On Formal Verification of ACT-R Architectures and Models

Vincent Langenfeld (langenfv@tf.uni-freiburg.de)
Department of Computer Science, Albert-Ludwigs-Universität Freiburg

Bernd Westphal (westphal@tf.uni-freiburg.de)
Department of Computer Science, Albert-Ludwigs-Universität Freiburg

Andreas Podelski (podelski@tf.uni-freiburg.de)
Department of Computer Science, Albert-Ludwigs-Universität Freiburg

Abstract

Subject of this article is the question whether the potential for automatic defect analysis for symbolic timed ACT-R models as demonstrated in earlier work can be developed into a scalable and comprehensible technique. We present a formal, operational model of an ACT-R architecture and a translation scheme of ACT-R models into timed automata. We have applied this translation to ACT-R models and report on scalability experiments with automatic defect analysis.

Keywords: ACT-R; Cognitive Architecture; Formal Methods; Timed Automata; Modelling

Introduction

ACT-R (Anderson, 1983, 2009) is a cognitive architecture (an implementation of a unified theory of cognition) that is widely used in cognitive modelling to validate psychological theories. A psychological theory is a hypothesis on how a given task is solved by humans. Psychological theories can be validated by constructing an ACT-R model that implements the psychological theory and comparing the model's predictions to experimental data.

The work (Langenfeld, Westphal, Albrecht, & Podelski, 2018) points out that it is critical for this approach that an ACT-R model *correctly* implements the psychological theory because an incorrect ACT-R model (wrt. the psychological theory) may lead to a false rejection or false acceptance of an invalid theory as follows. An incorrect ACT-R model may give predictions that *do not match* experimental data although the theory's predictions would (thus false rejection), or the model may give predictions that *do match* the experimental data although the theory's predictions would not (thus false acceptance).

(Langenfeld et al., 2018) introduce the notion of *model defect* in general, i.e., any kind of programming error in production rules like simple typing errors, forgotten conditions or requests, etc. They formally study the following three model properties that can indicate the presence of errors. The first considered model property is called *deadlock*, a situation where model execution cannot continue although the model is not in a final state. The second property is *correctness of the mental model*, that is, the questions whether it is possible to observe expected chunks (as defined by the psychological theory) during model executions and whether it is impossible to observe unexpected chunks. The third property is *timing feasibility*, that is, whether an ACT-R model is principally able to reproduce timing aspects that are observed in experimental data (e.g. given a model, is it possible to complete the necessary computation steps of the ACT-R model within the time frame observed with human participants). Using an abstract, formal semantics of ACT-R (Albrecht & Westphal, 2014b), it is principally possible, effective, and useful from a modeller's point of view to automatically and exhaustively analyse ACT-R models for the absence of defects (Langenfeld et al., 2018). Spotting such errors by simulation alone is, in contrast, tedious and time consuming in general.

Subject of this article is the question whether the potential for automatic defect analysis for symbolic timed ACT-R models as mentioned above can be developed into a scalable and comprehensible technique. To this end, we present a formal, operational model of an ACT-R architecture and a translation scheme of ACT-R models into the same formalism of timed automata (Alur & Dill, 1994), that allows us to easily model the discrete, timing, and concurrency aspects of ACT-R and that is well supported by existing analysis tools (Behrmann, David, & Larsen, 2004). We have applied this translation to artificial ACT-R models as well as an ACT-R model from the research literature and we have found the analysis of the resulting network of timed automata (using existing tools) to scale well, both in number of chunks and number of production rules. By using the formalism of timed automata, we obtain a comprehensible model including all architecture aspects, hence the potential to analyse theories regarding architectures in addition to psychological theories.

Related Work. Formalisations of the ACT-R semantics appear in (Albrecht & Westphal, 2014b) and later in (Gall & Frühwirth, 2014; Gall & Frühwirth, 2018), and are used towards comparing cognitive architectures (Ragni et al., 2018).

Preliminary results on the formal analysis of ACT-R models for defects have been presented in (Albrecht & Westphal, 2014a) and elaborated in (Langenfeld et al., 2018). The feasibility of such analyses is investigated by encoding simplified fragments of ACT-R architecture and model aspects and selected rules into logical formulae that can effectively be analysed for satisfiability. This work, in contrast, supports a wider range of analysis goals and aims at a comprehensible model of an architecture and a complete ACT-R model.

An analysis procedure for confluence of ACT-R models can be obtained by encoding an ACT-R architecture and mod-

els in constraint handling rules and solving the confluence problems in this domain (Gall & Frühwirth, 2017).

Preliminaries

F-ACT-R. The formal description of ACT-R (Albrecht, 2013; Albrecht & Westphal, 2014a) differentiates between a syntactical description of the ACT-R model and description of the semantics assigned to the constructs of the model by a cognitive architecture.

