Lógica computacional en Sevilla

(30 años en una hora)

José A. Alonso Jiménez

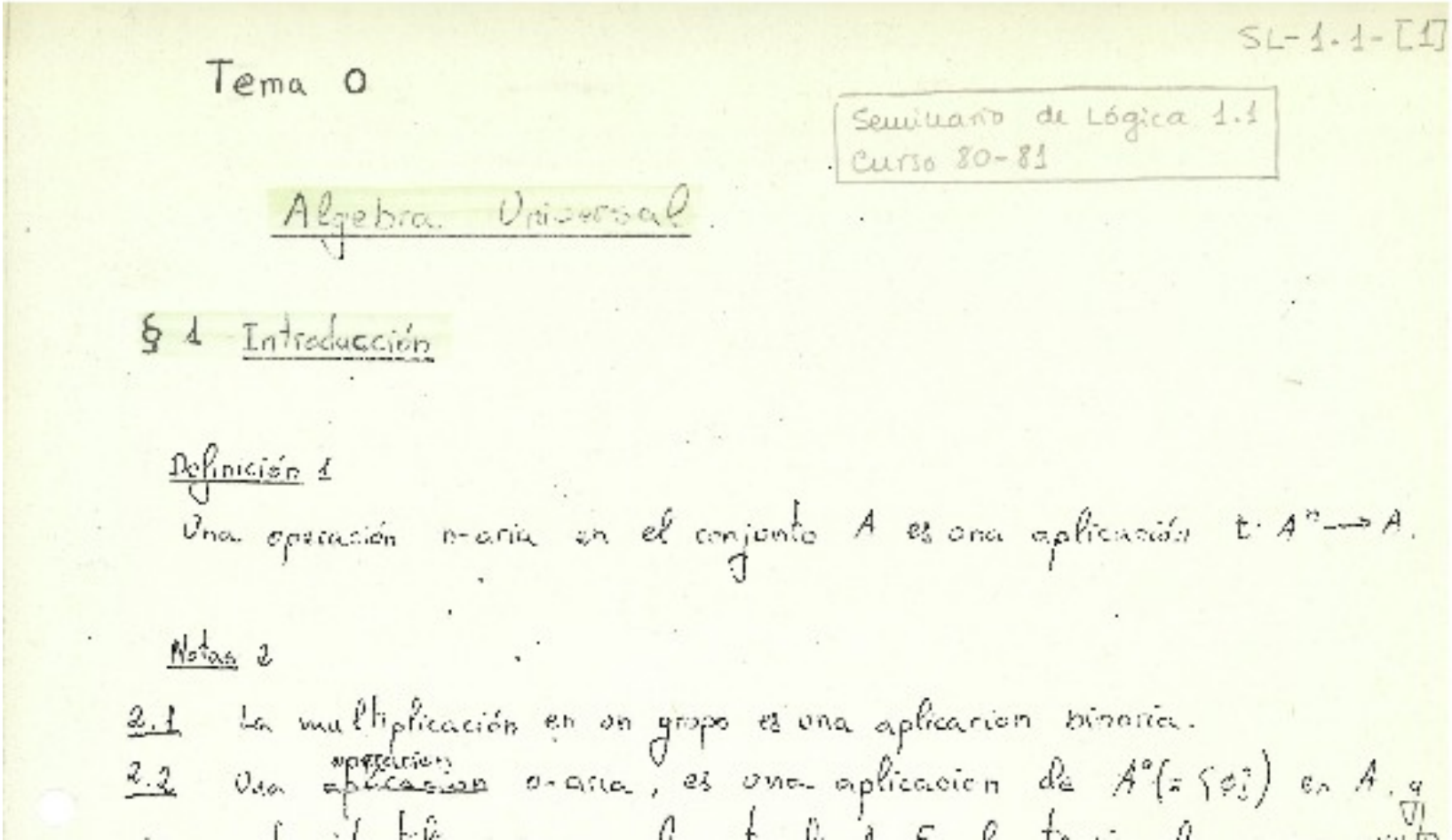
Grupo de Lógica Computacional
Departamento de Ciencias de la Computación e I.A.
Universidad de Sevilla

1 de Diciembre de 2010

Lógica computacional en Sevilla (30 años en una hora)

Seminario de Lógica Matemática (Curso 1980-81)

- Alejandro Fernández Margarit: **Introducción algebraica a la Lógica Matemática** (Barnes y Mack).



SL-1.1-[1]

Tema 0

Seminario de Lógica 1.1
Curso 80-81

Algebra Universal

§ 1 Introducción

Definición 1
Una operación n-aria en el conjunto A es una aplicación $t: A^n \longrightarrow A$.

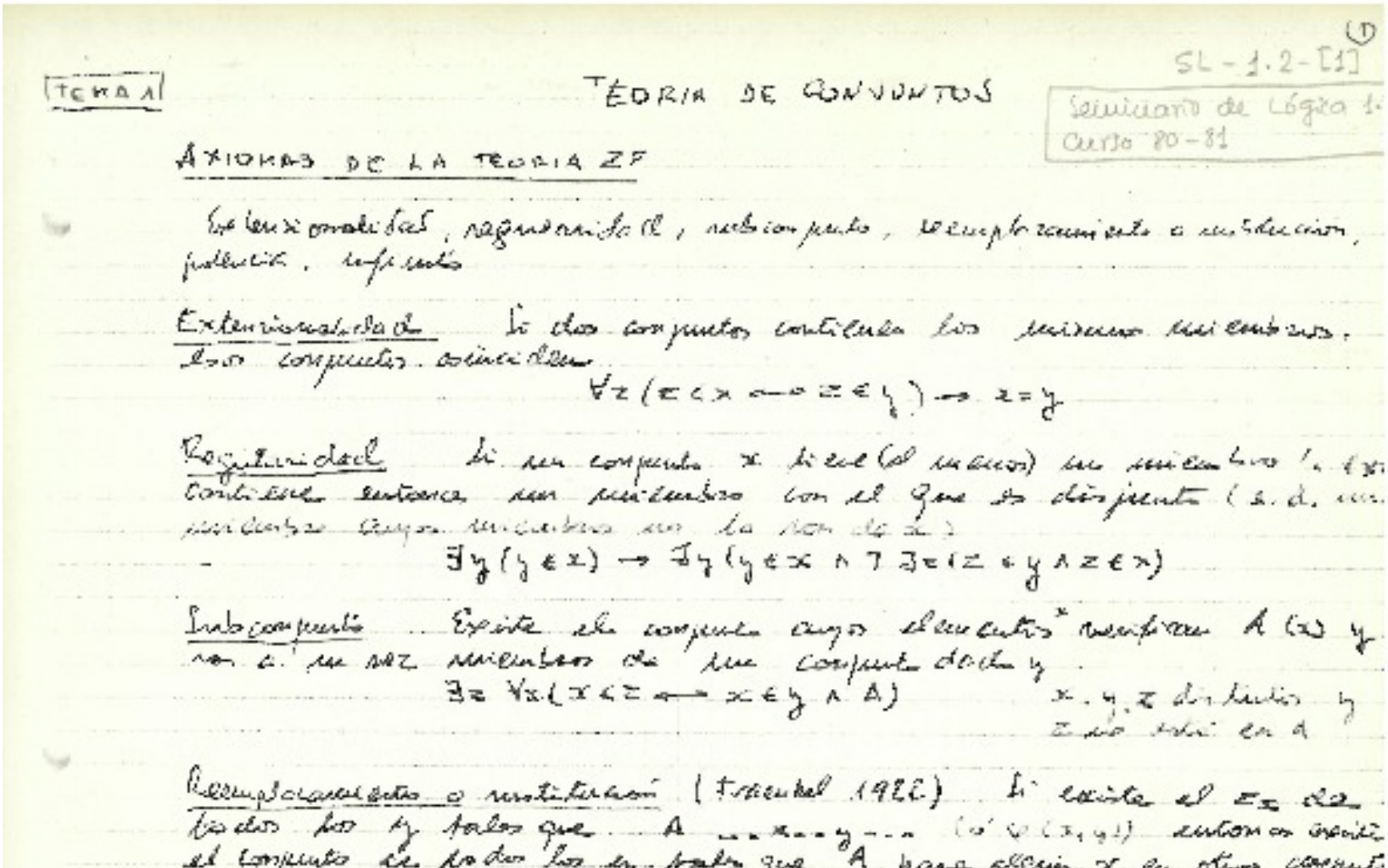
Notas 2

2.1 La multiplicación en un grupo es una aplicacion binaria.

2.2 Una aplicacion n-aria, es una aplicacion de $A^n (= \{s_i\})$ en A.

Seminario de Lógica Matemática (Curso 1980-81)

- Luis Laita: **Teoría axiomática de conjuntos** (Shoenfield).


TEMA 1

TEORIA DE CONJUNTOS

SL - 1.2 - [1]

Seminario de Lógica 1.
Curso 80-81

AXIOMAS DE LA TEORIA ZF

Extensionalidad

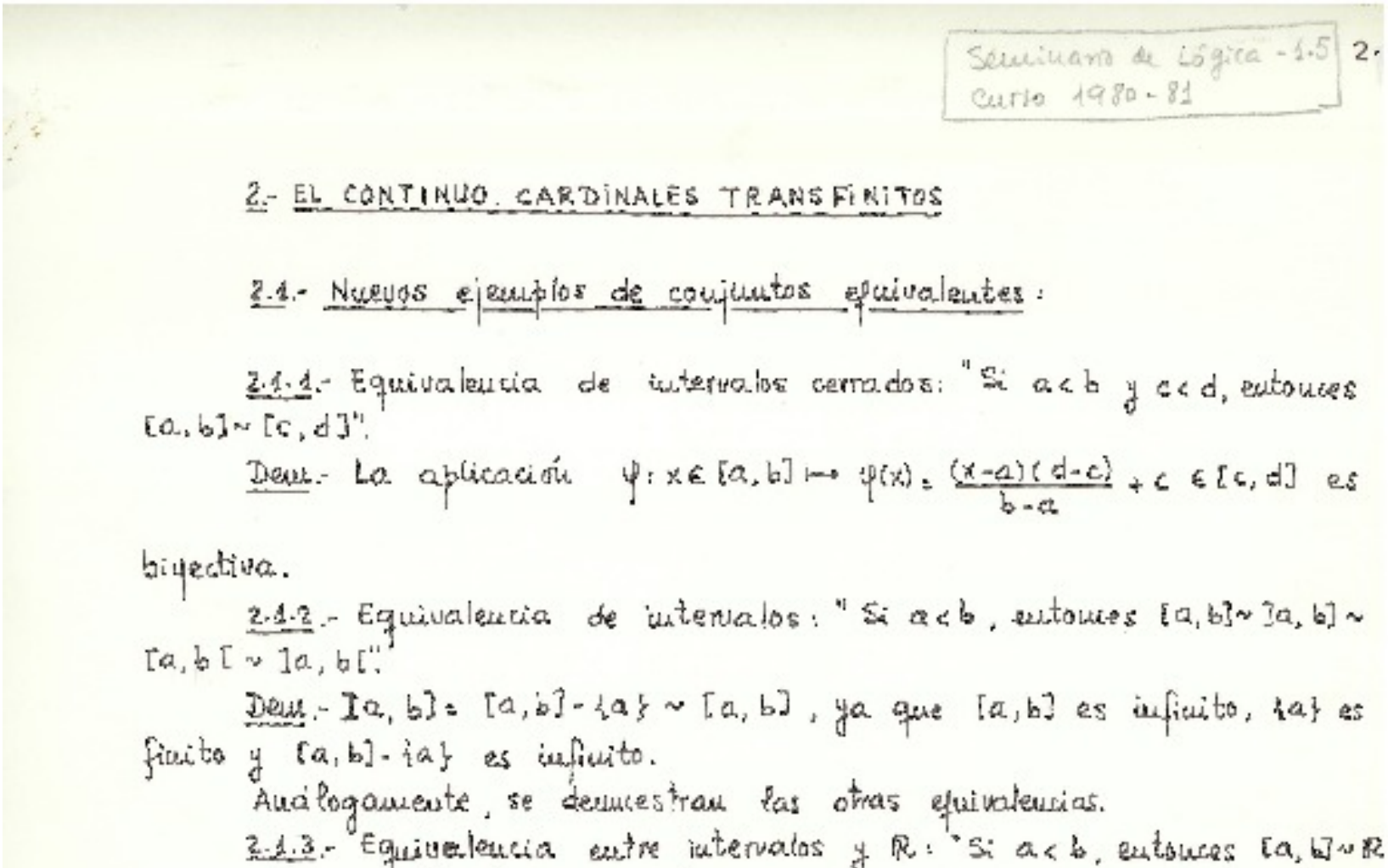
Regularidad

Subconjuntos

Reemplazamiento o sustitución (Fraenkel 1922)

Seminario de Lógica Matemática (Curso 1980-81)

- José A. Alonso: **Teoría de conjuntos** (Halmos).



