MAXIMA: el paquete logic algebra

por francisco miguel garcía olmedo

INICIATIVA POR EL SOFTWARE LIBRE

27 de abril de 2011

Tabla de Contenidos

introducción

Clave: como se ha visto, el álgebra de Boole provee las herramientas para entender la parte lógica de los circuitos de conmutación clásicos.

El `álgebra de boole` guarda un estrecho parentesco con la `lógica proposicional`.

introducción

Clave: como se ha visto, el álgebra de Boole provee las herramientas para entender la parte lógica de los circuitos de conmutación clásicos.

El `álgebra de boole` guarda un estrecho parentesco con la `lógica proposicional`.

- Si en el álgebra de las proposiciones definimos la relación de "ser equivalentes lógicamente", resulta una relación de equivalencia compatible con las operaciones.

introducción

Clave: como se ha visto, el álgebra de Boole provee las herramientas para entender la parte lógica de los circuitos de conmutación clásicos.

El `álgebra de boole` guarda un estrecho parentesco con la `lógica proposicional`.

- Si en el álgebra de las proposiciones definimos la relación de "ser equivalentes lógicamente", resulta una relación de equivalencia compatible con las operaciones.
- el cociente de dicha álgebra por esta relación de equivalencia compatible, es un álgebra de Boole especialmente "sencilla".

introducción

- Debido a esta relación y una vez conocida, el ámbito de trabajo lógico a nivel proposiconal puede ser transparente hasta cierto punto.

introducción

- Debido a esta relación y una vez conocida, el ámbito de trabajo lógico a nivel proposiconal puede ser transparente hasta cierto punto.
- Para el trabajo en lógica proposiconal–algebra de Boole usaremos —y cargaremos desde MAXIMA— el paquete logic.lisp.

introducción

- Los operadores reconocidos son los siguientes:

operador	tipo	preced.	descripción

introducción

- Los operadores reconocidos son los siguientes:

operador	tipo	preced.	descripción
`not`	prefijo	70	negación (NEG, $\neg$)

introducción

- Los operadores reconocidos son los siguientes:

operador	tipo	preced.	descripción
`not`	prefijo	70	negación (NEG, $\neg$)
`and`	n-ario	65	conjunción (AND, $\wedge$)

introducción

- Los operadores reconocidos son los siguientes:

operador	tipo	preced.	descripción
`not`	prefijo	70	negación (NEG, $\neg$)
`and`	n-ario	65	conjunción (AND, $\wedge$)
`nand`	n-ario	62	barra de Sheffer ($\vert$)

introducción

- Los operadores reconocidos son los siguientes:

operador	tipo	preced.	descripción
`not`	prefijo	70	negación (NEG, $\neg$)
`and`	n-ario	65	conjunción (AND, $\wedge$)
`nand`	n-ario	62	barra de Sheffer ($\mid$)
`nor`	n-ario	61	NOR ($\downarrow$)

introducción

- Los operadores reconocidos son los siguientes:

operador	tipo	preced.	descripción
`not`	prefijo	70	negación (NEG, $\neg$)
`and`	n-ario	65	conjunción (AND, $\wedge$)
`nand`	n-ario	62	barra de Sheffer ($\mid$)
`nor`	n-ario	61	NOR ($\downarrow$)
`or`	n-ario	60	diyunción (OR, $\vee$)

introducción

- Los operadores reconocidos son los siguientes:

operador	tipo	preced.	descripción
`not`	prefijo	70	negación (NEG, $\neg$)
`and`	n-ario	65	conjunción (AND, $\wedge$)
`nand`	n-ario	62	barra de Sheffer ($\vert$)
`nor`	n-ario	61	NOR ($\downarrow$)
`or`	n-ario	60	diyunción (OR, $\vee$)
`implies`	infijo	59	implicación ($\rightarrow$)

introducción

- Los operadores reconocidos son los siguientes:

operador	tipo	preced.	descripción
`not`	prefijo	70	negación (NEG, $\neg$)
`and`	n-ario	65	conjunción (AND, $\wedge$)
`nand`	n-ario	62	barra de Sheffer ($\mid$)
`nor`	n-ario	61	NOR ($\downarrow$)
`or`	n-ario	60	diyunción (OR, $\vee$)
`implies`	infijo	59	implicación ($\to$)
`eq`	n-ario	58	equivalencia ($\sim$)

introducción

- Los operadores reconocidos son los siguientes:

operador	tipo	preced.	descripción
`not`	prefijo	70	negación (NEG, $\neg$)
`and`	n-ario	65	conjunción (AND, $\wedge$)
`nand`	n-ario	62	barra de Sheffer ($\mid$)
`nor`	n-ario	61	NOR ($\downarrow$)
`or`	n-ario	60	diyunción (OR, $\vee$)
`implies`	infijo	59	implicación ($\rightarrow$)
`eq`	n-ario	58	equivalencia ($\sim$)
`xor`	n-ario	58	o exclusivo (XOR, $\oplus$)

