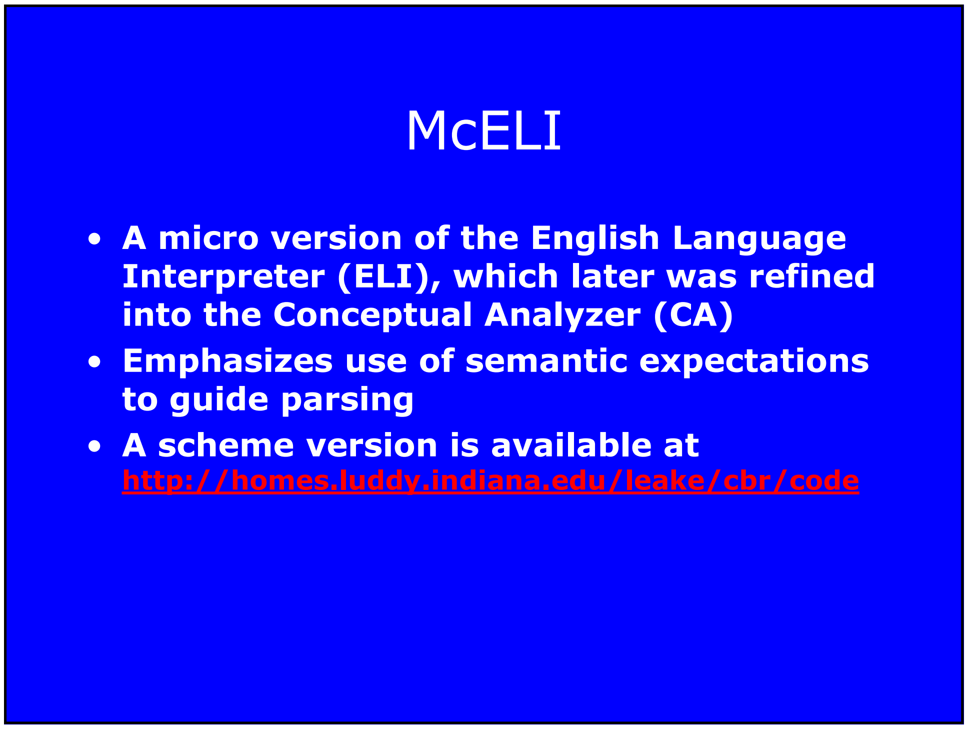
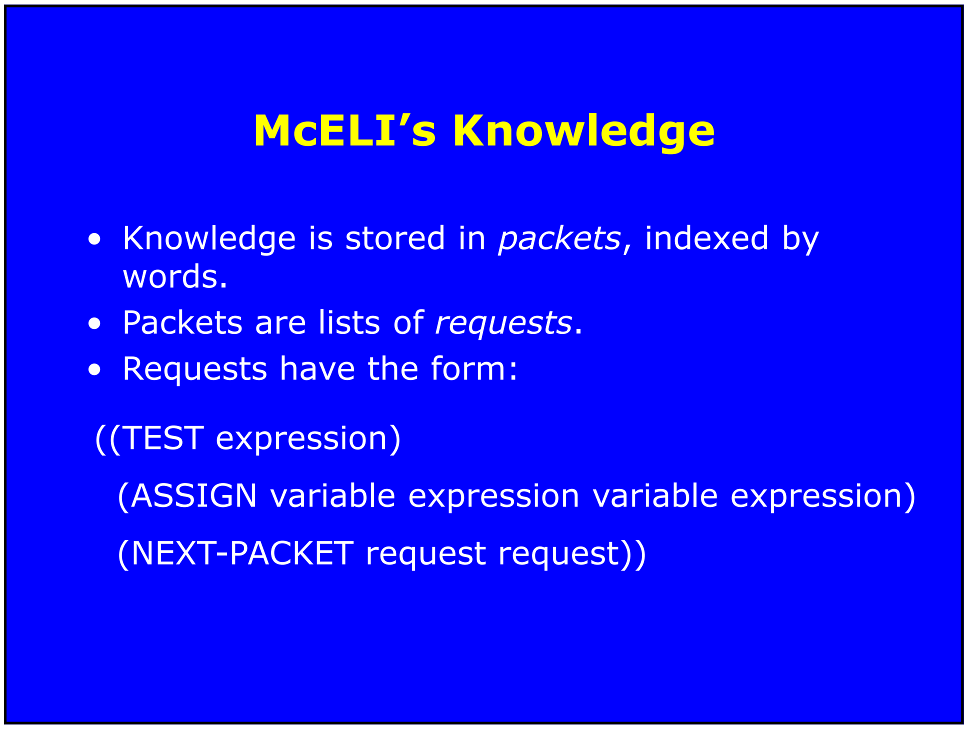


