Binary Decision Diagrams (BDDs)
Model Checking Simbólico

Alicia Villanueva

Universidad Politécnica de Valencia

villanue@dsic.upv.es

Introducción

- *Model Checking* es:
  - técnica de verificación **formal** y **automática**
  - aplicada a sistemas con un número **finito** de estados

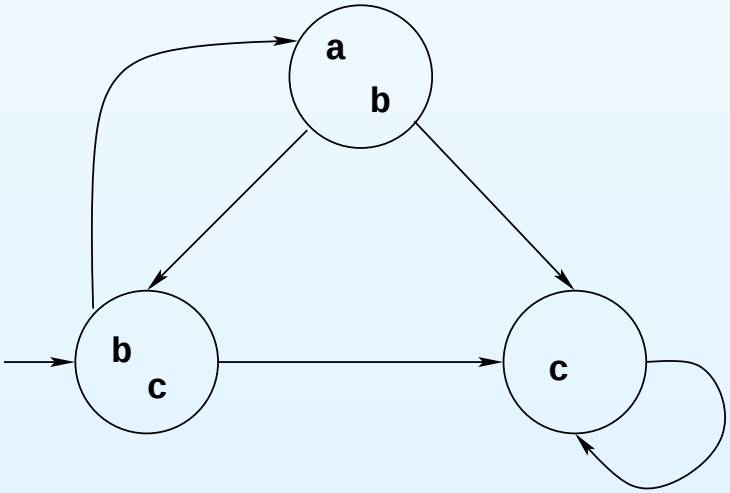
$$M \models \phi$$

Introducción

- *Model Checking* es:
  - técnica de verificación **formal** y **automática**
  - aplicada a sistemas con un número **finito** de estados

$$M \models \phi$$

- El problema de la explosión de estados
  - M es una *estructura de Kripke*

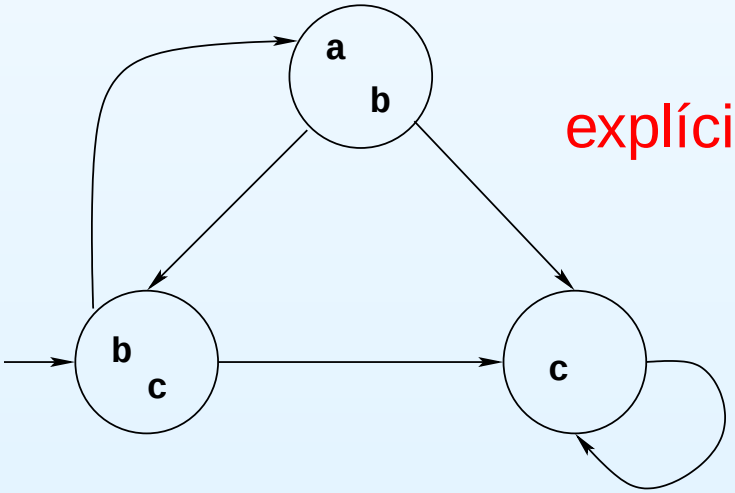


Introducción

- *Model Checking* es:
  - técnica de verificación **formal** y **automática**
  - aplicada a sistemas con un número **finito** de estados

$$M \models \phi$$

- El problema de la explosión de estados
  - M es una *estructura de Kripke*



explícita $\rightarrow$ demasiados nodos

Introducción

Soluciones al problema

- Interpretación Abstracta
- Evaluación Parcial
- Simetrías
- **Representación Simbólica**

- y más...

Introducción

Soluciones al problema

- Interpretación Abstracta
- Evaluación Parcial
- Simetrías
- **Representación Simbólica**
  - Clave 1: transformación de Estructuras de Kripke en fórmulas booleanas
  - Clave 2: algoritmos eficientes para manejar *Binary Decision Diagrams*
- y más...

Contenido de la charla

- Diagramas de Decisión Binarios (BDDs)
- Lógica CTL
- Algoritmo de *model checking* Simbólico
- El *model checker* SMV
- *Model checking* simbólico en ccp

Diagramas de Decisión Binarios (BDDs)

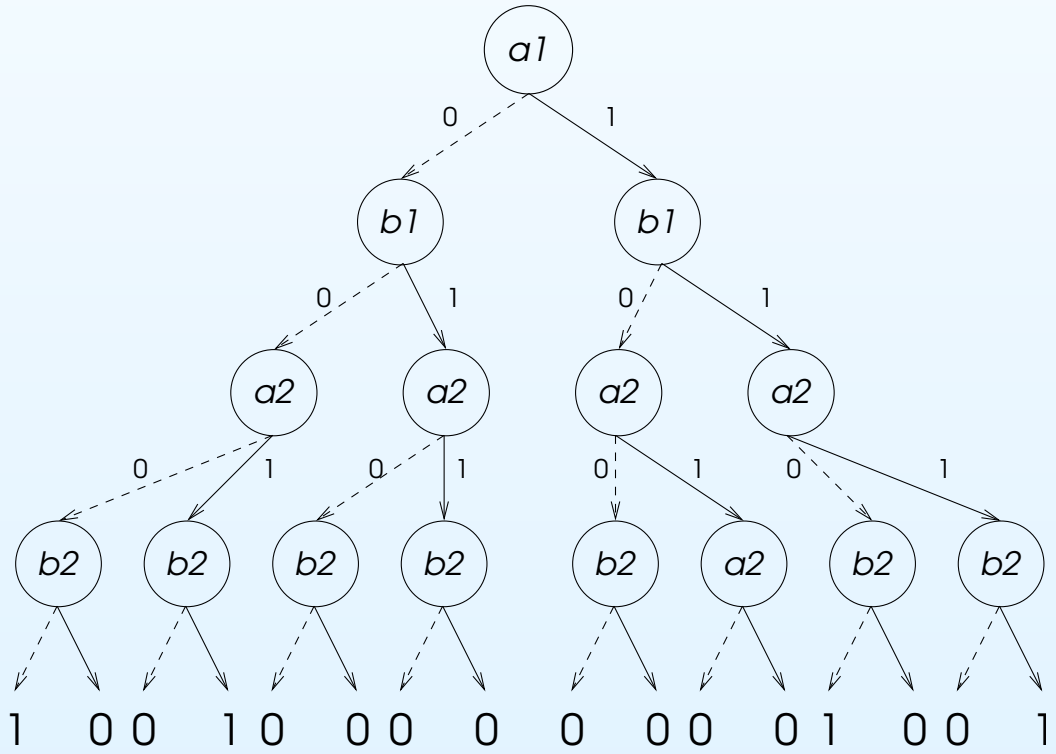
Introducción

- Sirven para representar simbólicamente los estados del sistema
- Introducidos en los años 70 por Bryant para representar funciones booleanas de forma compacta
- Los Diagramas de Decisión Binarios **Ordenados** (OBDDs) son representaciones canónicas de las funciones
- Son equivalentes a los autómatas de cadenas con alfabeto $\mathbb{B} = \{0, 1\}$

Diagramas de Decisión Binarios (BDDs)

Árboles de Decisión Binarios (BDTs)

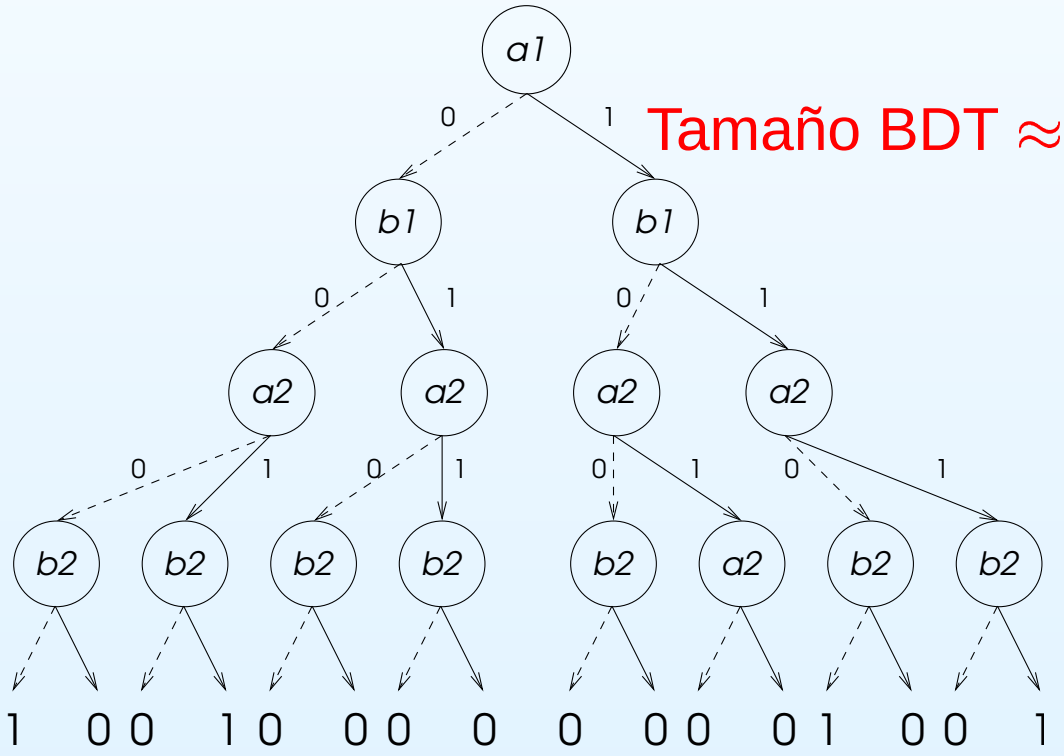
$$(a_1 \leftrightarrow b_1) \wedge (a_2 \leftrightarrow b_2)$$



