

INTELLIGENZA ARTIFICIALE

ANNO ACCADEMICO
2008 - 2009

La Rappresentazione della Conoscenza:
Il Linguaggio OPS5

1

SOMMARIO

- Introduzione al linguaggio OPS5
- La working memory
- Le regole di produzione
- Recognize-Act Cycle
- Esempio di programma in OPS5

2

IL LINGUAGGIO OPS5

- OPS5 è un linguaggio per sistemi di produzioni forward chaining, ossia un membro della famiglia di linguaggi basati sul modello dei **sistemi di produzioni**.
- Un programma in OPS5 consiste in una **declaration section** che descrive gli oggetti trattati dal programma, seguita da una **production section** che contiene le regole di produzione.
- Durante l'esecuzione, i dati elaborati dal programma sono mantenuti nella **working memory** e le regole nella **production memory**.

3

WORKING MEMORY

- La **Working Memory** (WM) è un data base di fatti relativi al problema da risolvere.
- Tali informazioni sono memorizzate sotto forma di elementi di WM che possono essere raggruppati in classi. Ogni classe è individuata da un nome (Class Name).
- La WM è dinamica. Durante la esecuzione di un programma OPS5, gli elementi di WM possono essere aggiunti, cancellati o modificati continuamente.

4

WORKING MEMORY ELEMENT

- Un **Working Memory Element** (WME) è una sequenza di atomi (simboli, numeri interi o floating-point) che rappresenta ad es. un oggetto o un concetto.
- Ogni atomo è memorizzato in un campo che possiamo etichettare con un nome (Attribute Name).
- Possiamo specificare un WME usando una combinazione di:
 - Class Name
 - Lista di attributi scalari e loro valori
 - "Vector attribute" e suo valore

5

WORKING MEMORY ELEMENT

- "Class Name" specifica la classe di appartenenza del WME.
- Gli attributi e i loro valori descrivono le caratteristiche del WME.
- Il valore di ciascun attributo scalare è un atomo.
- Il valore di un "vector attribute" è costituito da uno o più atomi.

6

WORKING MEMORY ELEMENT

- Il format per specificare un WME è il seguente:
`[class-name] [{scalar-attribute value} ...] [vector-attribute value]`
- Consideriamo il seguente elemento:
`(CHECK ^NUMBER 102 ^AMOUNT 10.06 ^COUNTED NO ^DATE 2 JAN 1985)`
- Questo statement rappresenta un WME della classe `CHECK`.

7

WORKING MEMORY ELEMENT

- Gli attributi `^NUMBER`, `^AMOUNT`, `^COUNTED`, e `^DATE` rappresentano quattro caratteristiche dell'elemento.
- I valori degli attributi scalari `^NUMBER`, `^AMOUNT` e `^COUNTED` sono gli atomi `102`, `10.06`, e `NO` rispettivamente.
- I valori del vector attribute `^DATE` è la lista degli atomi `2 JAN 1985`.

8

CLASS NAME

- Un "Class Name" è un simbolo che identifica un gruppo di elementi simili.
- Gli elementi che hanno lo stesso class name hanno gli stessi attributi, anche se i valori di tali attributi possono essere diversi.
- Ad esempio, i seguenti elementi hanno lo stesso class name
CHECK:

(CHECK ^NUMBER 102 ^AMOUNT 10.06 ^COUNTED NO ^DATE 2 JAN 1985)

(CHECK ^NUMBER 103 ^AMOUNT 22.45 ^COUNTED NO ^DATE 14 JAN 1985)

9

ATTRIBUTI

- Un attributo consiste in un operatore (^) e in un nome di attributo.
- L'operatore deve precedere il nome dell'attributo, ma è possibile inserire spazi, tabs, o altri caratteri "nonprinting" tra l'operatore e il nome.
- I nomi degli attributi descrivono le caratteristiche di un WME.
- E' possibile usare lo stesso nome di attributo in più di un WME anche se essi appartengono a classi diverse. Ad es.:

(CHECK ^NUMBER 102 ^AMOUNT 10.06 ^COUNTED NO)

