UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Sistemes basats en regles[1]

Alfons Juan
Albert Sanchis
Jorge Civera

DSIC

Departament de Sistemes Informàtics i Computació

[1] Per a una correcta visualització, es requereix l'Acrobat Reader v. 7.0 o superior

Objectius

- Descriure les bases dels sistemes basats en regles (Rule-Based Systems, RBS).
- Usar CLIPS per al disseny, construcció i execució d'RBS.

Índex

1 Introducció

- 1984: el grup d'IA del *NASA's Johnson Space Center* decideix desenvolupar una eina C de construcció de sistemes experts
- 1985: es desenvolupa la versió prototip de ***C Language Integrated Production System (CLIPS)***, idònia per a formació
- 1986: CLIPS es comparteix amb grups externs
- 1987–2002: millores de rendiment i noves funcionalitats; per exemple, programació procedural, OO i interfícies gràfiques
- Des de 2008: Gary Riley manté CLIPS fora de la NASA [1, 2] (URL: `https://www.clipsrules.net/`)
- Documentació:
  - ***Manual de referència I: Guia de programació bàsica*** [3]
  - Manual de referència II: Guia de programació avançada [4]
  - Manual de referència III: Guia d'interfícies [5]
  - Guia de l'usuari [6]

2 Sistemes basats en regles amb CLIPS

CLIPS permet construir SBRs amb tres components:

1. ***Base de fets (BF):***
   - Cada estat del problema sol representar-se amb un únic fet d'acord amb un cert patró de ***fet-estat***
   - A cada pas d'execució, els fets-estat representen estats del problema ja explorats o pendents d'exploració
   - La resta de fets conté informació ***estàtica*** del problema
2. ***Base de regles (BR):***
   - Cada possible acció aplicable a un o més estats del problema sol representar-se amb una única regla `esquerra=>dreta`
   - La part esquerra tria el conjunt d'estats al qual és aplicable
   - La part dreta sol resultar en nous fets-estat afegits (o eliminats) a la BF
3. ***Motor d'inferència:*** instanciació, selecció i execució de regles

- ***Motor d'inferència:***
  - ***Entrada:*** base de fets i base de regles inicials, BF i BR
  - ***Eixida:*** base de fets final, BF
  - ***Mètode:***

$CC = \emptyset$ // conjunt conflicte d'instàncies de regles

repetir

// afegim noves instàncies al CC a partir de nous fets:
$CC =$ Instancia(BF, BR, CC)

si $CC = \emptyset$: **eixir** // objectiu no aconseguit

// seleccionem una instància amb algun criteri:
$InstRule =$ Selecciona(CC)

// executem $InstRule$ i actualitzem BF i CC:
$(BF, CC) =$ Executa$(BF, CC, InstRule)$

fins_a objectiu aconseguit

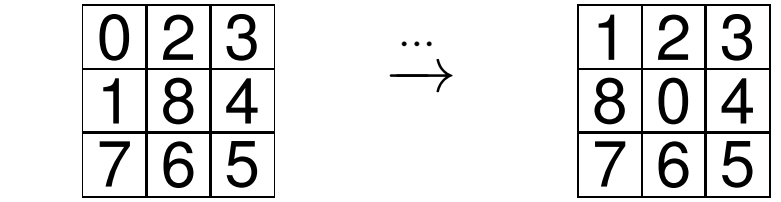
- ***Tres passos bàsics en inferència:***

1. ***Instancia:*** afegeix noves instàncies al CC a partir de nous fets, sense repetir instàncies afegides anteriorment ***(refracció)***
2. ***Selecciona:*** aplica un criteri de selecció com ara:
   - ***Profunditat:*** primer la instància més recent
   - ***Amplària:*** primer la instància més antiga
   - ***Prioritat:*** primer la instància de la regla més prioritària
3. ***Executa:*** aplica les ordres de la instància seleccionada:
   - ***Eliminació de fets*** en la BF
   - ***Eliminació d'instàncies*** en el CC amb fets eliminats
   - ***Inserció de fets*** nous en la BF ***sense repeticions***