Diagramas de Decisión Binarios (BDDs)

Árboles de Decisión Binarios (BDTs)

$$(a_1 \leftrightarrow b_1) \wedge (a_2 \leftrightarrow b_2)$$



Tamaño BDT $\approx$ Tabla de Verdad

Diagramas de Decision Binarios (BDDs)

Bryant demostró que

- si fijamos el orden de las variables (OBDDs) y
- eliminamos las redundancias

tenemos una representación **canónica** de la fórmula

Diagramas de Decision Binarios (BDDs)

Bryant demostró que

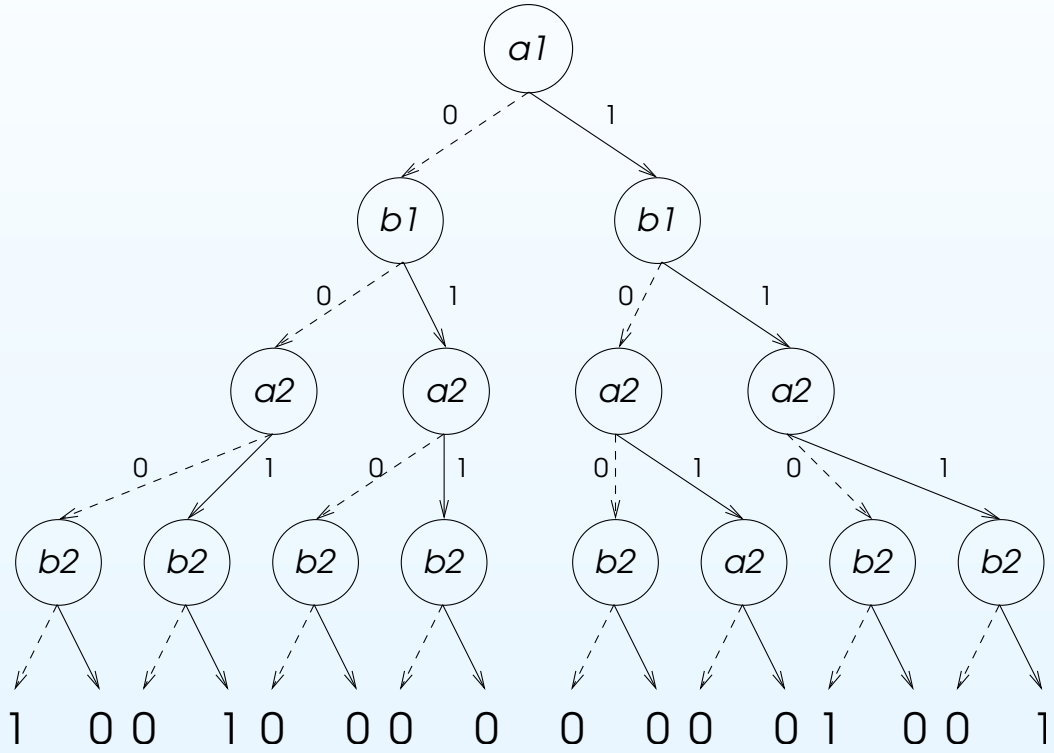
- si fijamos el orden de las variables (OBDDs) y
- eliminamos las redundancias

tenemos una representación **canónica** de la fórmula

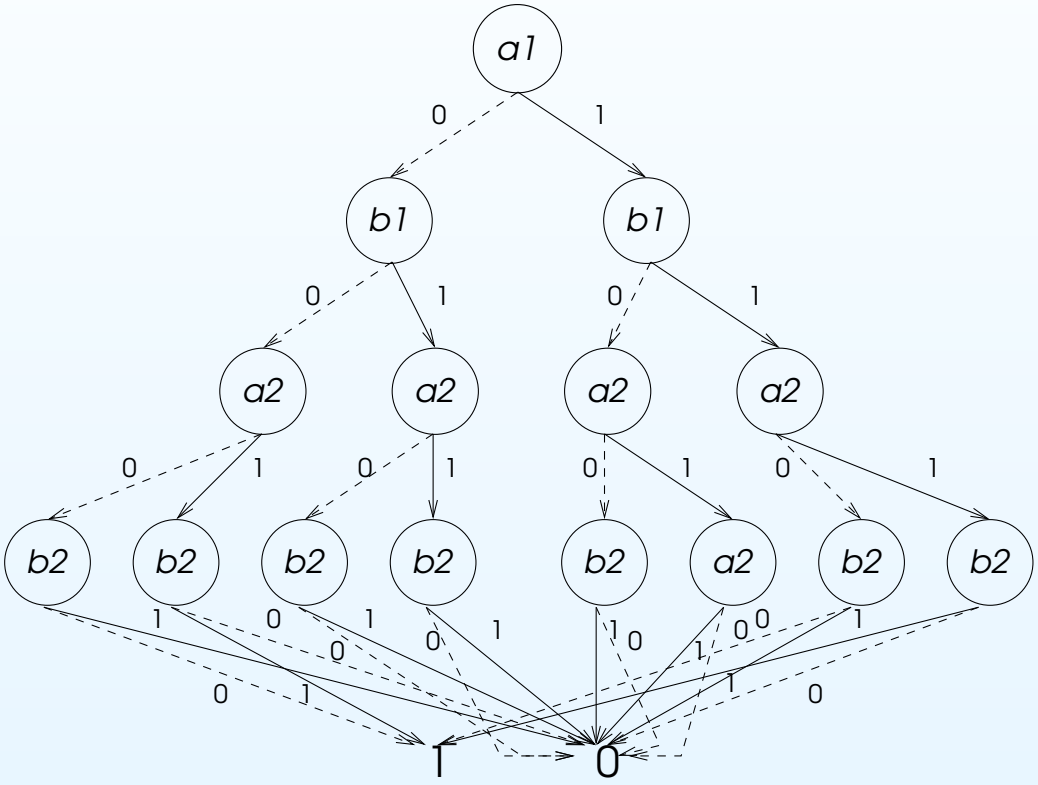
Eliminar las redundancias aplicando las reglas:

1. Quitar los nodos terminales repetidos $\Rightarrow$ dejamos un solo nodo $\mathbf{0}$ y uno $\mathbf{1}$
2. Quitar los nodos no terminales repetidos (coinciden la variable y los hijos)
3. Quitar tests redundantes (coinciden los dos hijos)

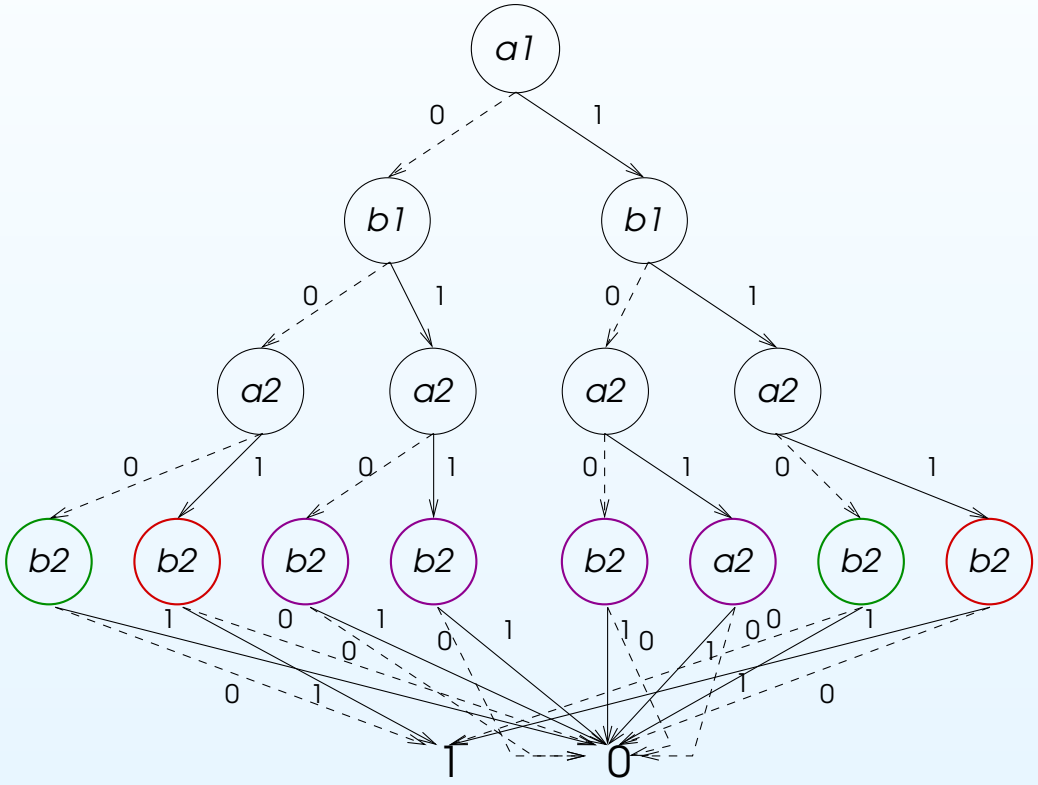
Diagramas de Decisión Binarios (BDDs)