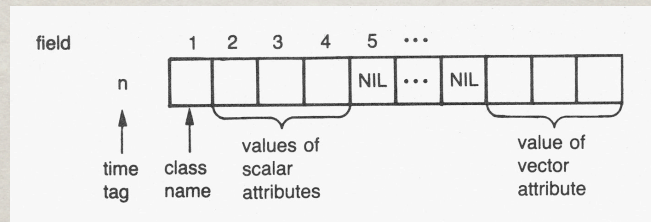
(TRANSACTION ^NUMBER 2560 ^TYPE DEPOSIT)

10

RAPPRESENTAZIONE INTERNA DEI WME

La rappresentazione interna di un elemento include un "time tag" e uno o più atomi che rappresentano la classe dell'elemento e i valori degli attributi.

La figura che segue illustra come gli atomi sono memorizzati nei vari campi:

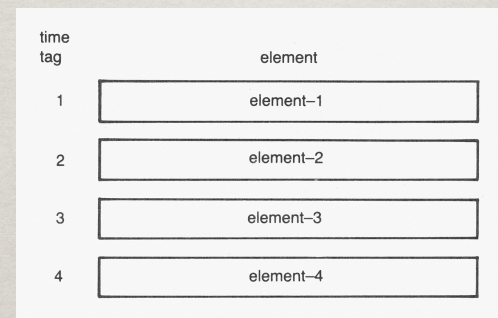


NIL è memorizzato nei campi ai quali non è ancora stato assegnato un valore.

11

TIME TAG

I time tag sono numeri interi consecutivi che il sistema usa per determinare quali siano gli elementi più recenti inseriti in Working Memory. Il sistema run-time assegna un unico time tag a ciascun elemento.



12

MEMORIZZAZIONE DEL CLASS NAME E DEI VALORI DEGLI ATTRIBUTI

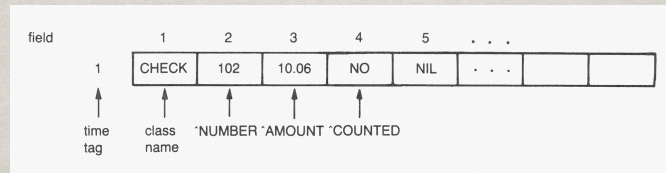
Il primo campo della struttura di un WME è riservato al class name dell'elemento.

Il compilatore assegna i campi ai nomi degli attributi quando essi sono dichiarati. Tali campi servono per memorizzare i valori.

Consideriamo il seguente WME:

(CHECK ^NUMBER 102 ^AMOUNT 10.06 ^COUNTED NO)

La figura illustra la sua rappresentazione interna:



13

MEMORIZZAZIONE DEL CLASS NAME E DEI VALORI DEGLI ATTRIBUTI

Il campo assegnato a ciascun nome di attributo è "globale".

Ciò significa che se un nome si riferisce ad un certo campo, tale nome si riferisce allo stesso campo per ogni element class in cui il nome appare.

Ad esempio, se consideriamo i seguenti elementi:

(CHECK ^NUMBER 102 ^AMOUNT 10.06 ^COUNTED NO)

(ORDER ^DEPARTMENT AUTOMOTIVE ^NUMBER 10-562)

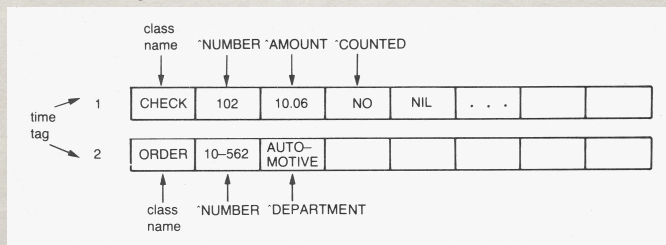
essi hanno differenti class name ma condividono l'attributo scalare ^NUMBER.

14

MEMORIZZAZIONE DEL CLASS NAME E DEI VALORI DEGLI ATTRIBUTI

Supponiamo che nel primo elemento il campo riservato all'attributo ^NUMBER sia il numero 2.