McELI

- **A micro version of the English Language Interpreter (ELI), which later was refined into the Conceptual Analyzer (CA)**
- **Emphasizes use of semantic expectations to guide parsing**
- **A scheme version is available at http://homes.luddy.indiana.edu/leake/cbr/code**

1

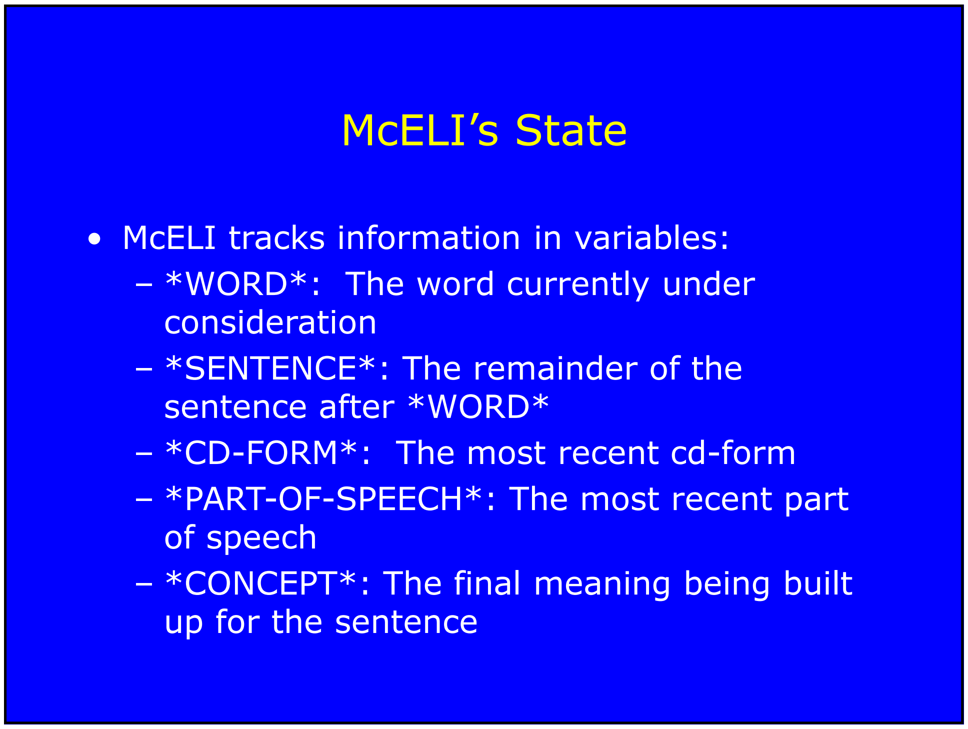


McELI's Knowledge

- Knowledge is stored in *packets*, indexed by words.
- Packets are lists of *requests*.
- Requests have the form:

```
((TEST expression)
  (ASSIGN variable expression variable expression)
  (NEXT-PACKET request request))
```