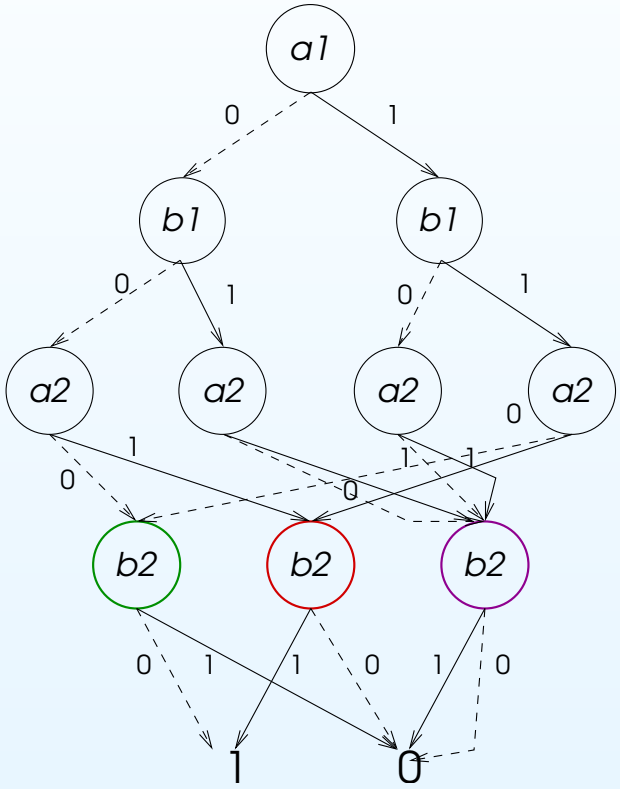
Diagramas de Decisión Binarios (BDDs)



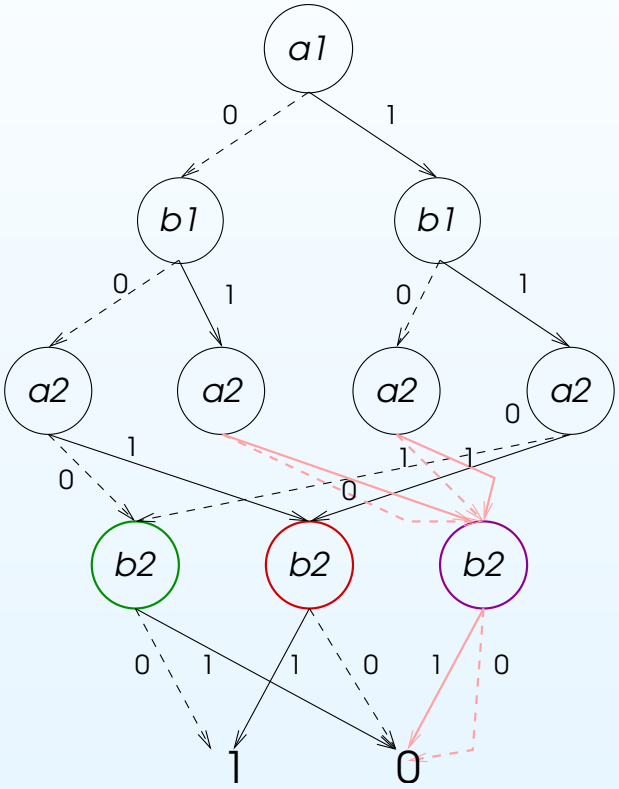
Diagramas de Decisión Binarios (BDDs)



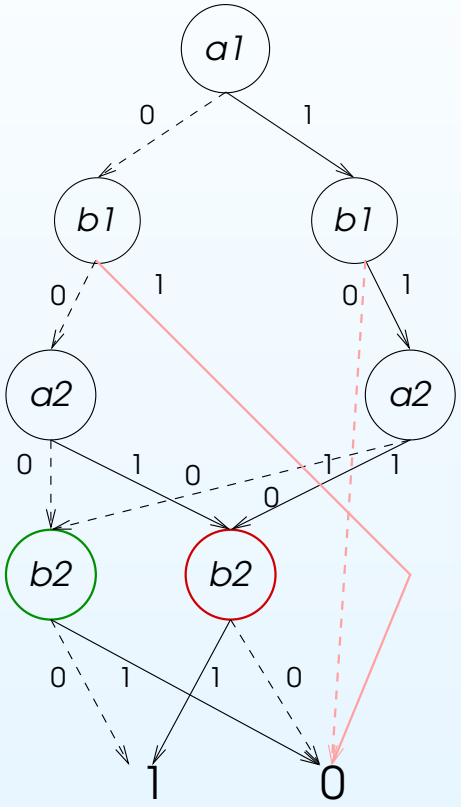
Diagramas de Decisión Binarios (BDDs)



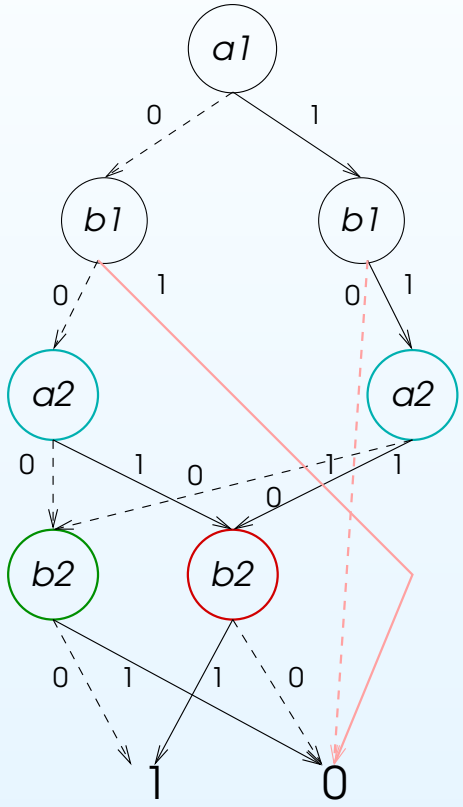
Diagramas de Decisión Binarios (BDDs)



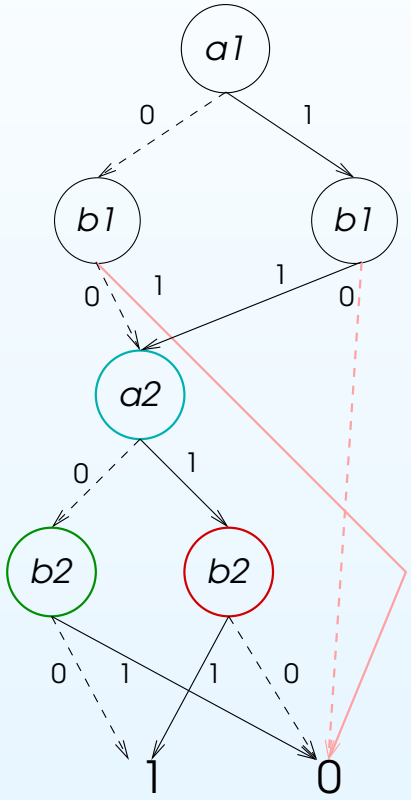
Diagramas de Decisión Binarios (BDDs)



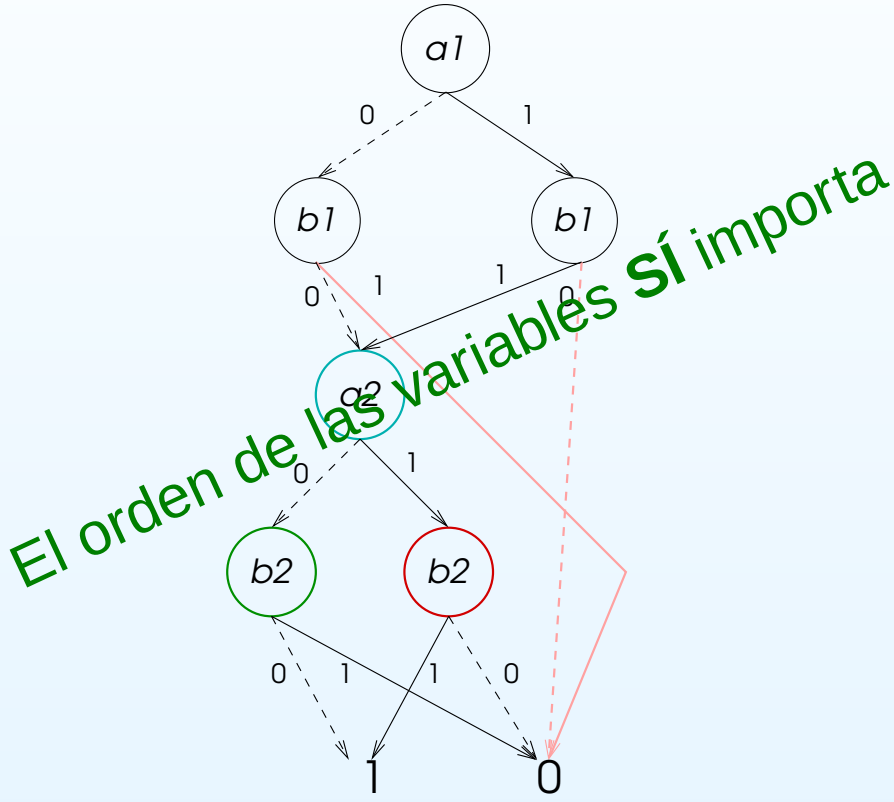
Diagramas de Decisión Binarios (BDDs)



Diagramas de Decisión Binarios (BDDs)



Diagramas de Decisión Binarios (BDDs)