Seminario de Lógica - 1.5
Curso 1980-81

2.

2.- EL CONTINUO. CARDINALES TRANSFINITOS

2.1.- Nuevos ejemplos de conjuntos equivalentes:

2.1.1.- Equivalencia de intervalos cerrados: "Si $a<b$ y $c<d$, entonces $[a,b]\sim[c,d]$".

Dem.- La aplicación $\varphi: x\in[a,b] \mapsto \varphi(x)=\frac{(x-a)(d-c)}{b-a}+c \in [c,d]$ es biyectiva.

2.1.2.- Equivalencia de intervalos: "Si $a<b$, entonces $[a,b]\sim]a,b]\sim[a,b[\sim]a,b[$".

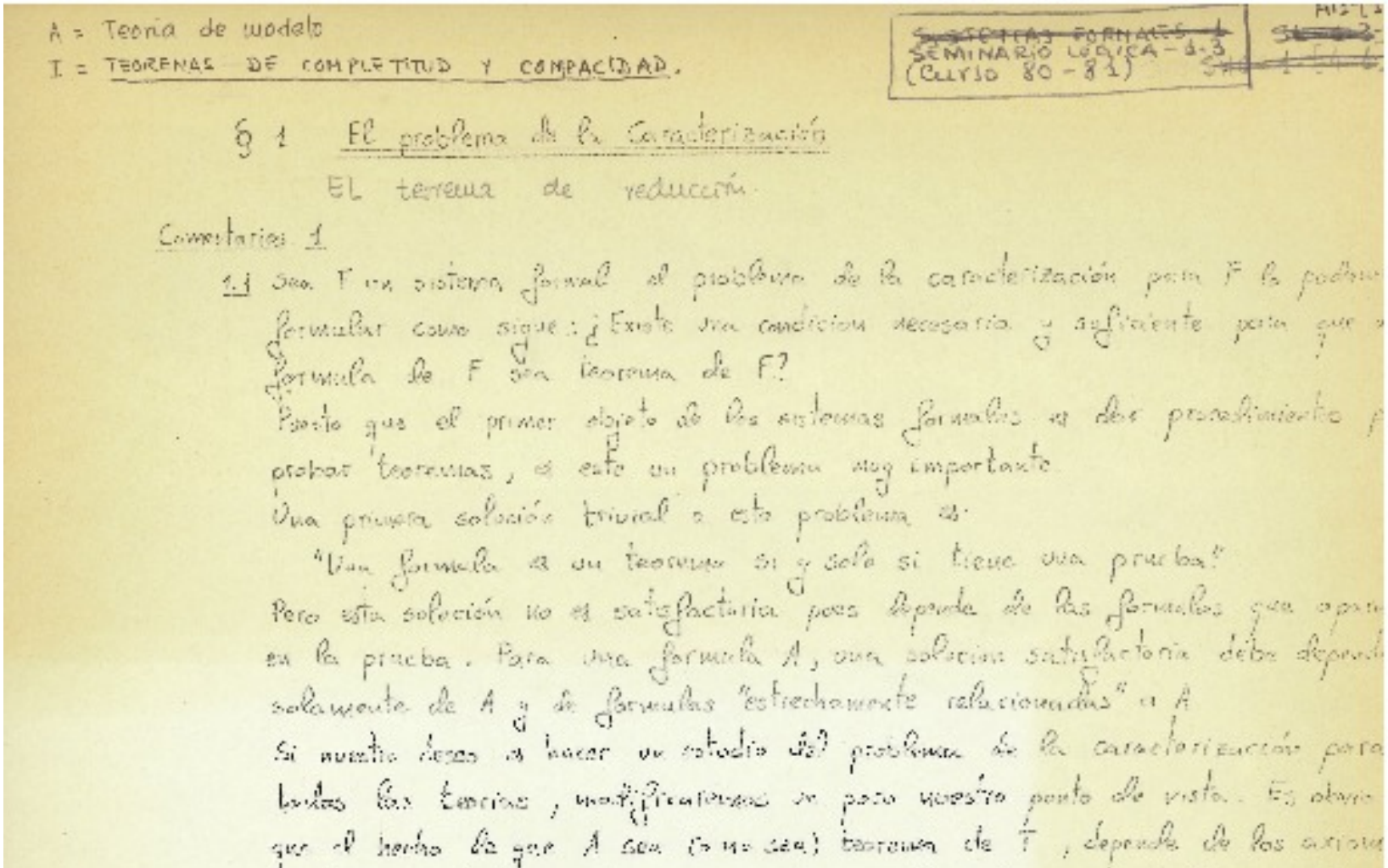
Dem.- $]a,b]=[a,b]-\{a\}\sim[a,b]$, ya que $[a,b]$ es infinito, $\{a\}$ es finito y $[a,b]-\{a\}$ es infinito.

Análogamente, se demuestran las otras equivalencias.

2.1.3.- Equivalencia entre intervalos y $\mathbb{R}$: "Si $a<b$, entonces $[a,b]\sim\mathbb{R}$

Seminario de Lógica Matemática (Curso 1980-81)

- Alejandro Fernández: **Teoría de modelos** (Chang, Keisler).


A = Teoría de modelo
I = TEOREMAS DE COMPLETITUD Y COMPACIDAD.

SEMINARIO LÓGICA - 8-3
(CURSO 80 - 81)

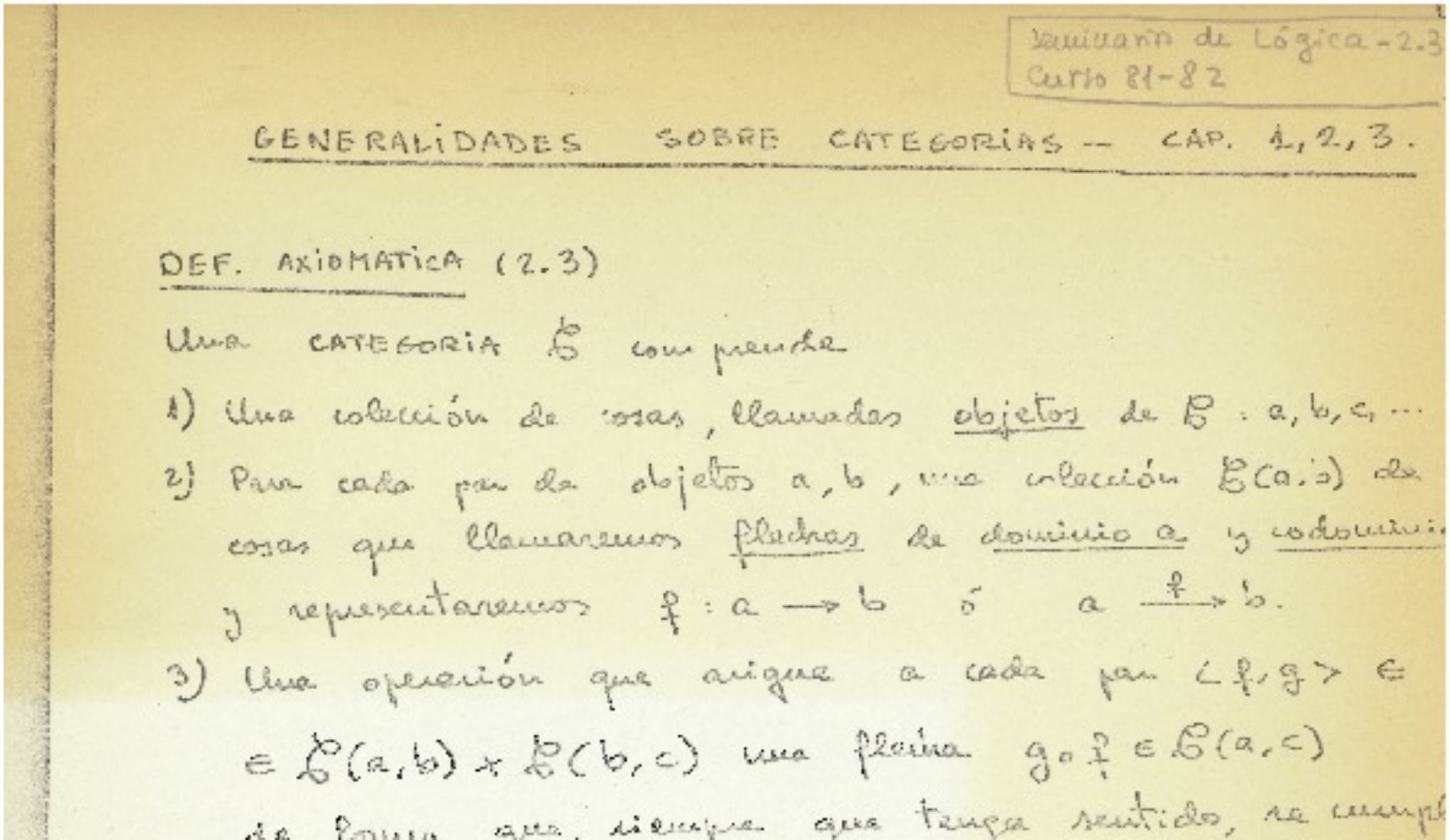
§ 1 El problema de la caracterización
El teorema de reducción

Comentarios 1

1.1 Sea T un sistema formal el problema de la caracterización para F lo podemos formular como sigue: ¿Existe una condición necesaria y suficiente para que una fórmula de F sea teorema de F?
Puesto que el primer objeto de los sistemas formales es dar procedimientos para probar teoremas, es este un problema muy importante.
Una primera solución trivial a este problema es:
"Una fórmula es un teorema si y solo si tiene una prueba"
Pero esta solución no es satisfactoria pues depende de las fórmulas que aparecen en la prueba. Para una fórmula A, una solución satisfactoria debe depender solamente de A y de fórmulas "estrechamente relacionadas" a A
Si nuestro deseo es hacer un estudio del problema de la caracterización para todas las teorías, modificaremos un poco nuestro punto de vista. Es obvio que el hecho de que A sea (o no sea) teorema de T, depende de los axiomas

Seminario de Lógica Matemática (Curso 1981-82)

- Agustín Riscos Fernández: **Topoi, un análisis categorial de la lógica** (Goldblatt)



Seminario de Lógica - 2.3
Curso 81-82

GENERALIDADES SOBRE CATEGORIAS -- CAP. 1, 2, 3.

DEF. AXIOMATICA (2.3)

Una CATEGORIA $\mathcal{C}$ comprende

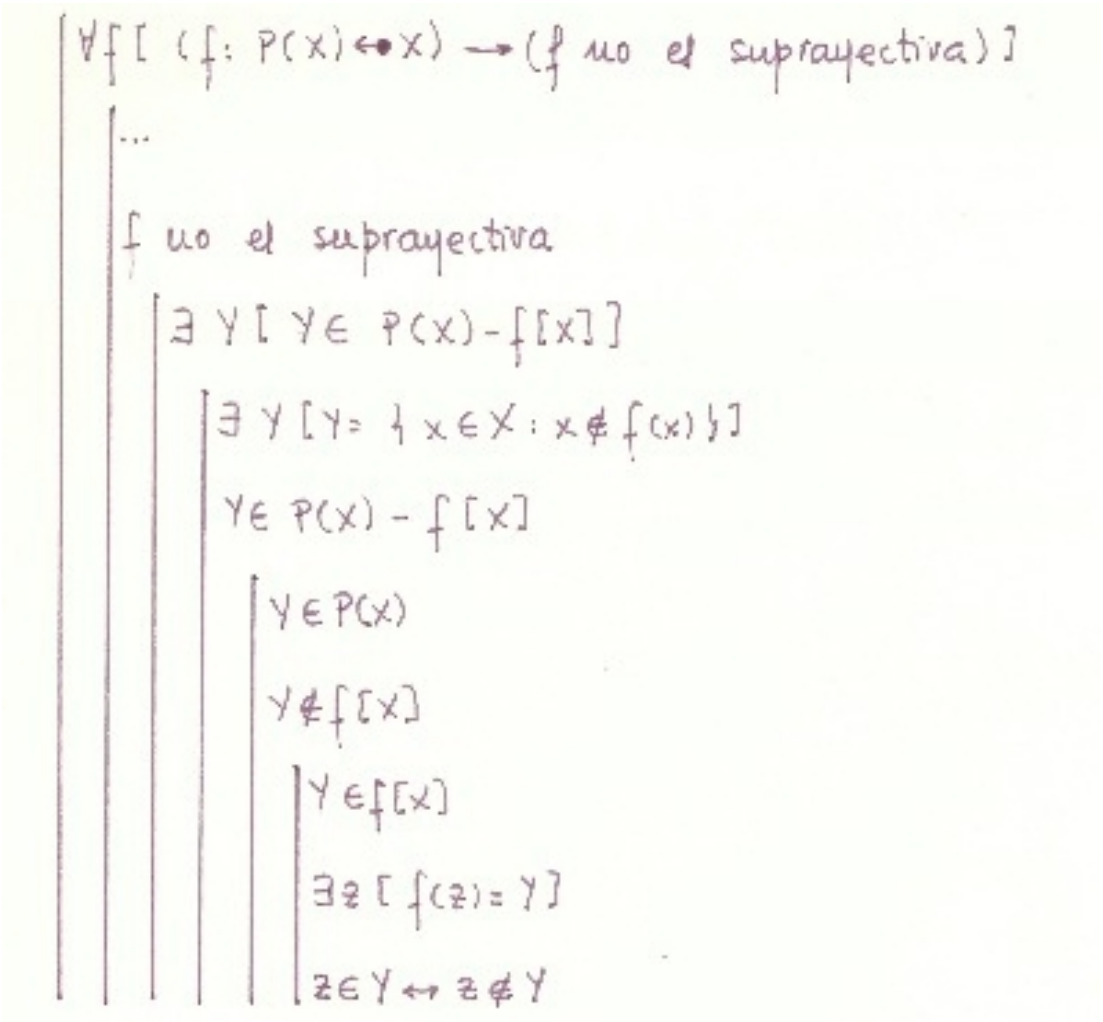
1) Una colección de cosas, llamadas objetos de $\mathcal{C}$: a, b, c, ...