Tabla de Contenidos

funciones básicas

Clave: trataremos la tabla de la función asociada a una expresión, la equivalencia lógica y el polinomio de Gegalkine.

Son funciones relacionadas con la semántica de las expresiones.

funciones básicas

Clave: trataremos la tabla de la función asociada a una expresión, la equivalencia lógica y el polinomio de Gegalkine.

Son funciones relacionadas con la semántica de las expresiones.

`characteristic_vector(expr,var_1, ...,var_n)`:

- `characteristic_vector(expr,var_1,...,var_n)` devuelve una lista de tamaño 2^n con todos los valores posibles de `expr`.

funciones básicas

Clave: trataremos la tabla de la función asociada a una expresión, la equivalencia lógica y el polinomio de Gegalkine.

Son funciones relacionadas con la semántica de las expresiones.

`characteristic_vector(expr,var_1, ...,var_n)`:

- `characteristic_vector(expr,var_1,...,var_n)` devuelve una lista de tamaño 2^n con todos los valores posibles de `expr`.
- `characteristic_vector(f(x,y,z),x,y,z)` equivale a:

```
[f(false,false,false),f(false,false,true),
 f(false,true,false),f(false,true,true),
 f(true,false,false),f(true,false,true),
 f(true,true,false),f(true,true,true)]
```

funciones básicas

- Si `var_1,...,var_n` es suprimido, entonces se supone `[var_1,...,var_n]=sort(listofvars(expr))`.

funciones básicas

- Si `var_1,...,var_n` es suprimido, entonces se supone `[var_1,...,var_n]=sort(listofvars(expr))`.
- Ejercicio: Calcular la tabla asociada a las expresiones: `true`, `not x`, `x or y`, `x xor y`, `x and y`, `x nand y`, `x eq y`.

funciones básicas

- Si `var_1,...,var_n` es suprimido, entonces se supone `[var_1,...,var_n]=sort(listofvars(expr))`.
- Ejercicio: Calcular la tabla asociada a las expresiones: `true`, `not x`, `x or y`, `x xor y`, `x and y`, `x nand y`, `x eq y`.
- Ejercicio: Comprobar, con una orden de MAXIMA, si coinciden `characteristic_vector(x implies y)` y `characteristic_vector(x implies y,y,x)`. Si la respuesta es `false`, explicar lo ocurrido.

funciones básicas

`zhegalkin_form (expr)`

- `zhegalkin_form (expr)` devuelve la representación de expr en la base (por medio de las operaciones) {`xor`, `and`, `true`}

funciones básicas

`zhegalkin_form (expr)`

- `zhegalkin_form (expr)` devuelve la representación de `expr` en la base (por medio de las operaciones) {`xor`, `and`, `true`}
- Bien visto, `zhegalkin_form (expr)` puede entenderse como el polinomio de Gegalkine de `expr`.

funciones básicas

`zhegalkin_form (expr)`

- `zhegalkin_form (expr)` devuelve la representación de expr en la base (por medio de las operaciones) {`xor`, `and`, `true`}
- Bien visto, `zhegalkin_form (expr)` puede entenderse como el polinomio de Gegalkine de expr.
- Ejercicio: Calcular la expresión de:

  en la base {`xor`, `and`, `true`}

funciones básicas

`zhegalkin_form (expr)`

- `zhegalkin_form (expr)` devuelve la representación de `expr` en la base (por medio de las operaciones) {`xor, and, true`}
- Bien visto, `zhegalkin_form (expr)` puede entenderse como el polinomio de Gegalkine de `expr`.
- Ejercicio: Calcular la expresión de:
  - x or y or z

en la base {`xor, and, true`}

funciones básicas

`zhegalkin_form (expr)`

- `zhegalkin_form (expr)` devuelve la representación de expr en la base (por medio de las operaciones) {`xor`, `and`, `true`}
- Bien visto, `zhegalkin_form (expr)` puede entenderse como el polinomio de Gegalkine de expr.
- Ejercicio: Calcular la expresión de:
  - x or y or z
  - (x implies y) or z

  en la base {`xor`, `and`, `true`}

funciones básicas

`logic_equiv (expr_1,expr_2)`

- `logic_equiv (expr_1,expr_2)` devuelve `true` en caso de que sean equivalentes `expr_1` y `expr_2` y `false` en caso contrario.

