Tema AR–3: El procedimiento de Davis y Putnam

José A. Alonso Jiménez
José L. Ruiz Reina

Dpto. de Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Objetivos del tema

- **Presentar el procedimiento de Davis y Putnam para decidir la inconsistencia de un conjunto de cláusulas.**
- **Aplicar el procedimiento de Davis y Putnam para decidir:**
  *** la validez de una fórmula y**
  *** si una fórmula es consecuencia de un conjunto**

Eliminación de tautologías

- **Tautologías:**
  - **Def.: C es tautología $\Longleftrightarrow$ C contiene un literal y su complementario.**
  - **Ejemplos:**
    $\{p, q, -p\}$ es tautología
    $\{p, q, -r\}$ no es tautología.
- **Reconocimiento de tautologías**

```
;;; (es-tautologia '(p q (- p)))  =>  T
;;; (es-tautologia '(p q (- r)))  =>  NIL
;;; (es-tautologia '())           =>  NIL
(defun es-tautologia (C)
  (if (some #'(lambda (L) (member (complementario L)
                                  C
                                  :test #'equal))
            C)
      t))
```

- **Propiedades:**
  - **C es tautología $\Longleftrightarrow$ C es válida.**

Eliminación de tautologías

- **Eliminabilidad de tautologías:**

```
Sean S un conjunto de cláusulas y
     C una tautología.
Entonces
     S es inconsistente <==> S-{C} es inconsistente
```

- **Ejemplo:**

```
     {{p,q}, {p,q,-p}} es consistente
<==> {{p,q}}           es consistente
```

- **Procedimiento de eliminación de tautologías**

```
;;; (elimina-tautologias '((p q)(p q (- p))))
;;; =>  ((P Q))
(defun elimina-tautologias (S)
  (remove-if #'(lambda (C) (es-tautologia C))
             S))
```

Eliminación de cláusulas unitarias

- **Cláusulas unitarias:**
  C es unitaria $\Longleftrightarrow$ C tiene sólo un literal

- **Reconocimiento de cláusulas unitarias**

```
;;; (es-unitaria '(p))      =>  T
;;; (es-unitaria '((- p)))  =>  T
;;; (es-unitaria '(p q))    =>  NIL
(defun es-unitaria (C)
  (= (length C) 1))
```

- **Eliminabilidad de cláusulas unitarias:**

```
Sean S  un conjunto de cláusulas,
     C  una cláusula unitaria de S,
     L  el literal de C,
     S1 el conjunto obtenido de S eliminando las
        cláusulas que tienen L y
     S2 el conjunto obtenido de S1 eliminando el
        complementario de L en las cláusulas que
        lo tengan.
Entonces S es inconsistente <==> S2 es inconsistente.
```

- **Ejemplo:**

```
     {{p,q,-r}, {p,-q}, {-p}, {r,u}} es inconsistente
<==> {{q,-r}, {-q}, {r,u}}           es inconsistente
<==> {{-r}, {r,u}}                   es inconsistente
<==> {{}, {u}}                       es inconsistente
```

Eliminación de cláusulas unitarias

- **Procedimiento de eliminación de cláusulas unitarias**

```
;;; (elimina-clausulas-unitarias '((p q (- r))(p (- q))((- p))(r)(u)))
;;; => (NIL (U))
;;; (elimina-clausulas-unitarias '((p q) ((- q)) ((- p) q (- r))))
;;; => NIL
;;; (elimina-clausulas-unitarias '(((- p) q) (p) (r u)))
;;; => ((R U))
(defun elimina-clausulas-unitarias (S)
  (if (member nil S)
      S
      (let ((C (find-if #'es-unitaria S)))
        (if C
            (elimina-clausulas-unitarias
             (elimina-clausula-unitaria (first C) S))
            S))))
```