2) Para cada par de objetos a, b, una colección $\mathcal{C}(a,b)$ de cosas que llamaremos flechas de dominio a y codominio b y representaremos $f : a \to b$ ó $a \xrightarrow{f} b$.

3) Una operación que asigna a cada par $\langle f, g \rangle \in \mathcal{C}(a,b) \times \mathcal{C}(b,c)$ una flecha $g \circ f \in \mathcal{C}(a,c)$

Seminario de Lógica Matemática (Curso 1981-82)

- José A. Alonso: **Teoría formalizada de conjuntos** (Jech).



Seminario de Lógica Matemática (Curso 1982-85)

- 1982–83: Demostraciones de independencia en teoría de conjuntos (Kunen).
- 1983–84: Lógica algebraica.
- 1984–85: Teoría de la recursión (Cutland).

Lógica computacional en Sevilla (30 años en una hora)

Seminario de Computación (Curso 1985-87)

- 1985-86: Programación en LeLisp (Queinnec, Winston).
- 1985-86: Inteligencia artificial (Nilsson, Delahaye).
- 1986-87: Cálculo simbólico con Macsyma.

Lógica computacional en Sevilla (30 años en una hora)

Primeros trabajos en lógica computacional

- 1985: Programación de cálculo simbólico con polinomios.
- 1986: Demostración automática en lógica proposicional.
- 1987: Demostración automática en lógicas trivalentes.
- 1988: Demostración automática en lógicas modales.

La Escuela Lógica

Etimología de la palabra "escuela" en Wikipedia

- El término **escuela** proviene del griego clásico (eskholé) por mediación del latín schola.
- El significado original en griego era de 'ocio, tranquilidad, tiempo libre', que luego derivó a **aquello que se hace en durante el tiempo libre** y, más concretamente, **aquello que merece la pena hacerse**.

Lógica computacional en Sevilla (30 años en una hora)

Sintaxis de las lógicas polivalentes

- Lenguaje de lógicas polivalentes:
  - n y s son enteros positivos.
  - $\{X_1, \ldots, X_n\}$ son las *variables proposicionales*.
  - $\{f_1, \ldots, f_s\}$ son las *conectivas*.
  - $\delta : \{1, \ldots, s\} \to \mathbb{N}$ es la *aridad*.
- Fórmulas proposicionales: $\mathbb{P}(X_1, \ldots, X_n)$
  - $X_i \in \mathbb{P}(X_1, \ldots, X_n)$
  - Si $j \in \{1, \ldots, s\}$ y $A_1, \ldots, A_{\delta(j)} \in \mathbb{P}(X_1, \ldots, X_n)$, entonces $f_j(A_1, \ldots, A_{\delta(j)}) \in \mathbb{P}(X_1, \ldots, X_n)$

Semántica de las lógicas polivalentes

- Valores de verdad
  - p es un número primo.
  - $\mathbb{Z}_p$ se interpretan como *valores de verdad*.
  - 0 se interpreta como *falso*.
  - 1 se interpreta como *verdadero*.
- Funciones de verdad
  - $H_j : \mathbb{Z}_p^{\delta(j)} \to \mathbb{Z}_p$ es la *función de verdad* de la conectiva f_j.
- Valoraciones
  - $v : \{X_1, \ldots, X_n\} \to \mathbb{Z}_p$ es una *valoración*
- Valor de verdad de una fórmula en una valoración v:

$$\hat{v} : \mathbb{P}(X_1, \ldots, X_n) \to \mathbb{Z}_p$$

$$\hat{v}(A) = \begin{cases} v(A), & \text{si } A \in \{X_1, \ldots, X_n\} \\ H_j(\hat{v}(A_1), \ldots, \hat{v}(A_{\delta(j)})), & \text{si } A \text{ es } f_j(A_1, \ldots, A_{\delta(j)}) \end{cases}$$

Consecuencia polivalente

- Modelo de fórmula
  - v es modelo de A si $v(A) = 1$.
- Modelo de un conjunto de fórmulas
  - v es modelo de $\{B_1, \ldots, B_m\}$ si $v(B_1) = \cdots = v(B_m) = 1$.
- Consecuencia lógica:
  - La fórmula A es *consecuencia lógica* de las fórmulas $B_1, \ldots, B_m$ si para todo modelo de $\{B_1, \ldots, B_m\}$ es modelo de A.
  - Se representa por $\{B_1, \ldots, B_m\} \models A$.

Ejemplo: lógica trivalente de Lukasiewicz

- $p = 3$
- 2 se interpreta como indeterminado
- Conectivas:
  $f_1 = \neg$ (negación),
  $f_2 = \Diamond$ (posibilidad),
  $f_3 = \Box$ (necesidad),
  $f_4 = \vee$ (disyunción),
  $f_5 = \wedge$ (conjunción),
  $f_6 = \rightarrow$ (implicación),
  $f_7 = \leftrightarrow$ (equivalencia)

Ejemplo: lógica trivalente de Lukasiewicz

- Funciones de verdad

a	$H_1(a)$	$H_2(a)$	$H_3(a)$
0	1	0	0
1	0	1	1
2	2	1	0

a	b	$H_4(a,b)$	$H_5(a,b)$	$H_6(a,b)$	$H_7(a,b)$
0	0	0	0	1	1
0	1	1	0	1	0
0	2	2	0	1	2
1	0	1	0	0	0
1	1	1	1	1	1
1	2	1	2	2	2
2	0	2	0	2	2
2	1	1	2	1	2
2	2	2	2	1	1

Lógica computacional en Sevilla (30 años en una hora)

Fórmulas y polinomios

- Transformación de fórmulas en polinomios:
  $\theta : \mathbb{P}(X_1, \ldots, X_n) \to \mathbb{Z}_p[X_1, \ldots, X_n]$ definida por
  $$\theta(A) = \begin{cases} A, & \text{si } A \in \{X_1, \ldots, X_n\}; \\ \theta_j(\theta(A_1), \ldots, \theta(A_{\delta(j)})), & \text{si } A = f_j(A_1, \ldots, A_{\delta(j)}) \end{cases}$$
  donde
  - $\theta_j(q_1, \ldots, q_{\delta(j)}) = \sum_{i_1, \ldots, i_{\delta(j)}} H_j(i_1, \ldots, i_{\delta(j)}) L_{i_1}(q_1) \ldots L_{i_{\delta(j)}}(q_{\delta(j)})$
  - $L_0(q) = 1 - q^{p-1}$
  - $L_i(q) = L_0(q - i)$ para $i \in \{1, \ldots, p-1\}$
- **Teorema**: Sean $A_0, A_1, \ldots, A_m \in \mathbb{P}(X_1, \ldots, X_n)$. Son equivalentes:
  - A_0 es consecuencia de $\{A_1, \ldots, A_m\}$
  - $\theta(A_0) - 1$ pertenece al ideal generado por $\{\theta(A_1) - 1, \ldots, \theta(A_m) - 1, X_1^p - X_1, \ldots, X_n^p - X_n\}$

Lógica computacional en Sevilla (30 años en una hora)

Decisión de consecuencia mediante BG

- Algoritmo de decisión de consecuencia mediante BG.
  - *Entrada*: Una fórmula A_0 y un conjunto finito $\Gamma = \{A_1, \ldots, A_m\}$ de fórmulas.
  - *Salida*: SÍ, si $\Gamma \models A_0$; NO, en caso contrario.
  - *Procedimiento*:
    $F := \{\theta(A_1) - 1, \ldots, \theta(A_m) - 1, X_1^p - X_1, \ldots, X_n^p - X_n\}$
    $G := BG(F - \{0\})$
    $q := FN(\theta(A_0) - 1, G)$
    si $q = 0$, devolver SÍ
    en otro caso devolver NO.
- Implementaciones: Lisp, Reduce, Maple, CoCoA.
- Aplicaciones:
  - Compilación de bases de conocimiento
  - Verificación de bases de conocimiento
  - Construcción de sistemas expertos

Lógica computacional en Sevilla (30 años en una hora)

Publicaciones

1987 Preuve automatique dans le calcul propositionnel et les logiques trivalentes. Congress on Computacional Topology and Geometry and Computation in Teaching Matemathics.

1988 Métodos algebraicos de razonamiento automático. Tesis Univ. de Sevilla.

1990 Lógicas polivalentes y bases de Gröbner. LNLF.

1991 Multi–valued logic and Gröbner bases with applications to modal logic. Journal of Symbolic Computation.

1999 A computer algebra approach to verification and deduction in many–valued knowledge systems. Soft Computing.

2009 A Groebner bases–based approach to backward reasoning in rule based expert systems. Ann. Math. Artif. Intell.

Lógica computacional en Sevilla (30 años en una hora)

Ejemplo de razonamiento automático con OTTER

- Teorema: *Sea G un grupo y e su elemento neutro. Si, para todo x de G, $x^2 = e$, entonces G es conmutativo.*
- Formalización:
  - Axiomas de grupo:
    $$(\forall x)[e.x = x]$$
    $$(\forall x)[x.e = x]$$
    $$(\forall x)[x.x^{-1} = e]$$
    $$(\forall x)[x^{-1}.x = e]$$
    $$(\forall x)(\forall y)(\forall z)[(x.y).z = x.(y.z)]$$
  - Hipótesis
    $$(\forall x)[x.x = e]$$
  - Conclusión
    $$(\forall x)(\forall y)[x.y = y.x]$$

Ejemplo de razonamiento automático con OTTER

- Entrada `grupos.in`

```
set(auto2).
op(400, xfy, *).
op(300, yf, ^).

list(usable).
e * x = x.                          % Ax. 1
x * e = x.                          % Ax. 2
x^ * x = e.                         % Ax. 3
x * x^ = e.                         % Ax. 4
(x * y) * z = x * (y * z).          % Ax. 5
x = x.                              % Ax. 6
x * x = e.
b * a != a * b.
end_of_list.
```

Ejemplo de razonamiento automático con OTTER

- Uso de OTTER

```
otter <grupos.in
```

- Resultado:

```
---------------- PROOF ----------------------------------
1 []                                                b*a!=a*b.
3,2 []                                              e*x=x.
5,4 []                                              x*e=x.
10 []                                               (x*y)*z=x*(y*z).
13 []                                               x*x=e.
17 [para_into,10.1.1.1,13.1.1,demod,3,flip.1]       x*(x*y)=y.
23 [para_into,10.1.1,13.1.1,flip.1]                 x*(y*(x*y))=e.
33 [para_from,23.1.1,17.1.1.2,demod,5,flip.1]       x*(y*x)=y.
37 [para_from,33.1.1,17.1.1.2]                      x*y=y*x.
38 [binary,37.1,1.1]                                F.
------------ end of proof ---------------------------------
```

Lógica computacional en Sevilla (30 años en una hora)

Ejemplo de conocimiento matemático

- Teorema: *Existen infinitos números primos.*
- Demostración usual:
  Hemos de probar que para cada número natural, x, existe un número primo y que es mayor que x. Lo haremos por reducción al absurdo.

  Supongamos que existe un número natural a tal que cualquier número primo es menor o igual que a. Sea x un número arbitrario. Puesto que el menor factor primo de $1 + x!$ es primo, debe ser menor o igual que a y, por tanto, divide al factorial de a. En particular, el menor factor primo de $1 + a!$ divide al factorial de a y, puesto que también divide a $1 + a!$, tiene que dividir a la diferencia y, por tanto, debe ser igual a 1. De donde se sigue que $a! = 0$. Lo que es una contradicción.