funciones básicas

`logic_equiv (expr_1,expr_2)`

- `logic_equiv (expr_1,expr_2)` devuelve `true` en caso de que sean equivalentes expr_1 y expr_2 y `false` en caso contrario.
- ¿Qué diálogo nos dará el siguiente guión?

```
(%i1) load ("logic.lisp")$
(%i2) e:((x or y) xor z) and u);
(%i3) zhegalkin_form (e);
(%i4) logic_equiv (%i2,%o3);
(%i5) is (characteristic_vector(%i2)
                   = characteristic_vector(%o3));
(%i6) logic_equiv (x and y eq x, x implies y);
```

funciones básicas

- Ejercicio: Interprétese y explíquese el diálogo producido por el guion anterior.

funciones básicas

- Ejercicio: Interprétese y explíquese el diálogo producido por el guion anterior.
- Ejercicio: Supóngase que nos es dado un conjunto finito de fórmulas del lenguaje proposicional: $\varphi_1, \ldots, \varphi_n, \psi$. Úsese lo visto para dar respuesta al problema $\varphi_1, \ldots, \varphi_n \models \psi$.

formas normales

Clave: trataremos las formas normales conjuntiva y disyuntiva de expresiones booleanas.

Son funciones que preparan las expresiones para un de las uso sintáctico posterior.

formas normales

Clave: trataremos las formas normales conjuntiva y disyuntiva de expresiones booleanas.

Son funciones que preparan las expresiones para un de las uso sintáctico posterior.

`boolean_form(expr)`

- `boolean_form(expr)` devuelve la expresión de `expr` en la base `{and,or,not}`.

formas normales

Clave: trataremos las formas normales conjuntiva y disyuntiva de expresiones booleanas.

Son funciones que preparan las expresiones para un de las uso sintáctico posterior.

`boolean_form(expr)`

- `boolean_form(expr)` devuelve la expresión de `expr` en la base {`and`,`or`,`not`}.
- Como se observará, si `expr` es una expresión booleana expresada en la base {`and`,`or`,`not`}, `demorgan(expr)` aplica las leyes de De Morgan a `expr`.

formas normales

- Por ejemplo:

```
(%i1) load ("logic.lisp")$
(%i2) boolean_form (a implies b implies c);
(%o2) (not ((not a) or b)) or c
(%i3) demorgan (%);
(%o3) ((not b) and a) or c
(%i4) logic_equiv(boolean_form(a implies b implies c),
      zhegalkin_form (a implies b implies c));
(%o4) true
(%i5) demorgan (boolean_form (a nor b nor c));
(%o5) (not a) and (not b) and (not c)
```

formas normales

Clave: las formas normales conjuntiva y/o disyuntiva de expresiones. . .

intervienen de forma esencial en algoritmos como el de Quine-MacCluskey (fnd y fnd), Davis-Putnam (fnc), forma prenexa y de Skolem (fnc), etc.

formas normales

Clave: las formas normales conjuntiva y/o disyuntiva de expresiones. . .

intervienen de forma esencial en algoritmos como el de Quine-MacCluskey (fnd y fnd), Davis-Putnam (fnc), forma prenexa y de Skolem (fnc), etc.

`fdnf (expr)` y `fcnf (expr)`

- `fdnf (expr)` devuelve la forma normal disyuntiva de `expr`.

formas normales

Clave: las formas normales conjuntiva y/o disyuntiva de expresiones. . .

intervienen de forma esencial en algoritmos como el de Quine-MacCluskey (fnd y fnd), Davis-Putnam (fnc), forma prenexa y de Skolem (fnc), etc.