Eliminación de cláusulas unitarias

```
;;; (elimina-clausula-unitaria '(- p) '((p q (- r))(p (- q))((- p))(r)(u)))
;;; =>  ((Q (- R)) ((- Q)) (R) (U))
;;; (elimina-clausula-unitaria '(- q) '((q (- r)) ((- q)) (r) (u)))
;;;  =>  (((- R)) (R) (U))
;;; (elimina-clausula-unitaria '(- r) '(((- r)) (r) (u)))
;;; =>  (NIL (U))
(defun elimina-clausula-unitaria (L S)
  (let ((L1 (complementario L)))
    (mapcar #'(lambda (C) (remove L1 C :test #'equal))
            (remove-if #'(lambda (C) (member L C :test #'equal))
                       S))))
```

Eliminación de literales puros

- **Literales de un conjunto de cláusulas**
  - **Def.: Literales(S) = $\bigcup S$**
  - **Ejemplo: Literales($\{\{p, q\}, \{p, q, \neg p\}\}$) = $\{p, q, \neg p\}$**
  - **Procedimiento**

```
;;; (literales '((p q)(p q (- p))))  =>  (P Q (- P))
(defun literales (S)
  (union-general S))
```

- **Literales puros:**
  - **Def.: L es un literal puro de $S \iff S$ no contiene el complementario de L**
  - **Ejemplos:**
    * **Sea $S = \{\{p, q\}, \{p, \neg q\}, \{r, q\}, \{r, \neg q\}\}$**
    * **p es un literal puro de S**
    * **q no es un literal puro de S**
  - **Procedimiento**

```
(defun es-literal-puro (L S)
  (every #'(lambda (C)
             (not (member (complementario L)
                          C :test #'equal)))
         S))
```

Eliminación de literales puros

- **Eliminabilidad de literales puros:**

```
Sean S  un conjunto de cláusulas,
     L  un literal puro de S y
     S1 el conjunto obtenido de S eliminando las cláusulas que tienen L.
Entonces, S es inconsistente <==> S1 es inconsistente.
```

- **Ejemplo:**

```
      {{p,q}, {p,-q}, {r,q}, {r,-q}} es inconsistente
<==> {{r,q}, {r,-q}}                 es inconsistente
<==> {}                              es inconsistente
```

Eliminación de literales puros

- **Procedimiento**

```
;;; (elimina-literales-puros '((p q) (p (- q)) (r q) (r (- q))))
;;; =>  NIL
;;; (elimina-literales-puros '((p q) (r (- s)) ((- r) s)))
;;; =>  ((R (- S)) ((- R) S))
(defun elimina-literales-puros (S)
  (let ((L1 (find-if #'(lambda (L) (es-literal-puro L S))
                     (literales S))))
    (if L1
        (elimina-literales-puros (elimina-literal-puro L1 S))
        S)))

;;; (elimina-literal-puro 'p '((p q) (p (- q)) (r q) (r (- q))))
;;; =>  ((R Q) (R (- Q)))
;;; (elimina-literal-puro 'r '((r q) (r (- q))))
;;; =>  NIL
(defun elimina-literal-puro (L S)
  (remove-if #'(lambda (C) (member L C :test #'equal))
             S))
```

Bifurcación

- **Regla de bifurcación:**

```
Sean S   un conjunto de cláusulas sin tautologías,
     L   un literal de S,
     S0  el conjunto de las cláusulas de S que no
         contienen a L ni a su complementario,
     S1a el conjunto de las cláusulas de S que
         contienen L,
     S1b el conjunto obtenido borrando L en las
         cláusulas de S1a,
     S1  la unión de S1 y S0,
     S2a el conjunto de las cláusulas de S que
         contienen el complementario de L,
     S2b el conjunto obtenido borrando el
         complementario de L en las cláusulas
         de S2a y
     S2  la unión de S2b y S0.
Entonces,
     S es inconsistente <==> S1 y S2 son inconsistentes.
```