The abstract syntax of ACT-R defines a model over a set of module signatures. A module signature consists of a finite set of buffers B, a finite set of module queries Q, and a finite set of action symbols A. A production rule r is a pair of a precondition and an action. A precondition is a proposition over buffer slots and module queries. An action is a set of similar propositions together with a buffer and an action symbol. An ACT-R model is a finite set of production rules $R = \{r_1, ...r_n\}$ and a finite set of chunks $\{c_0, ..., c_n\}$.

An ACT-R architecture consists of a interpretation function for the action symbols of modules and a production rule selection mechanism. A cognitive state is a function γ from buffers to pairs (c,d) where c is a chunk and d is a time delay. A pair in γ thus describes buffer contents (if $d = 0$) and buffer assignments in the future (if $d > 0$). The ACT-R architecture works in cycles of production rule selection and execution. Cognitive states γ, γ' are in a successor relation ($\gamma \xrightarrow{(r,t)} \gamma'$), if the selection mechanism chooses production rule r (consuming time t) whose precondition is fulfilled by the current cognitive state γ and γ' is the result of applying the interpretation of every action symbol of r to γ.

Running Example. The addition model from the ACT-R tutorial (Bothell, 2017b), Unit 1.7.1, models the addition of two numbers by counting up from the first number in as many steps as given by the second number. To implement counting, the model uses rules whose preconditions match the current number and retrieve the corresponding count fact, i.e. a pair of a number and its direct successor, from declarative memory. In our examples we use the production rule *initialize-addition*, which is only applicable at the beginning of the computation. Its precondition requires an empty goal buffer slot `sum`. To start the addition, its action assigns the first number of the addition to goal buffer slot `sum`, and 0 to the `count` slot that tracks counting of the second number. Then a retrieval for the successor of `sum` is started.

Timed Automata. Timed automata (Alur & Dill, 1994) are a formal, operational model of real-time systems, i.e., systems that have to compute outputs within certain time intervals. In the simplest case, a *timed automaton* $\mathcal{A}$ is a tuple $(L, B, \mathbb{X}, I, E, \ell_{ini})$ comprising a finite set of *locations* L (including the *initial location* ℓ_{ini}), a set of *channels* B, and a set of *clocks* $\mathbb{X}$. Function I labels each location with a clock constraint (called location *invariant*), and E is a finite set of edges. An edge $(\ell, \alpha, \varphi, \rho, \ell')$ comprises source and destination location ℓ and ℓ', action α (which can be the internal action τ, or an *output* $b!$ or *input* $b?$ on a channel $b \in B$), clock constraint φ as *guard*, and the *update* $\rho \subseteq \mathbb{X}$ that denotes the set of clocks to be reset.

The operational semantics of a *network of timed automata* $\mathcal{N} = \mathcal{A}_1 \| \cdots \| \mathcal{A}_n$ ('$\|$' denoting parallel composition) is a labelled transition system over *configurations* $\langle \vec{\ell}, \nu \rangle$ where $\vec{\ell}_i$ is the *current location* of automaton $\mathcal{A}_i$ and $\nu : \mathbb{X} \to \mathbb{R}_0^+$ is a valuation of the clocks. Two configurations are in transition relation $\langle \vec{\ell}, \nu \rangle \xrightarrow{\lambda} \langle \vec{\ell}', \nu' \rangle$ if and only if $\lambda \in \mathbb{R}_0^+$, $\vec{\ell} = \vec{\ell}'$, and $\nu' = \nu + \lambda$ satisfies the invariants of all locations in $\vec{\ell}$ (delay transition), or there is an edge $(\ell, \tau, \varphi, \rho, \ell') \in E_i$ such that $\vec{\ell}_i = \ell$, $\vec{\ell}'_i = \ell'$, φ is satisfied in ν, and ν' is obtained from ν by resetting the clocks in ρ to zero (internal transition), or there are two edges enabled in two different automata in $\mathcal{N}$ with complementary input and output actions (rendezvous transition). A *computation path* of a network of timed automata is a sequence of configurations starting with $\langle \vec{\ell}_0, \nu_0 \rangle$ where $\vec{\ell}_0$ comprises the initial locations, and ν_0 assigns value 0 to all clocks (and satisfies all location invariants), and subsequent configurations are in transition relation.

The modelling, simulation, and model-checking tool Uppaal (Behrmann et al., 2004) extends the simple case by features such as data variables, broadcast channels, and committed locations where no delay is possible. In the remaining article, we use a graphical representation of timed automata (see, e.g., Figure 1) where the double outline location is initial, locations marked by a 'C' are committed locations, and invariants (if any) are shown in purple. Edges are annotated with action (in cyan), guard (in green), and updates (in blue).

