries verified successfully, and PRISM produced concrete values for the bounded-horizon reachability probabilities of `"sick"` and `"any_error"`. These results indicated the absence of deadlocks, adherence to bounded-step semantics, and correct composition of the stochastic modules.

The Soar models, translation tool, configuration files, and PRISM models used in this study are available in our public repository. The translation process is fully configuration-driven, enabling regeneration of the verification models and systematic exploration of parameter variations without modifying the underlying cognitive model.

### 5.2 Generalization to Diverse Cognitive Architectures

Although this paper focuses on translating Soar task models to PRISM DTMCs, the core translation pattern is architecture-agnostic: (i) identify a finite set of state-bearing cognitive features, (ii) represent procedural knowledge as guarded state-update rules, and (iii) impose an explicit control automaton that captures the architecture's step semantics so that rule enablement and state updates occur in the intended order. Under this view, a cognitive architecture instance can be treated as a structured transition system that is compiled into a verification backend after a finite abstraction is chosen.

To generalize the translator to other architectures, we require that the source model expose the following interface elements. First, the model must provide an explicit state vocabulary $V$ that covers the subset of internal cognitive features and external I/O features relevant to the assurance question. In Soar this is derived from working-memory elements; in architectures such as ACT-R this corresponds to buffers, selected declarative memory features, and task-relevant perceptual/motor variables. Second, the model must provide a set of procedural rules $R$ whose semantics can be expressed as guarded updates over $V$. This can be satisfied either by direct access to the architecture's rule language or by compiling that rule language into a rule-tuple representation. Third, the model must expose a phase structure that determines when rules are eligible to fire and how conflicts or ties are resolved. This structure is captured in the translation by an explicit finite control variable, enabling the generated DTMC to respect architecture-specific ordering constraints (e.g., perception–cognition–action phases, or buffer update ordering). Finally, to obtain a DTMC, the translation must eliminate unresolved choice: in each translated state, the enabled behavior must define a unique probability distribution over next states. Any underspecified decisions in the source semantics must therefore be compiled either into a fixed tie-breaking rule or into explicit probabilistic branching. A key enabler of cross-architecture reuse is configuration driven abstraction. The same PRISM backend generation

pipeline applies once the configuration specifies: (i) the set of exposed state variables and their finite domains, (ii) the discrete-time step semantics and horizon, (iii) the policy for resolving ties or underspecified choices, and (iv) the external stochastic modules to be composed. This separation supports systematic variation across architectures and scenarios without changing the underlying source cognitive models.

The approach generalizes most directly to architectures whose operational semantics can be represented as a synchronous step relation over a finite abstraction. Architectures that rely on unbounded data structures, continuous variables, or dense-time semantics may require additional abstraction (e.g., discretization or predicate abstraction) or a different verification backend. Nevertheless, the translation framework provides a common blueprint for compiling cognitively grounded task logic into verification models that can be composed with probabilistic human-state and environment dynamics.

## 6 Conclusion and Future Work

We presented a translation framework that compiles Soar cognitive task models into PRISM discrete-time Markov chains, enabling probabilistic model checking of properties that depend on both cognitive task progression and stochastic dynamics such as cybersickness and performance effects. The translation maps a finite abstraction of working-memory features to finite-domain state variables, compiles productions into guarded updates, and introduces an explicit control automaton to preserve the ordering constraints of the Soar decision cycle. The resulting PRISM models support quantitative analysis using probabilistic temporal logics and rewards, and they compose naturally with externally parameterized stochastic modules.

Future work will extend the translator along three directions. First, we will expand the class of analyzable models by supporting richer stochastic plug-ins and by integrating reward structures for expected-time and expected-cost analyses aligned with mission assurance objectives. Second, we will strengthen semantic assurance by developing trace-based conformance checks between Soar executions and PRISM paths, including automated regression tests over representative use cases and configuration variants. Third, we will broaden architecture coverage by instantiating the same compilation pattern for additional cognitive architectures (e.g., ACT-R), including standardized interfaces that extract rule-tuple representations and architecture-specific phase semantics.