Extracción del conocimiento

- Símbolos no lógicos

fact (x)	factorial de x
mfp (x)	menor factor primo de x, si $x > 1$ y x en caso contrario
PR (x)	x es un número primo
x < y	x es menor que y
DIV (x,y)	x divide a y

- Hipótesis necesarias:
  - H1 Si un número divide a una suma y a uno de los sumandos, entonces divide al otro.
  - H2 El menor factor primo de un número divide a dicho número.
  - H3 Si el menor factor primo de un número es 1, entonces dicho número es 1.
  - H4 El menor factor primo de $1 + x$ es distinto de 0.
  - H5 El menor factor primo de $1 + x!$ es primo.
  - H6 Si $y \neq 0$ e $y \leq x$, entonces y divide a $x!$.
  - H7 El factorial de cualquier número es distinto de 0.

Representación del conocimiento en OTTER

- Representación en OTTER:

```
list(usable).
DIV(x, y + z), DIV(x, z) -> DIV(x, y).   % H1
-> DIV(mfp(x), x).                       % H2
mfp(x) = 1 -> x = 1.                     % H3
mfp(1 + x) = 0 ->.                       % H4
-> PR(mfp(1 + fact(x))).                 % H5
-> y = 0, x < y, DIV(y, fact(x)).        % H6
fact(x) = 0 ->.                          % H7
end_of_list.
```

Representación del conocimiento en OTTER

- Tesis

  Tesis: $\forall x \exists y (x < y \wedge PR(y))$

  Negación: $\exists x \forall y \neg (x < y \wedge PR(y))$

- Cláusula de la tesis

```
list(sos).
a < y , PR(y) ->.
end_of_list.
```

- Demoduladores

```
list(demodulators).
-> (x + y = x) = (y = 0).
-> DIV(x,1) = (x = 1).
end_of_list.
```

- Regla de inferencia

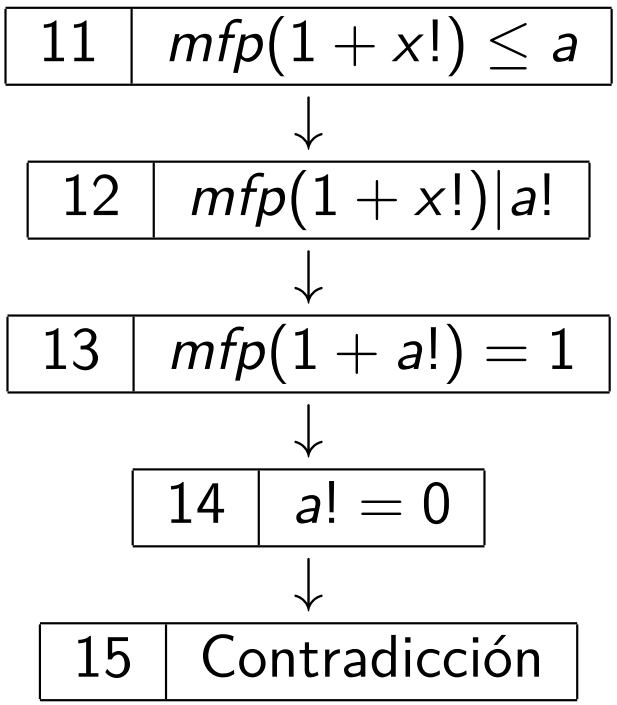
```
set(ur_res).
```

Procesamiento del conocimiento con OTTER

```
1 [] DIV(x,y+z), DIV(x,z) -> DIV(x,y).
2 []  -> DIV(mfp(x),x).
3 [] mfp(x)=1 -> x=1.
4 [] mfp(1+x)=0 -> .
5 []  -> PR(mfp(1+fact(x))).
6 []  -> y=0, x<y, DIV(y,fact(x)).
7 [] fact(x)=0 -> .
8 []  -> (x+y=x)= (y=0).
9 []  -> DIV(x,1)= (x=1).
10 [] a<y, PR(y) -> .
11 [ur,10,5] a<mfp(1+fact(x)) -> .
12 [ur,11,6,4]  -> DIV(mfp(1+fact(x)),fact(a)).
13 [ur,12,1,2,demod,9]  -> mfp(1+fact(a))=1.
14 [ur,13,3,demod,8]  -> fact(a)=0.
15 [binary,14.1,7.1]  -> .
```

Correspondencia de la prueba de OTTER y la inicial

- Grafo de la prueba:



Lógica computacional en Sevilla (30 años en una hora)

Publicaciones

1992 Automated proofs in group theory with OTTER. Symposium on Knowledge Engineering.

1993 Especificación formal de bases de datos con un demostrador automático de teoremas. PRODE.

1994 Automatización de la aritmética. LNLF.

2002 A methodology for the computer-aided cleaning of complex knowledge databases. IECON.

2003 Towards a practical argumentative reasoning with qualitative spatial databases. IEA/AIE.

2004 Ontology cleaning by mereotopological reasoning. DEXA.

2005 Razonamiento mereotopológico automatizado para la depuración de ontologías. Tesis Univ. de Sevilla.

2006 Foundational challenges in automated data and ontology cleaning in the Semantic Web. IEEE Intelligent Systems.

Lógica computacional en Sevilla (30 años en una hora)

Manifiesto QED (1993)

QED is the very tentative title of a project to build a computer system that effectively represents all important mathematical knowledge and techniques.

1. to help mathematicians cope with the explosion in mathematical knowledge
2. to help development of highly complex IT systems by facilitating the use of formal techniques
3. to help in mathematical education
4. to provide a cultural monument to "the fundamental reality of truth"
5. to help preserve mathematics from corruption
6. to help reduce the 'noise level' of published mathematics
7. to help make mathematics more coherent
8. to add to the body of explicitly formulated mathematics
9. to help improve the low level of self-consciousness in mathematics

Tesinas en lógica computacional (1994–95)

- 1994: **Sistemas de reescritura de término** (José L. Ruiz Reina).
- 1994: **Demostración automática mediante tableros semánticos** (Manuel E. Gegúndez Arias y Jesús Muñoz San Miguel).
- 1994: **Fundamentos de programación lógica** (Miguel A. Gutiérrez Naranjo).
- 1995: **Problemas de unificación: Unificación sintáctica y E-unificación** (Francisco J. Martín Mateos).

Lógica computacional en Sevilla (30 años en una hora)

El sistema ACL2

- **A C**omputacional **L**ogic for an **A**pplicative **C**ommon **L**isp.
- Desarrollado por Moore y Kaufmann.
- Sucesor de NQTHM de Boyer y Moore.
- Tres aspectos de ACL2:
  1. Lenguaje de programación.
  2. Lógica.
  3. Sistema de razonamiento automático.

ACL2 como lenguaje de programación

- Definición:

```
(defun concatena (x y)
  (if (endp x) y
      (cons (car x) (concatena (cdr x) y))))

(defun inversa (x)
  (if (endp x) nil
      (concatena (inversa (cdr x)) (list (car x)))))
```

- Evaluación:

```
ACL2 !> (concatena '(a b) '(c d e))
(A B C D E)
ACL2 !> (inversa '(a b c))
(C B A)
ACL2 !> (inversa (inversa '(a b c)))
(A B C)
```

ACL2 como lógica

- ACL2 es una lógica de primer orden con igualdad sin cuantificadores de la parte aplicativa de Common Lisp.
- Ejemplo de definiciones de la teoría de listas:

```
(defun true-listp (x)
  (if (consp x) (true-listp (cdr x))
                (eq x nil)))
```

- Ejemplo de axiomas de la teoría de lista:

```
(defaxiom car-cdr-elim
  (implies (consp x)
           (equal (cons (car x) (cdr x)) x)))
(defaxiom car-cons (equal (car (cons x y)) x))
(defaxiom cdr-cons (equal (cdr (cons x y)) y))
```

- Mediante `defun` se extiende la lógica de manera conservativa. Se necesitan condiciones de admisibilidad.

ACL2 como lógica: Admisibilidad

- Ejemplo 1:

```
ACL2 !> (defun f (x) y)

ACL2 Error in (DEFUN F ...): The body of F contains
a free occurrence of the variable symbol Y.
```

- Ejemplo 2:

```
ACL2 !> (defun f (x) (+ (f x) 1))

ACL2 Error in (DEFUN F ...): No :measure was
supplied with the definition of F. Our heuristics
for guessing one have not made any suggestions.
No argument of the function is tested along every
branch and occurs as a proper subterm at the same
argument position in every recursive call. You must
specify a :measure.
```

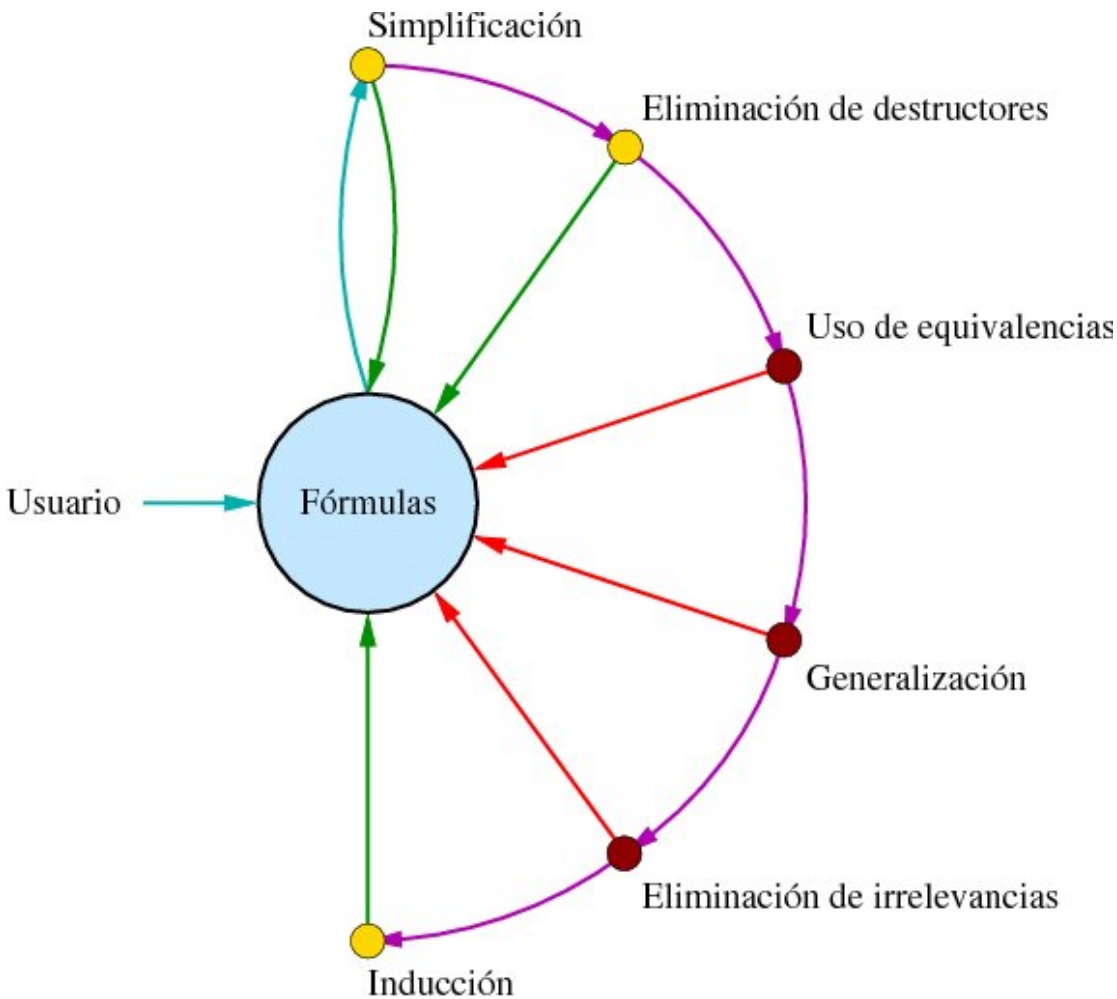
ACL2 como lógica: Admisibilidad

- Ejemplo 3:

```
ACL2 !> (defun inversa (x)
          (if (endp x)
              nil
              (concatena (inversa (cdr x))
                         (list (car x)))))

The admission of INVERSA is trivial, using the relation
EO-ORD-< (which is known to be well-founded on the
domain recognized by EO-ORDINALP) and the measure
(ACL2-COUNT X).  We observe that the type of INVERSA is
described by the theorem
(OR (CONSP (INVERSA X)) (EQUAL (INVERSA X) NIL)).
We used primitive type reasoning and the
:type-prescription rule CONCATENA.
```

ACL2 como SRA (sistema de razonamiento automático)



ACL2 como SRA: Inducción

```
ACL2 !> (defthm inversa-inversa
          (implies (true-listp x)
                   (equal (inversa (inversa x)) x)))

   Name the formula above *1.
   Perhaps we can prove *1 by induction. Two induction schemes
are suggested by this conjecture. These merge into one
derived induction scheme.
   We will induct according to a scheme suggested by (INVERSA
X). If we let (:P X) denote *1 above then the induction
scheme we'll use is
(AND (IMPLIES (AND (NOT (ENDP X)) (:P (CDR X)))
              (:P X))
     (IMPLIES (ENDP X) (:P X))).
```

El papel del usuario

- Frecuentemente los primeros intentos de demostración de resultados no triviales fallan.
- Ello significa que el demostrador necesita demostrar lemas previos.
- Estos lemas se obtienen:
  - de una demostración preconcebida a mano o
  - del anális del fallo del intento de prueba.
- Por tanto, el papel del usuario es:
  - Formalizar la teoría en la lógica.
  - Implementar una estrategia de la prueba, mediante una sucesión de lemas.
- El resultado es un fichero con definiciones y teoremas
  - *Un libro* en la terminología de ACL2.
  - El libro puede certificarse y usarse en otros libros.