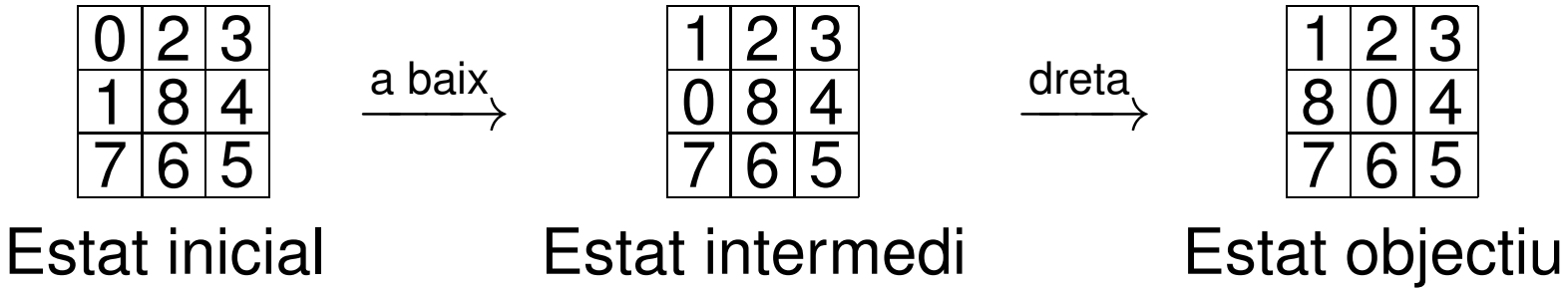
- ***No duplicitat de fets i refracció***:
  - Per defecte, els fets en CLIPS no es dupliquen en la BF
  - ***Refracció:*** Una regla només es pot instanciar una vegada amb el mateix fet i amb la mateixa instanciació de valors a variables
  - No duplicació i refracció prevenen de l'activació infinita de regles
  - La inserció d'un nou fet en la BF pot provocar l'instanciació de noves regles

3 El problema del 8-puzle en CLIPS

Donat un estat inicial, es vol arribar a un estat objectiu mitjançant moviments de la fitxa en blanc (fitxa 0): dreta, esquerra, a dalt i a baix



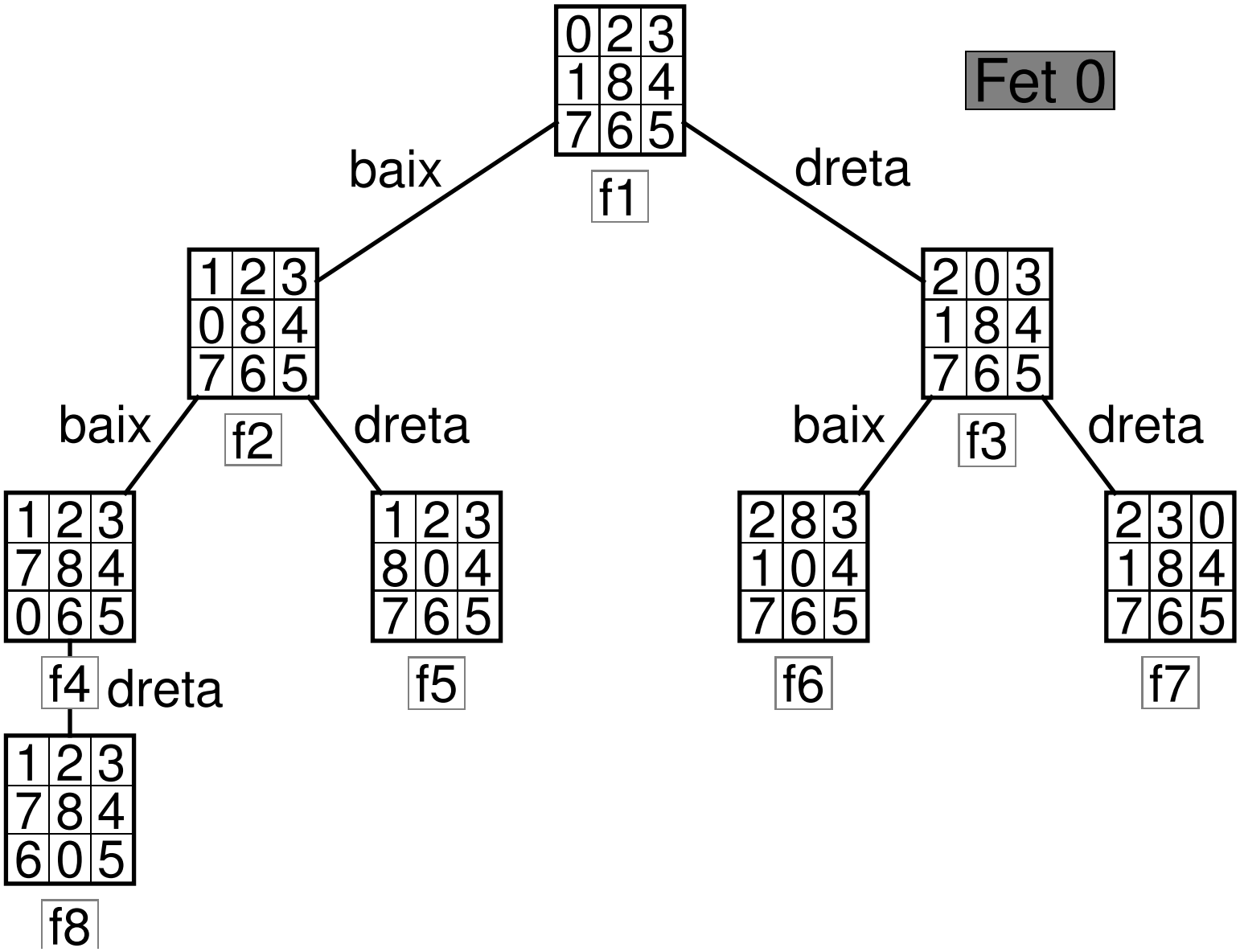
Solució per a l'estat inicial i objectiu anterior:



Un RBS senzill per al problema del 8-puzle

```
(deffacts bfini (puzzle 0 2 3 1 8 4 7 6 5))

(defrule esquerra
  (puzzle $?x ?y 0 $?z)
  (test (<> (length$ $?x) 2))
  (test (<> (length$ $?x) 5)) =>
  (assert (puzzle $?x 0 ?y $?z)))

(defrule dreta
  (puzzle $?x 0 ?y $?z)
  (test (<> (length$ $?x) 2))
  (test (<> (length$ $?x) 5)) =>
  (assert (puzzle $?x ?y 0 $?z)))

(defrule dalt
  (puzzle $?x ?a ?b ?c 0 $?y) =>
  (assert (puzzle $?x 0 ?b ?c ?a $?y)))

(defrule baix
  (puzzle $?x 0 ?a ?b ?c $?z) =>
  (assert (puzzle $?x ?c ?a ?b 0 $?z)))

(defrule objectiu
  (puzzle 1 2 3 8 0 4 7 6 5) =>
  (printout t "Solució trobada!" crlf)
  (halt))
```

Traça de BFS amb CLIPS per al 8-puzle



4 Fets

- Llista de símbols entre parèntesis on el primer indica la “relació” ***(puzzle 0 2 3 1 8 4 7 6 5)***
- Fets inicials: amb ***deffacts***

  `(deffacts <name> [<comment>] <fact>*)`

  on la sintaxi de cada ***fet*** és:

  `<fact> ::= (<symbol> <constant>*)`

- Exemple:

  `(deffacts bfini (puzzle 0 2 3 1 8 4 7 6 5))`

- Ordres sobre fets:
  - ***assert*** (per a inserir un nou fet en la BF)
  - ***retract*** (per a eliminar un fet de la BF)

5 Regles

- Consten de dos parts, la LHS i la RHS:
  - Antecedent o part esquerra (LHS):
    - ↦ Condicions a complir perquè s'execute la RHS
  - Conseqüent o part dreta (RHS):
    - ↦ Accions a executar si es compleix la LHS

- Sintaxi:

```
(defrule <name> <LHS> => <RHS>)
```

on la part esquerra ***(LHS)*** és:

```
LHS ::== <conditional-element>*
```

i la part dreta ***(RHS)***:

```
RHS ::= <action>*
```

- ***Activació i execució de regles***:
  - S'executa o dispara (*fire*) en funció de l'existència o no de fets amb els quals es complisquen les condicions
  - El motor d'inferència és l'encarregat d'encaixar (fer *matching* de) fets amb regles
    - ↦ Anomenem ***instàncies*** d'una regla als diferents matchings de fets amb la regla que es puguen fer (zero, un o més)
    - ↦ ***Agenda o conjunt conflicte:*** conjunt de instàncies de totes les regles pendents d'execució

- ***Cicle bàsic d'execució de regles***:
  - ***Motor d'inferència:*** bucle amb els següents passos bàsics

a) ***Selecció d'una instància (de regla) de l'agenda***

↦ Si no hi ha cap instància en l'agenda, s'acaba

b) ***Execució de la part dreta de la regla seleccionada***

c) ***Activació i desactivació d'instàncies de regles*** com a conseqüència de l'execució de la regla seleccionada

