Technical Report:
Porting the Lewis & Vasishth Parser to ACT-R 6

Felix Engelmann
University of Potsdam

Version dated June 16, 2012

Abstract

. . .

What's new in ACT-R 6

...

Syntactic Node Creation

Problems

There is a buffer for every type of constituent. In version 5 the buffers are handled by extra LISP code for creating node chunks in DM. In ACT-R 6 chunks are placed into DM by clearing any buffer, so we can omit the extra code.

So why not use one special node creation buffer? Because the several syntactic buffers have the purpose of creating more than one syntactic node in one production. That would not be possible with only one generic buffer. Options are:

1. Use one buffer and split the productions resulting in one production per node creation.
2. Create 3 special node-creation buffers that can handle up to three nodes maximum at once.
3. Use all syntactic buffers.

With all options the real problem is the "harvesting" of the buffers' contents, meaning the clean operation. This operation cannot be in the same production as the buffer request. Options are:

1. Create extra productions just for harvesting. That would take 50ms for each harvest :(
2. Somehow define the buffers to be cleared immediately after chunk creation.

. . .

3. Specify clearing of all syntactic buffers in one of the productions.

Solution

Node Creation. In order to maintain the possibility to create more than one node in one production I decided on option three: use all syntactic buffers. For that I defined a specific *Parsing Module*. It defines the following buffers:

`IPb, NPb, VPb, VP2b, DPb, CPb, PPb, AdjPb, AdvPb, lex`

Queries and requests are handled with the *goal-style-module* support that comes with ACT-R. That means the buffers mimic the behavior of the GOAL buffer. A request to a buffer creates a chunk of the desired specifications in the buffer.

Harvesting. The harvesting of all buffers happens once per parsing cycle inside the production `attend-to-next-word`:

```
(P attend-to-next-word
   =goal>
      ISA         comprehend-sentence
      state       "looking"
   =visual-location>
      ISA         visual-location
;; adjusted to act-r 6:
;    =visual-state>
;       ISA         module-state
;       modality    free
   ?visual>
      execution       free   ; use this for "saccadic suppression"
;       processor       free
==>
   !eval! (set-current-ip)
   =goal>
      state       "attending"
   +visual>
;       ISA         visual-object
      ISA         move-attention
      screen-pos  =visual-location
       -IPb>
       -CPb>
       -DPb>
       -NPb>
       -VPb>
       -VP2b>
       -PPb>
       -AdjPb>
       -AdvPb>
       -retrieval>
      =visual-location>
   ;    !eval! (delete-bufferchunk-ancestors-retrieval)
)
(spp attend-to-next-word :at 0)
```

Note: It might also be necessary to set the chunk creation time to 0

Unwanted Chunk Duplication

In ACT-R 5 chunks in Declarative Memory can be altered directly from within productions. In version 6, however, a retrieved chunk is always a copy of the original chunk. A retrieved copy of a chunk will only be merged with the original if all features and values are still identical. That has as a consequence that adding information to a retrieved syntactic constituent and releasing it into DM will result in a duplication of that constituent.

To deal with this we could just delete the original chunk when retrieving a copy. But this would leave dead references and reset the activation history of that chunk. A better solution is to merge the two chunks. Since in ACT-R 6 chunks can only be merged when they are exactly the same I introduced a function `called force-merge-chunks`:

```
(defun force-merge-chunks (original copy)
  (let* ((ctype (chunk-chunk-type-fct copy))
         (slots (chunk-type-slot-names-fct ctype)))
    (loop for slot in slots do
      (set-chunk-slot-value-fct original slot (chunk-slot-value-fct copy slot)))
    (if (and (equal-chunks-fct original copy) (merge-chunks-fct original copy))
      (format t " +++ force-merge-chunks succeess +++~%")
      (format t " !!! ERROR: force-merge-chunks failed! !!!"))
))
```

This function first sets all slots of the original to values of the copied chunk and then merges. To do that we have to know which original chunk the copy is from. This has to be done after retrieval. For that I created an event-hook that detects when a new chunk was set into the retrieval buffer and records the original - copy pair:

```
(setf *copied-chunks* '())

(defun detect-set-buffer-chunk (event)
     (when (and (eq (evt-action event) 'SET-BUFFER-CHUNK) (eq (first (evt-params event)) 'RETRIEVAL))
      (with-open-stream (*standard-output* (make-broadcast-stream))
        (setf copy (first (buffer-chunk-fct (list (first (evt-params event))))))
        ;(format t "Hook sees event with module ~S doing ~S with params ~S, bufferchunk is ~S~%" (evt-module event) (evt
        (record-copied-chunk copy))
      ))

(defun record-copied-chunk (copy)
  (when (eq (chunk-chunk-type-fct copy) 'SYN-OBJ)
    ;(format t " +++ record copied chunk ~s +++~%" copy)
    (let ((original (chunk-copied-from-fct copy)))
      (when original
        (if (chunk-p-fct original)
          (progn
            (format t " +++ Chunk ~s was copied from ~s. +++~%" copy original)
            (setf *copied-chunks* (acons copy original *copied-chunks*)))))
)))
```