Lógica computacional en Sevilla (30 años en una hora)

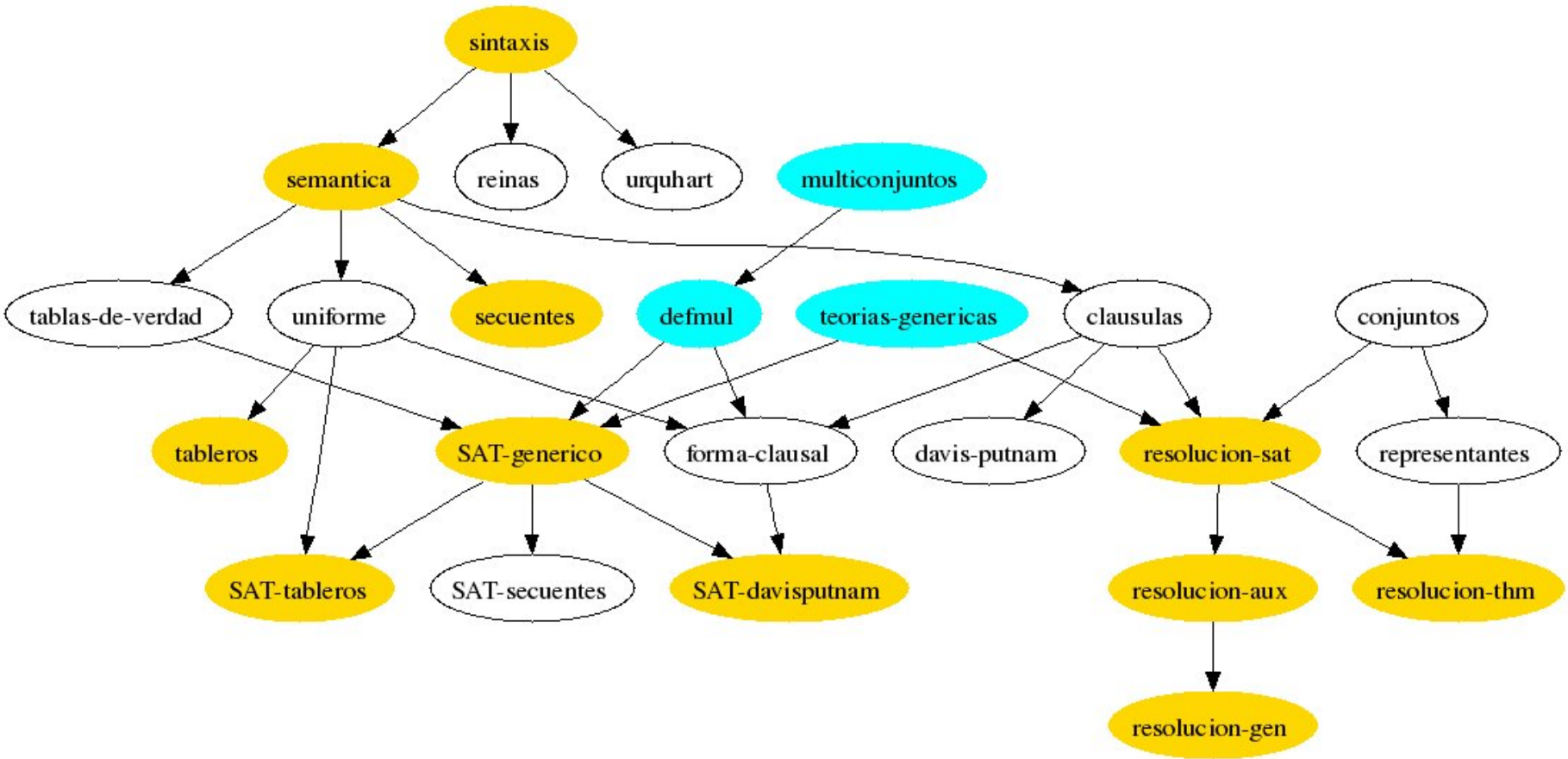
Cálculo de modelos mediante tableros semánticos

MOD(θ) :=
if endp(θ) **then** nil
elseif rama-cerrada(θ) **then** nil
else let F **be** seleccion(θ) **in**
    if doble-negacion(F)
    then MOD(añade(componente-neg-neg(F), elimina(F, θ)))
    elseif alfa-formula(F)
    then MOD(añade(componente-1(F),
                añade(componente-2(F), elimina(F, θ))))
    elseif beta-formula(F)
    then MOD(añade(componente-1(F), elimina(F, θ)) **or**
        MOD(añade(componente-2(F), elimina(F, θ))
    else θ

Cálculo de modelos mediante tableros semánticos

- Propiedades demostradas sobre MOD
  - Terminación.
  - Corrección y completitud: $MOD(\theta) \neq \mathbf{f} \Longleftrightarrow \theta$ tiene modelos.
- Otros algoritmos proposicionales
  - Satisfacibilidad por tableros: $SAT(F) \Longleftrightarrow MOD(\langle F \rangle) \neq \mathbf{f}$
  - Demostrabilidad por tableros: $DAT(F) \Longleftrightarrow MOD(\langle \neg F \rangle) = \mathbf{f}$
  - Deducibilidad por tableros:
    $DEDUC(\langle H_1, \ldots, H_n \rangle, F) \Longleftrightarrow MOD(\langle \neg F, H_1, \ldots, H_n \rangle) = \mathbf{f}$
- Terminación, corrección y completitud de SAT, DAT y DEDUC.

Verificación de procedimientos de decisión proposicionales



Verificación de procedimientos de decisión proposicionales

Libro	Líneas	Definiciones	Teoremas	Consejos
sintaxis.lisp	323	22	45	12
semantica.lisp	113	11	9	1
tablas-de-verdad.lisp	171	13	16	7
uniforme.lisp	206	10	18	5
tableros.lisp	349	13	34	12
secuentes.lisp	507	21	48	20
multiconjuntos.lisp	944	28	149	39
defmul.lisp	886	31	29	16
teorias-genericas.lisp	115	15	2	2
listas.lisp	146	15	10	1
SAT-generico.lisp	335	16	37	13
SAT-tableros.lisp	603	16	67	34
SAT-secuentes.lisp	427	20	36	9
clausulas.lisp	277	16	39	11
forma-clausal.lisp	371	12	47	17
davis-putnam.lisp	621	24	82	21
SAT-davisputnam.lisp	485	23	47	9
conjuntos.lisp	284	19	41	10
resolucion-sat.lisp	1069	27	124	46
representantes.lisp	229	11	25	11
resolucion-thm.lisp	861	13	92	50
resolucion-aux.lisp	333	0	36	13
resolucion-gen.lisp	489	20	50	11

Verificación de procedimientos de decisión proposicionales

1998 Verificación automática de sistemas de razonamiento (aplicación a la enseñanza de la Inteligencia Artificial). JENUI.

2000 Multiset relations: a tool for proving termination. ACL2 Workshop 2000.

2001 Verifying an applicative ATP using multiset relations. EUROCAST

2002 Desarrollo formal y verificación de sistemas proposicionales. IDEIA/IBERAMIA.

2002 Verification in ACL2 of a generic framework to synthesize SAT–provers. LPOSTR 2002.

2002 Teoría computacional (en ACL2) sobre cálculos proposicionales. Tesis Univ. de Sevilla.

2004 Formal verification of a generic framework to synthesize SAT–provers. Journal of Automated Reasoning.

Lógica computacional en Sevilla (30 años en una hora)

Verificación de procedimientos de decisión ecuacionales

- Principales bloques de la formalización:
  1. Retículo de términos de primer orden respecto de la subsunción.
  2. Formalización de las reducciones abstractas.
  3. Formalización de teorías ecuacionales.
- Resultados relevantes en cada uno de los bloques:
  1. Verificación formal de un algoritmo de unificación
  2. Demostración automática del lema de Newman.
  3. Demostración automática del teorema de pares críticos de Knuth y Bendix.

Algoritmo de unificación (Reglas de Martelli y Montanari)

$$\{t \approx t\} \cup R; T \Rightarrow_u R; T$$

$$\{x \approx t\} \cup R; T \Rightarrow_u \bot \quad \text{si } x \in \mathcal{V}(t) \text{ y } x \neq t$$

$$\{x \approx t\} \cup R; T \Rightarrow_u \theta(R); \{x \approx t\} \cup \theta(T) \quad \text{si } x \in X,\ x \notin \mathcal{V}(t) \text{ y } \theta = \{x \mapsto t\}$$

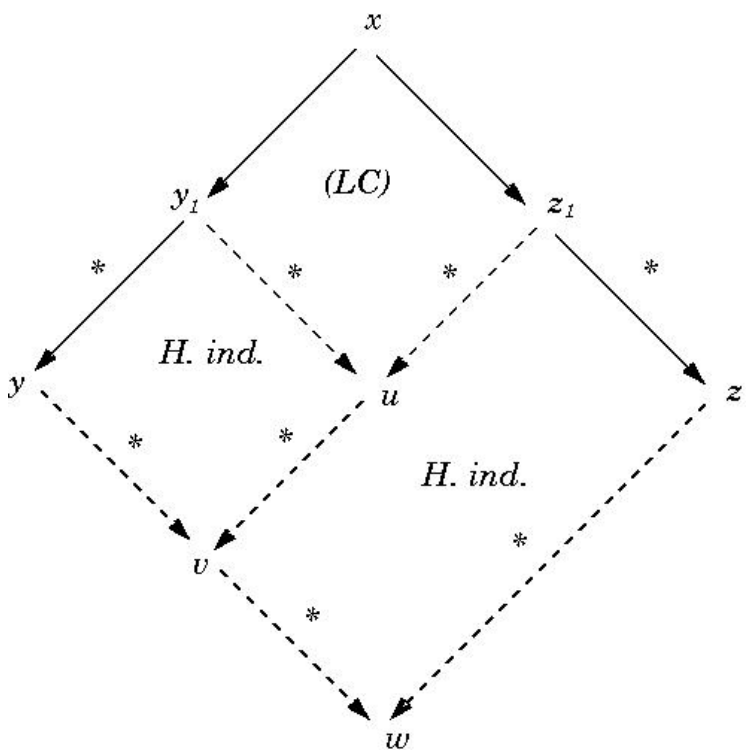
$$\{f(s_1, \ldots, s_n) \approx f(t_1, \ldots, t_n)\} \cup R; T \Rightarrow_u \{s_1 \approx t_1, \ldots, s_n \approx t_n\} \cup R; T$$

$$\{f(s_1, \ldots, s_n) \approx g(t_1, \ldots, t_m)\} \cup R; T \Rightarrow_s \bot, \text{ si } n \neq m \text{ ó } f \neq g$$

$$\{t \approx x\} \cup R; T \Rightarrow_u \{x \approx t\} \cup R; T \quad \text{si } x \in X,\ t \notin X$$

