Лекція 15.

CLIPS.

Правила (закінчення).

6.5.9. Автоматичне додавання і перегрупування умовних елементів

У деяких ситуаціях CLIPS автоматично додає додаткові зразки до лівої частини правил (звичайно для поліпшення алгоритму зіставлення зразків, використовуваного системою CLIPS). Існує два зразки, застосовуваних CLIPS за замовчуванням: зразок факту initial-object і зразок об'єкта initial-object.

Нижче приводиться визначення цих даних:

Визначення 9.24. Синтаксис визначеного факту й об'єкта

```
(initial-fact)
(object (is-a INITIAL-OBJECT) (name [initial-object]))
```

Безумовні правила

Якщо правило не містить умовних елементів у своїй лівій частині, то до передумов правила автоматично додається зразок initial-fact (конфігурацію CLIPS можна настроїти таким чином, щоб замість зразка факту додавався зразок об'єкта initial-object). Наприклад, правило, що випливає із приклада 9.46, буде перетворено так, як показано в прикладі 9.47:

Приклад 9.46. Правило без умов

```
(defrule  example
          => )
```

Приклад 9.47. Перетворене правило без умов

```
(defrule  example
          (initial-fact)
           =>)
```

Використання елементів *test* та *not* перед *and*

Умовні елементи test й not, що коштують перед and, додають зразок initial-fact або initial-object безпосередньо перед собою. Зразок initial-fact додається, якщо в першому умовному елементі використовується зразок факту. Зразок initial-object додається, якщо в першому умовному елементі використовується зразок об'єкта. Якщо в першому умовному елементі немає зразків, то тип зразка, що додає, визначається по наступному умовному елементі таким же методом. Якщо у всьому поточному умовному вираженні немає зразків, то система використає визначений факт initial-fact (хоча конфігурацію CLIPS можна настроїти таким чином, щоб замість зразка initial-fact додавався зразок initial-object).

Наприклад правила, що випливають із приклада 9.48, будуть змінені так, як у прикладі 9.49.

Приклад 9.48. Правила з умовами test й not перед and

```
(defrule       example1
          (test   (>  80   (startup-value)))
          =>
(defrule  example2
          (test (> 80 (startup-value)))
          (object (is-a MACHINE))
          =>)
(defrule  example3
          (machine ?x)
          (not (and (not (part ?x ?y))
          (inventoried ?x)))
          =>
```

Приклад 9.49. Перетворені правила з умовами test й not перед and

```
(defrule  example1
          (initial-fact)
          (test (> 80 (startup-value)))
          =>)
(defrule  example2
          (object (is-a INITIAL-OBJECT) (name [initial-
          object]))
          (test (> 80 (startup-value)))
          (object (is-a MACHINE))
          =>)
(defrule  example3
          (machine ?x)
          (not (and (initial-fact)
          (not (part ?x ?y))
          (inventoried ?x)))
          =>)
```

Використання елемента *not* перед *test*

Якщо відразу перед умовним елементом test використався умовний елемент not, то CLIPS автоматично переміщає умовний елемент not на місце першої умови безпосередньо наступного за test.

Наприклад, правило із приклада 9.50 зміниться на еквівалентне (приклад 9.51):

Приклад 9.50. Правило з умовами not перед елементом test

```
(defrule       example
               (a   ?x)
               (not   (b   ?x))
               (test   (>  ?x  5)}
               =>)
```

Приклад 9.51. Перетворене правило з умовами not перед test

```
(defrule       example
               (a  ?x)
               (test   (>   ?x  5))
               (not   (b  ?x) )
                =>)
```

Використання елемента *not* перед or

Якщо відразу перед умовним елементом or використався умовний елемент not, то CLIPS автоматично заміняє комбінацію not/or на еквівалентну комбінацію and/not.

Наприклад, що випливає правило (приклад 9.52) будуть змінено так, як показано в прикладі 9.53.

Приклад 9.52. Правило з умовами not перед or

```
(defrule  example
          (a ?x)
          (not (or (b ?x)
          (c ?x)))
          =>)
```

Приклад 9.53. Перетворене правило з умовами not перед or

```
(defrule       example
               (a  ?x)
               (and   (not   (b  ?x))
                         (not   (c  ?x)))
              =>)
```

Зауваження про автоматичне додавання й перегрупування умовних елементів

У завершення опису синтаксису лівої частини правил CLIPS оборотна увага на наступні важливі особливості:

1. Повна версія лівої частини правила містить неявний умовний елемент and.