To register the event-hook put this line into the model file:

```
(add-post-event-hook 'detect-set-buffer-chunk)
```

Now we need to know when to call the forced merge. That is done by putting a hook into the model's sgp parameter definition:

```
:chunk-add-hook merge-copied-syn-obj
```

which calls the following function:

```
(defun merge-copied-syn-obj (copy)
  (when (eq (chunk-chunk-type-fct copy) 'SYN-OBJ)
    ;(format t "  +++ merging copied chunk ~s +++" copy)
;    (format t " +++ chunk: ~S" copy)
    (let ((original (cdr (assoc copy *copied-chunks*))))
      (when original
        (if (chunk-p-fct original)
          (progn
            (format t " +++ Forcing merge unequal copy ~s with source chunk ~s. +++~%" copy original)
            (force-merge-chunks original copy))))
)))
```

Chunk Reference

In the 5.0 version non-existing chunks were referenced, their creation being requested in the same production (e.g. creating a new syn obj with +NPb. . . and referring to it in the SPEC of a chunk in another buffer). This is not allowed by standard ACT-R behavior.

Strictly, we would have to create those references in an extra production after the chunks had been created. But since this information is only relevant for post-hoc trace rebuilding but not important for the parsing process itself, these direct references can easily be substituted by other means.

The solution of choice was to create a unique ID for every newly created node and only use these IDs as references:

```
ID                  =ID-DP
```

In ACT-R 6.0 variables inside productions can be bound in the following way:

```
!bind! =ID-DP (new-name DP)
!bind! =ID-CP (new-name CP)
```

where the `new-name` function creates a unique symbol constructed of the argument and a number. When referring to a chunk inside another chunk (e.g in `COMP` or `SPEC` slot) the respective chunk ID is used instead of the chunk itself.

Lexical Retrieval

The LEX-RETRIEVAL buffer

Instead of the LEX-RETRIEVAL buffer I use the standard RETRIEVAL buffer, which in version 5 was used for syntactic retrieval only. Using one retrieval buffer for both purposes seems unproblematic because the syntactic and lexical retrieval never happen simultaneously.

Open Question: Does the LEX-RETRIEVAL buffer in version 5 support instantaneous retrieval of a word?

If so, in version 6 a lex-retrieval module would have to be created, which builds upon the mechanisms of the RETRIEVAL buffer but without a retrieval delay.

The Lexical Retrieval Production(s)

In version 5 there is a `lexical-retrieval-request-[WORD]` production created for every word in the lexicon. I do not see the reason, so I use one generic production retrieving the word that is in the visual buffer:

```
(p lexical-retrieval-request
    =goal>
       ISA         comprehend-sentence
       state    "attending"
    =visual>
       ISA         text
       value       =word
==>
    =goal>
       state       "lexical-retrieval"
       cue1        =word
       cue2        nil
       cue3        nil
       cue4        nil
    -visual-location>
;    +lex-retrieval>
    +retrieval>
       ISA         lexical-entry
       word        =word

;       !eval! (word-message =word)
)
```

The LEX Buffer

Its purpose seems to be to hold a copy of the content of LEX-RETRIEVAL so this buffer can be used for a new retrieval operation. This is also useful for completely excluding the LEX-RETRIEVAL buffer, because once the lexical content of the RETRIEVAL buffer is stored in LEX, the RETRIEVAL buffer can be used for syntactic retrieval in the next production.

The Copy Mechanism

Copying the content from LEX-RETRIEVAL to LEX in version 5 was done with a reference to the buffer:

```
   +lex>
      ISA               lexical-entry
      =lex-retrieval
```

In ACT-R 6 it looks like this:

```
   =lex>               =retrieval
```

Adding Information to an existing Chunk

Problem

In several productions the IP chunk is modified by a hack editing the chunk directly in memory (looking like a buffer modification). E.g.:

```
   =IP>                                        ;; This may not be legal
      isa              syn-obj
      number           sing
      subj-word        =word
```