Poiché tale nome si riferisce allo stesso campo per entrambi gli elementi, il valore dell'attributo ^NUMBER è messo nel campo n. 2 di entrambi gli elementi:



15

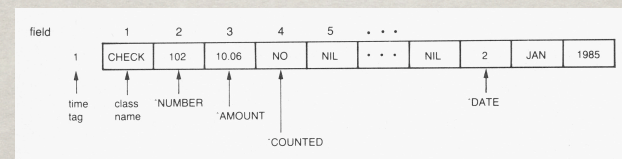
MEMORIZZAZIONE DEL CLASS NAME E DEI VALORI DEGLI ATTRIBUTI

Il compilatore assegna la parte finale della struttura di un elemento per il vector attribute.

Se consideriamo il seguente elemento:

(CHECK ^NUMBER 102 ^AMOUNT 10.06 ^COUNTED NO ^DATE 2 JAN 1985)

la sua rappresentazione interna è la seguente:



16

LA DICHIARAZIONE LITERALIZE

- La dichiarazione **LITERALIZE** ha il seguente effetto:
 - associa una classe ad una lista di nomi di attributi;
 - comunica al compilatore di assegnare i campi agli specificati nomi di attributi.

17

LA DICHIARAZIONE LITERALIZE

Esempio:

```
(LITERALIZE CHECK  
NUMBER  
AMOUNT  
COUNTED)
```

Questa dichiarazione associa il class name **CHECK** ai nomi di attributo **NUMBER**, **AMOUNT** e **COUNTED**.

18

LA DICHIARAZIONE LITERAL

- La dichiarazione **LITERAL** consente di assegnare in modo esplicito dei campi ai nomi di attributo. Ad esempio, questa dichiarazione:

```
(LITERAL NUMBER = 2  
AMOUNT = 4  
COUNTED = 7)
```

consente di assegnare il campo n. 2 all'attributo **NUMBER**, il campo n. 4 a **AMOUNT** e il n. 7 a **COUNTED**.

19

LA DICHIARAZIONE VECTOR-ATTRIBUTE

- La dichiarazione **VECTOR-ATTRIBUTE** consente di assegnare un campo al nome di un vector attribute. Il sistema run-time memorizza gli atomi relativi al valore dell'attributo partendo dal campo assegnato. Esempio:

```
(VECTOR-ATTRIBUTE DATE)
```

- Dopo tale dichiarazione, è possibile specificare il nome di tale attributo in una dichiarazione **LITERALIZE**.

20

LE REGOLE DI PRODUZIONE

• Una regola di produzione consiste di:

- un nome
- una left-hand side (**LHS**)
- una right-hand side (**RHS**)

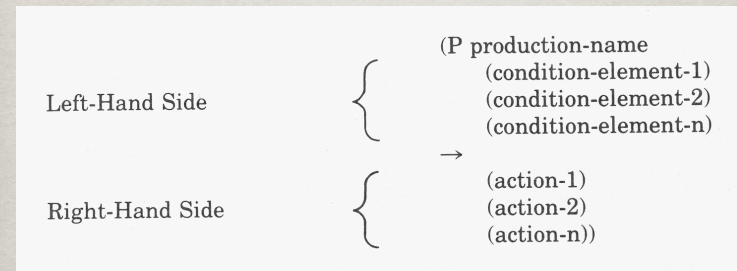
production-name: LHS → RHS

- La **LHS** contiene uno o più "condition element", ossia pattern da confrontare con gli elementi di WM.
- La **RHS** consiste in una o più azioni.

21

LE REGOLE DI PRODUZIONE

Una regola di produzione in **OPS5** assume una forma di questo tipo:



22

LE REGOLE DI PRODUZIONE

Esempio di regola:

```
(P COUNTED-CHECKS
 {<REPLY>
 (REPLY ^DATE {<DAY> <> STOP} <MONTH> <YEAR>)}
 -(CHECK ^DATE <DAY> <MONTH> <YEAR> ^COUNTED NO)
 {<COUNTER>
 (COUNT ^VALUE <VALUE>)}
 →
 (REMOVE <REPLY>)
 (REMOVE <COUNTER>)
 (MAKE START)
 (WRITE (CRLF) (CRLF) |There are| <VALUE> |checks dated| <DAY>
 <MONTH> <YEAR> (CRLF))
```

23

LHS: CONDITION ELEMENTS

- La **LHS** di una regola di produzione contiene una sequenza di condition elements.
- Il sistema run-time confronta gli atomi in un WME con i corrispondenti pattern in un condition element.
- I condition elements possono essere positivi o negativi.
- Una LHS deve contenere come minimo un condition element positivo, e il primo elemento deve essere positivo.