2. Перетворення умовних елементів forall й exists до еквівалентних виражень за допомогою not й and виконується перед додаванням відповідних зразків у ліву частину правила.
3. Умовний елемент test звичайно не використовується як перший елемент в умові and.
4. Команди, що виводять інформацію про умовні елементи в лівій частині правила, відображають інформацію про визначення правила у вигляді, у якому неї задав користувач. Інформація про перегрупування й додавання зразків initial-fact й initial-object не виводиться.

9.6. Команди й функції для роботи із правилами

Після того як ми повністю розібралися з поданням правил в CLIPS, розглянули внутрішні алгоритми обробки правил, стратегії дозволу конфліктів і синтаксис лівої частини правил, можна сміло переходити до вивчення функцій і команд, надаваних CLIPS для роботи із правилами. Повна специфікація цих функцій буде дана в *розд. 15* й *16*, у даній лекції ми розглянемо лише основні з них із прикладами використання.

9.6.1. Перегляд і видалення існуючих правил

Після створення правил за допомогою конструктора defrule цілком природно виникає бажання зробити що-небудь із уже існуючим правилом. CLIPS підтримує множина різних команд, що оперують із правилами. У даному розділі ми розглянемо найбільше часто використовувані команди: ppdefrule,list-defrules й undefrule.

За допомогою команди ppdefrule можна переглянути визначення правила в тому вигляді, у якому воно було створено за допомогою конструктора defrule.

Визначення 9.25. Синтаксис команди ppdefrule

```
(ppdefrule  <ім'я-правила>)
```

Для того щоб одержати повний список правил, присутніх в CLIPS у цей момент, використовується команда list-def rules.

Визначення 9.26. Синтаксис команди list-defrules

```
(list-defrules <ім'яі-модуля>)
```

Повний синтаксис цієї команди містить необов'язковий аргумент <ім'я-модуля> (про поняття модуля буде розказане в *розд. 12).* Якщо даний аргумент не заданий, то буде виведений список правил, певних у поточному модулі. У випадку явного завдання модуля буде список правил, що належать конкретному модулю. Даний аргумент може приймати значення *. У цьому випадку на екран буде виведений список всіх правил із всіх модулів.

Для видалення правила використовується команда undefrule.

Визначення 9.27. Синтаксис команди undefrule

```
(undefrule  <ім'яі-правила>)
```

Як параметр команда undefruie приймає ім'я правила, яке потрібно видалити. Якщо як ім'я правила був заданий символ *, то будуть вилучені всі правила.

Для демонстрації роботи команд, наведених у цьому й наступному розділах, будемо використати наступні правила:

Приклад 9.54. Необхідні для подальшої роботи правила

```
(defrule Make
  (a)
  (b)
  =>
  (assert (c)))
(defrule Make
  (c)
       (or (a)
  (    b))
  =>
  (assert (d)))
(defrule Make
  (d)
       (or (a)
       (b)
  (c))
  =>
  (assert (e)))
```

Введемо ці правила в середовище CLIPS, а потім виконаєте наступну послідовність команд:

Використання команд ppdefrule, list-defrules й undefrule

```
(ppdefrule Make)
(list-defrules)
(undefrule Make)
(list-defrules)
(undefrule *)
(list-defrules)
```

Якщо наведені вище дії були виконані правильно, то отриманий результат повинен відповідати мал. 9.6.