El orden de las variables **SÍ** importa

Encontrar el mejor orden es NP-completo

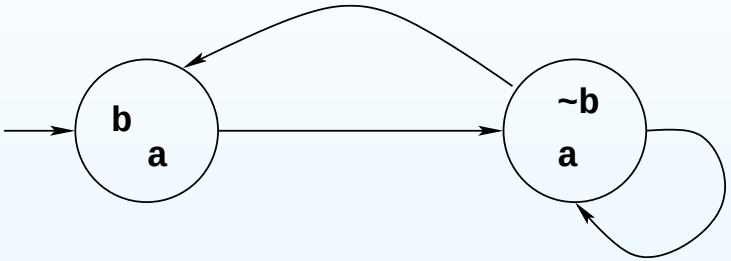
Diagramas de Decisión Binarios (BDDs)

Operaciones

Algoritmo	Complejidad Temporal
Reduce	$O(\|G\| \cdot \log \|G\|)$
Apply	$O(\|G_1\| \cdot \|G_2\|)$
Restrict	$O(\|G\| \cdot \log \|G\|)$
Compose	$O(\|G_1\|^2 \cdot \|G_2\|)$
Satisfy-one	$O(n)$
Satisfy-all	$O(n \cdot \|S_f\|)$
Satisfy-count	$O(\|G\|)$

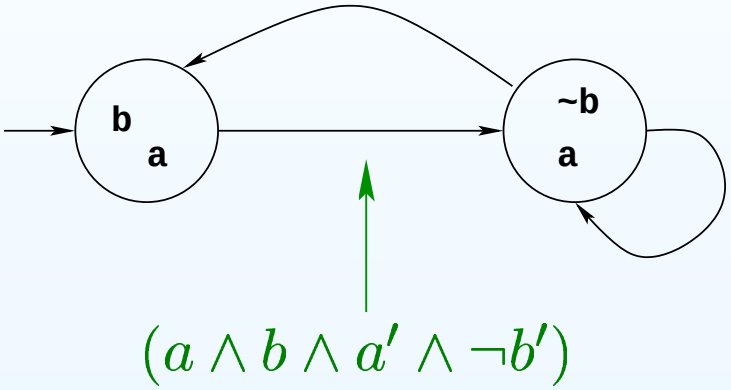
Representación de Estructuras de Kripke

Un OBDD puede representar una estructura de Kripke



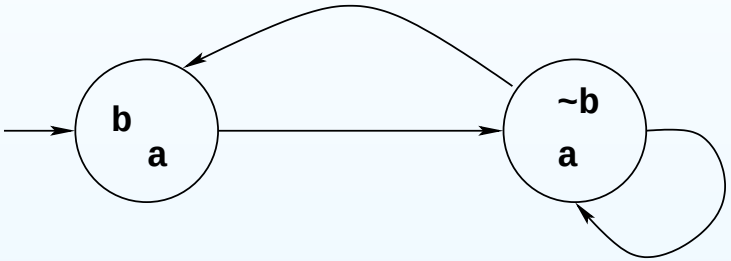
Representación de Estructuras de Kripke

Un OBDD puede representar una estructura de Kripke



Representación de Estructuras de Kripke

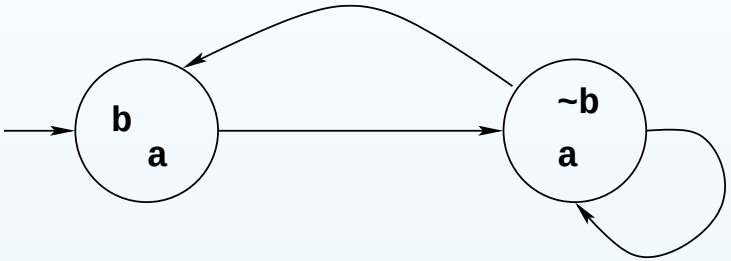
Un OBDD puede representar una estructura de Kripke



$$(a \wedge b \wedge a' \wedge \neg b') \vee (a \wedge \neg b \wedge a' \wedge \neg b') \vee (a \wedge \neg b \wedge a' \wedge b')$$

Representación de Estructuras de Kripke

Un OBDD puede representar una estructura de Kripke



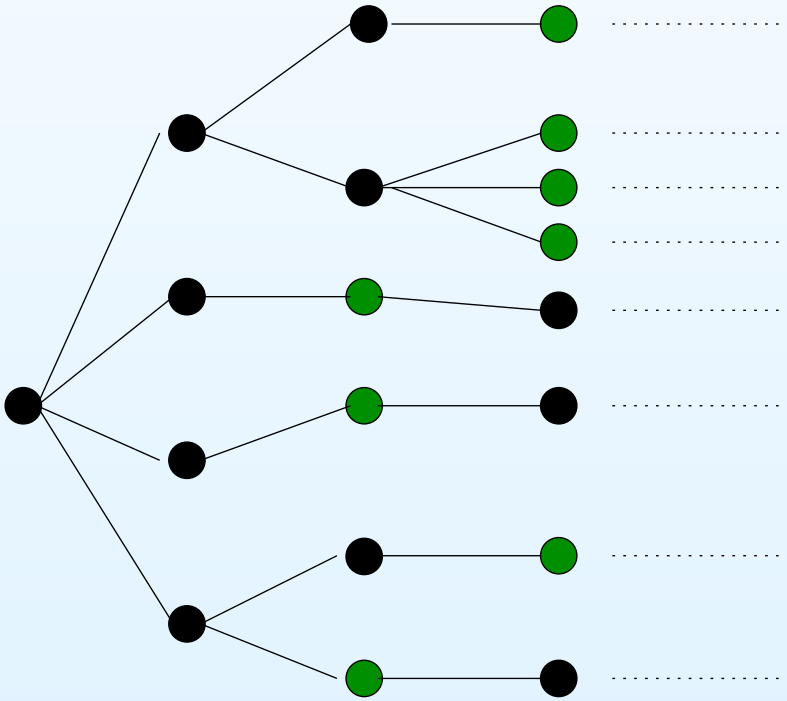
$$(a \wedge b \wedge a' \wedge \neg b') \vee (a \wedge \neg b \wedge a' \wedge \neg b') \vee (a \wedge \neg b \wedge a' \wedge b')$$

- variables booleanas
- tenemos algoritmos para construir un OBDD a partir de la fórmula
- en realidad NO se parte de la estructura de Kripke

Lógica CTL (*Computation Tree Logic*)

- Lógica temporal ramificada
- Operadores

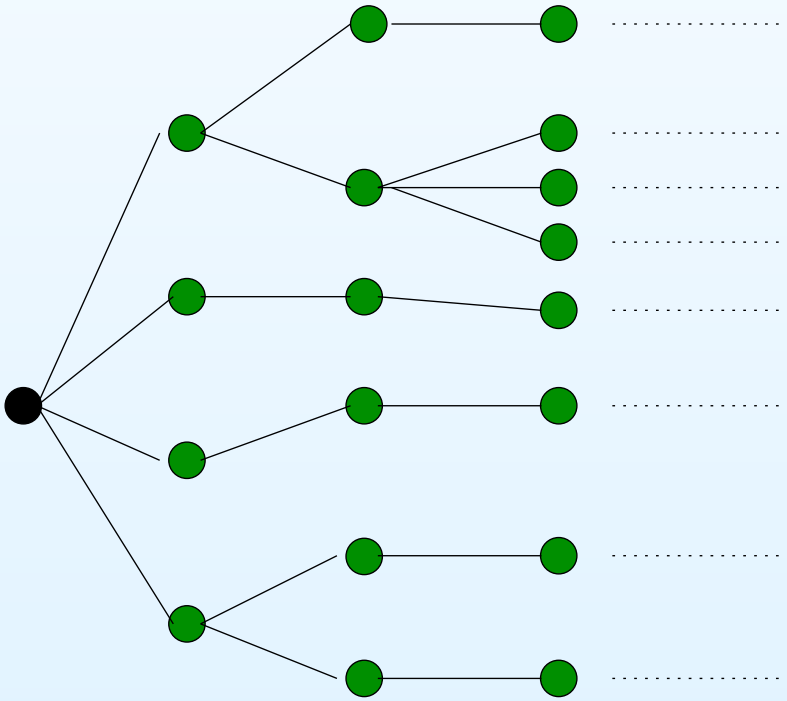
AFp: En todos los posibles futuros, eventualmente se satisface p



Lógica CTL (*Computation Tree Logic*)

- Lógica temporal ramificada
- Operadores

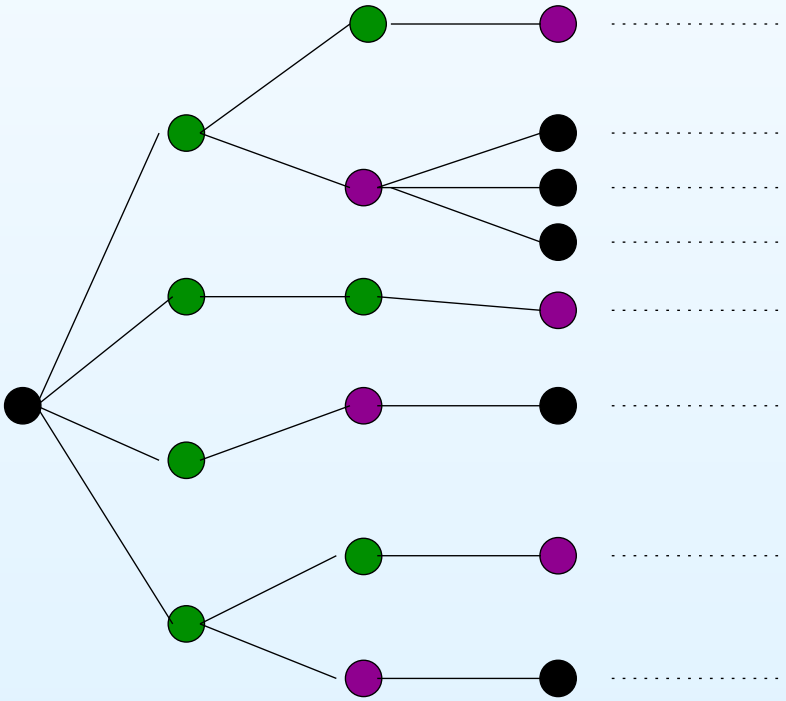
AGp: En todos los posibles futuros, siempre se satisface p