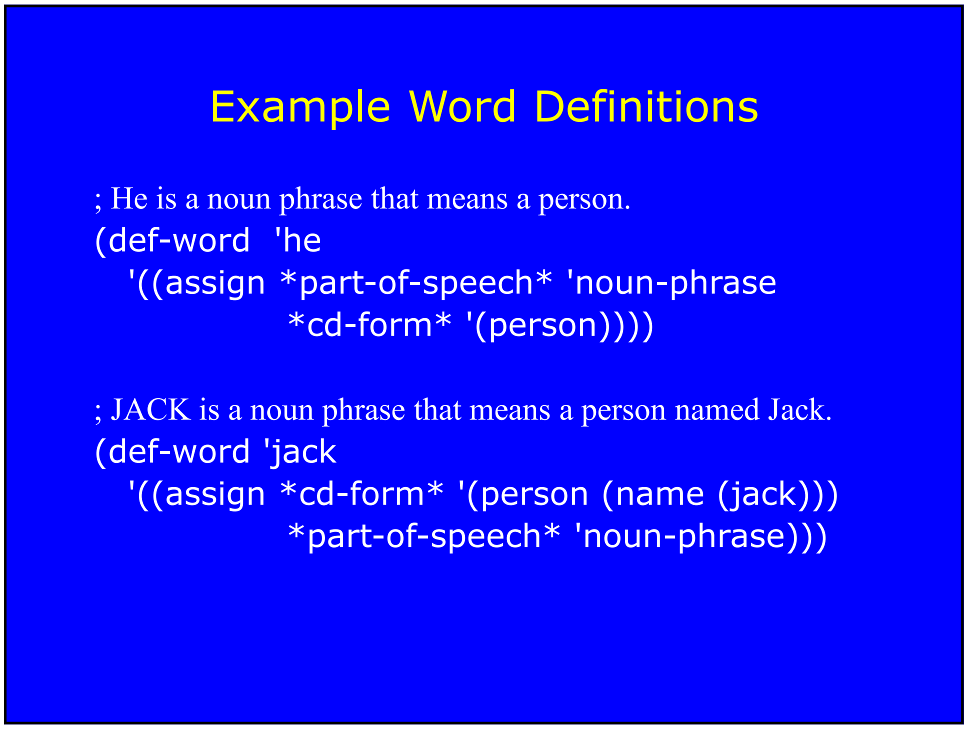
2



McELI’s State

- McELI tracks information in variables:
  - *WORD*: The word currently under consideration
  - *SENTENCE*: The remainder of the sentence after *WORD*
  - *CD-FORM*: The most recent cd-form
  - *PART-OF-SPEECH*: The most recent part of speech
  - *CONCEPT*: The final meaning being built up for the sentence

3

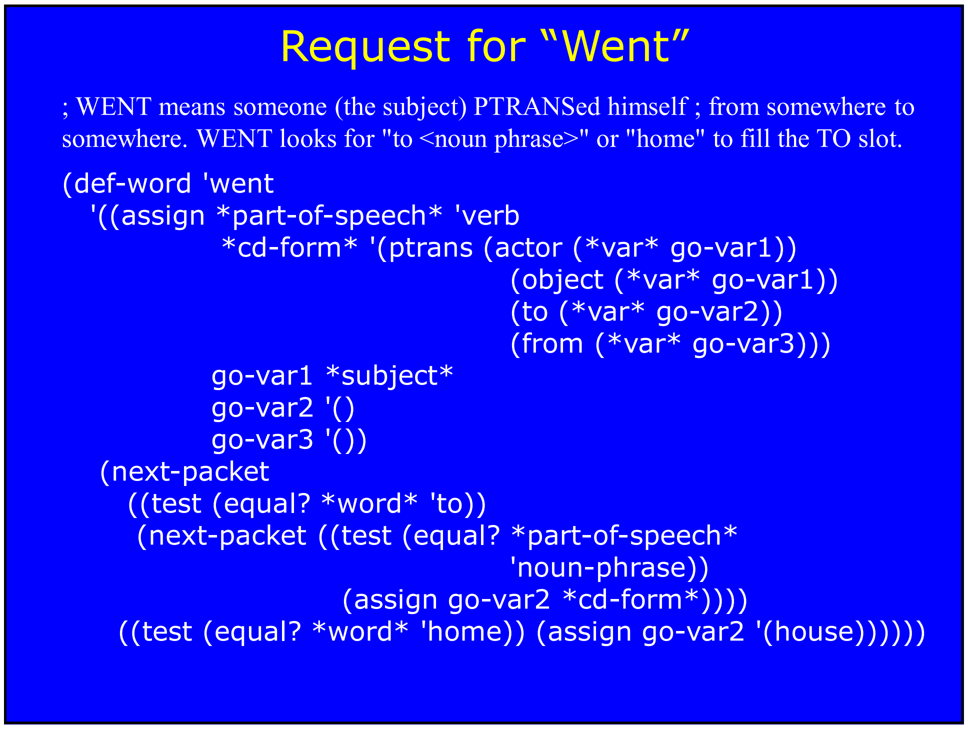


Example Word Definitions

```
; He is a noun phrase that means a person.
(def-word  'he
   '((assign *part-of-speech* 'noun-phrase
                 *cd-form* '(person))))

; JACK is a noun phrase that means a person named Jack.
(def-word 'jack
   '((assign *cd-form* '(person (name (jack)))
                 *part-of-speech* 'noun-phrase)))
```