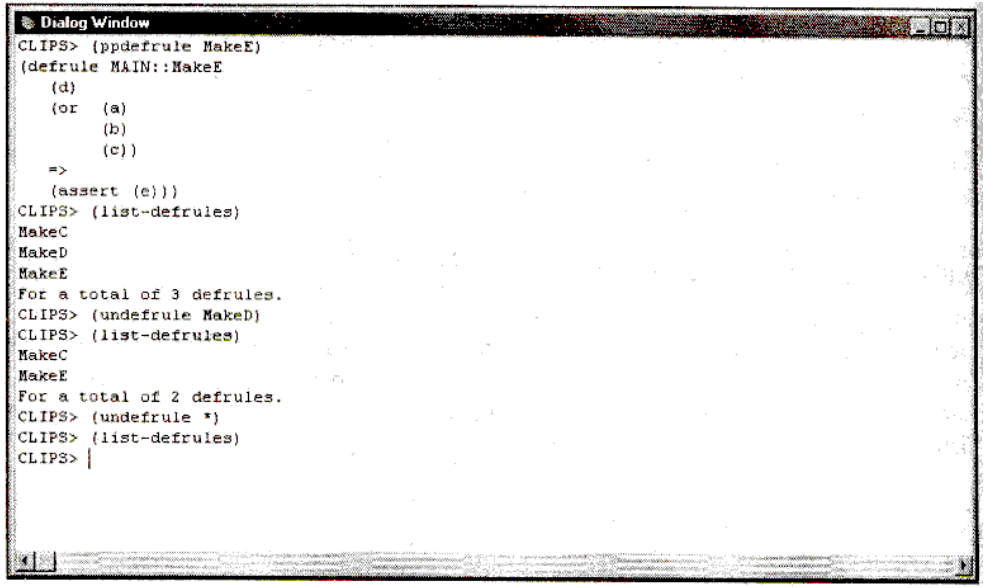

Рис. 9.6. Результат застосування команд ppdefrule, list-defrules й undefrule

Як уже згадувалося в лекції 8 користувачам Windows-версії CLIPS доступний інструмент за назвою **Defrule Manager** (Менеджер правил). Якщо в цей момент у середовищі CLIPS відсутні правила, то пункт **Defrule Manager** меню **Browse** не буде доступний. Якщо ви повторно заведете наведені вище правила й відкриєте менеджер правил, то повинні будете побачити результат, наведений на мал. 9.7. Менеджер відображає список всіх правил, доступних у цей момент. Загальна кількість правил відображається в заголовку вікна менеджера, у цей момент це **Defrule Manager — 3 Items.** За допомогою кнопок **Remove й Pprint** можна видаляти й виводити визначення обраного правила відповідно. Вся інформація, одержувана від менеджера правил, відображається безпосередньо в головному вікні CLIPS.

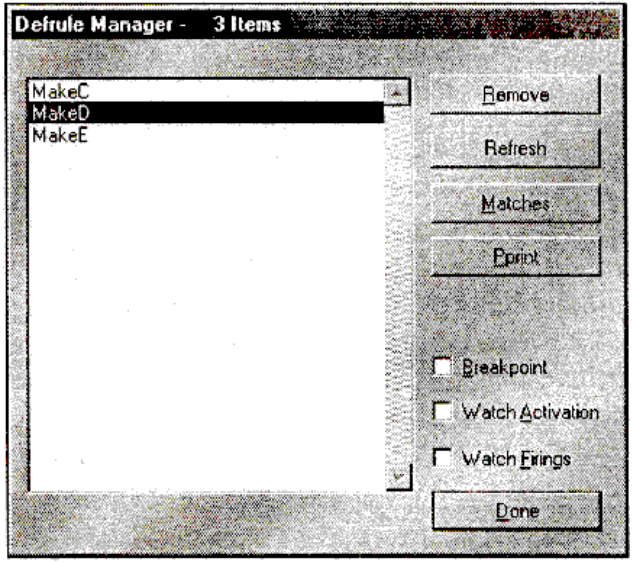

Рис. 9.7. Перегляд списку правил за допомогою менеджера правил

CLIPS не містить спеціальних команд для зміни існуючих правил. Щоб змінити існуюче правило, користувачеві необхідно заново визначити таке правило за допомогою конструктора def rule. При цьому існуюче визначення правила буде автоматично вилучено із системи, навіть якщо новий конструктор містив помилки, і нове правило додане не було.

