Threaded Cognition

User Guide and Theoretical Notes

Dario D. Salvucci & Niels A. Taatgen

1. Introduction

Threaded cognition (Salvucci & Taatgen, submitted) is an integrated theory of concurrent multitasking — that is, doing two or more things at once. The theory posits that streams of thought can be represented as cognitive threads that are executed together on a resource-limited cognitive processor without the need for specialized executive processes to control them. By specifying the particular limitations of processing on cognitive, perceptual, and motor processes in a computational modeling framework, threaded cognition provides explicit predictions of the bottlenecks that do and do not arise in multitasking behavior for two or more given tasks. The theory has been validated in illustrative domains ranging from simple psychological laboratory tasks such as dual-choice tasks to complex real-world domains such driving.

The theory of threaded cognition has been implemented as a computational model within the framework of the ACT-R cognitive architecture (Anderson et al., 2004). The implementation has no effect on existing ACT-R models, but adds significant new functionality to the architecture in terms of accounting for multitasking behavior. This document provides theoretical and practical details for those interested in using the mechanisms of threaded cognition, or "threading," in their own ACT-R modeling efforts. We assume a basic familiarity with the workings of ACT-R and model development in this framework; please refer to the ACT-R web site (act.psy.cmu.edu) for more information about the architecture.

2. Installation

The implementation of threaded cognition requires ACT-R  ~~To install the code into your ACT-R setup, place the file "threads.lisp" into the directory "actr6/modules/".~~ The code will compile and load automatically during the next startup of the ACT-R system. The provided sample model can then be loaded and run. All code and models have been tested on a Macintosh running OS 10.4.9 and MCL 5.1 with ACT-R version 1.1 [r222].

3. Modeling with Threads

Threaded cognition as implemented in ACT-R allows for multiple model threads to be executed together to produce multitasking behavior. In essence, threaded cognition extends the goal buffer into a set of active goals that progress together in a multitasking fashion. In addition, the theory defines how conflicts for both the procedural resource (i.e., production firing) and the

other resources (declarative, perceptual, and motor) are resolved. The theory and implementation thus provide, for any two or more models, predictions of multitasking behavior when executing the models together.

3.1. Syntax and Semantics

To set a new goal, all current ACT-R models utilize a +goal> construct that replaces the current goal with a new goal chunk. For example, the following rule illustrates how, in the current ACT-R, a rule terminates the current goal and initiates a new goal:

```
(p Start-Next-Goal
   =goal>
      isa current-goal
==>
   +goal>
      isa next-goal)
```

Threading maintains this semantics for the first instance of +goal> in a given rule. However, it extends the semantics by allowing for additional +goal> specifications in the same, where each subsequent specification adds a new goal to the goal set. For example, we can modify the above rule as follows to produce a new rule that initiates two goals:

```
(p Start-Two-Goals
   =goal>
      isa current-goal
==>
   +goal>
      isa next-goal-1
   +goal>
      isa next-goal-2)
```

When this rule fires, the current goal is replaced in the goal set by next-goal-1, and then next-goal-2 is added to this set, resulting in both goals being present in the active set.

For model initialization where the goal-focus command is used to set the initial goal, multiple calls to the this function have an effect analogous to that of +goal> — for instance:

```
(goal-focus goal-1)
(goal-focus goal-2)
```

Here the model begins with both goals in the goal set, and threading takes over to manage the execution of both goal's threads.

When a goal terminates without needing to start a subsequent goal, a model should use –goal> to denote termination as follows:

```
(p Terminate-Goal
   =goal>
      isa current-goal
==>
   -goal>)
```

This command removes the current goal from the goal set. Note that for previous ACT-R models, this would essentially halt the production system, but in the presence of threading this

terminates only the thread corresponding to the current goal and allows other threads to continue execution.

It is critical to note that threading does not change the semantics or execution of existing ACT-R models. Because all previous models have at most one +goal> specification in the rule actions, this specification behaves in an identical way (replacing the current goal), and thus execution is exactly the same with or without threading. In essence, we can think of previous models as all being single-threaded and thus not being affected in any way by the new threading capabilities.

3.2. Modeling Guidelines

Models that run with the new threading mechanism are largely identical to those already developed or under development in the ACT-R architecture. There is only one additional guideline imposed by threading related to checking the status of processing modules and buffers. It is already typical for ACT-R models to test whether a module is "free" on the left-hand side of a rule before initiating a module process on the right-hand side; for example, the following rule tests the motor module and ensures that the system waits for any existing motor movements to complete before firing the rule and starting another movement:

```
(p Move-Hand
   =goal>
      isa goal
   ?manual>
      state free
==>
   +manual>
      isa punch
      hand left
      finger index)
```

Some module processes, including perceptual (visual and aural) and declarative retrieval processes, place a result chunk in the module's buffer for subsequent use. It is important that, before starting on a new process on these modules, rules also test to make sure that the buffer is empty, as shown in these two examples:

```
(p Attend-Visual-Location
   =goal>
      isa goal
   ?visual-location>
      state free
      buffer empty
==>
   +visual-location>
      isa visual-location)

(p Retrieve-Chunk
   =goal>
      isa goal
   ?retrieval>
      state free
      buffer empty
==>
   +retrieval>
      isa chunk)
```

The checks for both a free module and an empty buffer are critical for threading. When multiple threads run concurrently, the check for a free module ensures that no other thread is using that module; in fact, this occurs in many current ACT-R models as well. The check for an empty buffer ensures that, when a perceptual or retrieval process completes and the result is stored in the buffer, another thread does not start a process and overwrite the results before the requesting thread can use the results.

Some models already follow these guidelines, which are indeed good guidelines for any ACT-R models, threaded or not. But without concurrent processing, such checks are not normally needed. We suggest that all models include these checks, whether or not the models themselves utilize concurrent processing.

4. Issues and Future Work

As described in the full paper (Salvucci & Taatgen, submitted), we have validated the threading mechanism for several laboratory and real-world domains. However, there are at least two issues that we hope to address with future work. First, there is currently no limitations on the number of goals that can be stored in the goal set (in much the same way that there are no limitations on the number of slots in a declarative chunk). Clearly there should be some limits here, and it seems reasonable that a chunk in the goal set, or any chunk in any module buffer for that matter, can and should decay away, providing a natural limit to the size of the goal set and duration of a goal. However, we currently have no theory to specify this decay in more detail, and more importantly, it is not clear that we have sufficient empirical evidence for guiding such a development. Thus, just as for the number of chunk slots, the limits are left to the modeler with the understanding that models should keep a reasonable number of goals in the goal set and not overload the set with an unrealistic number of chunks and/or maintain goals for an unreasonable period of time.

Another future development involves an integration of threaded cognition with recent modeling work on task switching (e.g., Altmann & Trafton, 2002). As detailed in the full paper, we believe that threading provides a complementary mechanism to those explored for task switching, and the two approaches are very amenable for a unified approach; however, we have not yet attempted such a unification. Further work should clarify the roles of these various mechanisms for different types of multitasking.

References

Altmann, E. M., & Trafton, J. G. (2002). Memory for goals: An activation-based model. *Cognitive Science*, *26*, 39-83.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036-1060.

Salvucci, D. D., & Taatgen, N. A. (submitted). Threaded cognition: An integrated theory of concurrent multitasking. Revised manuscript submitted to *Psychological Review*.