TA-ACT-R

In this section, we describe how we represent ACT-R models by networks of timed automata that can then be analysed for ACT-R model defects. Recall that an ACT-R model R is a finite set $\{r_1, \ldots, r_n\}$ of production rules that has computations *on* an architecture A.

Given an ACT-R model R and an architecture A, we construct networks $\mathcal{N}^R$ and $\mathcal{N}^A$ of timed automata such that we can conclude from analysis results of the network $\mathcal{N}^R \| \mathcal{N}^A$ to the presence or absence of model defects in the ACT-R model on architecture A. Constructing the network $\mathcal{N}^A$ can be considered a one-time effort: In the case described below, we consider timed automata that follow the production rule selection mechanism and the behaviour of models in the ACT-R tool. Network $\mathcal{N}^A$ can be composed with any $\mathcal{N}^R$, i.e., with any network of a specific ACT-R model, as long as model R is compatible with the modules offered by A.

In the following paragraphs, we first describe the construction of the timed automata in $\mathcal{N}^A$ that model module behaviour, here on the example of the declarative module. Then we describe the construction of production rule automata to obtain $\mathcal{N}^R$, and conclude with the production rule selection mechanism in $\mathcal{N}^A$. Figure 5 visualises the overall structure and potential communication between the timed automata in $\mathcal{N}^R \| \mathcal{N}^A$ over shared channels.

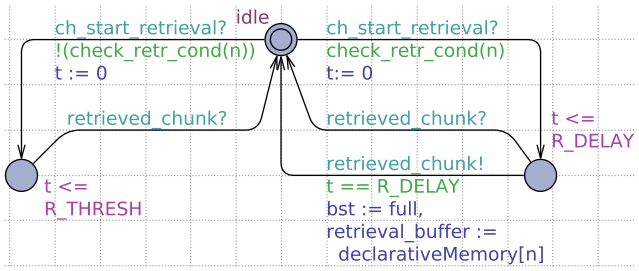

Figure 1: TA-ACT-R chunk automaton $\mathcal{A}^{ch}$.

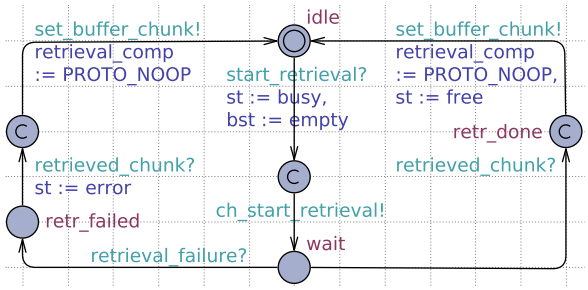

Figure 2: TA-ACT-R automaton $\mathcal{A}^{D}$ (declarative).

Chunks and the Declarative Module. The declarative module is responsible for memory management, i.e. to maintain and recall chunks of previously learned information. Actions of production rules can initiate a recall of information, e.g., the successor of a number in the addition model, and the declarative module delivers one (of possibly many) matching chunks or none at all. Recalling information takes time: In ACT-R, the declarative module takes a certain amount of time to recall information (retrieval delay) or considers a recall failed after a time limit (retrieval threshold).

Our TA-ACT-R model of the declarative module is a set of timed automata that realise the behaviour described above. It comprises one timed automaton $\mathcal{A}^{D}$, and one timed automaton $\mathcal{A}^{ch}$ for each chunk in memory. The idea of this structure is that, for each recall action, $\mathcal{A}^{D}$ stimulates all chunk automata at once, and each chunk automaton with a chunk matching the current recall action offers its chunk to $\mathcal{A}^{D}$. Selection between matching chunks is non-deterministic (and exhaustively considered in analysis).

Figure 1 shows the TA-ACT-R chunk automaton $\mathcal{A}^{ch}$. From the initial location `idle` there are two possible ways of handling a recall action: Either the chunk managed by $\mathcal{A}^{ch}$ matches the request (continue to the right) or not (continue to the left). The distinction between the two cases is made by function `check_retr_cond()`, which hides the details of comparing the request (as specified by shared variables). Both edges (to the right and to the left of `idle`) reset clock t to 0. By the invariants of the bottom right (or left) locations (shown in purple), either an amount of time units corresponding to retrieval delay or retrieval threshold pass. In case of a match (bottom right location), automaton $\mathcal{A}^{ch}$ can send or receive on the broadcast channel `retrieved_chunk`. If multiple chunks match, exactly one chunk automaton nondeterministically acts as sender and all others receive simultaneously. In any case (including no matching chunk), the synchronisation moves the automaton back to location `idle` and the recall action is completed from the perspective of the chunk automaton. Only the chunk automaton acting as sender writes its chunk into the shared variable `retrieval_buffer` that models the retrieval buffer of the declarative module and sets the buffer flag `bst` to indicate that there is a chunk in the retrieval buffer. This update corresponds to placing the retrieved chunk in the declarative module's retrieval buffer.