24

LHS: CONDITION ELEMENTS

- Ogni componente di un condition element (class name, attributi scalari, vector attribute) e il suo valore è considerato un **termine**. Ad es., il seguente condition element:

(CHECK ^NUMBER 102)

ha due termini, CHECK e l'attributo ^NUMBER che ha valore 102.

- L'interprete confronta ogni atomo in un WME con il valore del termine corrispondente nel condition element.

25

LHS: CONDITION ELEMENTS

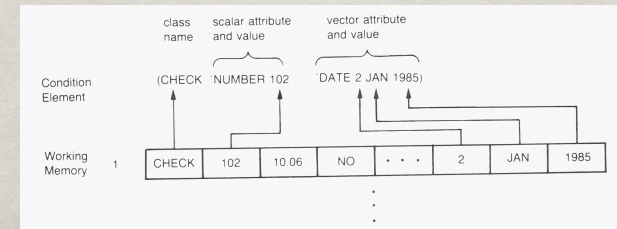
Supponiamo che la working memory contenga il seguente elemento:

(CHECK ^NUMBER 102 ^AMOUNT 10.06 ^COUNTED NO ^DATE 2 JAN 1985)

Consideriamo ora il seguente condition element:

(CHECK ^NUMBER 102 ^DATE 2 JAN 1985)

Il sistema effettua il confronto come illustrato in figura:



26

LHS: VARIABILI

- Una **variabile** è un simbolo racchiuso tra parentesi angolari.
- Una variabile può legarsi ad un atomo di un WME.
- La prima volta che la variabile compare nella regola di produzione, essa si lega all'atomo nel WME che si unifica con il condition element in cui la variabile è presente.
- A questo punto, tutte le successive occorrenze della variabile all'interno della regola in questione rappresentano lo stesso atomo. Ad es., se in LHS si ha:

(REPLY ^DATE <DAY> <MONTH> <YEAR>)
- (CHECK ^DATE <DAY> <MONTH> <YEAR>)

e l'interprete trova un match per il primo condition element, le variabili <DAY> <MONTH> e <YEAR> sono legate agli stessi valori anche nel secondo condition element.

27

LHS: PREDICATI

- I **predicati** sono operatori che possono precedere valori (costanti o variabili) nei termini dei condition element.

- I predicati sono:

= equal
<> not equal
<=> same type
< less than
<= less than or equal
> greater than
>= greater than or equal

28

LHS: ELEMENT VARIABLES

• Un **element variable** è un simbolo racchiuso tra parentesi angolari che consente di riferirsi ad un WME che soddisfa un condition element.

• Per specificare un element variable, si racchiude tra parentesi graffe tale variabile e il condition element positivo:

```
{<COUNTER>  
(COUNT ^VALUE <VALUE>)}
```

• La variabile <COUNTER> è legata al WME che si unifica con il condition element: (COUNT ^VALUE <VALUE>).

29

LHS: CONGIUNZIONI E DISGIUNZIONI

• Una **congiunzione (conjunction)**, delimitata da parentesi graffe, specifica condizioni multiple che devono essere tutte soddisfatte (è un AND logico). Ad es.:

```
(PERSON ^NAME <CHILD> ^AGE {> 0 < 5})  
(CHECK ^NUMBER {> 102 < 105 <NUMBER>})
```

• Una **disgiunzione (disjunction)**, delimitata da doppie parentesi angolari, specifica un insieme di valori uno dei quali deve corrispondere all'elemento di WM (è un XOR logico). Ad es.:

```
(CITY ^NAME <NEWENGLAND> ^STATE << CT MA ME NH RI VT >>)  
(CHECK ^NUMBER << 103 105 108 >>)
```

30

RHS: AZIONI

• La **RHS** di una regola di produzione consiste in una o più **azioni**.