Bifurcación

- **Ejemplo:**

```
Sean S   = {{p,-q}, {-p,q}, {q,-r}, {-q,-r}}
     L   = p
     S0  = {{q,-r}, {-q,-r}}
     S1a = {{p,-q}}
     S1b = {{-q}}
     S1  = {{-q}, {q,-r}, {-q,-r}}
     S2a = {{-p,q}}
     S2b = {{q}}
     S2  = {{q}, {q,-r}, {-q,-r}}
Entonces
     {{p,-q}, {-p,q}, {q,-r}, {-q,-r}} es inconsistente
<==> {{-q}, {q,-r}, {-q,-r}}           es inconsistente y
     {{q}, {q,-r}, {-q,-r}}            es inconsistente
```

Bifurcación

- **Procedimiento**

```
;;; (bifurcacion '((p (- q)) ((- p) q) (q (- r)) ((- q)(- r))) 'p)
;;; =>  ((((- Q)) ((- Q) (- R)) (Q (- R))) ;
;;;       ((Q) ((- Q) (- R)) (Q (- R))))
(defun bifurcacion (S L)
  (let ((L1 (complementario L))
        (S-con-L    ())
        (S-con-L1   ())
        (S-sin-L-L1 ()))
    (loop for C in S do
      (cond ((member L C :test #'equal)
             (push (remove L C :test #'equal) S-con-L))
            ((member L1 C :test #'equal)
             (push (remove L1 C :test #'equal) S-con-L1))
            (t (push C S-sin-L-L1))))
    (values (n-union S-con-L S-sin-L-L1 :test #'equal)
            (n-union S-con-L1 S-sin-L-L1 :test #'equal))))
```

Método de Davis y Putnam

- **Procedimiento de inconsistencia de Davis y Putnam:**

```
* es-inconsistente-por-dp(S)
  Argumento: S es un conjunto de cláusulas.
  Valor: T, si S es inconsistente; NIL, en caso contrario.
  Procedimiento: Aplicar el procedimiento es-inconsistente-por-dp-aux al
                 resultado de eliminar las tautologías de S.
* es-inconsistente-por-dp-aux(S)
  Argumento: S es un conjunto de cláusulas sin tautologías.
  Valor: T, si S es inconsistente; NIL, en caso contrario.
  Procedimiento
  1. Si S es vacío, devolver NIL.
  2. Si S contiene la cláusula vacía, devolver T.
  3. Sea S1 el resultado de aplicar la elimación de cláusulas unitarias a S.
  4. Si S1 es distinta de S aplicar es-inconsistente-por-dp-aux a S1.
  5. Si S1 es igual que S, sea S2 el resultado de aplicar la elimación de
     literales puros a S1.
  6. Si S2 es distinta de S1 aplicar es-inconsistente-por-dp-aux a S2.
  7. Si S2 es igual que S1, sean S3a y S3b los conjuntos obtenidos bifurcando
     S2 por el primer literal de la primera cláusula de S2.
  8. Aplicar el procedimiento es-inconsistente-por-dp A S3a y S3b.
```