That is, the timing behaviour of the declarative module is modelled in the chunk automata. Note that our model can support any number of chunks in memory, yet an upper bound on the number of $\mathcal{A}^{ch}$ automata needs to be fixed before analysing the model. This constraint corresponds to the observation that the majority of ACT-R models considers cognitive tasks that are solved in bounded time, and there is the assumption that only finitely many chunks can be used in bounded time. Yet an analysis of a TA-ACT-R model can detect if a given upper bound is sufficient to support a given ACT-R cognitive model. If not, the upper bound on the number of chunks can be increased and the analysis restarted, which in particular allows us to analyse the maximum number of chunks actually considered in a given ACT-R model. Note, that the specified number of chunks only restricts memory size but not memory content. Chunk content may be set in advance modelling pre-existing declarative knowledge (as in the addition model) and content may be acquired by chunk automata during run time (modelling learning of declarative knowledge).

Checking for matching chunks, retrieving one matching chunk (if any), and reporting the result of the recall action is organised by the timed automaton $\mathcal{A}^{D}$ shown in Figure 2.

In the TA-ACT-R model, the recall action is started by a synchronisation on channel `start_retrieval` with a rule automaton (see below) and taking the edge from location `idle` downwards. Without intermediate delay, the module flags are updated to indicate that the declarative module is busy and then the chunk automata are triggered by sending on channel `ch_start_retrieval` (cf. Figures 2 and 1) and changing to location `wait`. From location `wait`, there are two cases: either at least one chunk matched or none. The first case is handled by the sequence of edges to the right of `wait` (by synchronising with the chunk automata as explained above) and updating some more shared variables of the network, so that other automata in the complete network can access the recalled chunk. The second case (no chunk matched) is handled by the mostly symmetric sequence of edges to the left of `wait`, which sets the module's flags accordingly. In both cases, before returning to location `idle` and being ready for the next recall action, the procedural automaton is notified of completion by synchronisation on channel `set_buffer_chunk`.

Note that Figure 1 shows a simplified chunk automaton for

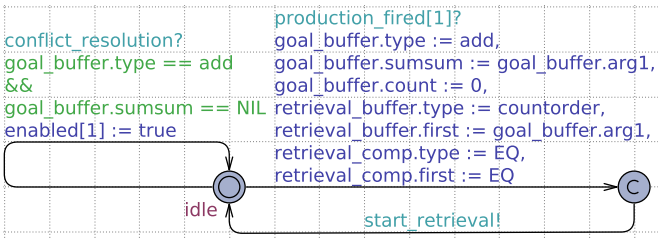

Figure 3: TA-ACT-A automaton $\mathcal{A}^r$ of the production rule *initialize-addition* of the addition model.

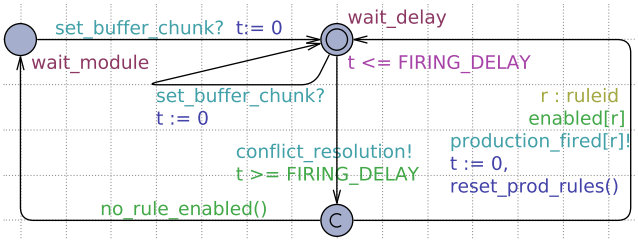

Figure 4: TA-ACT-R automaton $\mathcal{A}^P$ (procedural).

brevity of this presentation. In general, the declarative module is able to learn new chunks. Further note that TA-ACT-R models the purely symbolic variant of ACT-R declarative memory, that is, fulfilling a request is a sufficient condition for a chunk being retrieved from memory. In general, retrieval through the declarative module is affected by an *activation* value that models the effect of frequent usage of a chunk and prevents chunks with an activation value below a given threshold from being retrieved from memory. The analysis of the TA-ACT-R model presented here hence detects model errors like deadlock or (in)correctness of the mental model under the assumption of perfect memory. These errors do not disappear when considering activation hence such errors should be removed before considering more expensive analysis with activation (which is future work).

Production Rules. Given an ACT-R model R consisting of the rules $r_1, \ldots, r_n$, the network $\mathcal{N}^R$ is the parallel composition of n rule automata, i.e. $\mathcal{N}^R = \mathcal{A}^{r_1} \| \cdots \| \mathcal{A}^{r_n}$.

Figure 3 shows a concrete rule automaton to illustrate the principal construction of rule automata. Each rule automaton has two cycles (or phases) starting in the initial location `idle`. To the left is a single edge that synchronises with the procedural module automaton (see below) to determine the currently enabled production rules. The principle is similar to the selection of a matching chunk from the chunk automata by the declarative module automaton $\mathcal{A}^D$ in that it uses a broadcast channel (here `conflict_resolution`).