↦ Les activades s'afegeixen a l'agenda

↦ Les desactivades s'eliminen de l'agenda

d) ***Re-avaluació de prioritats dinàmiques d'instàncies en l'agenda:*** si utilitzem prioritats dinàmiques amb `salience`

- ***Estratègies de resolució de conflictes***
  - ***Ordenació d'instàncies de regles en l'agenda:***
    - ***Per prioritat:*** les noves instàncies se situen damunt (davant) de les de menor prioritat i baix (darrere) de les de major. `(salience <integer>)` defineix la prioritat de la regla
      - Prioritat mínima: -10000; màxima: 10000; per defecte: 0

```
(defrule <name>
   (declare (salience <integer>))
   <LHS> => <RHS>)
```

      - Exemple:

```
(defrule objectiu
   (declare (salience 1))
   (puzzle 1 2 3 8 0 4 7 6 5) =>
   (printout t "Solució trobada!" crlf)
   (halt))
```

    - ***Profunditat (*depth*)***: Les noves instàncies se situen damunt de totes les d'igual prioritat; és l'estratègia per omissió
    - ***Amplària (*breadth*)***: Les noves regles se situen baix de totes les de igual prioritat

Sintaxi de la LHS

- ***Elements condicionals (CEs):*** sèrie de zero, un o més elements de que consta la LHS d'una regla i que s'han de satisfer perquè s'afegisca una instància de la regla al conjunt conflicte (o agenda)
- Hi ha huit tipus de CEs, però només fem ús de cinc:
  - ***CEs patró:*** restriccions sobre els fets que el satisfan
  - ***CEs test:*** avaluen expressions durant l'encaix de patrons
  - ***CEs or:*** donat un grup de CEs, almenys un s'ha de satisfer
  - ***CEs and:*** donat un grup de CEs, tots s'han de satisfer
  - ***CEs not:*** donat un CE, *no* s'ha de satisfer

```
<conditional-element> ::=
  <pattern-CE> | <assigned-pattern-CE> |
  <test-CE> | <or-CE> | <and-CE> | <not-CE>
```

CE patrón

- ***<pattern-CE>:*** llista ordenada amb un símbol inicial, seguit de constants, comodins i variables

```
<pattern-CE> ::= (<symbol> <constraint>*)
<constraint> ::= <constant> | <variable> | ? | $?
<constant> ::= <symbol> | <string> | <integer> | <float>
<variable> ::=  <single-vble> | <multi-vble>
<single-vble> ::= ?<symbol>
<multi-vble> ::= $?<symbol>
```

- Exemple:

```
(puzzle $?x ?y 0 $?z)
```

  - ***$?x*** és una variable multi-avaluada que es pot instanciar a zero o més elements
  - ***?y*** és una variable mono-avaluada que es pot instanciar a exactament un element
  - ***0*** és una constant

- ***comodins ?*** i ***$?*** es comporten igual que les variables mono i multi-avaluades, respectivament, però els valors instanciats no s'emmagatzemen

Pattern matching

- Possibles “encaixos” (***matchings***) entre patrons i fets
  - un patró “encaixa” una vegada amb un fet

Fets	Patrons
f-1: ***(puzzle 1 2 3 0 7 4 8 6 5)***	***(puzzle $?x 0 $?y 7 $?z)***
f-2: ***(puzzle 4 7 1 5 8 0 6 3 2)***	

Match#	***$?x***	***$?y***	***$?z***
f-1	(1 2 3)	()	(4 8 6 5)

  - un patró “encaixa” una vegada amb diferents fets

Fets	Patrons
f-1: ***(puzzle 1 2 3 0 7 4 8 6 5)***	***(puzzle $?x 0 $?y)***
f-2: ***(puzzle 4 7 1 5 8 0 6 3 2)***	

Match#	***$?x***	***$?y***
f-1	(1 2 3)	(7 4 8 6 5)
f-2	(4 7 1 5 8)	(6 3 2)

▷ diferentes patrons “encaixen” amb el mateix fet

	Fets	Patrons
f-1:	***(puzzle 1 2 3 0 7 4 8 6 5)***	***(puzzle $?x 0 ?y $?z)***
		(puzzle $?x ?y 0 $?z)

Match#	***$?x***	***?y***	***$?z***
f-1	(1 2 3)	(7)	(4 8 6 5)
f-1	(1 2)	(3)	(7 4 8 6 5)