4

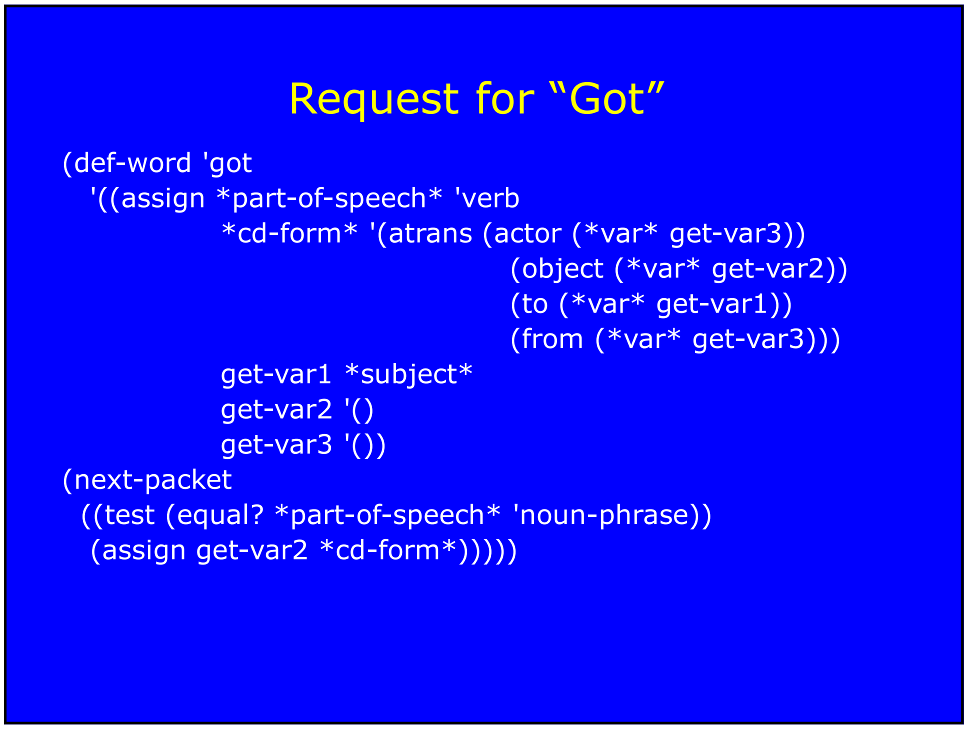


Request for “Went”

; WENT means someone (the subject) PTRANSed himself ; from somewhere to somewhere. WENT looks for "to <noun phrase>" or "home" to fill the TO slot.

```
(def-word 'went
  '((assign *part-of-speech* 'verb
            *cd-form* '(ptrans (actor (*var* go-var1))
                               (object (*var* go-var1))
                               (to (*var* go-var2))
                               (from (*var* go-var3)))
            go-var1 *subject*
            go-var2 '()
            go-var3 '())
    (next-packet
      ((test (equal? *word* 'to))
       (next-packet ((test (equal? *part-of-speech*
                               'noun-phrase))
                     (assign go-var2 *cd-form*))))
      ((test (equal? *word* 'home)) (assign go-var2 '(house))))))
```

5



Request for “Got”

```
(def-word 'got
  '((assign *part-of-speech* 'verb
            *cd-form* '(atrans (actor (*var* get-var3))
                               (object (*var* get-var2))
                               (to (*var* get-var1))
                               (from (*var* get-var3)))
            get-var1 *subject*
            get-var2 '()
            get-var3 '())
(next-packet
  ((test (equal? *part-of-speech* 'noun-phrase))
   (assign get-var2 *cd-form*)))))
```