This work advances the use of formal methods for cognitive security and human-in-the-loop assurance by providing a practical and semantically grounded pathway from executable cognitive models to verification environments.

**Acknowledgments.** This effort was sponsored by the Defense Advanced Research Projects Agency (DARPA) under agreement number HR0011-24-9-0439. The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

**Disclosure of Interests.** The authors have no competing interests to declare that are relevant to the content of this article.

## References


1. Amundson, I., Babar, J., Herencia-Zapana, H., Rollini, S.F., Brussee, B., Wu, P., Wang, T.E., Newendorp, A.K., Kohl, A.R., Fieffer, S.J., Khan, S.S., Sanaei, M., Muscala, M., Gilbert, S.B., Winer, E., Dorneich, M.C., Lathrop, J., Musliner, D., P. Goldman, R., Gottlieb, J., Ganeriwala, P., Chambers, C., Bhattacharyya, S.: Modeling and formal analysis of high-assurance mixed-reality systems. In: 2025 AIAA DATC/IEEE 44th Digital Avionics Systems Conference (DASC). pp. 1–10 (2025). https://doi.org/10.1109/DASC66011.2025.11257175
2. Anderson, J.R., Bothell, D., Byrne, M.D., Douglass, S., Lebiere, C., Qin, Y.: An integrated theory of the mind. Psychological review **111**(4), 1036 (2004)
3. Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)
4. Behrmann, G., David, A., Larsen, K.G.: A tutorial on uppaal. Formal methods for the design of real-time systems pp. 200–236 (2004)
5. Bhattacharyya, S., Neogi, N., Eskridge, T., Carvalho, M., Stafford, M.: Formal assurance for cooperative intelligent agents. In: NASA Formal Methods Symposium LNCS. vol. 10811 (2018). https://doi.org/10.1007/978-3-319-77935-5_2
6. Bhattacharyya, S., Davis, J., Gupta, A., Narayan, N., Matessa, M.: Assuring increasingly autonomous systems in human-machine teams: An urban air mobility case study. In: Formal Methods for Autonomous Systems. vol. 348, pp. 150–166 (2021). https://doi.org/10.4204/eptcs.348.11, https://doi.org/10.4204/eptcs.348.11
7. Bolton, M.L., Bass, E.J.: Enhanced operator function model (eofm): A task analytic modeling formalism for including human behavior in the verification of complex systems. In: The handbook of formal methods in human-computer interaction, pp. 343–377. Springer (2017)
8. Bolton, M.L., Bass, E.J., Siminiceanu, R.I.: Using formal verification to evaluate human-automation interaction: A review. IEEE Transactions on Systems, Man, and Cybernetics: Systems **43**(3), 488–503 (2013)
9. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuxmv symbolic model checker. In: CAV. pp. 334–342 (2014). https://doi.org/10.1007/978-3-319-08867-9_22, https://doi.org/10.1007/978-3-319-08867-9_22
10. Duan, J., Li, C., Yang, G., Qu, C., Chang, E., Zhang, Z., Che, X.: Study of cybersickness in augmented reality railway inspections applications. IEEE Access **12**, 143252–143262 (2024)
11. Ganeriwala, P., Matessa, M., Bhattacharyya, S., Jones, R.M., Davis, J., Kaur, P., Rollini, S.F., Neogi, N.: Design and validation of learning aware hmi for learning-enabled increasingly autonomous systems. In: 2025 IEEE International Systems Conference (SysCon). pp. 1–8. IEEE (2025). https://doi.org/10.1109/SysCon64521.2025.11014784, https://doi.org/10.1109/SysCon64521.2025.11014784
12. Ganeriwala, P., Matsumuro, M., Ritter, F.E., Bhattacharyya, S.: Enabling formal verification in a common model of cognition (2025), https://sbp-brims.org/2025/papers/working-papers/2025_SBP-BRiMS_paper_18.pdf, sBP-BRiMS 2025 Working Paper