• Le azioni possono svolgere le seguenti operazioni:

- Modificare la Working Memory
- Salvare e fare il restore della WM e del Conflict Set
- Fermare l'esecuzione del programma
- Legare variabili
- Manipolare files
- Scrivere su dispositivi di output
- Controllare loops
- Aggiungere nuove regole durante l'esecuzione di un pgm
- Chiamare routine esterne

31

RHS: AZIONI

• Una **azione** può essere specificata in questo modo:

```
(action-name argument-1 argument-2 .....
```

• Consideriamo questo esempio:

```
(MAKE CHECK ^NUMBER 102 ^AMOUNT 10.06 ^COUNTED NO  
^DATE 2 JAN 1985)
```

il nome dell'azione è **MAKE**, e ciò che segue sono i valori degli argomenti dell'azione. Tale azione inserisce in WM un WME della classe **CHECK** con i valori degli attributi specificati nell'istruzione.

32

RHS: AZIONI

• Lista (parziale) di **azioni** disponibili in OPS5:

- Make
- Modify
- Remove
- Write
- Halt
- Bind
- Cbind
- Openfile
- Closefile
- Call
- Accept e Acceptline
- Substr

33

ESEMPIO

```
(P FIND-CHECKS
 {<REPLY>
 (REPLY ^DATE {<DAY> <> STOP} <MONTH> <YEAR>)}
 {<CHECK>
 (CHECK ^NUMBER <NUMBER> ^AMOUNT<AMOUNT>
 ^COUNTED NO ^DATE <DAY> <MONTH> <YEAR>)}
 {<COUNTER>
 (COUNT ^VALUE <VALUE>)}
 →
 (WRITE (CRLF) (CRLF) | Found check number | <NUMBER>
 | for $ | <AMOUNT>
 | dated | (SUBSTR <REPLY> DATE INF))
 (MODIFY <CHECK> ^COUNTED YES)
 (MODIFY <COUNTER> ^VALUE (COMPUTE 1 + <VALUE> )))
```

34

RECOGNIZE-ACT CYCLE

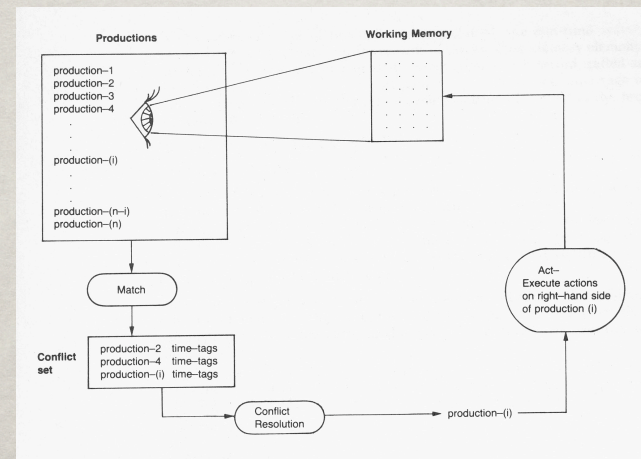
• In OPS5 il sistema run-time usa il “**recognize-act cycle**”, illustrato nella figura che segue, per eseguire i programmi.

• Il ciclo consiste nei passi seguenti:

1. MATCH
2. CONFLICT RESOLUTION
3. ACT
4. Go to step 1

35

RECOGNIZE-ACT CYCLE



36

FASE DI MATCH

- Durante la fase di **match**, il sistema run-time confronta gli elementi in WM con ciascun condition element presente nella LHS di ciascuna regola di produzione.
- Una LHS di una produzione è soddisfatta quando ci sono WME che soddisfano condition element positivi e non ci sono WME che soddisfano condition element negativi.

37

FASE DI MATCH: ESEMPIO

```
(P COUNTED-CHECKS
 {<REPLY>
 (REPLY ^DATE {<DAY> <> STOP} <MONTH> <YEAR>)}
 - (CHECK ^DATE <DAY> <MONTH> <YEAR> ^COUNTED NO)
 {<COUNTER>
 (COUNT ^VALUE <VALUE>)})
→
(REMOVE <REPLY>)
(REMOVE <COUNTER>)
(MAKE START)
(WRITE (CRLF) (CRLF) |There are | <VALUE> | checks dated |
<DAY> <MONTH> <YEAR> (CRLF))
```

38

FASE DI MATCH

- Se ci sono LHS di regole che sono soddisfatte da WME, il sistema crea un **conflict set**.
- Il conflict set contiene le **istanze di regole (instantiation)** che sono soddisfatte.
- Per istanza di regola si intende una regola di produzione associata alla lista dei WME che soddisfano la sua LHS.
- Un esempio di instantiation è: `FIND-CHECKS 12 3 11`
`FIND-CHECKS` è il nome della regola. Gli interi `12`, `3` e `11` sono i **time tag** dei 3 WME che soddisfano i 3 condition element presenti nella LHS della regola.