9.6.2. Збереження правил

Як ви вже встигли переконатися, створювати правила конструктором defrule щораз, у міру необхідності використовуючи для цього середовище CLIPS, досить незручно. Для полегшення участи користувача CLIPS дозволяє завантажувати конструктори правил (як, втім, і всі інші конструктори) з текстового файлу. Для цього використовується наступна команда:

Визначення 9.28. Синтаксис команди load

```
(load <ім'я-файлу>)
```

Ім'я файлу повинне бути рядком, тобто полягати в лапки. Ім'я файлу може містити повний шлях до файлу. У противному випадку система буде шукати файл у поточному каталозі. Для створення файлу в принципі можна використати будь-який ASCII-редактор, але краще застосовувати убудований редактор, надаваний середовищем CLIPS. Убудований редактор підтримує кілька додаткових функцій, надзвичайно корисних при розробці програм. По-перше, він здатний перевіряти синтаксис функцій, баланс відкриваючих і закриваючих дужок, допомагає в розміщенні й видаленні коментарів і т.д. Якщо ви будете використати убудований редактор для створення серйозної експертної системи, ви по достоїнству оціните ці можливості. По-друге, убудований редактор дозволяє швидко завантажувати в середовище окремі конструктори й команди. Ця можливість допомагає перевіряти й тестувати велику експертну систему. І, нарешті, по-третє, редактор надає допомогу по середовищу й мові, що буває надзвичайно корисної, навіть при наявності великого досвіду роботи в CLIPS. За замовчуванням файли, створені в убудованому редакторі CLIPS, одержують розширення clp. Для початку роботи з редактором просто виберіть пункт New меню **File.**

Створимо в CLIPS файл example1.CLP із трьома наведеними вище правилами. Після чого очистите CLIPS за допомогою команди clear і виконаєте команду (load "example1.CLP"). Отриманий результат повинен відповідати мал. 9.8.

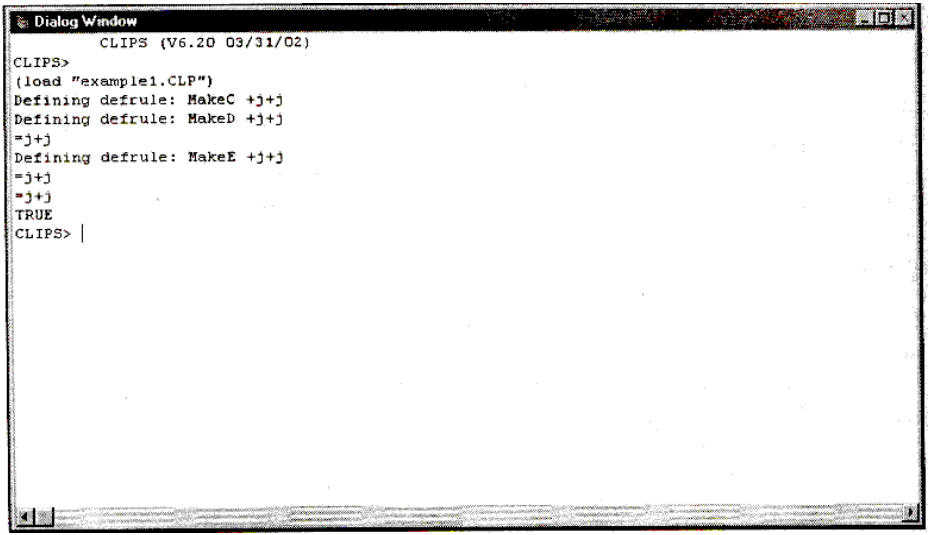

Рис. 9.8. Результат завантаження файлу example1.CLP

Команда load відображає процес завантаження кожного конструктора. У випадку успішного завантаження всіх певних у файлі конструкторів команда повертає

значення TRUE, у противному випадку — інформацію про помилку. У випадку якщо була знайдена помилка, процес завантаження файлу припиняється.

CLIPS підтримує також команду load*. Ця команда повністю ідентична load за винятком того, що вона не відображає процесу завантаження конструкторів.

Визначення 9.29. Синтаксис команди load*

```
(load*   <ім'я-файлу>)
```

CLIPS надає також команду save, що дозволяє зберігати в текстовий файл всі конструктори, певні в цей момент у системі. Синтаксис цієї команди ідентичний синтаксису команд load й load*.