Lógica CTL (*Computation Tree Logic*)

- Lógica temporal ramificada
- Operadores

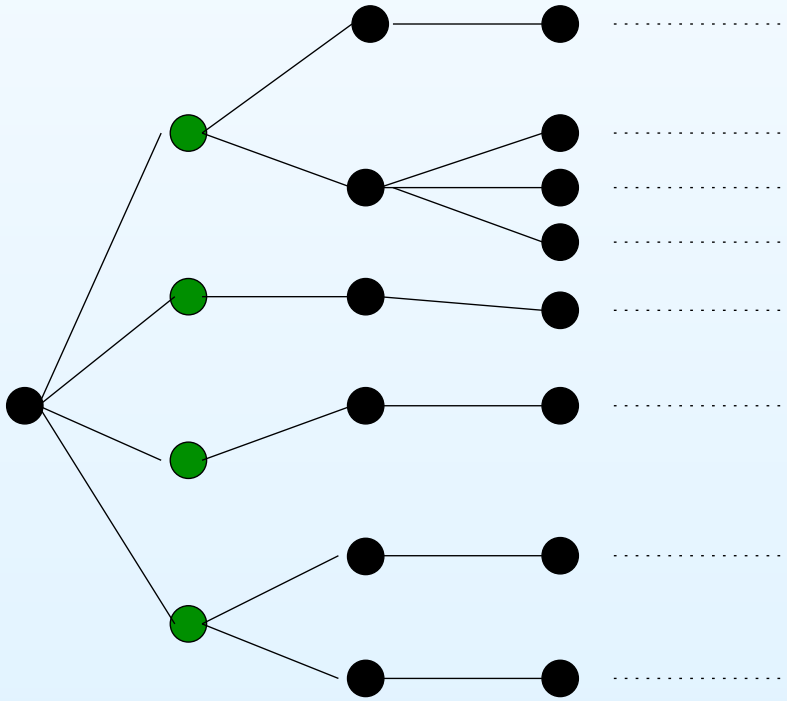
A[p**U**q**]**: En todos los posibles futuros, p se satisface hasta que lo haga q



Lógica CTL (*Computation Tree Logic*)

- Lógica temporal ramificada
- Operadores

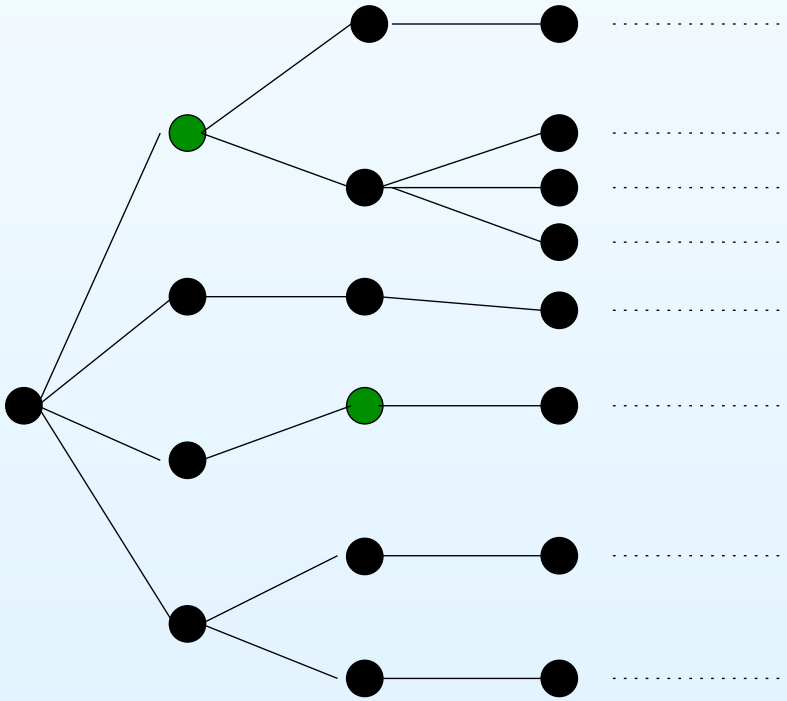
AXp: En todos los posibles futuros,en el siguiente instante de tiempo se satisface p



Lógica CTL (*Computation Tree Logic*)

- Lógica temporal ramificada
- Operadores

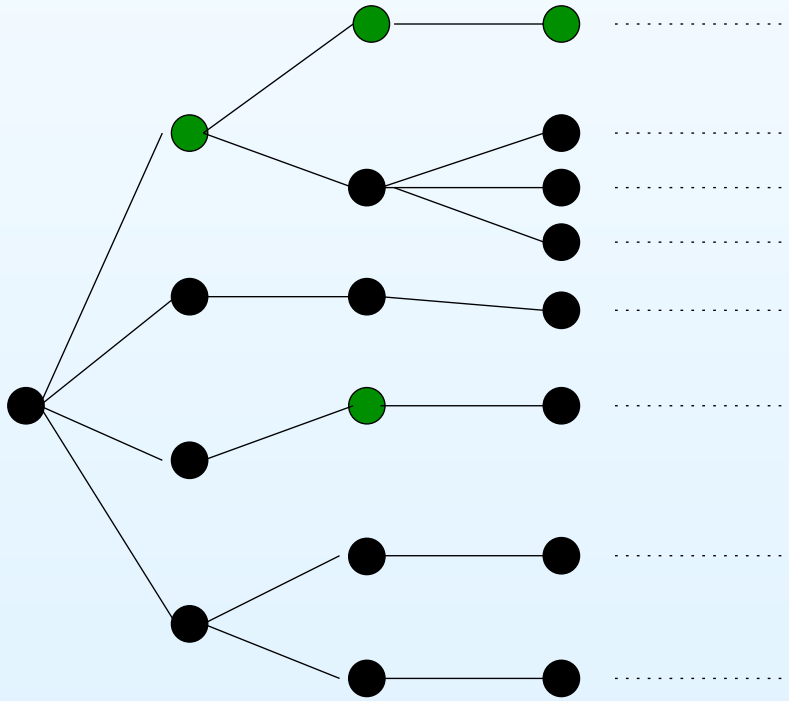
EFp: Existe un futuro en el que eventualmente se satisface p



Lógica CTL (*Computation Tree Logic*)

- Lógica temporal ramificada
- Operadores

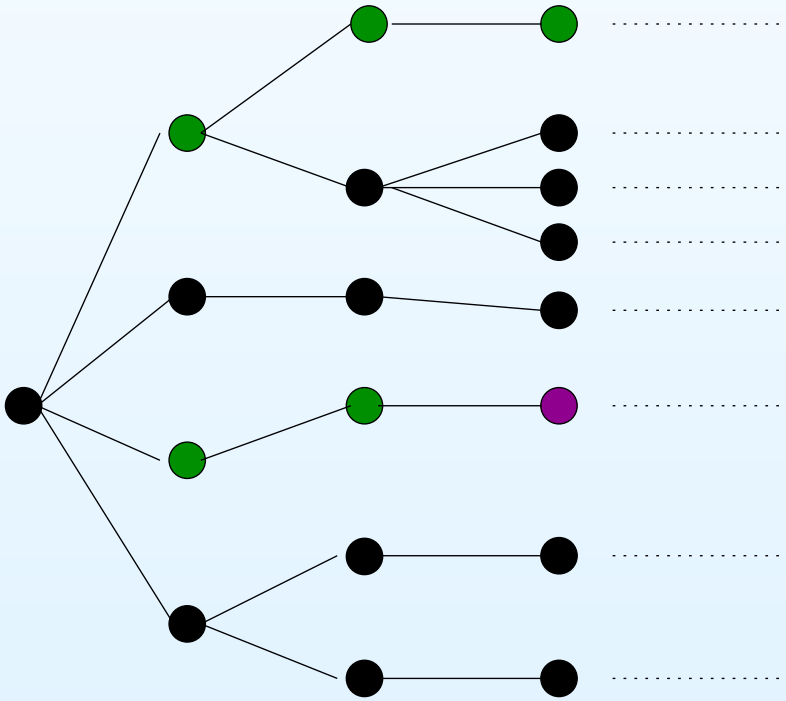
EGp: Existe un futuro en el que siempre se satisface p



Lógica CTL (*Computation Tree Logic*)