39

FASE DI MATCH

- Spesso accade che più di un insieme di WME soddisfino la LHS di una stessa regola di produzione.
- In tal caso il conflict set contiene più di una instantiation della stessa regola.
- Ad esempio, il conflict set potrebbe contenere:

```
FIND-CHECKS 12 3 11
FIND-CHECKS 12 4 11
FIND-CHECKS 12 5 11
FIND-CHECKS 12 6 11
FIND-CHECKS 12 2 11
```

40

FASE DI CONFLICT RESOLUTION

- In questa fase il sistema usa una **strategia di risoluzione dei conflitti** per selezionare l'istanza di regola migliore da applicare. Se il conflict set è vuoto, il programma si ferma.
- Le strategie disponibili sono basate sulle seguenti regole:
 - REFRACTION
 - RECENCY
 - SPECIFICITY

41

REFRACTION

- La regola di **refraction** impone di selezionare ed eseguire una **istanza di regola** solo una volta.
- Tale regola previene la possibilità di entrare in un loop infinito sugli stessi dati.
- Due **istanze di regole** sono uguali se:
 - contengono lo stesso nome della produzione
 - contengono gli stessi time tag

42

RECENCY

- La regola di **recency** comporta che l'ordine in base al quale il sistema seleziona le istanze di regole privilegia le istanze i cui WME sono stati inseriti più di recente in WM, ossia quelli con i time tag maggiori.
- Ad esempio, se il conflict set contiene le seguenti istanze:
 - FIND-CHECKS 12 3 11
 - FIND-CHECKS 12 4 11il sistema confronta i time tag maggiori. Essendo uguali (12 per entrambe le istanze), passa a confrontare i successivi. Poiché anche questi sono uguali (11 per entrambe), confronta i rimanenti. In questo caso è scelta la seconda istanza poiché il suo ultimo time tag (4) è maggiore dell'altro (3).

43

SPECIFICITY

- In base a questa regola, l'istanza è scelta in base alla sua specificità, che in genere viene determinata in base al numero dei test condizionali (**conditional test**) della LHS della regola relativa. Maggiore è il numero di test più specifica è la regola.
- Un conditional test è relativo a:
 - un class name
 - una disgiunzione
 - un valore costante preceduto da un predicato (con esclusione del caso della disgiunzione)
 - una occorrenza di una variabile (con esclusione della prima)

44

SPECIFICITY: ESEMPIO

La LHS della regola che segue ha 8 conditional test:

```
(P COUNTED-CHECKS
 {<REPLY>
 (REPLY ^DATE {<DAY> <> STOP} <MONTH> <YEAR>}}
 - (CHECK ^DATE <DAY> <MONTH> <YEAR> ^COUNTED NO)
 {<COUNTER>
 (COUNT ^VALUE <VALUE>}}
→
 (REMOVE <REPLY>)
 (REMOVE <COUNTER>)
 (MAKE START)
 (WRITE (CRLF) (CRLF) | There are | <VALUE> | checks dated | <DAY>
 <MONTH> <YEAR> (CRLF))
```

45

SPECIFICITY: ESEMPIO

- Il primo condition element contiene due test, relativi a: **REPLY** e **<> STOP**.
- Le variabili **<DAY>**, **<MONTH>** e **<YEAR>** non sono test, poiché esse appaiono per la prima volta nella regola e, come sappiamo, vengono legate a degli atomi di un certo WME.
- Il secondo condition element contiene 5 test: **CHECK**, **<DAY>**, **<MONTH>**, **<YEAR>** e **NO**. In questo caso le variabili sono contate come test perché sono già state legate a degli atomi.
- Il terzo condition element contiene un test: **COUNT**. La variabile **<VALUE>** non è contata come test poiché, apparendo per la prima volta, è legata ad un atomo.