In order to avoid hacking, however, the chunk would have to be retrieved first to be modified. I wrote a new hack that tracks the chunk representing the currently closest IP node in a variable `*current-ip*`. That way the current IP chunk can always be modified without the need to retrieve it:

```
(defun set-current-ip nil
  (with-open-stream (*standard-output* (make-broadcast-stream))
    (setf ipchunk (first (buffer-chunk IPb)))
    (if ipchunk (setf *current-ip* ipchunk)))
  (format t " +++ Setting current IP chunk +++")
)

(defun mod-current-ip (modlist)
  (mod-chunk-fct *current-ip* modlist)
  )
```

In order to make sure the variable always contains the current IP the following line is put into the `attend-to-next-word` production:

```
!eval! (set-current-ip)
```

When modifying the current IP in a production now the following code is used (here as an example):

```
   !eval! (mod-current-ip (list
     'number 'sing
     'subj-word =word
     ))
```

Set Goal Category

I rewrote the goal category setting functions:

```
(defun map-goal-category (next-goal)
  (when *verbose* (format t "

 Setting goal to ~A.

  " next-goal))
  (cdr (assoc next-goal *goal-cat-mappings*))
)

(defun map-next-goal-category (next-goal)
  (when *verbose* (format t "

 Setting next-goal to ~A.

  " next-goal))
  (car (rassoc next-goal *goal-cat-mappings*))
)
```

So now instead of accessing the chunks via lisp code with

```
!eval! (set-next-goal-category *IP* =goal-cat)
```

and

```
!eval! (set-goal-category =goal =next-goal)
```

we now only use lisp code to map the goal categories but modify the chunks in ACT-R style:

```
    next-goal        =goal-IP
 !bind! =goal-IP (map-next-goal-category =goal-cat)
```

and

```
     goal-cat        =goal-cat   ; (for goal buffer)
   !bind! =goal-cat (map-goal-category =next-goal)
```

Add Reference

E.g.:

```
!eval! (add-reference (wme-references (get-wme =subj-pred)))
```

The `(add-reference)` call needs to be replaced by some equivalent function. Its purpose is to boost activation of a chunk that is in a slot of the retrieved chunk.

Some Ideas

```
(sdp CHUNK :reference-count 3.0)
(setf rc (first (first (sdp CHUNK :reference-count))))
(sdp-fct CHUNK (list ':reference-count (+ rc 1)))
```

Other Changes

- The production `attach-aux-verb-no-gap` did not fire because the next-goal slot of the IP in the RETRIEVAL buffer is not set.
- The variable *time* - a global ACT-R 5 variable defined in actr5.lisp - has disappeared in ACT-R 6. Now (mp-time) is used for the same purpose. This mainly affects: `!eval! (set-begin-time =word)`.
- The spp parameter `:c` (cost in units of time) in ACT-R 5 (in `(spp Set-Retrieval-Cues-Input-Wh-Pronoun :c .1)`) is now called `:at` (action time).
- the `SetSimilarities` function in the model definition has changed its name to `set-similarities`.
- When modifying buffer do not use `ISA` slot!
- in `+visual-location>` the `attended` slot must be `:attended`.

Further Productions that have to be replaced or added

Replace the productions `find-first-word`, `find-location-of-next-word`, and `stop-marker`.

```
(P find-first-word
   =goal>
      ISA          comprehend-sentence
      state        nil
==>
   !bind! =ID-IP (new-name IP)
   =goal>
      state        "looking"
   +visual-location>
      ISA          visual-location
      screen-x     lowest
      :attended     nil
   +IPb>
      ISA                 syn-obj
      cat                 IP
      ID                  =ID-IP
      waiting-for-cat     wait-for-IP

      waiting-for-finite  wait-for-finite

      finite              finite
      next-goal           next-*done*
)


(P find-location-of-next-word
   =goal>
      ISA          comprehend-sentence
      state        "read"
;; New ACT-R 6 syntax
;    =visual-state>
;      ISA               module-state
;      modality          free
   ?visual>
      state             free
==>
   +visual-location>
      ISA          visual-location
;      screen-x          greater-than-current
    > screen-x          current
      screen-x          lowest
;      :nearest           current
   =goal>
      state        "looking"

   !eval! (set-end-time)
)


(p stop-marker
    =goal>
       ISA         comprehend-sentence
       state    "attending"
    =visual>
       ISA         text
       value       "*"
==>
    =goal>
       state       "stop"
       !eval! (word-message "*")
)
```

Currently open questions

• Why has production `attach-CP-as-SR-modifier-of-retrieved-singular-NP` a `!eval! (set-begin-time =word)` call? Begin time is usually set in the productions that set retrieval cues.

• *vlshort* and *vllong* stops at the preposition. One reason seams to be that the production `SET-RETRIEVAL-CUES-INPUT-PREP` tries to retrieve a chunk with `cat NP-VP`