Lema de Newman

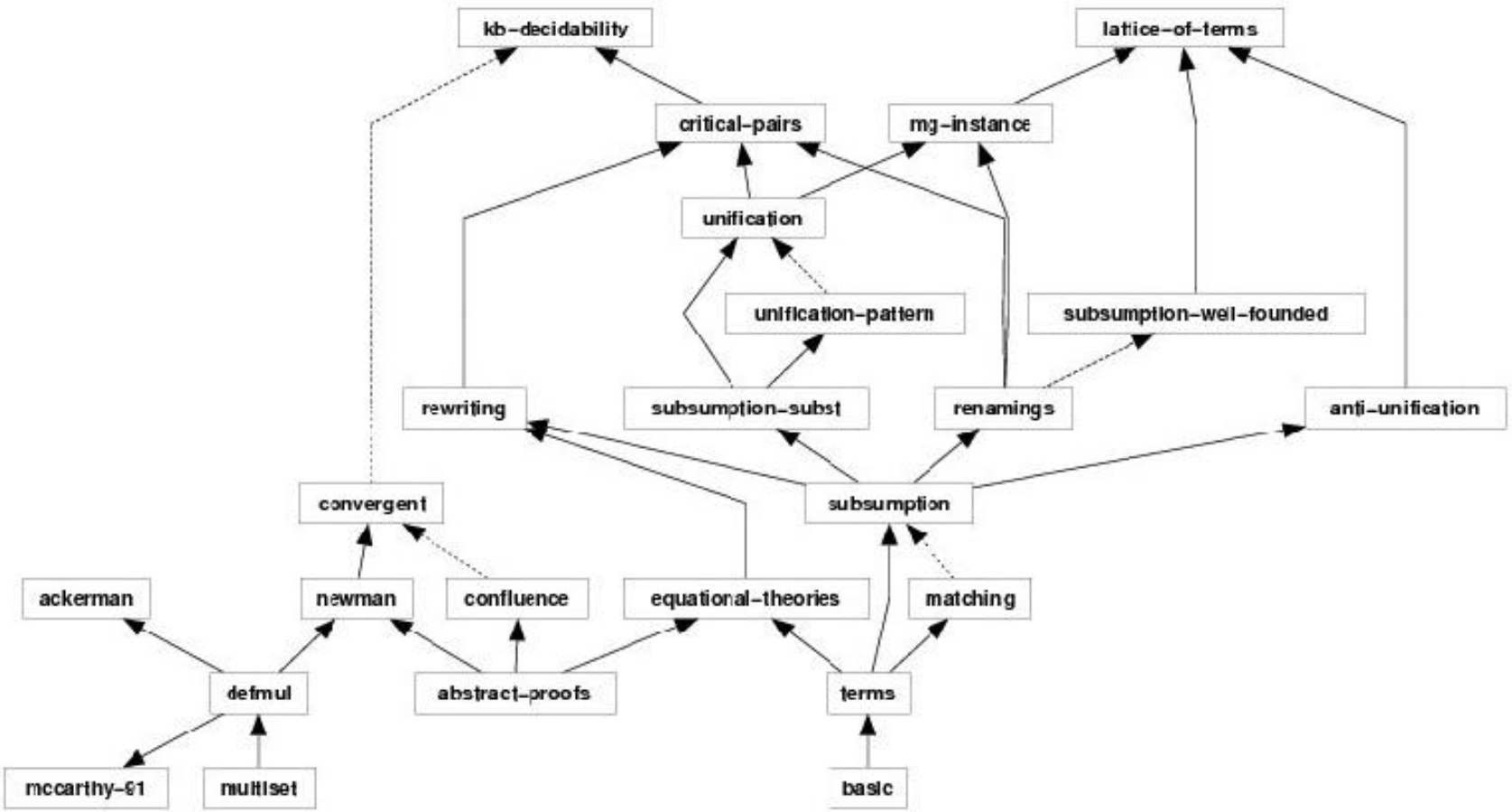
- Noetherianidad (NT): no existe $x_1 \to x_2 \to x_3 \ldots$
- Confluencia local (LC): $y \leftarrow x \to z \Longrightarrow y \overset{*}{\to} u \overset{*}{\leftarrow} z$
- Church-Rosser (C-R): $y \overset{*}{\leftrightarrow} z \Longrightarrow y \overset{*}{\to} u \overset{*}{\leftarrow} z$
- Lema de Newman: (NT) + (LC) $\Longrightarrow$ (C-R)



Teorema de pares críticos de Knuth y Bendix

- Una regla de reescritura $(l \to r)$ en $T(\Sigma, X)$ es un par ordenado (l, r) tal que $l, r \in T(\Sigma, X)$, $\mathcal{V}(r) \subseteq \mathcal{V}(l)$ y $l \notin X$.
- (u, v) es un par crítico determinado por la superposición de $l_2 \approx r_2$ sobre $l_1 \approx r_1$ en la posición p si
  - $l_1 \approx r_1$ y $l_2 \approx r_2$ son dos ecuaciones en $T(\Sigma, X)$,
  - $l_1' \approx r_1'$ y $l_2' \approx r_2'$ son dos ecuaciones con variables separadas y equivalentes, respectivamente, a las ecuaciones $l_1 \approx r_1$ y $l_2 \approx r_2$,
  - $p \in \mathcal{P}(l_1)$ (y por tanto $p \in \mathcal{P}(l_1')$) es tal que $l_1/p \notin X$,
  - σ es unificador de máxima generalidad de l_1'/p y l_2',
  - $u = \sigma(r_1')$ y $v = \sigma(l_1'[p \leftarrow r_2'])$.
- Ejemplo: Un par crítico determinado por la superposición de $i(x) * x \approx e$ sobre $(x * y) * z \approx x * (y * z)$ en la posición 1 es $(e * z, i(v) * (v * z))$.
- Teorema de pares críticos de Knuth y Bendix: $\to_E$ es localmente confluente si para cualquier $(u, v) \in pc(E)$, se tiene $u \downarrow_E v$.

Verificación de procedimientos de decisión ecuacionales



Verificación de procedimientos de decisión ecuacionales

Libro	Líneas	Definiciones	Teoremas	Consejos
basic	378	22	79	2
terms	770	53	76	12
matching	325	7	48	8
subsumption	295	13	29	18
subsumption–subst	327	16	38	13
Subtotal	2095	111	270	53
renamings	578	9	64	25
subsumption–well–founded	216	3	30	7
anti–unification	434	10	37	6
unification–pattern	808	7	105	33
unification	277	12	24	8
mg–instance	159	3	17	11
lattice–of–terms	148	17	20	5
Subtotal	2620	61	297	95
multiset	576	24	69	16
defmul	652	24	14	13
ackermann	127	10	10	4
mccarthy–91	110	10	13	3
Subtotal	1465	68	106	36
abstract–proofs	152	18	17	0
confluence	242	13	32	7
newman	451	15	56	10
convergent	269	22	20	8
Subtotal	1114	68	125	25
equational–theories	219	12	24	7
rewriting	528	18	54	13
critical–pairs	1710	54	152	41
kb–decidability	370	17	24	7
Subtotal	2827	101	254	68
Total	10121	409	1052	277

Verificación de procedimientos de decisión ecuacionales

1999 Mechanical verification of a rule–based unification algorithm in the Boyer–Moore theorem prover. En Joint Conference on Declarative Programming.

2000 Multiset relations: a tool for proving termination. En ACL2 Workshop.

2000 A mechanical proof of Knuth–Bendix critical pair theorem (using ACL2). En FTP'2000.

2000 Formalizing rewriting in the ACL2 theorem prover. En AISC 2000.

2001 Una teoría computacional acerca de la lógica ecuacional (formalización n ACL2 de la lógica ecuacional y demostración automática de sus propiedades). Tesis Univ. de Sevilla

2002 Formal proofs about rewriting using ACL2. En Annals of Mathematics and Artificial Intelligence.

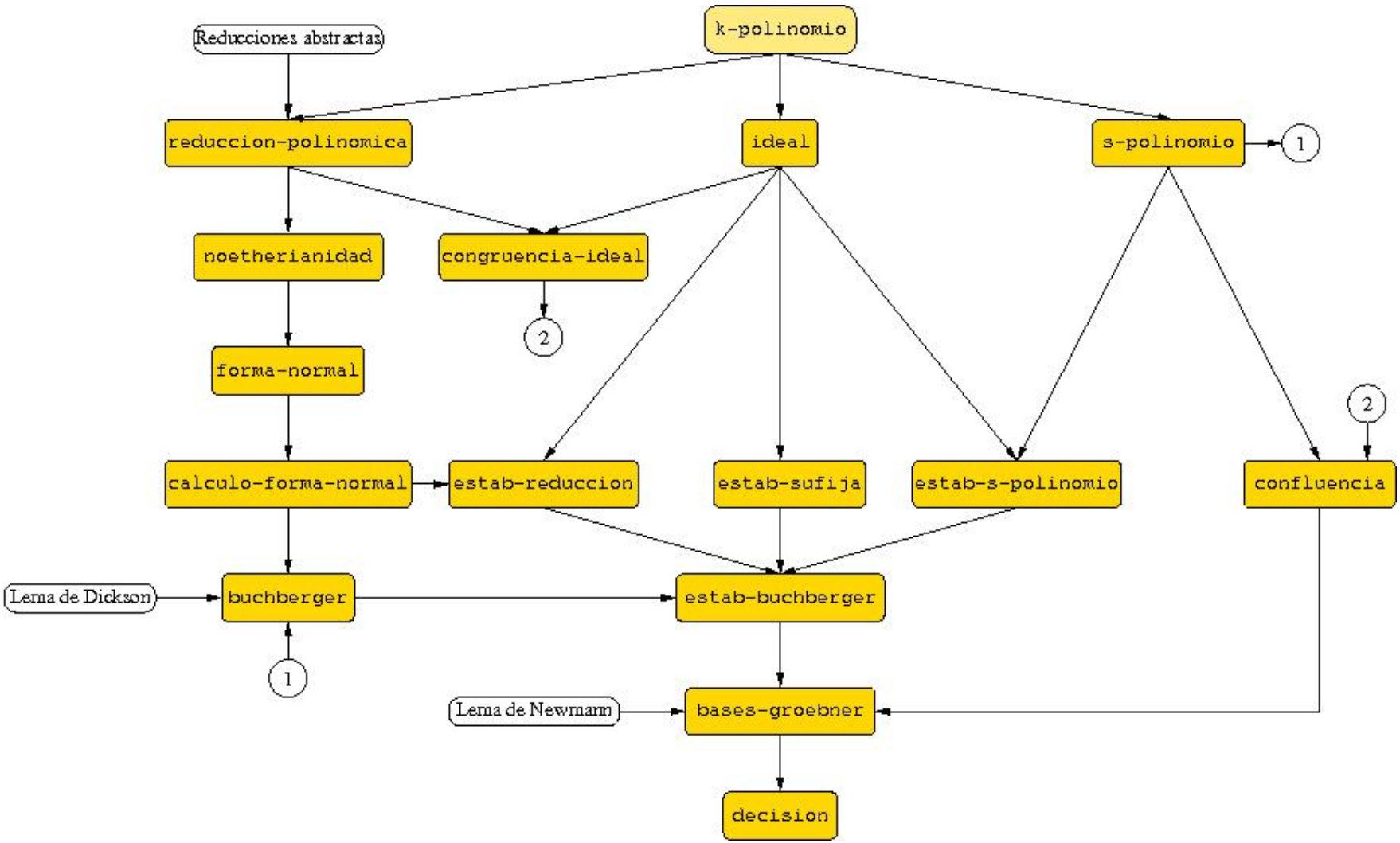
2003 Termination in ACL2 using multiset relation. En Thirty Five Years of Automating Mathematics.

Lógica computacional en Sevilla (30 años en una hora)

El algoritmo de Buchberber

- $Buchberger : F \to G$
  $G \leftarrow F$
  $C \leftarrow [(f_i, f_j) : 1 \leq i < j \leq n]$
  mientras $C \neq \emptyset$
  $(p, q) \leftarrow primero(C)$
  $C \leftarrow resto(C)$
  $h \leftarrow red^*_F(Spolinomio(p, q))$
  si $h \neq 0$ $C \leftarrow [(h, f_i) : f_i \in G] + +C$
  $G \leftarrow h : G$
- El algoritmo de Buchberger termina.
- El algoritmo de Buchberger devuelve una base de Gröbner:
  $$p \in ideal(F) \iff p \to^*_{Buchberger(F)} 0$$
- El algoritmo de Buchberger da un procedimiento de decisión:
  $$p \in ideal(F) \iff red^*_{Buchberger(F)}(p) = 0$$

Verificación de procedimientos de decisión polinomiales



Verificación de procedimientos de decisión polinomiales

Libro	Líneas	Def.	Teoremas	Pistas
reduccion–polinomica	123	17	7	3
noetherianidad	278	2	35	25
congruencia–ideal	602	18	51	31
forma–normal	88	4	9	4
calculo–forma–normal	188	8	37	18
ideal	152	8	26	8
s–polinomio	54	2	7	5
buchberger	183	12	19	8
estabilidad–reduccion	64	0	10	2
estabilidad–s–polinomio	18	0	1	1
estabilidad–sufija	67	2	14	2
estabilidad–buchberger	95	3	20	2
confluencia	687	15	57	25
bases–groebner	560	17	51	31
decision	26	1	2	2
Total	3185	109	346	167

Verificación de procedimientos de decisión polinomiales

Libro	Líneas	Def.	Teoremas	Pistas
Directorio polinomios				
coeficiente	185	7	36	12
termino	115	7	16	2
monomio	135	14	24	8
polinomio	17	5	1	0
forma–normal	145	8	23	4
suma	46	1	11	3
congruencias–suma	22	0	5	2
opuesto	81	1	13	8
producto	105	5	19	9
congruencias–producto	120	1	18	4
Subtotal	971	49	166	52
Directorio polinomios–normalizados				
polinomio–normalizado	167	9	26	16
orden	35	2	6	0
Subtotal	202	11	32	16
Directorio polinomios–racionales				
racional	281	9	61	27
termino	137	9	20	3
termino–division	125	3	27	0
monomio	176	16	38	7
polinomio	46	5	1	1
forma–normal	147	7	10	7
suma	63	2	5	5
congruencias–suma	26	0	10	10
opuesto	52	2	3	2
producto	82	5	9	7
congruencias–producto	11	0	4	4
polinomio–normalizado	267	12	44	22
orden	95	3	5	5
defun–k	81	15	0	0
pertenencia	312	3	57	25
k–polinomio	410	18	68	47
Subtotal	2311	109	362	172
Total	3484	169	560	240