`fdnf (expr)` y `fcnf (expr)`

- `fdnf (expr)` devuelve la forma normal disyuntiva de `expr`.
- `fcnf (expr)` devuelve la forma normal conjuntiva de `expr`.

formas normales

- Por ejemplo:

```
(%i1) load ("logic.lisp")$
(%i2) pdnf (x implies y);
(%o2) (x and y) or ((not x) and y)
                or ((not x) and (not y))
(%i2) pcnf (x implies y);
(%o2) (not x) or y
```

formas normales

`logic_simp (expr)`

- `logic_simp (expr)` devuelve una versión simplificada equivalente de `expr`.

formas normales

logic_simp (expr)

- logic_simp (expr) devuelve una versión simplificada equivalente de expr.
- ```
  (%i1) load ("logic.lisp")$
  (%i2) logic_simp (a or (b or false or (a or b)));
  (%o2) a or b
  (%i3) logic_simp (b eq a eq false eq true);
  (%o3) eq a eq b false
  (%i4) logic_simp ((a xor true) xor b xor true);
  (%o4) a xor b
  ```
  ```

Tabla de Contenidos

dualidad

Clave: Conocemos el teorema de dualidad,

Ahora se trata de encontrar la expresión dual de otra dada. . . y cuestiones aledañas.

dualidad

Clave: Conocemos el teorema de dualidad,

Ahora se trata de encontrar la expresión dual de otra dada. . . y cuestiones aledañas.

`dual_function(expr)`

- `dual_function(expr)` devuelve la expresión dual de `expr`.

dualidad

Clave: Conocemos el teorema de dualidad,

Ahora se trata de encontrar la expresión dual de otra dada. . . y cuestiones aledañas.

`dual_function(expr)`

- `dual_function(expr)` devuelve la expresión dual de `expr`.
-
```
(%i1) load ("logic.lisp")$
(%i2) dual_function (x or y);
(%o2) not ((not x) or (not y))
(%i3) demorgan (%);
(%o3) x and y
```

dualidad

self_dual(expr)

- self_dual(expr) devuelve true cuando, y sólo cuando, expr es equivalente a dual_function(expr).

dualidad

`self_dual(expr)`

- `self_dual(expr)` devuelve `true` cuando, y sólo cuando, `expr` es equivalente a `dual_function(expr)`.
-
```
(%i1) load ("logic.lisp")$
(%i2) self_dual (a);
(%o2) true
(%i3) self_dual (not a);
(%o3) true
(%i4) self_dual (a eq b);
(%o4) false
```

Tabla de Contenidos

completitud funcional

`functionally_complete (expr_1,...,expr_n)`:

- `functionally_complete (expr_1,...,expr_n)` devuelve true sii `expr_1,...,expr_n` es un sistema funcionalmente completo.

completitud funcional

`functionally_complete (expr_1,...,expr_n)`:

- `functionally_complete (expr_1,...,expr_n)` devuelve true sii `expr_1,...,expr_n` es un sistema funcionalmente completo.
- Considérese el siguiente diálogo:

```
(%i1) load ("logic.lisp")$
(%i2) functionally_complete (x and y, x xor y);
(%o2) false
(%i3) functionally_complete (x and y, x xor y, true);
(%o3) true
(%i4) functionally_complete (x and y, x or y, not x);
(%o4) true
```

completitud funcional

`logic_basis(expr_1,...,expr_n)`:

- `logic_basis (expr_1,...,expr_n)` devuelve true sii `expr_1,...,expr_n` es un sistema funcionalmente completo minimal (base lógica).

completitud funcional

`logic_basis(expr_1,...,expr_n)`:

- `logic_basis (expr_1,...,expr_n)` devuelve `true` sii `expr_1,...,expr_n` es un sistema funcionalmente completo minimal (base lógica).
- Considérese el siguiente diálogo:

```
(%i1) load ("logic.lisp")$
(%i2) logic_basis (x and y, x or y);
(%o2) false
(%i3) logic_basis (x and y, x or y, not x);
(%o3) false
(%i4) logic_basis (x and y, not x);
(%o4) true
```

completitud funcional

- y su continuación:

```
(%i5) logic_basis (x or y, not x);
(%o5) true
(%i6) logic_basis (x and y, x xor y, true);
(%o6) true
```

completitud funcional

- Ejercicio: hacer un programa que dé un listado de todas las bases lógicas construidas sobre: `not x`, `x nand y`, `x nor y`, `x implies y`, `x and y`, `x or y`, `x eq y`, `x xor y`, `true`, `false`

Tabla de Contenidos

cerrado por false

closed_under_f(expr):

- closed_under_f(f(x_1,...,x_n)) devuelve true si, y sólo si, f(false,...,false)=false.

cerrado por false

closed_under_f(expr):

- closed_under_f(f(x_1,...,x_n)) devuelve true si, y sólo si, f(false,...,false)=false.
- Considérese el siguiente diálogo:

```
(%i1) load ("logic.lisp")$
(%i2) closed_under_f (x and y);
(%o2) true
(%i3) closed_under_f (x or y);
(%o3) true
(%i4) closed_under_f (x implies y);
(%o4) false
```

cerrado por true

`closed_under_t(expr)`:

- `closed_under_t(f(x_1,...,x_n))` devuelve `true` si, y sólo si, `f(true,...,true)=true`.

cerrado por true

`closed_under_t(expr)`:

- `closed_under_t(f(x_1,...,x_n))` devuelve `true` si, y sólo si, `f(true,...,true)=true`.
- Considérese el siguiente diálogo:

```
(%i1) load ("logic.lisp")$
(%i2) closed_under_t (x and y);
(%o2) true
(%i3) closed_under_f (x or y);
(%o3) true
```

cerrado por true

closed_under_t(expr):

- closed_under_t(f(x_1,...,x_n)) devuelve true si, y sólo si, f(true,...,true)=true.
- Considérese el siguiente diálogo:

```
(%i1) load ("logic.lisp")$
(%i2) closed_under_t (x and y);
(%o2) true
(%i3) closed_under_f (x or y);
(%o3) true
```

- Ejercicio: encontrar una expresión expr tal que closed_under_t(expr) dé el valor false.

monotonia

`monotonic(expr)`:

- `monotonic(expr)` devuelve true sii el vector característico de `expr` es monótono,

monotonia

monotonic(expr):

- monotonic(expr) devuelve true sii el vector característico de expr es monótono,
- es decir, en pseudocódigo:

```
charvec : characteristic_vector(expr)
charvec[i] <= charvec[i+1], i = 1, ..., n-1
      where a<=b := (a=b or (a=false and b=true)).
```

monotonia

`monotonic(expr)`:

- `monotonic(expr)` devuelve true sii el vector característico de `expr` es monótono,
- es decir, en pseudocódigo:

```
charvec : characteristic_vector(expr)
charvec[i] <= charvec[i+1], i = 1, ..., n-1
     where a<=b := (a=b or (a=false and b=true)).
```

- Considérese el siguiente diálogo:

```
(%i1) load ("logic.lisp")$
(%i2) monotonic (a or b);
(%o2) true
(%i3) monotonic (a and b);
(%o3) true
```

derivada

- y su continuación:

```
(%i1) load ("logic.lisp")$
(%i4) monotonic (a implies b);
(%o4) false
(%i5) monotonic (a xor b);
(%o5) false
```

linealidad

`linear(expr)`:

- `linear(expr)` devuelve true sii el polinomio de Gegalkine de `expr`, `zhegalkin_form(expr)`, es linear

linealidad

`linear(expr)`:

- `linear(expr)` devuelve `true` sii el polinomio de Gegalkine de `expr`, `zhegalkin_form(expr)`, es linear
- Considérese el siguiente diálogo:

```
(%i1) load ("logic.lisp")$
(%i2) linear (a or b);
(%o2) false
(%i3) linear (a eq b);
(%o3) true
(%i4) zhegalkin_form (a or b);
(%o4) (a and b) xor a xor b
(%i5) zhegalkin_form (a eq b);
(%o5) a xor b xor true
```

Tabla de Contenidos

derivada

Clave: está muy extendida la opinión de que el concepto de derivada es consustancial al de límite.

Sin embargo, la obra de EULER sugiere que podría existir el primero sin el segundo.

derivada

Clave: está muy extendida la opinión de que el concepto de derivada es consustancial al de límite.

Sin embargo, la obra de EULER sugiere que podría existir el primero sin el segundo.

- El álgebra de Boole es el escenario de un ejemplo de derivada no proveniente de un límite.

derivada

Clave: está muy extendida la opinión de que el concepto de derivada es consustancial al de límite.

Sin embargo, la obra de EULER sugiere que podría existir el primero sin el segundo.

- El álgebra de Boole es el escenario de un ejemplo de derivada no proveniente de un límite.
- La definición es la siguiente:

```
logic_diff(f(x_1,...,x_k,...,x_n),x_k) :=
  f(x_1,...,true,...,x_n)
               xor
  f(x_1,...,false,...,x_n)
```

derivada

`logic_diff(f,x)`

- `logic_diff(f,x)` devuelve `df/dx` de `f` respecto de su variable `x`.

derivada

logic_diff(f,x)

- logic_diff(f,x) devuelve df/dx de f respecto de su variable x.
- Considérese el siguiente diálogo:

```
(%i1) load ("logic.lisp")$
(%i2) logic_diff (a or b or c, a);
(%o2) (b and c) xor b xor c xor true
(%i3) logic_diff (a and b and c, a);
(%o3) b and c
(%i4) logic_diff (a or (not a), a);
(%o4) false
```

derivada

Ejercicio:

- Considerar $f(x, y) = x \to y$

derivada

Ejercicio:

- Considerar $f(x, y) = x \to y$
- Calcular `df/dx`, `df/dy` y `df/dxdy`.

derivada

Ejercicio:

- Considerar $f(x, y) = x \to y$
- Calcular `df/dx`, `df/dy` y `df/dxdy`.
- Evaluar cada una de las tres funciones anteriormente obtenida en el punto de coordenadas booleanas `(0,0)`.

derivada

Ejercicio:

- Considerar $f(x, y) = x \rightarrow y$
- Calcular `df/dx`, `df/dy` y `df/dxdy`.
- Evaluar cada una de las tres funciones anteriormente obtenida en el punto de coordenadas booleanas `(0,0)`.
- Efectuar las siguientes operaciones:

$$f(0,0) \oplus (\frac{df}{dx}(0,0) \wedge x \oplus \frac{df}{dy}(0,0) \wedge y) \oplus (\frac{df}{dxdy}(0,0) \wedge x \wedge y)$$

y si a esta nueva función boolena la representamos por $g(x, y)$

derivada

- ¿qué relación hay entre $f(x,y)$ y $g(x,y)$? Encontrarla usando MAXIMA.

derivada

- ¿qué relación hay entre $f(x, y)$ y $g(x, y)$? Encontrarla usando MAXIMA.
- ¿Qué similitud existirá entre el desarrollo de McLaurin de una función real de variable real y el polinomio de Gegalkine de una función booleana?

derivada

- ¿qué relación hay entre $f(x, y)$ y $g(x, y)$? Encontrarla usando MAXIMA.
- ¿Qué similitud existirá entre el desarrollo de McLaurin de una función real de variable real y el polinomio de Gegalkine de una función booleana?
- ¿Habrá un desarrollo en serie de Taylor para funciones boooleanas?

derivada

- ¿qué relación hay entre $f(x, y)$ y $g(x, y)$? Encontrarla usando MAXIMA.
- ¿Qué similitud existirá entre el desarrollo de McLaurin de una función real de variable real y el polinomio de Gegalkine de una función booleana?
- ¿Habrá un desarrollo en serie de Taylor para funciones boooleanas?
- ¿Habrá un concepto de integral de una función booleana?

derivada

- ¿qué relación hay entre $f(x, y)$ y $g(x, y)$? Encontrarla usando MAXIMA.
- ¿Qué similitud existirá entre el desarrollo de McLaurin de una función real de variable real y el polinomio de Gegalkine de una función booleana?
- ¿Habrá un desarrollo en serie de Taylor para funciones boooleanas?
- ¿Habrá un concepto de integral de una función booleana?
- ¿Habrá una teoría de ecuaciones diferenciales booleanas?

derivada

- El polinomio de Gegalkine de una expresión booleana es el desarrollo de McLaurin de la función asociada a la expresión.

derivada

- El polinomio de Gegalkine de una expresión booleana es el desarrollo de McLaurin de la función asociada a la expresión.
- Las tres últimas preguntas tienen respuesta afirmativa.

derivada

- El polinomio de Gegalkine de una expresión booleana es el desarrollo de McLaurin de la función asociada a la expresión.
- Las tres últimas preguntas tienen respuesta afirmativa.
- El concepto de derivada de una expresión booleana interviene de forma esencial en un tercer método de optimización de circuitos.

derivada

- El polinomio de Gegalkine de una expresión booleana es el desarrollo de McLaurin de la función asociada a la expresión.
- Las tres últimas preguntas tienen respuesta afirmativa.
- El concepto de derivada de una expresión booleana interviene de forma esencial en un tercer método de optimización de circuitos.
- Estas teorías están teniendo gran influencia en los desarrollos de nanotecnología, por ejemplo, y en el desarrollo de circuitos de entrada multivalente.