13. Ganeriwala, P., Narayan, N., Jones, R.M., Matessa, M., Bhattacharyya, S., Davis, J., Rollini, S.F., Purohit, H., Neogi, N.: Systems engineering with architecture modeling, formal verification, and human interactions for learning-enabled autonomous agent. Systems Engineering (2025)
14. Grumberg, O., Clarke, E., Peled, D.: Model checking. In: International Conference on Foundations of Software Technology and Theoretical Computer Science; Springer: Berlin/Heidelberg, Germany (1999)
15. Guntupalli, L., Li, F.Y.: Dtmc modeling for performance evaluation of dw-mac in wireless sensor networks. In: 2016 IEEE Wireless Communications and Networking Conference. pp. 1–6. IEEE (2016)
16. Kotseruba, I., Tsotsos, J.K.: 40 years of cognitive architectures: core cognitive abilities and practical applications. Artificial Intelligence Review **53**(1), 17–94 (2020)
17. Kwiatkowska, M., Norman, G., Parker, D.: Prism: Probabilistic model checking for performance and reliability analysis. ACM SIGMETRICS Performance Evaluation Review **36**(4), 40–45 (2009)
18. Kwiatkowska, M., Norman, G., Parker, D.: Prism 4.0: Verification of probabilistic real-time systems. In: International conference on computer aided verification. pp. 585–591. Springer (2011)
19. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic model checking and autonomy. Annual review of control, robotics, and autonomous systems **5**(1), 385–410 (2022)
20. Laird, J.: The SOAR Cognitive Architecture. MIT Press (2012). https://doi.org/10.7551/mitpress/7688.001.0001
21. Laird, J., Rosenbloom, P., Newell, A.: Chunking in soar: The anatomy of a general learning mechanism. Machine Learning **1**, 11–46 (03 1986). https://doi.org/10.1007/BF00116249
22. Laird, J.E., Lebiere, C., Rosenbloom, P.S.: A standard model of the mind: Toward a common computational framework across artificial intelligence, cognitive science, neuroscience, and robotics. Ai Magazine **38**(4), 13–26 (2017)
23. Langenfeld, V., Westphal, B., Podelski, A.: On formal verification of act-r architectures and models. In: Proceedings of the Annual Meeting of the Cognitive Science Society. vol. 41 (2019)
24. Langley, P., Laird, J.E., Rogers, S.: Cognitive architectures: Research issues and challenges. Cognitive Systems Research **10**(2), 141–160 (2009)
25. Narayan, N., Ganeriwala, P., Jones, R.M., Matessa, M., Bhattacharyya, S., Davis, J., Purohit, H., Rollini, S.F.: Assuring learning-enabled increasingly autonomous systems*. In: 2023 IEEE International Systems Conference (SysCon). pp. 1–7 (2023). https://doi.org/10.1109/SysCon53073.2023.10131227
26. Nason, S., Laird, J.: Soar-rl: Integrating reinforcement learning with soar. Cognitive Systems Research **6**, 51–59 (03 2005). https://doi.org/10.1016/j.cogsys.2004.09.006
27. Neogi, N., Bhattacharyya, S., Griessler, D., Kiran, H., Carvalho, M.: Assuring intelligent systems: Contingency management for uas. IEEE Transactions on Intelligent Transportation Systems **22**(9), 6028–6038 (2021). https://doi.org/10.1109/TITS.2021.3076399
28. Norris, J.R.: Markov Chains. Cambridge University Press (1998)
29. Romero, O.J.: Cogarch-adl: Toward a formal description of a reference architecture for the common model of cognition. Procedia computer science **145**, 788–796 (2018)
30. Sun, R.: Anatomy of the mind: exploring psychological mechanisms and processes with the Clarion cognitive architecture. Oxford University Press (2016)
31. Tomaszewska, J.: Application of markov chains, mtbf and machine learning in air transport reliability. Aviation and Security Issues **4**(2), 83–106 (2023)

32. Wang, T.E., Amundson, I., Babar, J., Wu, P.: Formal analysis of vulnerabilities in mixed-reality systems. In: IEEE International Conference on Systems, Man, and Cybernetics (2025)
33. Wickens, C.D.: Situation awareness and workload in aviation. Current directions in psychological science **11**(4), 128–133 (2002)