Método de Davis y Putnam

- **Ejemplo**

```
(es-inconsistente-por-dp
 '((a) ((- a) b) (b d) (c d) (c (- d)) ((- c) e) ((- c)(- e)) (a (- a))))
1. (ES-INCONSISTENTE-POR-DP
    '((A) ((- A) B) (B D) (C D) (C (- D)) ((- C) E) ((- C) (- E)) (A (- A))))
2. (ELIMINA-TAUTOLOGIAS
    '((A) ((- A) B) (B D) (C D) (C (- D)) ((- C) E) ((- C) (- E)) (A (- A))))
2. ELIMINA-TAUTOLOGIAS ==>
   ((A) ((- A) B) (B D) (C D) (C (- D)) ((- C) E) ((- C) (- E)))
2. (ES-INCONSISTENTE-POR-DP-AUX
    '((A) ((- A) B) (B D) (C D) (C (- D)) ((- C) E) ((- C) (- E))))
3. (ELIMINA-CLAUSULAS-UNITARIAS
    '((A) ((- A) B) (B D) (C D) (C (- D)) ((- C) E) ((- C) (- E))))
4. (ELIMINA-CLAUSULAS-UNITARIAS
    '((B) (B D) (C D) (C (- D)) ((- C) E) ((- C) (- E))))
5. (ELIMINA-CLAUSULAS-UNITARIAS '((C D) (C (- D)) ((- C) E) ((- C) (- E))))
5. ELIMINA-CLAUSULAS-UNITARIAS ==> ((C D) (C (- D)) ((- C) E) ((- C) (- E)))
4. ELIMINA-CLAUSULAS-UNITARIAS ==> ((C D) (C (- D)) ((- C) E) ((- C) (- E)))
3. ELIMINA-CLAUSULAS-UNITARIAS ==> ((C D) (C (- D)) ((- C) E) ((- C) (- E)))
```

Método de Davis y Putnam

```
3. (ES-INCONSISTENTE-POR-DP-AUX '((C D) (C (- D)) ((- C) E) ((- C) (- E))))
4. (ELIMINA-CLAUSULAS-UNITARIAS '((C D) (C (- D)) ((- C) E) ((- C) (- E))))
4. ELIMINA-CLAUSULAS-UNITARIAS ==> ((C D) (C (- D)) ((- C) E) ((- C) (- E)))
4. (ELIMINA-LITERALES-PUROS '((C D) (C (- D)) ((- C) E) ((- C) (- E))))
4. ELIMINA-LITERALES-PUROS ==> ((C D) (C (- D)) ((- C) E) ((- C) (- E)))
4. (BIFURCACION '((C D) (C (- D)) ((- C) E) ((- C) (- E))) 'C)
4. BIFURCACION ==> (((- D)) (D)), (((- E)) (E))
4. (ES-INCONSISTENTE-POR-DP '(((- D)) (D)))
5. (ELIMINA-TAUTOLOGIAS '(((- D)) (D)))
5. ELIMINA-TAUTOLOGIAS ==> (((- D)) (D))
5. (ES-INCONSISTENTE-POR-DP-AUX '(((- D)) (D)))
6. (ELIMINA-CLAUSULAS-UNITARIAS '(((- D)) (D)))
7. (ELIMINA-CLAUSULAS-UNITARIAS '(NIL))
7. ELIMINA-CLAUSULAS-UNITARIAS ==> (NIL)
6. ELIMINA-CLAUSULAS-UNITARIAS ==> (NIL)
6. (ES-INCONSISTENTE-POR-DP-AUX '(NIL))
4. ES-INCONSISTENTE-POR-DP ==> T
```

Método de Davis y Putnam

```
4. (ES-INCONSISTENTE-POR-DP '(((- E)) (E)))
5. (ELIMINA-TAUTOLOGIAS '(((- E)) (E)))
5. ELIMINA-TAUTOLOGIAS ==> (((- E)) (E))
5. (ES-INCONSISTENTE-POR-DP-AUX '(((- E)) (E)))
6. (ELIMINA-CLAUSULAS-UNITARIAS '(((- E)) (E)))
7. (ELIMINA-CLAUSULAS-UNITARIAS '(NIL))
7. ELIMINA-CLAUSULAS-UNITARIAS ==> (NIL)
6. ELIMINA-CLAUSULAS-UNITARIAS ==> (NIL)
6. (ES-INCONSISTENTE-POR-DP-AUX '(NIL))
4. ES-INCONSISTENTE-POR-DP ==> T
3. ES-INCONSISTENTE-POR-DP-AUX ==> T
2. ES-INCONSISTENTE-POR-DP-AUX ==> T
1. ES-INCONSISTENTE-POR-DP ==> T
T
```