6

McELI's Algorithm

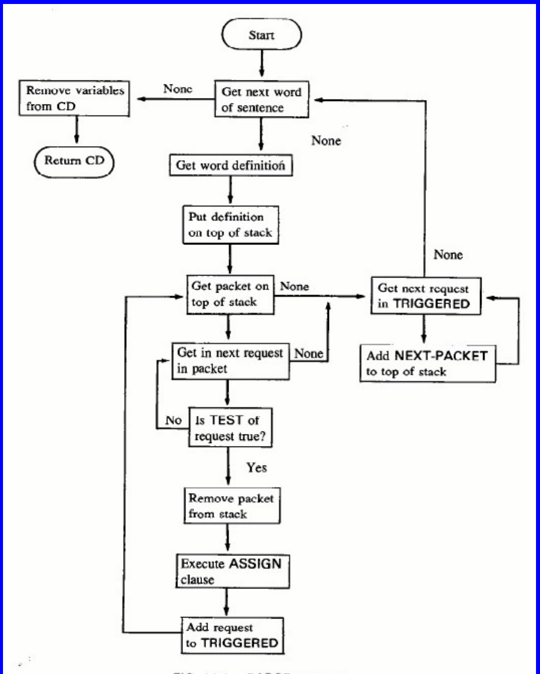


FIG. 14.1. PARSE(sentence).

7

SAM

- SAM was a model of script-based story understanding
  - Scripts are fossilized inference chains
  - Scripts guide choices of inferences and help to distinguish routine and non-routine situations
  - Scripts guided inferencing
- SAM could summarize stories, translate them, and answer questions

8

Sample Q/A Dialogue

- Input text:
- John went to a restaurant. He ordered a hot dog. The waiter said they didn’t have any. He asked for a hamburger. When the hamburger came, it was burnt. He left the restaurant.

9

- Q4: What did the waiter serve John?
- The waiter served John a hamburger.
- Q5: Why didn’t John eat the hamburger?
- Because the hamburger was overdone.
- Q6: Did John pay the check?
- No, John was angry because the hamburger was overdone and so he left the restaurant.

10

Questions and Answers

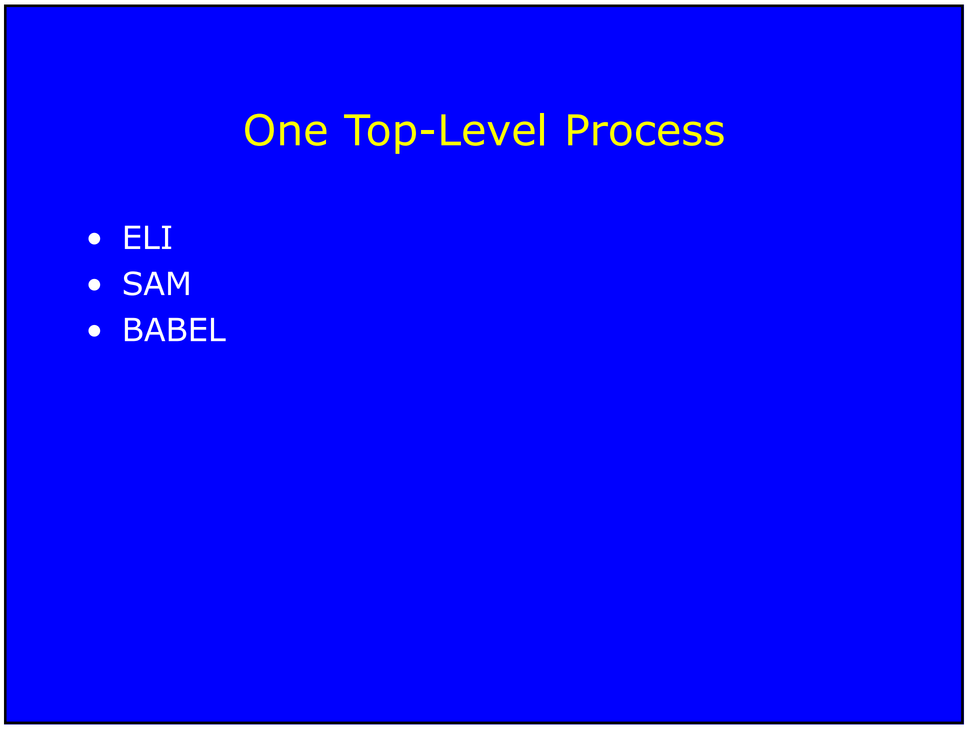
- Q1: Did John sit down in the restaurant?
- Probably.
- Q2: Did John order a hot dog?
- Yes.
- Q3: Did John eat a hot dog?
- No, the waiter told John the management was unable to give it to him.