Визначення 9.30. Синтаксис команди save

```
(save  <ім'я-файлу>)
```

Текстовий формат не єдиний спосіб зберігання конструкторів CLIPS. Команди bsave й bload дозволяють зберігати й завантажувати конструктори у двійковому виді. Двійкові файли завантажуються набагато швидше, ніж текстові, але

займають більше місця (тому що крім конструкторів вони зберігають повну інформацію про поточний стан середовища). Ще однією незручністю використання двійкових файлів є те, що створювати їх можна тільки безпосередньо в середовищі CLIPS.

Більшість описаних вище команд для роботи з файлами (а саме load, save, bsave й bload) доступні в меню **File** Windows-версії середовища CLIPS. Це команди **Load, Save, Save Binary й Load Binary** відповідно. Усі використають стандартні Windows-діалоги для вибору файлів.

9.6.3. Запуск і зупинка програми

Як було замічено, для запуску CLIPS-програм використовується команда run.

Визначення 9.31. Синтаксис команди run

```
(run <вцілочисельний-вираз>)
```

Цілочисельне вираження є необов'язковим аргументом команди run. У найпростішому випадку як цей аргумент можна використати будь-яку целую константу. Якщо даний аргумент заданий і він позитивний, то CLIPS запустить на виконання задане число правил із плану рішення задачі. Якщо дане число більше числа правил у плані рішення задачі, то буде запущені всі правила. У випадку якщо аргумент не заданий або є негативним, план рішення задачі також буде виконаний повністю.

В Windows-версії CLIPS у меню **Execution** доступні дві версії команди run — **Run** й **Step.** Перша команда використає версію команди run без аргументів і запускає всі правила із плану рішення задачі. Програма Step дозволяє трасувати програму й виконувати задане число правил. За замовчуванням це число дорівнює 1, але цю установку середовища можна змінити за допомогою діалогового вікна **Preferences.** Для запуску цього діалогового вікна виберіть пункт **Preferences** меню **Execution.** Загальний вид діалогового вікна показаний на мал. 9.9. Кількість правил, запущених за один крок трасування, відображається в поле **Step Rule Firing Increment.**

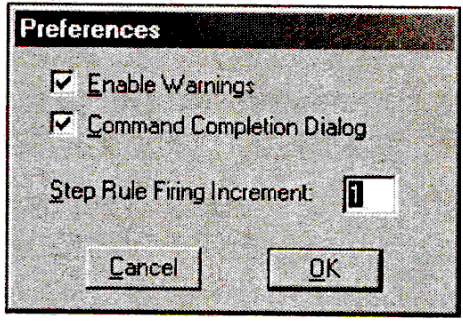

Рис. 9.9. Діалогове вікно **Preferences**

Крім виконання програми по кроках, CLIPS дозволяє установку точок зупину (breakpoints) на окремих правилах.

Визначення 9.32. Синтаксис команди set-break

```
(set-break <ім'я-правила>)
```

Якщо крапка останова визначена для заданого правила, то виконання програми припиниться перед запуском цього правила. Крапка останова не зупиняє правило, якщо це перше правило в плані рішення задачі.

Видалити крапки останова можна за допомогою команди remove-break.

Визначення 9.33. Синтаксис команди remove-break

```
(remove-break <ім'я-правила>)
```

У випадку виконання команди remove-break без параметрів CLIPS видалить всі певні раніше крапки останова.

Установлювати й знімати крапки останова також можна за допомогою менеджера правил, зовнішній вигляд якого представлений на мал. 9.7. Для цього виберіть правило й установите прапорець **Breakpoint.**

У випадку якщо виконання програми необхідно зупинити, використайте команду halt без аргументів.

Визначення 9.34. Синтаксис команди halt

```
(halt)
```

Діалогове вікно **Watch Options** (пункт **Watch** меню **Execution)** дозволяє встановити прапорець **Statistics,** як показано на мал. 9.10.

У цьому випадку після виконання кожної команди run CLIPS буде виводити статистичну інформацію про кількість запущених правил, повному й середньому часі виконання правил, кількості доданих фактів і т.д.