- Lógica temporal ramificada
- Operadores

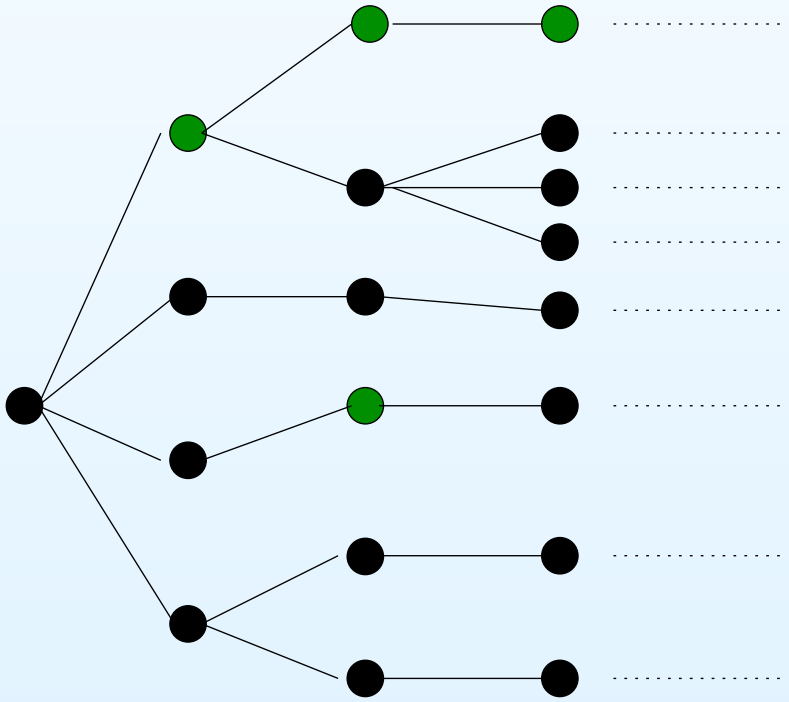
E[p**U**q**]**: Existe un futuro en el que se satisface p hasta que lo hace q



Lógica CTL (*Computation Tree Logic*)

- Lógica temporal ramificada
- Operadores

EXp: Existe un futuro en el que en el siguiente instante de tiempo se satisface p



Lógica CTL (*Computation Tree Logic*)

Sintaxis

- Toda proposición atómica es una fórmula CTL
- Si f y g son fórmulas CTL, entonces $\neg f, (f \wedge g), AX\,f, EX\,f, A(f\,U\,g), E(f\,U\,g)$ lo son también.

Los demás operadores se derivan de estos...

$$f \vee g = \neg(\neg f \wedge \neg g)$$
$$AFg = A(\mathrm{true}\,U\,g)$$
$$EFg = E(\mathrm{true}\,U\,g)$$
$$AGf = \neg E(\mathrm{true}\,U\neg f)$$
$$EGf = \neg A(\mathrm{true}\,U\neg f)$$

Lógica CTL (*Computation Tree Logic*)

Caracterización Punto Fijo

- Es la base para el algoritmo de Model Checking Simbólico

Lógica CTL (*Computation Tree Logic*)

Caracterización Punto Fijo

- Es la base para el algoritmo de Model Checking Simbólico
- Una fórmula vendrá identificada por el **conjunto** de estados en los que se satisface

Lógica CTL (*Computation Tree Logic*)

Caracterización Punto Fijo

- Es la base para el algoritmo de Model Checking Simbólico
- Una fórmula vendrá identificada por el **conjunto** de estados en los que se satisface
  - *true* es el conjunto S de todos los estados

Lógica CTL (*Computation Tree Logic*)

Caracterización Punto Fijo

- Es la base para el algoritmo de Model Checking Simbólico
- Una fórmula vendrá identificada por el **conjunto** de estados en los que se satisface
  - *true* es el conjunto S de todos los estados
  - *false* es el conjunto vacío

Lógica CTL (*Computation Tree Logic*)

Caracterización Punto Fijo

- Es la base para el algoritmo de Model Checking Simbólico
- Una fórmula vendrá identificada por el **conjunto** de estados en los que se satisface
  - *true* es el conjunto S de todos los estados
  - *false* es el conjunto vacío
- Dada τ, un punto fijo es un conjunto P tal que $\tau(P) = P$

Lógica CTL (*Computation Tree Logic*)

Caracterización Punto Fijo

- Es la base para el algoritmo de Model Checking Simbólico
- Una fórmula vendrá identificada por el **conjunto** de estados en los que se satisface
  - *true* es el conjunto S de todos los estados
  - *false* es el conjunto vacío
- Dada τ, un punto fijo es un conjunto P tal que $\tau(P) = P$
- $\mu P.\tau(P)$ denota el menor punto fijo (lfp)

Lógica CTL (*Computation Tree Logic*)

Caracterización Punto Fijo

- Es la base para el algoritmo de Model Checking Simbólico
- Una fórmula vendrá identificada por el **conjunto** de estados en los que se satisface
  - *true* es el conjunto S de todos los estados
  - *false* es el conjunto vacío
- Dada τ, un punto fijo es un conjunto P tal que $\tau(P) = P$
- $\mu P.\tau(P)$ denota el menor punto fijo (lfp)
- $\nu P.\tau(P)$ es el mayor punto fijo (gfp)

Lógica CTL (*Computation Tree Logic*)

Caracterización Punto Fijo

- $AF f_1 = \mu Z.(f_1 \vee AXZ)$
- $EF f_1 = \mu Z.(f_1 \vee EXZ)$
- $AG f_1 = \nu Z.(f_1 \wedge AXZ)$
- $EG f_1 = \nu Z.(f_1 \wedge EXZ)$
- $A[f_1 U f_2] = \mu Z.(f_2 \vee (f_1 \wedge AXZ)$
- $E[f_1 U f_2] = \mu Z.(f_2 \vee (f_1 \wedge EXZ)$

Lógica CTL (*Computation Tree Logic*)

Caracterización Punto Fijo

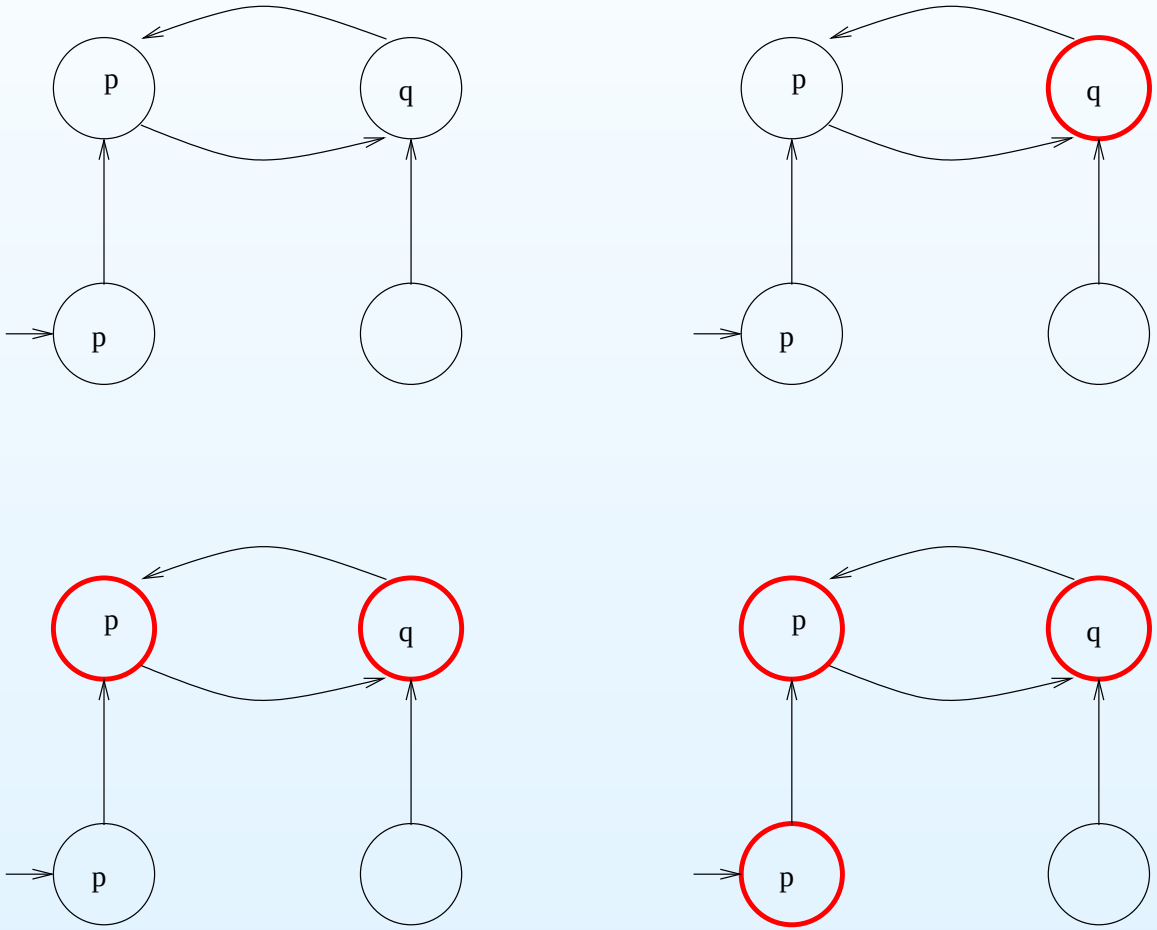
- $AF f_1 = \mu Z.(f_1 \vee AXZ)$
- $EF f_1 = \mu Z.(f_1 \vee EXZ)$
- $AG f_1 = \nu Z.(f_1 \wedge AXZ)$
- $EG f_1 = \nu Z.(f_1 \wedge EXZ)$
- $A[f_1 U f_2] = \mu Z.(f_2 \vee (f_1 \wedge AXZ)$
- $E[f_1 U f_2] = \mu Z.(f_2 \vee (f_1 \wedge EXZ)$

un ejemplo...