Verificación de procedimientos de decisión polinomiales

2000 Automatic verification of polynomial rings: fundamental properties in ACL2. En ACL2 Workshop 2000.

2001 A certified polynomial–based decision procedure for propositional logic. En TPHOLs 2001.

2002 Implementation in ACL2 of well-founded polynomial orderings. En ACL2 Workshop 2002.

2002 Algoritmos polinómicos en ACL2 (Una aproximación al algoritmo de Buchberger). En IDEIA/IBERAMIA 2002.

2003 A formal proof of Dickson's lemma in ACL2. En LPAR 2003.

2003 Verificación formal en ACL2 del algoritmo de Buchberger. Tesis Univ. de Sevilla.

2004 Verified computer algebra in ACL2 (Gröbner bases computation). En AISC 2004.

2009 A verified Common Lisp implementation of Buchberger´s algorithm in ACL2. En JSC 2009.

Lógica computacional en Sevilla (30 años en una hora)

Verificación de algoritmos eficientes

- Motivación: Verificar programas ACL2 **eficientes**.
- Medio: Objetos de hebra simple (stobj).
- Resultados:

2002 Progress report: term dags using stobjs. En Workshop on ACL2.
2002 Verificación formal y eficiencia: Un caso de estudio aplicado a la unificación de términos. En IDEIA/IBERAMIA.
2003 Formal reasoning about efficient data structures: A case study in ACL2. LOPSTR.
2004 A Formally Verified Quadratic Unification Algorithm. En Workshop on ACL2.
2006 Formal correctness of a quadratic unification algorithm. En Journal of Automated Reasoning.
2008 Efficient execution in an automated reasoning environment. En Journal of Functional Programming.

Lógica computacional en Sevilla (30 años en una hora)

Simulación de computación no–convencional en ACL2

- Objetivos:
  - Formalizar en ACL2 el modelo de computación molecular de Adleman.
  - Implementar en el modelo el experimento de Lipton para resover el problema SAT.
  - Demostrar en ACL2 la adecuación y completitud.
- Resultados:

2002 Molecular computation models in ACL2: a simulation of Lipton's experiment solving SAT. En Workshop on ACL2.

2003 Formal verification of molecular computational models in ACL2: A case study. En CAEPIA.

2003 Especificación y verificación de programas moleculares en PVS. Tesis Univ. de Sevilla.

Lógica computacional en Sevilla (30 años en una hora)

Historia de PVS

- PVS:
  - Nombre: *P*rototype *V*erification *S*ystem
  - Autores: N. Shankar, S. Owre y J.M. Rushby (SRI, USA)
  - Def: "PVS is a verification system: that is, a specification language integrated with support tools and a theorem prover"
  - Historia: HDM (1970), EHDM (1984), PVS (1991), PVS 4.2 (2008).
- Propósitos:
  - The primary purpose of PVS is to provide formal support for conceptualization and debugging in the early stages of the lifecycle of the design of a hardware or software system
  - The primary emphasis in the PVS proof checker is on supporting the construction of readable proofs

El problema de las monedas

- Enunciado: Demostrar que con monedas de 3 y 5 se puede obtener cualquier cantidad que sea mayor o igual a 8.
- Especificación

```
monedas : THEORY
BEGIN
 n, a, b: VAR nat
 monedas: LEMA
   (FORALL n: (EXISTS a, b: n+8 = 3*a + 5*b))
END monedas
```

El problema de las monedas

- Demostración manual: Por inducción en n
  - Base $n = 0$: $0 + 8 = 3 \times 1 + 5 \times 1$. Basta elegir $a = b = 1$
  - Paso $n + 1$: Supongamos que existen a y b tales que $n + 8 = 3 \times a + 5 \times b$. Vamos a distinguir dos casos según que $b = 0$.

    Caso 1 Sea $b = 0$, entonces $n + 8 = 3 \times a$, $a > 3$ y
    $$(n + 1) + 8 = 3 \times (a - 3) + 5 \times 2$$
    Caso 2 Sea $b \neq 0$, entonces
    $$(n + 1) + 8 = 3 \times (a + 2) + 5 \times (b - 1)$$

El problema de las monedas

- Demostración con PVS

```
monedas :

  |-------
{1}   (FORALL n: (EXISTS a, b: n + 8 = 3 * a + 5 * b))

Rule? (induct "n")
Inducting on n on formula 1, this yields  2 subgoals:
monedas.1 :

  |-------
{1}   EXISTS a, b: 0 + 8 = 3 * a + 5 * b

Rule? (inst 1 1 1)
Instantiating the top quantifier in 1 with the terms: 1, 1,
this simplifies to:
```

El problema de las monedas

```
monedas.1 :

  |-------
{1}   0 + 8 = 3 * 1 + 5 * 1

Rule? (assert)
Simplifying, rewriting, and recording with decision procedures,
This completes the proof of monedas.1.

monedas.2 :

  |-------
{1}   FORALL j:
        (EXISTS a, b: j + 8 = 3 * a + 5 * b) IMPLIES
         (EXISTS a, b: j + 1 + 8 = 3 * a + 5 * b)

Rule? (skosimp*)
Repeatedly Skolemizing and flattening, this simplifies to:
```

El problema de las monedas

```
monedas.2 :
{-1}  j!1 + 8 = 3 * a!1 + 5 * b!1
  |-------
{1}   EXISTS a, b: j!1 + 1 + 8 = 3 * a + 5 * b

Rule? (case "b!1=0")
Case splitting on b!1 = 0, this yields  2 subgoals:
monedas.2.1 :
{-1}  b!1 = 0
[-2]  j!1 + 8 = 3 * a!1 + 5 * b!1
  |-------
[1]   EXISTS a, b: j!1 + 1 + 8 = 3 * a + 5 * b

Rule? (inst 1 "a!1-3" "2")
Instantiating the top quantifier in 1 with the terms:
a!1-3, 2, this yields  2 subgoals:
```

El problema de las monedas

```
monedas.2.1.1 :
[-1]  b!1 = 0
[-2]  j!1 + 8 = 3 * a!1 + 5 * b!1
  |-------
{1}   j!1 + 1 + 8 = 3 * (a!1 - 3) + 5 * 2

Rule? (assert)
Simplifying, rewriting, and recording with decision procedure

This completes the proof of monedas.2.1.1.
```

El problema de las monedas

```
monedas.2.1.2 (TCC):
[-1]  b!1 = 0
[-2]  j!1 + 8 = 3 * a!1 + 5 * b!1
  |-------
{1}   a!1 - 3 >= 0

Rule? (assert)
Simplifying, rewriting, and recording with decision procedure
This completes the proof of monedas.2.1.2.
This completes the proof of monedas.2.1.
```

El problema de las monedas

```
monedas.2.2 :
[-1]  j!1 + 8 = 3 * a!1 + 5 * b!1
   |-------
{1}   b!1 = 0
[2]   EXISTS a, b: j!1 + 1 + 8 = 3 * a + 5 * b

Rule? (inst 2 "a!1+2" "b!1-1")
Instantiating the top quantifier in 2 with the terms:
a!1+2, b!1-1, this yields  2 subgoals:
```

El problema de las monedas

```
monedas.2.2.1 :
[-1]  j!1 + 8 = 3 * a!1 + 5 * b!1
   |-------
[1]   b!1 = 0
{2}   j!1 + 1 + 8 = 3 * (a!1 + 2) + 5 * (b!1 - 1)

Rule? (assert)
Simplifying, rewriting, and recording with decision procedure

This completes the proof of monedas.2.2.1.
```

El problema de las monedas

```
monedas.2.2.2 (TCC):

[-1]  j!1 + 8 = 3 * a!1 + 5 * b!1
  |-------
{1}   b!1 - 1 >= 0
[2]   b!1 = 0

Rule? (assert)
Simplifying, rewriting, and recording with decision procedures,

This completes the proof of monedas.2.2.2.

This completes the proof of monedas.2.2.

This completes the proof of monedas.2.
Q.E.D.
```

Demostración PVS

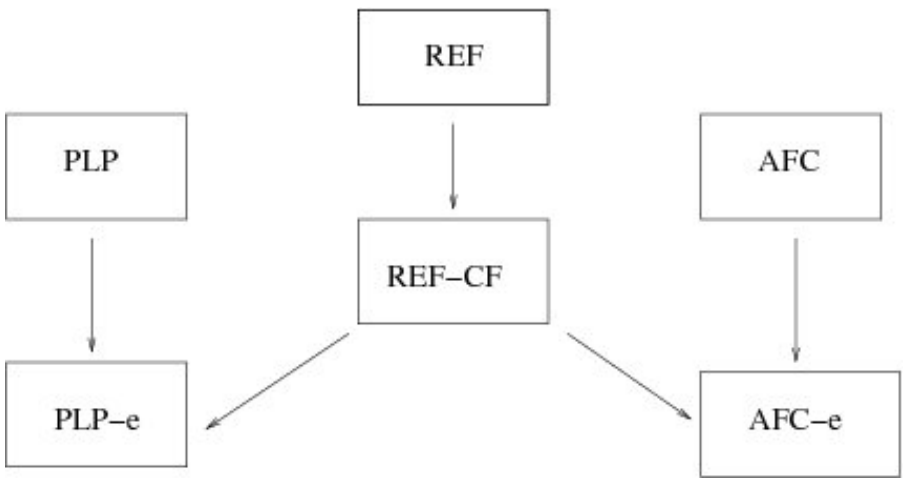
- Llamada: `M-x edit-proof`
- Demostración

```
("" 
 (INDUCT "n")
 (("1" (INST 1 1 1) (ASSERT))
  ("2"
   (SKOSIMP*)
   (CASE "b!1=0")
   (("1" (INST 1 "a!1-3" "2") (("1" (ASSERT)) ("2" (ASSERT))))
    ("2" (INST 2 "a!1+2" "b!1-1") (("1" (ASSERT)) ("2" (ASSERT)))
```

Lógica computacional en Sevilla (30 años en una hora)

Desarrollo de teorías computacionales con PVS: PLP y AFC

- Motivación:
  - Aumentar la potencia expresiva.
  - Obtener especificaciones evaluables vía refinamientos.
- Teorías desarrolladas:
  - Programación lógica proposicional.
  - Análisis formal de conceptos.
- Estructura del estudio realizado:



Desarrollo de teorías computacionales con PVS: PLP y AFC

- Un modelo formal de la lógica proposicional con PVS:
  - Sintaxis
  - Semántica proposicional
  - Cláusulas de Horn
  - Semántica de cláusulas
- Semántica de los programas lógicos en PVS:
  - Semántica declarativa.
  - Semántica del punto fijo.
  - Semántica procedimental.
- Un modelo del análisis formal de conceptos en PVS:
  - Introducción.
  - El retículo de los conceptos.
  - Algoritmo de generación de los conceptos.
  - Implicaciones entre atributos.
  - Base de Duquenne–Guigues.

Desarrollo de teorías computacionales con PVS: PLP y AFC

- Marco genérico para refinamientos en PVS:
  - Refinamiento de tipos.
  - Refinamiento de operaciones.
  - Caso de estudio: un refinamiento de conjuntos finitos.
- Una formalización evaluable de la PLP:
  - Algoritmo de cálculo de la base de Herbrand.
  - Algoritmo de cálculo del menor modelo de Herbrand.
  - Algoritmo de cálculo del operador de consecuencia.
  - Algoritmo de cálculo del menor punto fijo.
  - Algoritmos de resolución SLD.
- Una formalización evaluable del AFC.
  - Representación de los contextos formales.
  - Especificaciones evaluables de los operadores de derivación.
  - Refinamiento del tipo de los conceptos.
  - Especificaciones evaluables del ínfimo y el supremo.
  - Algoritmo de cálculo de los conceptos.

Desarrollo de teorías computacionales con PVS: PLP y AFC

Biblioteca	Lemas	Sintaxis	Tipos	Pruebas
Conjuntos	66	0.01	27.29	9.69
Listas	114	0.00	22.19	16.66
Marco para refinamiento	234	0.01	150.52	53.63
Programación lógica proposicional	479	0.01	564.26	202.64
Análisis formal de conceptos	296	0.02	335.29	191.00

Lógica computacional en Sevilla (30 años en una hora)

Desarrollo de teorías computacionales con PVS: ALC

- Sintaxis y semántica de ALC.
  - Sintaxis de ALC.
  - Los conceptos de ALC.
  - Definiciones sobre la sintaxis de ALC.
  - Conocimiento terminológico y asertivo.
  - Semántica de los conceptos de ALC.
  - Semántica de las bases de conocimiento.
- Razonamiento en ALC.
  - Forma normal negativa.
  - Árboles de expansión de una caja–A.
  - Propiedades de las expansiones de las cajas–A.
  - Adecuación del proceso.
  - Completitud y terminación.
  - Medida de terminación.