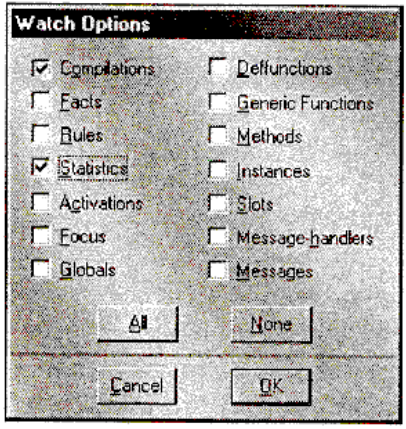

Рис. 9.10. Установка режиму з висновком статистичної інформації

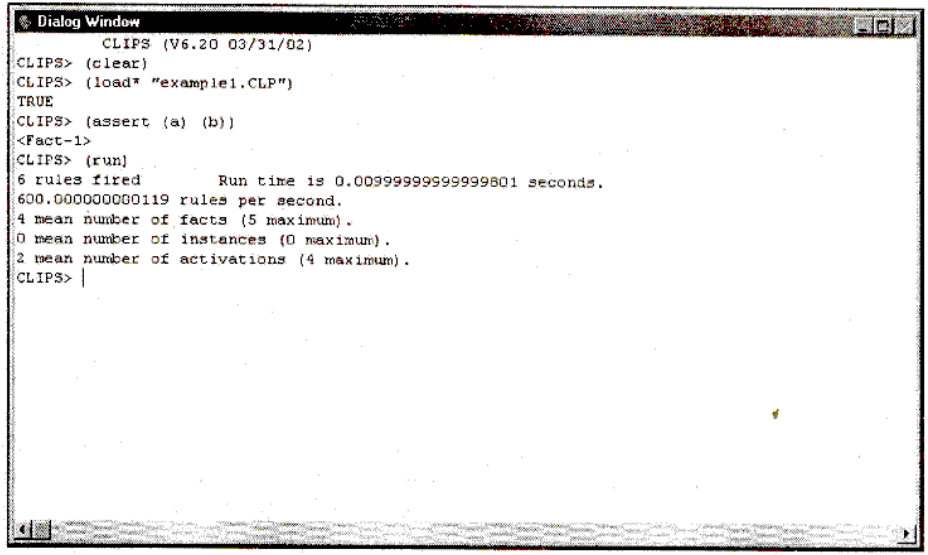

Рис. 9.11. Одержання статистичної інформації

Якщо ви встановите прапорець **Statistics,** завантажимо файл example 1.CLP, додамо факти (а) і (Ь) і запустимопрограму, то побачимо результати, представлені на мал. 9.11.

9.6.4. Перегляд плану рішення задачі

План рішення задачі (agenda) можна переглядати різними способами. Найпростіший з них - команда agenda, набрана в головному вікні CLIPS. Очистите CLIPS, завантажите файл example1.CLP, додайте факти a, b, c и d і викличте команду agenda. Отриманий результат повинен відповідати наведеному на мал. 9.12.