Lógica CTL (*Computation Tree Logic*)

$$E[pUq]$$

$$\tau \longrightarrow q \vee (p \wedge EXZ)$$



Algoritmo de *model checking* Simbólico

Fórmulas Booleanas Cuantificadas (QBF)

- Para obtener una notación más concisa de las operaciones sobre fórmulas booleanas
- Sintaxis. Dado un conjunto V de variables proposicionales
  - las variables de V son fórmulas
  - si f y g son fórmulas, entonces $\neg f$, $f \vee g$ y $f \wedge g$ también lo son
  - si f es una fórmula y $v \in V$, entonces $\exists v f$ y $\forall v f$ son fórmulas

$$\sigma\langle v \leftarrow a\rangle(w) = \begin{cases} a & \text{if } v = w \\ \sigma(w) & \text{otherwise} \end{cases}$$

Algoritmo de *model checking* Simbólico

Fórmulas Booleanas Cuantificadas (QBF)

- Para obtener una notación más concisa de las operaciones sobre fórmulas booleanas
- Sintaxis. Dado un conjunto V de variables proposicionales
  - las variables de V son fórmulas
  - si f y g son fórmulas, entonces $\neg f$, $f \vee g$ y $f \wedge g$ también lo son
  - si f es una fórmula y $v \in V$, entonces $\exists v f$ y $\forall v f$ son fórmulas

$$\sigma\langle v \leftarrow a\rangle(w) = \begin{cases} a & \text{if } v = w \\ \sigma(w) & \text{otherwise} \end{cases}$$

Aquí es donde se usa *restrict* de OBDDs

Algoritmo de *model checking* Simbólico

Algoritmo *Check*

- Entrada: lógica CTL que se quiere verificar
- Salida: OBDD que representa exactamente los estados del sistema que satisfacen la fórmula
- El OBDD que representa la relación de transición es un parámetro implícito

Algoritmo de *model checking* Simbólico

Algoritmo *Check*

- Entrada: lógica CTL que se quiere verificar
- Salida: OBDD que representa exactamente los estados del sistema que satisfacen la fórmula
- El OBDD que representa la relación de transición es un parámetro implícito

Intuición

- Se define inductivamente sobre la estructura de la fórmula
  - *Check(a)* será el OBDD del conjunto de estados donde se satisface a
  - *Check*($f_1 \wedge f_2$) será *Apply(Check*(f_1)*,Check*(f_2)*)*
  - para EX, EU y EG hay algoritmos especiales...

Algoritmo de *model checking* Simbólico

- *Check*(EX f) = *CheckEX(Check(f))*

$$\textit{CheckEX}(f(v)) = \exists v'[f(v') \wedge R(v, v')]$$

- *Check*(E[f U g]) = *CheckEU(Check(f),Check(g))*

$$\text{E}[f_1 \text{ U } f_2] = \mu Z.f_2 \vee (f_1 \wedge EXZ)$$

  - El cálculo del punto fijo es **finito**
  - Comprobar si hemos llegado al punto fijo fácil ya que los OBDDs son canónicos

- *Check*(EG f) = *CheckEG(Check(f))*

$$\text{EG } f_1 = \nu Z.f_1 \wedge EXZ$$

Estamos trabajando sobre los OBDDs que representan f y R.

Algoritmo de *model checking* Simbólico

Fairness

- las condiciones de *fairness* se expresan como fórmulas CTL

$$F = \{P_1, \ldots, P_n\}$$

- *CheckFair* será el nuevo algoritmo de *model checking*
- tenemos también *CheckFairEX*, *CheckFairEU* y *CheckFairEG*

Algoritmo de *model checking* Simbólico

Fairness - Idea

Dada la fórmula EGf y las condiciones de *fairness* F, el conjunto Z de estados que la satisfacen cumplen

1. que todos satisfacen f y
2. que para todas las condiciones $P_k \in F$ y todos los estados $s \in Z$, existe una secuencia de estados de longitud uno o más de uno desde s hasta un estado en Z el cual satisface P_k de forma que todos los estados del camino satisfacen f.

Algoritmo de *model checking* Simbólico

Fairness - Idea

Dada la fórmula EGf y las condiciones de *fairness* F, el conjunto Z de estados que la satisfacen cumplen

1. que todos satisfacen f y
2. que para todas las condiciones $P_k \in F$ y todos los estados $s \in Z$, existe una secuencia de estados de longitud uno o más de uno desde s hasta un estado en Z el cual satisface P_k de forma que todos los estados del camino satisfacen f.

$$CheckFairEG(f(v)) = \nu Z(v).f(v) \wedge \bigwedge_{k=1}^{n} EX(EU(f(v), Z(v) \wedge P_k))$$

Algoritmo de *model checking* Simbólico

Contraejemplos y Testimonios

- Contraejemplo: si una fórmula AG es falsa, sirve de justificación
- Testimonio: cuando una fórmula EF es cierta, sirve de justificación

Algoritmo de *model checking* Simbólico

Contraejemplos y Testimonios

- Contraejemplo: si una fórmula AG es falsa, sirve de justificación
- Testimonio: cuando una fórmula EF es cierta, sirve de justificación

Testimonio para EGf

- La idea es construir el camino π incrementalmente
- Cada vez se comprueba que el camino puede extenderse a un camino *fair*

$$\nu Z.f \wedge \bigwedge_{k=1}^{n} EX(E[fU(Z \wedge P_k)])$$

Algoritmo de *model checking* Simbólico

Contraejemplos y Testimonios

- Contraejemplo: si una fórmula AG es falsa, sirve de justificación
- Testimonio: cuando una fórmula EF es cierta, sirve de justificación

Testimonio para EGf

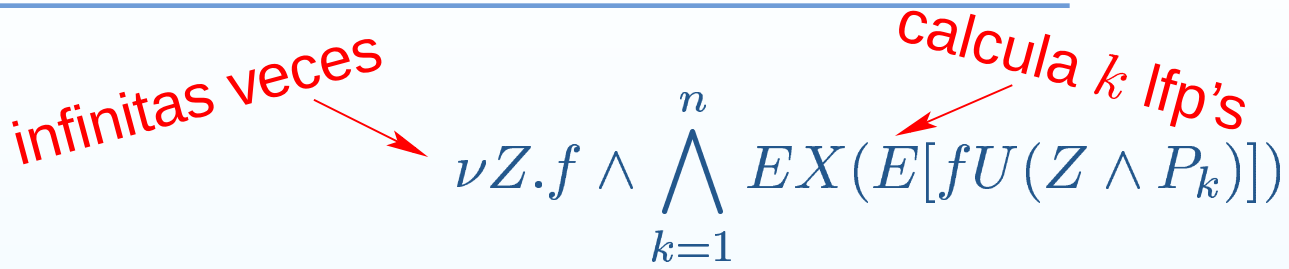
- La idea es construir el camino π incrementalmente
- Cada vez se comprueba que el camino puede extenderse a un camino *fair*

asegura que se repite infinitas veces

calcula k lfp's

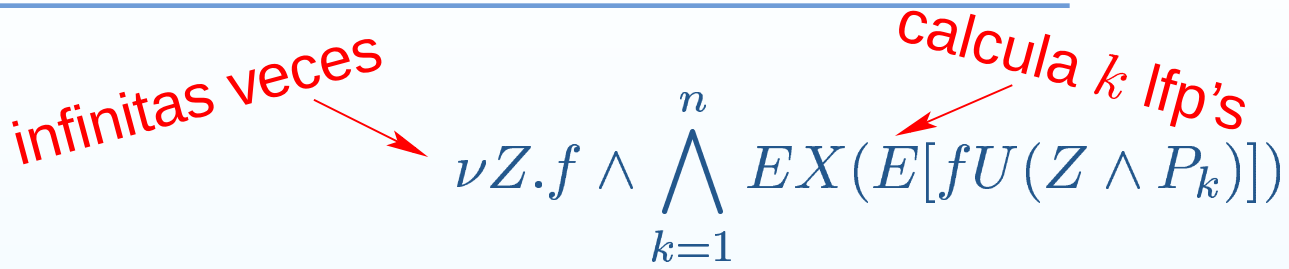
$$\nu Z.f \wedge \bigwedge_{k=1}^{n} EX(E[fU(Z \wedge P_k)])$$

Algoritmo de *model checking* Simbólico



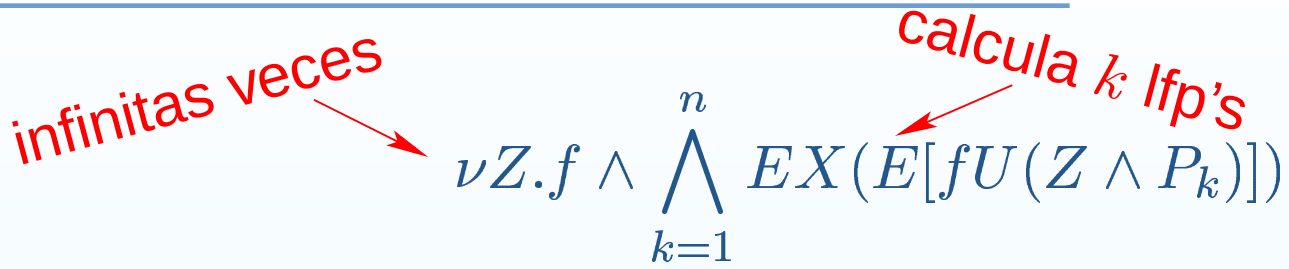
- Para cada P_i se calcula una secuencia de aproximaciones $Q_0^P \subseteq Q_1^P \subseteq \ldots Q_l^P$