- diferents “encaixos” entre un patró i un fet degut a diferents instanciacions de variables

	Fets	Patrons
f-1:	*(puzzle 1 2 3 0 7 4 8 6 5)*	*(puzzle $?x ?y $?z)*

Match#	*$?x*	*?y*	*$?z*
f-1	()	(1)	(2 3 0 7 4 8 6 5)
f-1	(1)	(2)	(3 0 7 4 8 6 5)
f-1	(1 2)	(3)	(0 7 4 8 6 5)
f-1	(1 2 3)	(0)	(7 4 8 6 5)
f-1	(1 2 3 0)	(7)	(4 8 6 5)
f-1	(1 2 3 0 7)	(4)	(8 6 5)
f-1	(1 2 3 0 7 4)	(8)	(6 5)
f-1	(1 2 3 0 7 4 8)	(6)	(5)
f-1	(1 2 3 0 7 4 8 6)	(5)	()

Assignació de patrons a variables

- ***<assigned-pattern-CE>:*** Assignació de l'índex d'un fet amb el qual fa ***matching*** un patró a una variable amb la finalitat d'esborrar el fet de la base de fets mitjançant una acció ***retract*** en la RHS

```
<assigned-pattern-CE> ::= <single-vble> <- <pattern-CE>
```

- Exemple:

```
(defrule esquerra
  ?f <- (puzzle $?x ?y 0 $?z)
  (test (<> (length$ $?x) 2))
  (test (<> (length$ $?x) 5))
  =>
  (retract ?f)
  (assert (puzzle $?x 0 ?y $?z)))
```

CE Test

- ***<test-CE>*** se satisfà si ***<function-call>*** no torna ***False***

```
<test> ::= (test <function-call>)
<function-call> ::=  (<function-name> <expression>*)
<function-name> ::=  > | < | = | <> | eq | neq | <member>
<expression> ::= <constant> | <variable> | <function-call>
```

- Exemple:

```
(test (<> (length$ $?x) 2))
```

  - ***<>*** és l'operador "desigualtat" per a ***<integer>*** o ***<float>***
  - ***length$*** és una funció pre-definida que calcula la longitud d'una llista

- Cal tenir en compte que s'usa notació prefixa (l'operador va davant dels operands)

CEs or, and, not

- `(or <CE>+)` se satisfà si qualsevol dels `<CE>+` ho fa

```
(or (test (= ?x 1)) (test (= ?y 2)))
```

- `(and <CE>+)` se satisfà si tots els `<CE>+` ho fan

```
(and (test (= ?x 1)) (test (= ?y 2)))
```

- `(not <CE>+)` se satisfà si `<CE>` no ho fa

```
(not (test (= ?x 0)))
```

Sintaxi de la RHS

- Accions en la RHS permeten inserir i eliminar fets, mostrar text, detenir el motor d'inferència, etc.:

```
<action> ::=
(assert <fact>+) |
(retract <fact-index>+)  |
(printout t <string> crlf) |
(halt)
<fact-index> ::= <integer> | <single-vble>
```

- Exemples:

```
(defrule baix
  ?f <- (puzzle $?x 0 ?a ?b ?c $?z) =>
  (retract ?f)
  (assert (puzzle $?x ?c ?a ?b 0 $?z)))
(defrule objectiu
  (puzzle 1 2 3 8 0 4 7 6 5) =>
  (printout t "Solució trobada!" crlf)
  (halt))
```

6 Funcions multicamp

- crea valor multicamp

```
(create$ a b)
(a b)
```

- n-èsim camp del multicamp

```
(nth$ 2 (create$ a b))
 b
```

- longitud de multicamp

```
(length$ (create$ a b c))
3
```

- posicion(s) de valor en multicamp

```
(member$ b (create$ a b b))
 2
(member$ (create$ b b) (create$ a b b))
(2 3)
(member$ c (create$ a b b))
 FALSE
```

Referències

[1] G. Riley. CLIPS: A Tool for Building Expert Systems.

[2] G. Riley. CLIPS: SourceForge Project Page.

[3] C. Culbert et al. CLIPS Reference Manual I: Basic Programming Guide (v6.4.1).

[4] C. Culbert et al. CLIPS Reference Manual II: Advanced Programming Guide (v6.4.1).

[5] C. Culbert et al. CLIPS Reference Manual III: Interfaces Guide (v6.4.1).

[6] J. Giarratano. CLIPS User's Guide (v6.4.1).