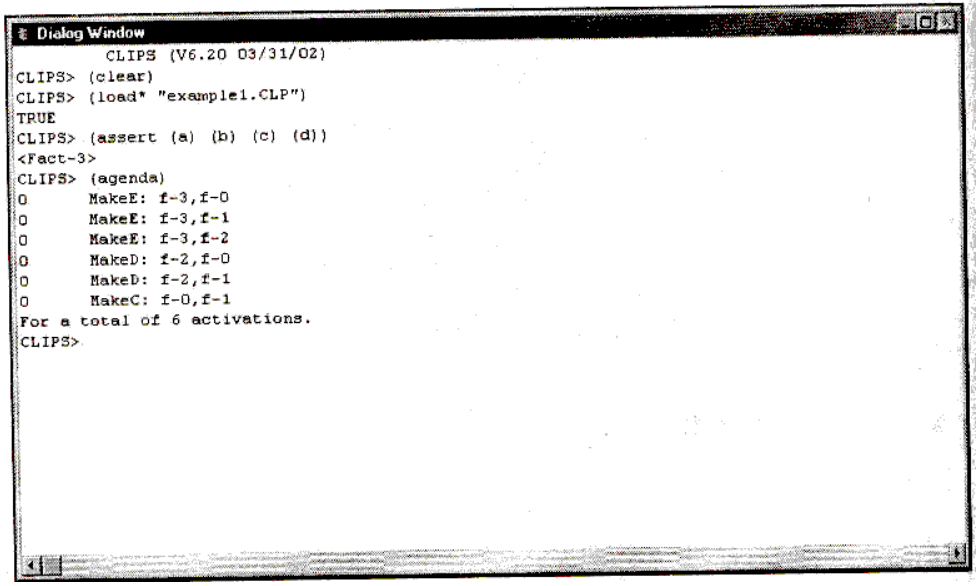

Рис. 9.12. Перегляд плану рішення задачі

План рішення задачі містить 6 активацій правил. По команді agenda всі ці активації будуть виведені на екран разом із пріоритетом правил (ліворуч від імені правила) і списком даних, що активували правило (праворуч від імені правила). Порядок правил у плані рішення задачі сильно залежить від обраної стратегії дозволу конфліктів і пріоритету правил.

Крім цього, Windows-версія CLIPS дозволяє виводити план рішення задачі в окремому вікні — **Agenda.** Для того щоб зробити вікно видимим, скористайтеся пунктом **Agenda Window** меню **Window.** Зовнішній вигляд цього вікна показаний на мал. 9.13, його вміст повністю відповідає інформації, одержуваної за допомогою команди agenda. Даний інструмент надзвичайно корисний при налагодженні програм або для спостереження за зміною плану рішення задачі в процесі виконання програми.

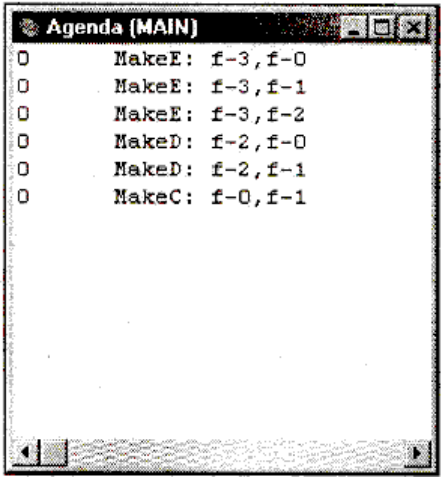

Рис. 9.13. Вікно **Agenda**

Крім вікна **Agenda,** що дозволяє тільки перегляд, CLIPS надає ще один зручний візуальний інструмент — **Agenda Manager** (Менеджер плану рішення задачі), що дозволяє якщо буде потреба коректувати план рішення задачі. Для виклику менеджера плану рішення задачі виберіть пункт **Agenda Manager** меню **Browse.** Зовнішній вигляд цього інструмента наведений на мал. 9.14. З його допомогою можна видаляти із плану рішення задачі окремі активації правил або запускати правила в деякому довільному порядку.

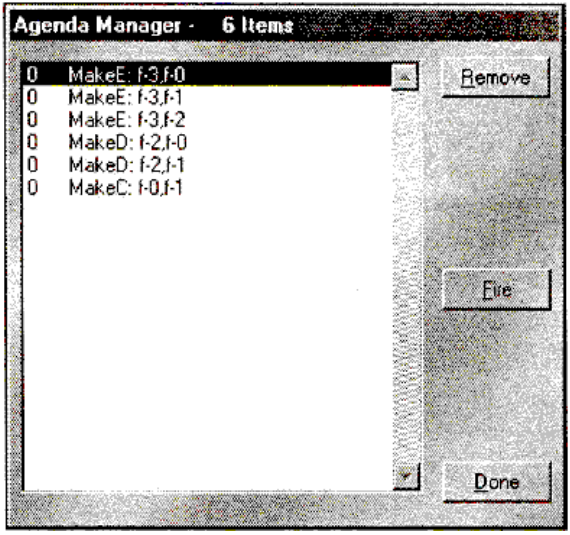

Рис. 9.14. Вікно менеджера плану рішення задачі

За допомогою діалогового вікна **Watch Options** (див. мал. 9.10) або менеджера правил можна задавати режим відображення активацій й/або запуску правил. У цьому випадку користувач буде одержувати відповідне інформаційне повідомлення при додаванні правила в план рішення задачі або при видаленні правила з нього, а також при кожному запуску правила.

9.6.5. Перегляд даних, здатних активувати правило

CLIPS надає можливість переглядати списки наборів даних (фактів або об'єктів), здатних активувати задане правило.