If the procedural module automaton sends on `conflict_resolution`, all rule automata simultaneously take the left edge from `idle` to `idle` if the guard is satisfied. Automaton $\mathcal{A}^r$ writing a value 1 into their position of the shared array variable `enabled` indicates that it is possible to fire rule r under the current module configuration.

If rule r is selected by the procedural module, the latter sends on the rendezvous channel `production_fired`$[r]$ such that the rule automaton takes the sequence of edges to the right of `idle` (cf. Figure 3). The first edge in the sequence has updates according to the actions of the rule, followed by a sequence of edges that trigger activities of modules (in this example, of the declarative module discussed above).

Figure 3 actually shows the rule automaton of the production rule *initialize-addition* (cf. Preliminaries). The precondition of this rule, r_{ia} for short, is satisfied if the goal buffer does not yet hold an intermediate or final result. This precondition has a direct translation to the guard (shown in green) of the left edge in Figure 3. The action of r_{ia} updates the goal buffer and prepares the buffer of the declarative module for a chunk retrieval. This action directly translates to the update (shown in blue) of the right edge in Figure 3.

The general translation of a rule from an ACT-R model uses the structure shown in Figure 3. A translation of the rule's precondition becomes the guard of the left edge and a translation of the rule's action becomes the update of the right edge, followed by a sequence of synchronisations to initiate behaviour of the modules referred to in the rule.

Procedural Module. Figure 4 shows the automaton that realises the behaviour of the procedural module which selects enabled rules for execution. As explained with the rule automaton above, there are two phases. A rule execution cycle starts in location `wait_delay` by sending on channel `conflict_resolution` on the downward edge. Rules whose preconditions are fulfilled receive, and update the shared array `enabled` accordingly. If at least one rule is enabled, the lower location is exited to the right. One enabled rule is selected non-deterministically and sending on the corresponding `production_fired` channel triggers the execution of the action of the selected rule. The shared array `enabled` is also reset on the right edge back to `wait_delay`. In location `wait_delay`, the invariant and the guard of the outgoing edge ensure that the next rule is executed at least `FIRING_DELAY` time units later. In case that no rule is enabled, the procedural module automaton waits for any module to change state, since only a change in the cognitive state makes it necessary to check again for an enabled rule (Anderson, 2009; Bothell, 2017a). The self loop on the location `wait_delay` ensures that the firing delay is observed if the cognitive state changes during the procedural module waiting for the next rule execution cycle to start.

The Rule Execution Cycle. Figure 5 shows everything put together. The network $\mathcal{N}^A$ modelling the (ACT-R model independent) architecture is the parallel composition $\mathcal{A}^P \| \mathcal{A}^D \| \cdots$ of all module automata (as discussed above). The network $\mathcal{N}^R$ representing the behaviour of the considered ACT-R model $R = \{r_1, \ldots, r_n\}$ is the parallel composition $\mathcal{A}^{r_1} \| \cdots \| \mathcal{A}^{r_n} \| \mathcal{A}^{ch}_1 \| \cdots \| \mathcal{A}^{ch}_m$ of the rule automata for R with m chunk automata (memory size).