11

Breakout Groups

- Write basic (very high level) steps for a high-level algorithm for a script-based understander. It should take a natural language story as input and output a summary of the story

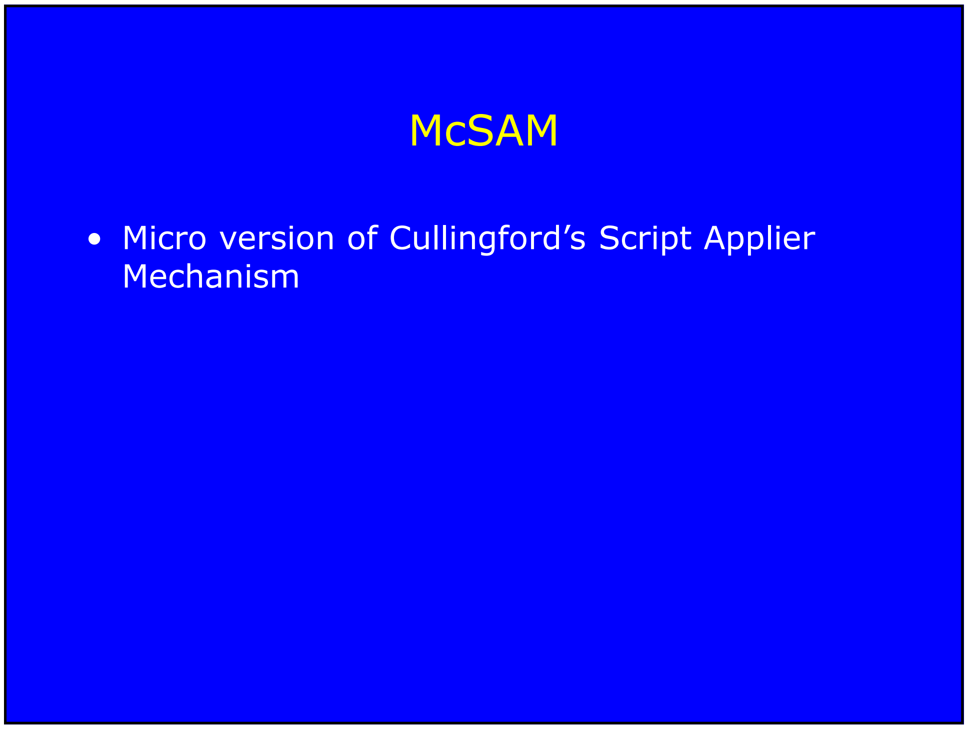
12



One Top-Level Process

- ELI
- SAM
- BABEL

13

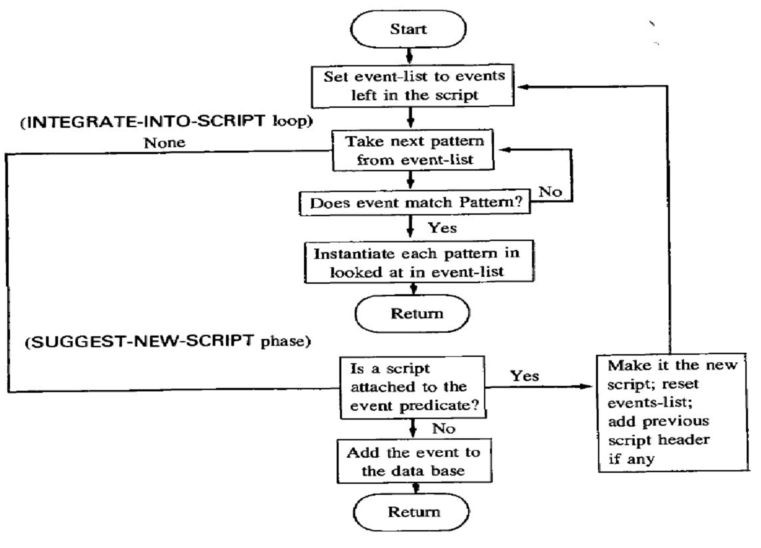


McSAM

- Micro version of Cullingford’s Script Applier Mechanism

14

McSAM's Algorithm Given a CD



Start
Set event-list to events left in the script
(INTEGRATE-INTO-SCRIPT loop)
None
Take next pattern from event-list
Does event match Pattern?
No
Yes
Instantiate each pattern in looked at in event-list
Return
(SUGGEST-NEW-SCRIPT phase)
Is a script attached to the event predicate?
Yes
Make it the new script; reset events-list; add previous script header if any
No
Add the event to the data base
Return

FIG. 6.1. PROCESS-CD(event).

15