Визначення 9.35. Синтаксис команди matches

```
(matches <ім'я-правила>)
```

Команда matches виводить інформацію про всі можливі набори даних, здатних активувати це правило. Подивитеся на результати виконання даної команди для правил Make й Make (наявність фактів a, b і з обов'язково), наведені на мал. 9.15 й 9.16 відповідно.

На цьому ми закінчимо вивчення синтаксису правил CLIPS й основних команд і функцій для роботи з ними.

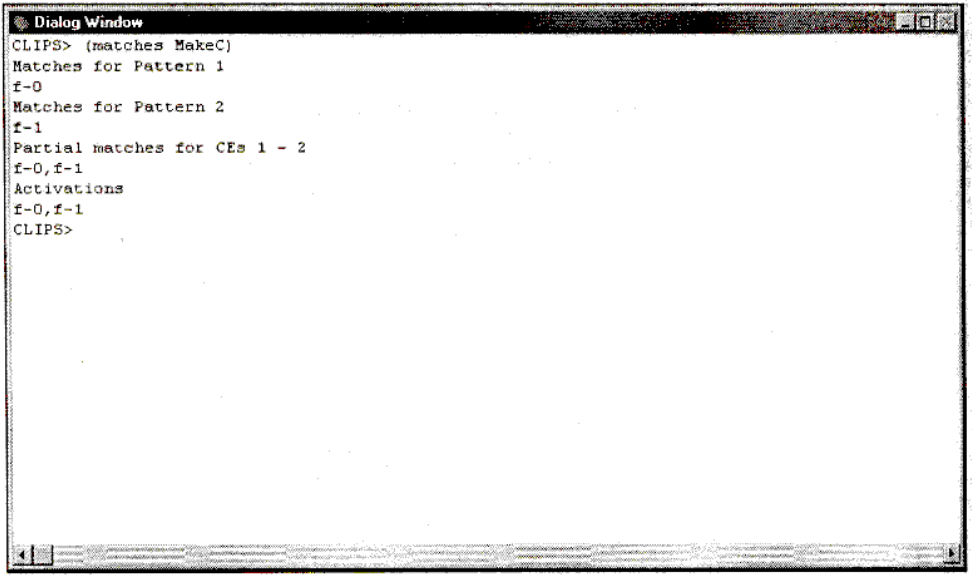

Рис. 9.15. Дані, що активують правило Make

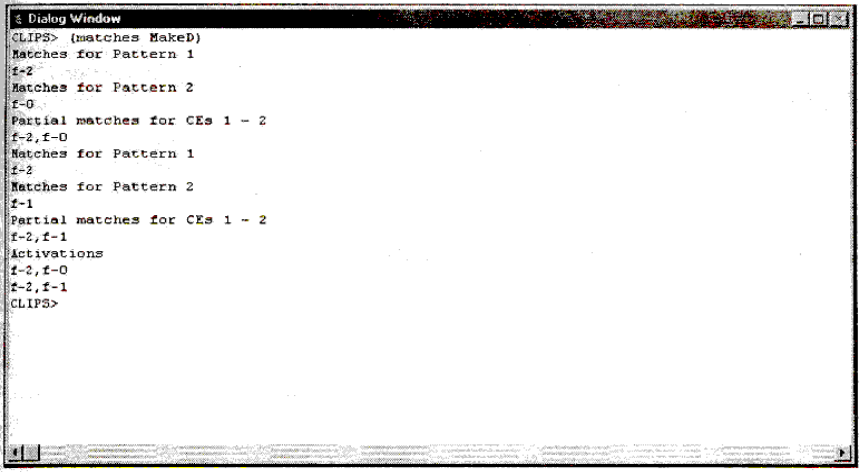
Dialog Window
CLIPS> (matches MakeD)
Matches for Pattern 1
f-2
Matches for Pattern 2
f-0
Partial matches for CEs 1 - 2
f-2,f-0
Matches for Pattern 1
f-2
Matches for Pattern 2
f-1
Partial matches for CEs 1 - 2
f-2,f-1
Activations
f-2,f-0
f-2,f-1
CLIPS>

Рис. 9.16.