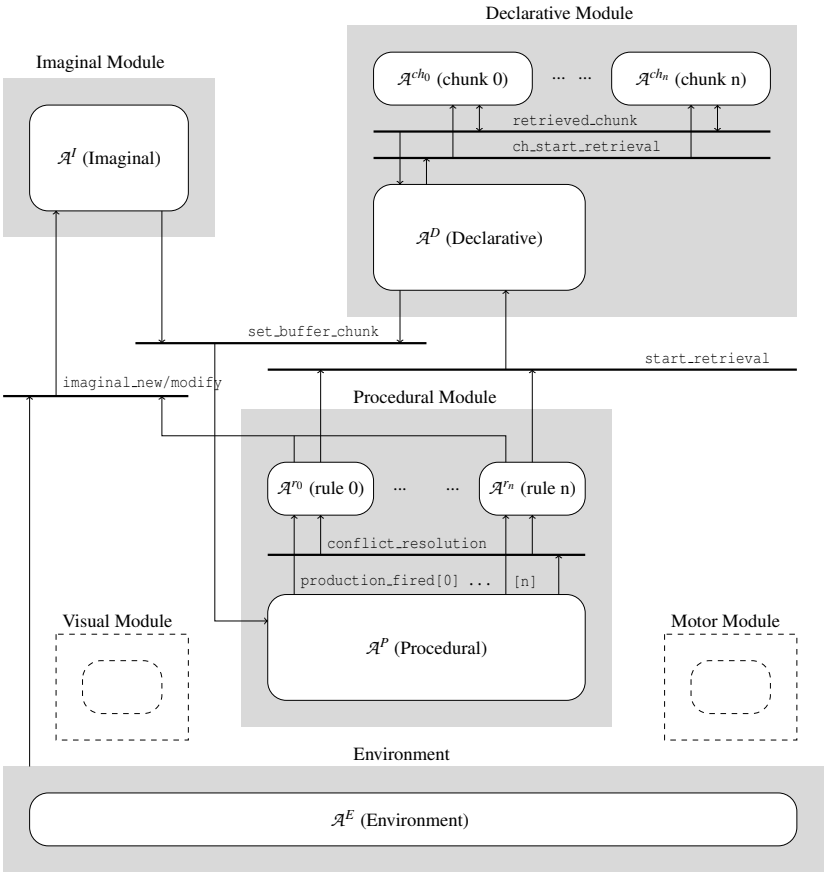

Figure 5: Structure of TA-ACT-R. Each white rectangle represents one timed automaton in the TA-ACT-R model. Arrows show potential synchronisation and are directed from senders to receivers and labelled with the channel name. The grey boxes group together those timed automata that together model an architecture module. Note that the environment does not directly interact with the imaginal module in ACT-R, but through, e.g., the visual module. For our experiments, we have abstracted from this indirection, a timed automaton model of, e.g., the visual module would be placed in the structure as shown by the dashed boxes.

The cognitive architecture of ACT-R interprets ACT-R models by repetitive application of the following, 3-step rule execution cycle: 1.) wait a fixed time, 2.) check rules' preconditions on the current cognitive state to determine the set of enabled rules, and 3.) executing the action of an enabled rule (if any; otherwise wait for a change of the cognitive state). Modules may work during the waiting time in Step 1. The same execution cycle is directly visible in our TA-ACT-R model where Steps 1 to 3 are driven by automaton $\mathcal{A}^P$ and the steps are conducted in cooperation with the rule automata. The basic rule execution cycle is controlled by $\mathcal{A}^P$ (acting as sender on different channels), yet during its waiting time module automata may work concurrently.

Execution of the TA-ACT-R addition model would start by $\mathcal{A}^P$ waiting for the fixed time in location `wait_delay` (Step 1; cf. Figure 4) and then triggering each rule automaton to update their enabled flag (Step 2; cf. Figure 4 and 3). In our TA-ACT-R addition model, the shared variables are initialised such that the initialisation production rule (as shown in Fig. 3) is enabled. Hence $\mathcal{A}^P$ would then trigger this rule automaton (Step 3; cf. Figure 4 and 3). The rule automaton executes the actions of its rule (possibly in cooperation with module automata) while $\mathcal{A}^P$ is already back in location `wait_delay`, that models Step 1. The retrieval action started by the rule automaton is processed (including the retrieval delay) by the declarative module and chunk automata (cf. Figure 2 and 1). After completion of this retrieval, $\mathcal{A}^P$ is notified about the changed cognitive state and commences the next rule execution cycle (Step 1).

Formally, we observe sequences of timed automata configurations that are related by delay or synchronisation transitions. In these transition sequences of the TA-ACT-R network, we can clearly identify those configurations that correspond to situations *right before* starting a new rule execution cycle. In the more abstract F-ACT-R semantics, a rule execution cycle basically corresponds to one transition between two cognitive states, namely $\gamma_0 \xrightarrow{(r_1,50)} \gamma_1$ where $\gamma_0 = \{\text{goal} \mapsto (c_0,0), \text{retrieval} \mapsto (\bot,0)\}$ is the initial cognitive state and $\gamma_1 = \{\text{goal} \mapsto (c_0,0), \text{retrieval} \mapsto (c_2,50)\}$ is the cognitive state at the end of the rule execution cycle yet waiting for the retrieval action to complete.

The abstract, F-ACT-R computation paths of a given ACT-R model are hence refined by TA-ACT-R computation paths (one transition in the F-ACT-R model is related to a sequence of transitions in the TA-ACT-R model), which in turn is refined by computations of the ACT-R tool. In all three cases, we can clearly pinpoint the configuration right before the next rule execution and thus conclude from, e.g., an analysis of a TA-ACT-R model to the reachable cognitive states in the more abstract F-ACT-R view.

Discussion. Figures 1 to 4 show an abstract, comprehensible, readable and simulatable *model* of an ACT-R architecture. Using this architecture model, it becomes remarkably easy to evaluate ACT-R models under different architecture assumptions of a much wider range than the parameters of the ACT-R simulator allow. For example, other retrieval delays are obtained by redefining constants in $\mathcal{A}^D$ (cf. Fig.2); counting presentations (to support activation values) can be realised by increasing a counter in the successful case of a chunk automaton; unsuccessful retrieval of chunks in memory (sporadic forgetting) can be realised by removing the left edge from `idle` in the chunk automaton; etc.

