Should Act-R Include Production Refraction?

Richard M Young

Psychology Department
University of Hertfordshire

Talk presented at Act-R Workshop
Carnegie Mellon University
25-27 July 2003

Conflict Resolution and Instantiations

- *Conflict resolution* is the term used with production systems to mean how, on each cycle, from among the possibly several or many productions whose conditions are satisfied, one or more productions are chosen to be fired.
- Act-R fires just a single production on each cycle. Conflict resolution consists of computing the quantity E = PG–C for each satisfied production, adding noise, and choosing the production with the highest resulting E value.
- Strictly speaking, conflict resolution applies not to productions but to *instantiations*, where an instantiation is a production together with the data that are matched to its variables. In many production system architectures, a single production can give rise to several instantiations, perhaps only one of which can be allowed to fire.

  However, in Act-R from version 5.0, each production can give rise to at most one instantiation, so the distinction (between productions and instantiations) is less important.

Importance of Conflict Resolution

- The definitive analysis of conflict resolution strategies is:
  McDermott, J. & Forgy, C. (1978) Production system conflict resolution strategies. In D. A. Waterman & F. Hayes-Roth (Eds.), *Pattern-Directed Inference Systems*, 177-199. Academic Press.
- McDermott & Forgy point out the several ways in which a well-chosen conflict resolution strategy can support the aims of a production system architecture
  - — such as sensitivity to the environment, stability (i.e. focus on a goal), learnability, and so on;
  - — although Act-R differs enough from the kind of production systems McD & R were considering that their conclusions need caution in applying to Act-R.
- The Act-R community has a long-standing culture of not being particularly concerned with these points, instead being content to work with *hand-tuned* production systems,
  - — i.e. where if the analyst makes a change in one production, s/he adjusts the other productions as needed.
- Act-R models also tend to be *over-programmed*, in the sense that the productions are carefully (and often ingeniously) written to fire in a pre-ordained sequence.

Importance of Conflict Resolution — 2

- Reasons why this is undesirable include
  1) not get full advantage of potential modularity and independence of productions, e.g. for “lesioning” or adding individual productions;
  2) makes learning harder (automatic acquisition);
  3) over-programming prevents the architecture from playing its role.

Rule Refraction

- One of the conflict resolution methods McDermott & Forgy discuss is *refraction*, where — roughly speaking — once a production has fired, it cannot fire again matched to the same data. Or more accurately: an instantiation cannot fire more than once during its lifetime.
  - — Note that if an item of matching data is removed, and then re-created, then the production can fire again.
- Refraction supports the independence and modularity of productions in several ways, especially by making unnecessary various kinds of fiddly housekeeping operations, which in turn makes learning (automatic acquisition of productions) more feasible.
- Look at a couple of examples …

Example 1. Retrieval from Memory

- In Act-R 5.0, a standard retrieval from memory consists of a *request* production followed by a *harvest* production. To prevent the request production from firing repeatedly, a signal has to be used to indicate that the request has been made.

```
Prequest:  C, not-yet-requested, not-yet-retrieved ==>
                    +retrieval>cue, signal-requested
Pharvest:  C, =retrieval> cue+more  ==>    A
```

With refraction, this could be simplified to

```
Prequest:  C                        ==>  +retrieval>cue
Pharvest:  C, =retrieval> cue+more  ==>  A
```

- Note that one of the issues here, quite apart from refraction, is that the memory buffer does not have an associated status buffer to show when it is busy.
  - — others are aware of that point

But note that the role of refraction goes beyond what a status buffer would do. Once Prequest had fired for a particular (incarnation of) C, it will not fire again (on that same incarnation).

Example 2. Shifting from Compute to Retrieve

- An interesting pattern is where, once a result has been computed one or more times, Act-R begins to retrieve it instead of re-computing it.

  This is typically rather messy. To provide sufficient state signals, the modeller may split the initiation of the computation between the memory-request production and the initiate-computation production, something like this:

```
Rrequest:  C, not-yet-initiated  ==>
                    +retrieval>cue, half-initiate
Rinitiate: C, half-initated   ==>  initiate computation
```


From Compute to Retrieve — 2

With refraction, a much cleaner (and more interesting) model can be written:

```
Prequest:  C, answer=nil     ==>   +retrieval>cue
Pinitiate: C, answer=nil     ==>   initiate computation
Pharvest:  C, answer=nil, retrieved result
                             ==>         set answer
Pcomputed: C, answer=nil, computed result
                             ==>         set answer
```

- Here,
  - — no artificial signals are needed
  - — retrieval & computation in parallel
    - * started in an order determined by the architecture
  - — process terminates when the first one finishes
  - — the retrieval productions can be added after the computation productions, without needing to change them in any way.

Refraction and Parameter Learning

- Worth looking at the interaction between refraction and the parameter learning mechanisms in Act-R.
- Take the basic pattern to be

```
P1:    C1        ==>     A1
P2:    C1, C2    ==>     A2
```

- Usual behaviour is for P1 to fire first, then when C2 becomes true, P2 to fire.
- Parameter learning will cause P1 to be slightly less attractive than P2, because of the extra effort of P1.
- That is correct, because if C2 already holds, we would normally want the model to proceed directly to P2.
- Note that, unlike with some other production system architectures, this mechanism resists being misused as a quick & dirty way to get sequencing. To get cheap sequencing, we'd need higher E (expected gain) for the earlier rules. But Act-R will, correctly, learn a slightly *lower* E for the earlier rules. So the modeller would end up fighting the architecture.

Discussion/Conclusions

- Case that refraction would make a useful addition to Act-R.
- Specifically, it would
  - simplify models
  - lead to more independent and modular productions.
- This would
  - make it easier to construct models, by adding productions to the existing ones, without having to alter the existing ones;
  - make it easier to learn rules automatically.
- That said, the case is not as compelling as it is in McDermott & Forgy's (1978) analysis. The main reason is that OPS-like productions systems tend to use a large, cumulative, internal state based on a large dynamic memory that supports multiple instantiations.
  - Whereas Act-R 5.0 has moved to a small, configural state that *changes* rather than *grows*.
- We should attempt to evolve Act-R culture, by encouraging awareness and good practice in writing LHS conditions that minimise the use of contrived signals, maximise the modularity and independence of productions, and avoid over-programming.