Método de Davis y Putnam

- **Propiedades del procedimiento**
  `es-inconsistente-por-dp(S)`
  - **Terminación: Siempre termina**
  - **Adecuación: Si devuelve `T`, *S* es inconsistente**
  - **Completitud: Si *S* es inconsistente, devuelve `T`**
- **Procedimiento**

```
(defun es-inconsistente-por-dp (S)
  (catch 'resultado
    (es-inconsistente-por-dp-aux (elimina-tautologias S))))

(defun es-inconsistente-por-dp-aux (S)
  (cond ((null S)       (throw 'resultado nil))
        ((member nil S) (throw 'resultado t))
        (t (let ((S1 (elimina-clausulas-unitarias S)))
             (if (not (equal S S1))
                 (es-inconsistente-por-dp-aux S1)
                 (let ((S2 (elimina-literales-puros S1)))
                   (if (not (equal S1 S2))
                       (es-inconsistente-por-dp-aux S2)
                       (multiple-value-bind (S3a S3b)
                         (bifurcacion S2 (first (first S2)))
                         (and (es-inconsistente-por-dp S3a)
                              (es-inconsistente-por-dp S3b))))))))))
```

Validez mediante Davis y Putnam

- **Decisión de validez mediante Davis y Putnam: Son equivalentes:**
  - $\models F$
  - `es-inconsistente-por-dp(negacion(F)) = T`
- **Procedimiento**

```
;;; (es-valida-por-dp '((p -> q) / ( q -> p)))  =>  T
;;; (es-valida-por-dp '(p -> q))                =>  NIL
(defun es-valida-por-dp (F)
  (es-inconsistente-por-dp (clausulas (negacion F))))
```

Consecuencia mediante Davis y Putnam

- **Decisión de consecuencia mediante Davis y Putnam:**
  Son equivalentes
  - $\boldsymbol{S \models F}$
  - `es-inconsistente-por-dp(S U {negacion(F)}) = T`
- **Procedimiento**

```
;;; (es-consecuencia-por-davis-putnam '((p -> q) (q -> r)) '(p -> r))  =>  T
;;; (es-consecuencia-por-davis-putnam '(p) '(p & q))                    =>  NIL
(defun es-consecuencia-por-davis-putnam (S F)
  (es-inconsistente-por-dp
   (clausulas-conjunto (cons (negacion F) S))))
```

Problema de los animales

```
> (setf S '(((tiene_pelos / da_leche) -> es_mamifero)
            ((es_mamifero & (tiene_pezugnas / rumia)) -> es_ungulado)
            ((es_ungulado & tiene_cuello_largo) -> es_jirafa)
            ((es_ungulado & tiene_rayas_negras) -> es_cebra)
            (tiene_pelos & (tiene_pezugnas & tiene_rayas_negras)))
        F 'es_cebra)
ES_CEBRA

> (time (es-consecuencia-por-davis-putnam S F))
Real time: 0.175328 sec.
Run time: 0.18 sec.
Space: 8876 Bytes
T
```

Problema de los animales

```
> (time (setf S1 (clausulas-conjunto (cons (negacion F) S))))
Real time: 0.13575 sec.
Space: 3936 Bytes
(((- ES_CEBRA))
 ((- TIENE_PELOS) ES_MAMIFERO)
 ((- DA_LECHE) ES_MAMIFERO)
 ((- ES_MAMIFERO) (- TIENE_PEZUGNAS) ES_UNGULADO)
 ((- ES_MAMIFERO) (- RUMIA) ES_UNGULADO)
 ((- ES_UNGULADO) (- TIENE_CUELLO_LARGO) ES_JIRAFA)
 ((- ES_UNGULADO) (- TIENE_RAYAS_NEGRAS) ES_CEBRA)
 (TIENE_PELOS)
 (TIENE_PEZUGNAS)
 (TIENE_RAYAS_NEGRAS))

> (time (es-inconsistente-por-dp S1))
Real time: 0.038179 sec.
Space: 4940 Bytes
T
```