By using a formal modelling language like timed automata, we obtain a precisely defined semantics. In contrast to a textual description of ACT-R's behaviour, it is unambiguously determined which delays or edges are possible in each model configuration. The Uppaal tool uses this fact to offer a convenient simulation environment that shows, in each configuration, the enabled edges and allows a user to choose the next one. If a model analysis finds a defect, the simulator can be used to inspect one computation path that exhibits the defect.

From these two aspects, we also envision a use of our TA-ACT-R models in teaching ACT-R: We see our model to fill a gap between a slide presentation of the concepts and principles of ACT-R and the ACT-R tool. Instructors could use the timed automata simulator in order to present the dynamic behaviour of the ACT-R architecture from rule selection to module activities before referring students to the ACT-R tool.

Evaluation

A highly relevant question on model analysis techniques and tools is about scalability. To be practically useful, a tool needs to be able to analyse ACT-R models that are used in cognitive science research.

Our investigation of the scalability of our TA-ACT-R-based approach to model analysis considers the following three research questions: (1) How does the number of chunks in the declarative memory affect the consumption of computational resources? (2) How does the length of the cognitive computation path affect the consumption of computational resources? (3) How does the number of rules in the ACT-R model affect the consumption of computational resources?

Addition Model. We have investigated the scalability of our approach using a parameterised ACT-R model of the addition task. Table 1a reports measurements of the classical addition model with four rules that we apply to a given number of count order chunks in declarative memory. The goal, that is, the number of count steps necessary to complete the addition, is fixed and thereby we isolate the effect on computational resource consumption to the number of chunks. The analysis checks that for each TA-ACT-R computation path, we finally observe the correct result in the goal buffer. Table 1a shows that the analysis of this parameterised addition model easily scales to 1,000 chunks considered for retrieval, while the length of the TA-ACT-R model computation path remains constant as expected from the fixed goal. Time consumption increases about linearly because each step of the analysis algorithm needs to check each chunk automaton for whether it offers a matching chunk; the reason for increased memory consumption is that the number of automata in the network uniformly increases the size of each TA-ACT-R configuration.

Table 1b reports measurements from the same model discussed above but with increasing addition goal. The model is supposed to apply the highest number of count steps possible with the given chunks, i.e. the instance with 1,000 chunks is supposed to conduct 999 count steps. The time needed for the analysis in the table scales roughly linearly in both, number of chunks and length of computation; the numbers of reachable TA-ACT-R configurations in the table grow linearly in the length of the computation. Table 1b shows that an exhaustive analysis of the model with a few hundred chunks takes not much more than a minute. With an analysis time in this low order of magnitude, we anticipate that our TA-ACT-R analysis can be effectively used during the process of cognitive modelling, that is, to analyse an ACT-R model for common errors, and, in case errors are found, to fix these errors and re-run the analysis. For large chunk numbers and computation lengths, the time needed to complete the analysis becomes more noticeable. We suggest to value the computation time wrt. the obtained outcome: After (in case of the addition model) about 4 minutes, *all possible computations* of the cognitive model have been considered.

Table 1c reports measurements from a different addition model where each count fact is modelled as its own production rule. That is, in order to, e.g., do 100 count steps, there are 100 different rules. Table 1c shows that the analysis of this parameterised addition model easily scales to 1,000 rules.

Preferred Mental Model Theory. To evaluate the performance of our TA-ACT-R-based approach on a cognitive model from the research literature, we have considered the PMMT[1] model that has been used in (Langenfeld et al., 2018) to illustrate the usefulness of checking models for the absence of deadlocks (a deadlock is a cognitive state where no production rule is able to fire while the end of the modelled behaviour has not been reached).

[1]The preferred mental model theory (PMMT; Ragni, Knauff, & Nebel, 2005; Ragni & Knauff, 2013) is the most recent refinement of the established mental model theory (MMT; Johnson-Laird, 1980), that aims to explain human spatial reasoning.

Decl.	Time	States	Memory
25	0.09 s	165	8.1 MiB
50	0.17 s	165	8.9 MiB
100	0.29 s	165	10.7 MiB
500	1.20 s	165	24.5 MiB
1,000	2.70 s	165	40.8 MiB

(a) Computational resources used with increasing number of chunks in the declarative memory (Decl.) for fixed addend 9.

Decl.	Time	States	Memory
25	0.1 s	681	8.4 MiB
50	0.7 s	1,431	10.6 MiB
100	2.5 s	2,931	17.8 MiB
500	65.2 s	14,931	177.6 MiB
1,000	254.8 s	29,931	653.9 MiB

(b) Resource consumption with increasing number of chunks (Decl.), with highest possible addend (chunk number minus 1).