46

STRATEGIE DI RISOLUZIONE DEI CONFLITTI

- L'interprete **OPS5** supporta due **strategie di risoluzione dei conflitti**:
 - **Lexicographic-Sort (LEX)**
 - **Means-Ends-Analysis (MEA)**

47

STRATEGIE DI RISOLUZIONE DEI CONFLITTI

- Entrambe le strategie applicano le regole nel seguente ordine: **REFRACTION**, **RECENCY**, **SPECIFICITY**.
- Tuttavia, la strategia **MEA** include un extra step dopo la **REFRACTION**, che aiuta ad organizzare programmi di grandi dimensioni.
- La strategia di default è la **LEX**. E' possibile cambiare strategia mediante il comando **STRATEGY**. Ad es:

```
(STRATEGY MEA)
```

48

STRATEGIA LEX

La strategia **LEX** usa le seguenti regole in sequenza per ordinare le istanze nel conflict set:

1. Si applica il criterio di **REFRACTION** rimuovendo dal conflict set le istanze che l'interprete ha selezionato nel ciclo precedente.
2. Si ordinano le istanze che rimangono in base alla loro **RECENCY**, e si seleziona quella che ha il valore più alto della **RECENCY**.
3. Se più di una istanza ha lo stesso valore più alto della **RECENCY**, si ordinano tali istanze in base alla **SPECIFICITY**, e si seleziona quella con il valore più alto.
4. Se più di una istanza ha lo stesso valore più alto della **SPECIFICITY**, si sceglie arbitrariamente l'istanza da applicare.

49

STRATEGIA LEX: ESEMPIO

- Supponiamo che la regola **FIND-CHECKS** contenga 10 conditional test, che la regola **COUNTED-CHECKS** ne contenga 8 e che il conflict set contenga le seguenti istanze di regole:

```
FIND-CHECKS 3 6 20
COUNTED-CHECKS 20 3 6
```

Dopo aver applicato il criterio di **REFRACTION**, la strategia valuta le istanze in base alla **RECENCY**.

- Poiché entrambe le istanze contengono gli stessi time tag (anche se in ordine diverso) le istanze hanno la stessa **RECENCY**.
- La istanza che contiene **FIND-CHECKS** è però più specifica (10 test contro 8) e viene quindi scelta.

50

STRATEGIA MEA

- La strategia **MEA** dà la massima priorità alle regole che hanno il primo condition element che si unifica con il WME più recente.
- Tale strategia si usa quindi se mettiamo in prima posizione nella LHS il più importante condition element.
- Questo extra step valuta la **RECENCY** dei time tag per i WME che soddisfano tali condition elements.
- Pertanto, è conveniente utilizzare tal strategia per problemi che possiamo dividere in vari task.

51

STRATEGIA MEA

La strategia **MEA** usa le seguenti regole in sequenza per ordinare le istanze nel conflict set:

1. Si applica il criterio di **REFRACTION** rimuovendo dal conflict set le istanze che l'interprete ha selezionato nel ciclo precedente.
2. Confronta il primo time tag di ogni istanza e seleziona quella con il time tag più alto.
3. Si ordinano le istanze che rimangono in base alla loro **RECENCY** (usando tutti i time tag) e si seleziona quella che ha il valore più alto della **RECENCY**.
4. Se più di una istanza ha lo stesso valore più alto della **RECENCY**, si ordinano tali istanze in base alla **SPECIFICITY**, e si seleziona quella con il valore più alto.
5. Se più di una istanza ha lo stesso valore più alto della **SPECIFICITY**, si sceglie arbitrariamente l'istanza da applicare.

52

STRATEGIA MEA: ESEMPIO

- Riprendiamo l'esempio precedente, in cui si è supposto che il conflict set contenga le seguenti istanze di regole:

```
FIND-CHECKS 3 6 20
COUNTED-CHECKS 20 3 6
```

Dopo aver applicato il criterio di **REFRACTION**, la strategia valuta le istanze in base alla **RECENCY** del primo time tag in ciascuna istanza.

- Poiché il primo time tag della prima istanza è 3 e quello della seconda istanza è 20, l'istanza relativa alla regola **COUNTED-CHECKS** ha il valore più alto, e viene quindi scelta.

