gactar

A tool for creating & running basic ACT-R models on multiple implementations using a single declarative file format

Andy Maloney

Independent Researcher

Jennifer Schellinck

Renan Ozen

Brendan Conway-Smith

Robert L. West

Carleton University

Why gactar?

Situation: Different ACT-R implementations have been developed

- Are outputs consistent with models implemented in ACT-R 7.0?

Solution: “gactar” provides formal validation via transpilation

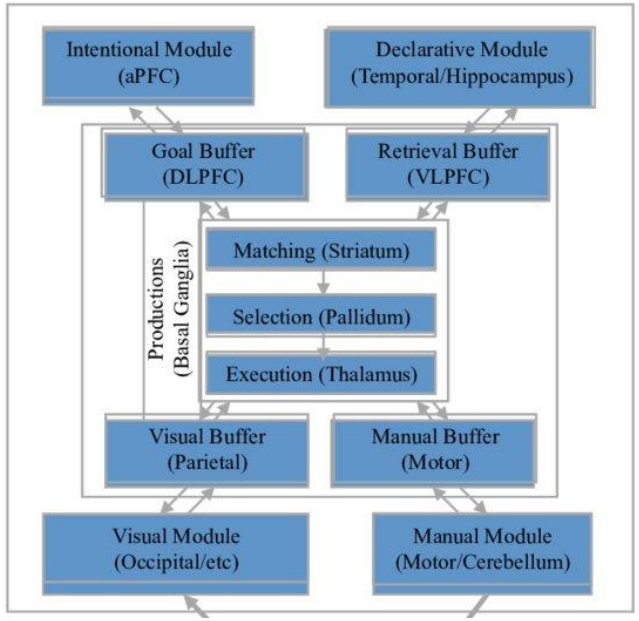
- taking source code written in one language and transforming into another language that has a similar level of abstraction.
- Exposes potential divergences from theory or expected behaviours

amod → Lisp
→ Python

Why transpiling?

- to assist congruence
- people want to use ACT-R in different languages and situations
- makes clear what the other versions don't do
- is Python ACT-R really doing what Lisp ACT-R is doing?
- a formal way of showing equivalence
- trust issues with the Python code

What is gactar?



Open-source project developed by Andy Maloney

- Data scientist, software engineer

https://github.com/asmaloney/gactar

- Collaboration is welcome!

Why gactar?

Accessibility for novice modelers

- Abstracts away programming to focus on modelling
- amod format is easy to read & understand

Production descriptions are restricted to a small set of actions to prevent programming “outside the model”

Buffers

Goal

- Stores current goal

Retrieval

- Stores chunks retrieved from declarative memory

Imaginal

- keeps track of what you are doing

Chunks

Structure of chunks are declared in the *config* section of an amod file

```
[chunk_name: slot_name1 slot_name2 ...]

   [count: first second]
   [word: form category]
   [property: object attribute value]
```

Productions

```
(production_name) {
      match {
            (buffer condition)
      }
      do {
            (action)
      }
}
```

Productions

match section describes chunk patterns that will fire the production when matched in some buffer

```
(production_name) {
        match {
                (buffer condition)
        }
        do {
                (action)
        }
}
```

Productions

match section describes chunk patterns that will fire the production when matched in some buffer.

do section prescribes action to take when production fires

```
(production_name) {
      match {
            (buffer condition)
      }
      do {
            (action)
      }
}
```

Actions

Command	Example
clear *(buffer name)+*	**clear** goal, retrieval
print (string/var/number)+	**print** 'text', ?var, 42
recall (pattern)	**recall** [car: ?color]
set (buffer name).(slot name) **to** (string/var/number)	**set** goal.wall_color **to** ?color
set (buffer name) **to** (pattern)	**set** goal **to** [start: 6 nil]

Productions

Example: increment

```
increment {
    match {
        goal [countFrom: ?x !?x counting]
        retrieval [count: ?x ?next]
    }
    do {
        print ?x
        recall [count: ?next ?]
        set goal.start to ?next
    }
}
```

DEMO

https://github.com/asmaloney/gactar

References

Maloney, A. (2021). gactar: A Tool For Exploring ACT-R Modelling. https://dx.doi.org/10.13140/RG.2.2.25387.36642