Proc.	Time	States	Memory
100	0.8 s	11,806	9.0 MiB
1,000	11.9 s	17,230	58.1 MiB

(c) Resource consumption with increasing number of production rules (Proc.), with highest possible addend.

Table 1: Evaluation results for time and memory consumption of an exhaustive analysis of addition models with `verifyta` 4.1.19 (Behrmann et al., 2004). Column 'States' gives the number of reachable configurations of the network of timed automata (cf. Preliminaries). The figures given above are averaged over ten runs (i7-6500/2.5 GHz, 8 GiB, Windows 10/64bit laptop).

The considered ACT-R model of the PMMT is technically non-trivial as it makes use of multiple modules (often in the same rule) and depends on complex preconditions including buffer requests and module queries. From its design parameters (about 40 production rules, less than 10 learned chunks), we would have expected an exhaustive analysis of the computational space of the TA-ACT-R model to take at most one second considering the figures in Table 1. In fact, the analysis was much faster: The analysis tool `verifyta` (Behrmann et al., 2004) reported the absence of deadlocks for every possible combination of two premises and a conclusion within 146 *ms* (storing 701 TA-ACT-R states in 8.7 MiB of memory). Thus there are complex ACT-R research models that can be very efficiently analysed for the absence of model defects (Langenfeld et al., 2018).

Conclusion and Future Work

As future work we will automate the translation of the production rules of an ACT-R model to the according automata to enable the analysis of models without manual translation. We will also extend TA-ACT-R by hybrid processes like chunk activation and retrieval delays. We will also integrate hybrid processes (e.g. calculation of chunk activation and retrieval delays) into TA-ACT-R to replace the non-deterministic sub symbolic layer for more precise analysis of ACT-R models.

In this article we investigated the potential for automatic defect analysis of ACT-R models. We developed a formal but easy to comprehend model of the ACT-R architecture and a translation scheme for ACT-R models. Benchmark results show, that the analysis of useful properties scales well for high numbers of chunks and production rules so that it can be applied during model development.

Acknowledgements

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) PO 279/2-1.

References

Albrecht, R. (2013). *Towards a Formal Description of the ACT-R Unified Theory of Cognition*. Unpublished master's thesis, Albert-Ludwigs-Universität Freiburg.

Albrecht, R., & Westphal, B. (2014a). Analysing Psychological Theories with F-ACT-R. *Cogn. Processing*, *15*, 77–79.

Albrecht, R., & Westphal, B. (2014b). F-ACT-R: Defining the Architectural Space. *Cogn. Processing*, *15*, 79-81.

Alur, R., & Dill, D. L. (1994). A theory of timed automata. *Theoretical Computer Science*, *126*(2), 183–235.

Anderson, J. R. (1983). *The Architecture of Cognition*. Psychology Press.

Anderson, J. R. (2009). *How can the human mind occur in the physical universe?* Oxford University Press.

Behrmann, G., David, A., & Larsen, K. G. (2004). *A Tutorial on Uppaal*. In *SFM* (Vol. 3185, p. 200-236). Springer.

Bothell, D. (2017a). *ACT-R 7 reference manual.* Retrieved from `http://act-r.psy.cmu.edu/actr7`

Bothell, D. (2017b). *ACT-R Tutorial.* Retrieved from `http://act-r.psy.cmu.edu/actr7/`.

Gall, D., & Frühwirth, T. (2018). An operational semantics for the cognitive architecture ACT-R and its translation to constraint handling rules. *ACM TCL*, *19*(3), 22:1–22:42.

Gall, D., & Frühwirth, T. W. (2014). A formal semantics for the cognitive architecture ACT-R. In *LOPSTR* (Vol. 8981, pp. 74–91). Springer.

Gall, D., & Frühwirth, T. W. (2017). A decidable confluence test for cognitive models in ACT-R. In *RuleML+RR* (Vol. 10364, pp. 119–134). Springer.

Johnson-Laird, P. (1980). Mental models in cognitive science. *Cognitive Science*, *4*, 71–115.

Langenfeld, V., Westphal, B., Albrecht, R., & Podelski, A. (2018). But does it really do that? Using formal analysis to ensure desirable ACT-R model behaviour. In *CogSci 2018* (pp. 659–664).

Ragni, M., & Knauff, M. (2013). A theory and a computational model of spatial reasoning with preferred mental models. *Psychological review*(3), 561–588.

Ragni, M., Knauff, M., & Nebel, B. (2005). A Computational Model for Spatial Reasoning with Mental Models. In *Proc. of the 27th annual Cog. Sci. Conf.* (pp. 1064–1070).

Ragni, M., Sauerwald, K., Bock, T., Kern-Isberner, G., Friemann, P., & Beierle, C. (2018). Towards a formal foundation of cognitive architectures. In *CogSci 2018* (pp. 2321–2326).