53

ESEMPIO DI PROGRAMMA OPS5

```
(VECTOR-ATTRIBUTE DATE)
(LITERALIZE CHECK
  NUMBER AMOUNT COUNTED DATE)
(LITERALIZE COUNT
  VALUE)
(LITERALIZE REPLY
  DATE)
(STARTUP
  (MAKE CHECK ^NUMBER 102 ^AMOUNT 10.06 ^COUNTED NO ^DATE 2 JAN 1985 )
  (MAKE CHECK ^NUMBER 103 ^AMOUNT 22.45 ^COUNTED NO ^DATE 14 JAN 1985 )
  (MAKE CHECK ^NUMBER 104 ^AMOUNT 56.00 ^COUNTED NO ^DATE 14 JAN 1985 )
  (MAKE CHECK ^NUMBER 105 ^AMOUNT 27.25 ^COUNTED NO ^DATE 14 JAN 1985 )
  (MAKE CHECK ^NUMBER 106 ^AMOUNT 250.00 ^COUNTED NO ^DATE 14 JAN 1985 )
  (MAKE CHECK ^NUMBER 107 ^AMOUNT 16.15 ^COUNTED NO ^DATE 14 JAN 1985 )
  (MAKE CHECK ^NUMBER 108 ^AMOUNT 13.10 ^COUNTED NO ^DATE 25 JAN 1985 )
  (MAKE CHECK ^NUMBER 101 ^AMOUNT 40.30 ^COUNTED NO ^DATE 2 JAN 1985 )
  (MAKE CHECK ^NUMBER 109 ^AMOUNT 45.80 ^COUNTED NO ^DATE 30 JAN 1985 )
  (MAKE START)
  (STRATEGY MEA))
```

54

ESEMPIO DI PROGRAMMA OPS5

```
(P WHAT-DATE
  {<START>
  (START) }
  →
  (REMOVE <START>)
  (WRITE (CRLF) (CRLF) |What date do you want to search for?|)
  (WRITE (CRLF) (CRLF) |Enter the day, the first three|
    |letters of the month, and the year.|
    (CRLF)
    |For example - 4 JAN 1985 |
    (CRLF) (CRLF)
    |Type STOP to halt the program. |
    (CRLF) (CRLF)
    |Date >>>:|
  (MAKE COUNT ^VALUE 0)
  (MAKE REPLY ^DATE (ACCEPTLINE)))
```

55

ESEMPIO DI PROGRAMMA OPS5

```
(P FIND-CHECKS
  {<REPLY>
  (REPLY ^DATE {<DAY> <> STOP} <MONTH> <YEAR>)}
  {<CHECK>
  (CHECK ^NUMBER <NUMBER> ^AMOUNT<AMOUNT>
    ^COUNTED NO ^DATE <DAY> <MONTH> <YEAR>)}
  {<COUNTER>
  (COUNT ^VALUE <VALUE>)}
  →
  (WRITE (CRLF) (CRLF) | Found check number| <NUMBER>
    |for $| <AMOUNT>
    |dated| (SUBSTR <REPLY> DATE INF))
  (MODIFY <CHECK> ^COUNTED YES)
  (MODIFY <COUNTER> ^VALUE (COMPUTE 1 + <VALUE> )))
```

56

ESEMPIO DI PROGRAMMA OPS5

```
(P COUNTED-CHECKS
  {<REPLY>
  (REPLY ^DATE {<DAY> <> STOP} <MONTH> <YEAR>)}
  - (CHECK ^DATE <DAY> <MONTH> <YEAR> ^COUNTED NO)
  {<COUNTER>
  (COUNT ^VALUE <VALUE>)}
  →
  (REMOVE <REPLY>)
  (REMOVE <COUNTER>)
  (MAKE START)
  (WRITE (CRLF) (CRLF) |There are | <VALUE> |checks dated | <DAY> <MONTH>
  <YEAR> (CRLF)))
```

57

ESEMPIO DI PROGRAMMA OPS5

```
(P STOP-COUNT
  {<REPLY>
  (REPLY ^DATE STOP)}
  →
  (REMOVE <REPLY>)
  (HALT))
```

58