Desarrollo de teorías computacionales con PVS: ALC

- Algoritmo evaluable de ALC.
  - Refinamiento del tipo de los conceptos de ALC.
  - Refinamiento de la sintaxis de ALC.
  - Refinamiento de la semántica de ALC.
  - Refinamiento del proceso de completación de ALC.
  - Refinamiento del tipo de las activaciones.
  - Algoritmo genérico de satisfacibilidad para ALC.
  - Procedimiento de decisión de satisfacibilidad para ALC.
- Desarrollo literario de teorías en PVS.

Lógica computacional en Sevilla (30 años en una hora)

Publicaciones sobre teorías computacionales con PVS

2002 Una introducción al análisis formal de conceptos en PVS. En IDEIA/IBERAMIA.

2004 Teoría computacional (en PVS) de la programación lógica y del análisis formal de conceptos. Tesis Univ. de Sevilla.

2004 Verification of the formal concept analysis. En RACSAM (Revista de la Real Academia de Ciencias).

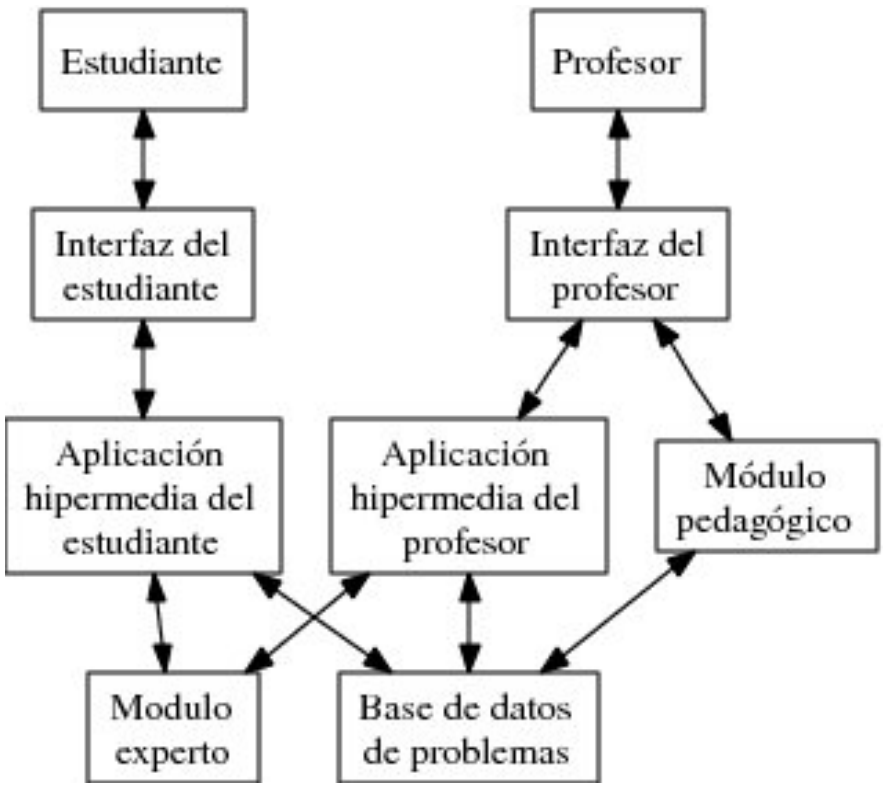
2005 A formally verified proof (in PVS) of the strong completeness theorem of propositional SLD–resolution. En EUROCAST.

2007 A formally verified prover for the ALC description logic. En TPHOLs.

2008 Constructing formally verified reasoners for the ALC description logic. En ENTCS.

Aplicación para la enseñanza de la RCyR

- Problema: Dificultades para enseñar la representación de argumentos en lógica de primer orden con igualdad.
- Solución: APLI^2 = APLIcación de Ayuda Para Lógica Informática
- Arquitectura del APLI2



Aplicación para la enseñanza de la RCyR

- Sistemas de RA usados: OTTER y MACE.
- Publicaciones:

  2006 FITS: Formalization With an Intelligent Tutor System. En Current Developments in Technology–Assisted Education.

  2007 KRRT: Knowledge Representation and Reasoning Tutor System. En EUROCAST.
- Asignaturas que lo usan:
  - "Lógica informática" desde el curso 2007–08.
  - "Razonamiento automático" desde el curso 2008–09.
- Para hacer:
  - Editor de demostraciones por tableros.
  - Minería de datos pedagógicos.

Lógica computacional en Sevilla (30 años en una hora)

El sistema Isabelle/HOL/Isar

- Isabelle es un marco lógico basado en lógica de orden superior.
- Isabelle/HOL es la especialización de Isabelle para HOL (Higher-Order Logic).
- Isar (Intelligible semi–automated reasoning) es un lenguaje de pruebas para Isabelle.
- Isabelle/HOL se ha desarrollado en la Universidad de Cambridge (Larry Paulson) y la Universidad de Munich (Tobias Nipkow).
- Isar se ha desarrollado en Universidad de Munich (Markus Wenzel).

Ejemplo de prueba en Isabelle/Isar

- Teorema de Cantor: Sea X cualquier conjunto y $\wp(X)$ su conjunto potencia. Entonces no existen aplicaciones suprayectivas de X en $\wp(X)$.
- Teorema de Cantor: Para cualquier función f de X en $\wp(X)$, existe un conjunto S tal que $S \notin \mathit{rango}(f)$.
- Representación en Isabelle:

```
theorem Cantor:
  fixes f :: "'α ⇒ 'α set"
  shows "∃S. S ∉ range f"
```

Ejemplo de prueba en Isabelle/Isar

- Teorema de Cantor: Sea X cualquier conjunto y $\wp(X)$ su conjunto potencia. Entonces no existen aplicaciones suprayectivas de X en $\wp(X)$.
- Teorema de Cantor: Para cualquier función f de X en $\wp(X)$, existe un conjunto S tal que $S \notin \mathit{rango}(f)$.
- Representación en Isabelle:

```
theorem Cantor:
  fixes f :: "'α ⇒ 'α set"
  shows "∃S. S ∉ range f"
```

Ejemplo de prueba en Isabelle/Isar

- Teorema de Cantor: Sea X cualquier conjunto y $\wp(X)$ su conjunto potencia. Entonces no existen aplicaciones suprayectivas de X en $\wp(X)$.
- Teorema de Cantor: Para cualquier función f de X en $\wp(X)$, existe un conjunto S tal que $S \notin \mathit{rango}(f)$.
- Representación en Isabelle:

```
theorem Cantor:
  fixes f :: "'α ⇒ 'α set"
  shows "∃S. S ∉ range f"
```

Demostración a nivel de primero de Matemáticas

Demostración natural

Sea $A = \{x.x \notin f(x)\}$. Vamos a demostrar que $A \notin \mathit{rango}(f)$. Lo haremos por reducción al absurdo. Supongamos que $A \in \mathit{rango}(f)$, entonces existe un b tal que $A = f(b)$. Vamos a demostrar una contradición distinguiendo dos casos:

- Caso 1: Supongamos que $b \in A$. Entonces, $b \notin f(b)$ (por la definición de A) y $b \notin A$ (por la definición de b). Contradicción con $b \in A$.
- Caso 2: Supongamos que $b \notin A$. Entonces, $b \in f(b)$ (por la definición de A) y $b \in A$ (por la definición de b). Contradicción con $b \notin A$.

Demostración a nivel de primero de Matemáticas

Demostración verificada en Isabelle/Isar

```
proof
  let ?A = "{x. x ∉ f x}"
  show "?A ∉ range f"
  proof
    assume "?A ∈ range f"
    then obtain b where "?A = f b" ..
    show False
    proof cases
      assume "b ∈ ?A"
      hence "b ∉ f b" by simp
      hence "b ∉ ?A" using `?A = f b` by simp
      thus False using `b ∈ ?A` by contradiction
    next
      assume "b ∉ ?A"
      hence "b ∈ f b" by simp
      hence "b ∈ ?A" using `?A = f b` by simp
      thus False using `b ∉ ?A` by contradiction
    qed
  qed
qed
```

Demostración a nivel de tercero de Matemáticas

Demostración natural

Consideremos el conjunto $A = \{x.x \notin f(x)\}$. Vamos a demostrar que $A \notin \mathit{rango}(f)$. Lo haremos por reducción al absurdo. Supongamos que $A \in \mathit{rango}(f)$, entonces existe un b tal que $A = f(b)$. Se obtiene una contradición distinguiendo dos casos según que $b \in A$.

Demostración a nivel de tercero de Matemáticas

Demostración verificada en Isabelle/Isar

```
proof
  let ?A = "{x. x ∉ f x}"
  show "?A ∉ range f"
  proof
    assume "?A ∈ range f"
    then obtain a where "?A = f a" ..
    show False
    proof cases
      assume "a ∈ ?A"
      show False using `?A = f a` by blast
    next
     assume "a ∉ ?A"
     show False using `?A = f a` by blast
    qed
```

Demostración a nivel de quinto de Matemáticas

- Teorema de Cantor: Para cualquier función f de X en $\wp(X)$, existe un conjunto S tal que $S \notin \mathit{rango}(f)$.
- Demostración natural:
  Trivial.
- Teorema y demostración verificada en Isabelle/Isar

```
theorem Cantor_3:
  fixes f :: "'α ⇒ 'α set"
  shows "∃A. A ∉ range f"
by best
```

Demostración a nivel de quinto de Matemáticas

- Teorema de Cantor: Para cualquier función f de X en $\wp(X)$, existe un conjunto S tal que $S \notin \mathit{rango}(f)$.
- Demostración natural:
  Trivial.
- Teorema y demostración verificada en Isabelle/Isar

```
theorem Cantor_3:
  fixes f :: "'α ⇒ 'α set"
  shows "∃A. A ∉ range f"
by best
```

Lógica computacional en Sevilla (30 años en una hora)

Formalización de pruebas en Isabelle/Isar

- Publicaciones
  - 2008 Introducción a la demostración asistida por ordenador (con Isabelle/Isar).
  - 2008 Elementos de matemáticas formalizadas en Isabelle/Isar. Trabajo de investigación de Doctorado.
  - 2009 Formalización de la demostración de completitud de la lógica proposicional en Isabell/Isar. En CLAI/CAEPIA.
- Objetivo: Verificar resultados de la metateoría de la lógica proposicional y de primer orden.

Lógica computacional en Sevilla (30 años en una hora)

Trabajos futuros

- Verificación de Kenzo
  - 2009 "ACL2 verification of simplicial degeneracy programs in the Kenzo system". En Calculemus.
  - 2009 "Polinomios simpliciales: una herramienta para la formalización de la Topología Simplicial en ACL2". En CLAI/CAEPIA.
  - 2009 "Verificación y eficiencia en programas para el cálculo simbólico: estudio de un caso". En PROLE.
- Verificación de sistema de decisión proposicional basado en polinomios.
  - 2008 "Extending attribute exploration by means of Boolean derivates". En CLA.
  - 2009 "Conservative Retractions of Propositional Logic Theories by Means of Boolean Derivatives: Theoretical Foundations". En Calculemus.
  - 2009 "Sistema certificado de decisión proposicional basado en polinomios". En CLAI/CAEPIA.

Trabajos futuros

- Verificación de sistemas de razonamiento para la Web semántica:
  - Realizado: Verificación en PVS se un sistema de razonamiento para ALC.
  - Adaptarla a Isabelle/Isar y comparar las formalizaciones.
  - Extenderla a lógicas descriptivas más expresivas.
- Verificación de la metateoría de la lógica clásica.
  - Realizado: Verificación en Isabelle/Isar de la adecuación y completitud de un sistema axiomático de la lógica proposicional.
  - Ampliarlo a lógica de primer orden.
  - Ampliarlo a tableros semánticos (orden de multiconjuntos).
- Aplicaciones a la enseñanza:
  - Continuar el desarrollo de APLI2.
  - Desarrollar cursos escritos en Isabelle/Isar.

Lógica computacional en Sevilla (30 años en una hora)

Comentarios finales sobre alternativas frecuentes

1. Verificación de programas o desarrollo computacional de teorías matemáticas.
2. Eficiencia de los programas o simplicidad en su verificación.
3. Verificación de programas o generación de código certificado.
4. Demostración automática (ATP) o interactiva (ITP).
5. Sistemas de razonamiento de núcleo pequeño o grande.
6. Lógicas elementales (SAT, LPO) o superiores (HOL).
7. Un sistema de razonamiento o varios.
8. Desarrollo descendente o ascendente.
9. Desarrollo individual o colectivo.
10. Verdad o belleza.
11. Sueño o realidad.