Algoritmo de *model checking* Simbólico



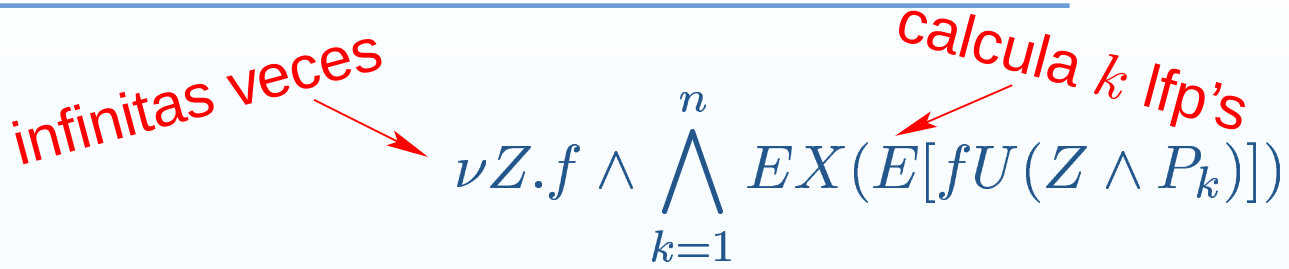
- Para cada P_i se calcula una secuencia de aproximaciones $Q_0^P \subseteq Q_1^P \subseteq \ldots Q_l^P$
- busco un camino (mínimo) desde un estado s que satisface EGf hasta un punto t donde se satisface P_k (estará en la secuenca de Q's)

Algoritmo de *model checking* Simbólico



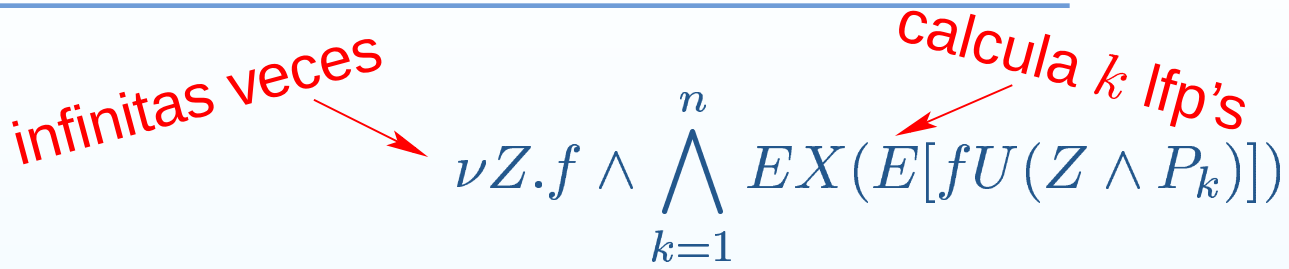
- Para cada P_i se calcula una secuencia de aproximaciones $Q_0^P \subseteq Q_1^P \subseteq \ldots Q_l^P$
- busco un camino (mínimo) desde un estado s que satisface EGf hasta un punto t donde se satisface P_k (estará en la secuenca de Q's)
- en el Q anterior tendré un sucesor u de t (porque lfp se construye desde el bottom).

Algoritmo de *model checking* Simbólico



- Para cada P_i se calcula una secuencia de aproximaciones $Q_0^P \subseteq Q_1^P \subseteq \ldots Q_l^P$
- busco un camino (mínimo) desde un estado s que satisface EGf hasta un punto t donde se satisface P_k (estará en la secuenca de Q's)
- en el Q anterior tendré un sucesor u de t (porque lfp se construye desde el bottom).
- repito para todos los P_i

Algoritmo de *model checking* Simbólico



- Para cada P_i se calcula una secuencia de aproximaciones $Q_0^P \subseteq Q_1^P \subseteq \ldots Q_l^P$
- busco un camino (mínimo) desde un estado s que satisface EGf hasta un punto t donde se satisface P_k (estará en la secuenca de Q's)
- en el Q anterior tendré un sucesor u de t (porque lfp se construye desde el bottom).
- repito para todos los P_i
- si s' no está en una SCC se repite todo cogiendo como nodo inicial s'.

El *model checker* SMV

Symbolic Model Verifier

- Herramienta para comprobar que un sistema con un número **finito** de estados satisface una fórmula CTL
- Tiene un lenguaje propio de especificación del sistema
- Soporta modularidad
- Los módulos se pueden componer de forma síncrona **o** asíncrona
- Admite no determinismo en las transiciones de estados
- Las relaciones de transición se pueden definir de forma explícita o implítita (en términos de valores de variables)

El *model checker* SMV

```
1   MODULE main -- two process mutual exclusion program

2   VAR
3   s0: {noncritical, trying, critical};
4   s1: {noncritical, trying, critical};
5   turn: boolean;
6   pr0: process prc(s0, s1, turn, 0);
7   pr1: process prc(s1, s1, turn, 1);

8   ASSIGN
9   init(turn) := 0;

10  FAIRNESS   !(s0 = critical)
11  FAIRNESS   !(s1 = critical)
```

El *model checker* SMV

```
12  SPEC EF((s0 = critical) & (s1 = critical))
13  SPEC AG((s0 = trying) -> AF (s0 = critical))
14  SPEC AG((s1 = trying) -> AF (s1 = critical))
15  SPEC AG((s0 = critical) -> A[(s0 = critical) U
16   (!(s0 = critical) & !E[!(s1=critical) U (s0 = critical)])])
17  SPEC AG((s1 = critical) -> A[(s1 = critical) U
18   (!(s1 = critical) & !E[!(s0=critical) U (s1 = critical)])])
```

El *model checker* SMV

```
19  MODULE prc(state0, state1, turn, turn0)
20  ASSIGN
21  init(state0) := noncritical;
22  next(state0) :=
23      case
24          (state0 = noncritical) : {trying,noncritical};
25          (state0 = trying) & (state1 = noncritical): critical;
26          (state0 = trying) & (state1 = trying) & (turn = turn0): critical;
27          (state0 = critical) : {critical,noncritical};
28          1: state0;
29      esac;
30  next(turn) :=
31      case
32          turn = turn0 & state0 = critical: !turn;
33          1: turn;
34      esac;
35  FAIRNESS running
```

El *model checker* SMV

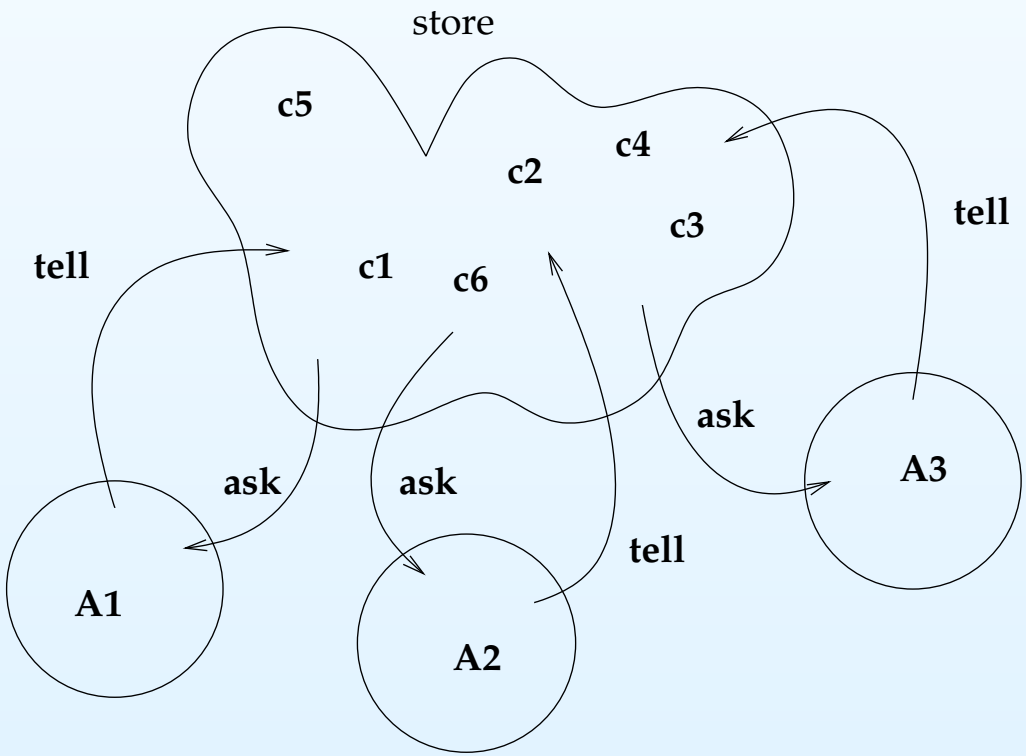
Salida

```
-- specification EF (s0 = critical & s1 = critical) is false
-- specification AG (s0 = trying -> AF s0 = critical) is true
-- specification AG (s1 = trying -> AF s1 = critical) is true
-- specification AG (s0 = critical -> A(... is false
-- specification AG (s1 = critical -> A(... is false

resource used:
user time: 1.15 s, system time: 0.3 s
BDD nodes allocated: 2405

BDD nodes representing transition relation: 56 + 1
```

Model checking simbólico para ccp

Timed Concurrent Constraint Paradigm

- Extensión de ccp
- *Store as constraint* vs *Store as valuation*



Model checking simbólico para ccp

Idea

- Estructura de Kripke $\longrightarrow$ Estructura tccp
- Fórmula temporal $\longrightarrow$ Lógica temporal sobre restricciones
- OBDD's $\rightarrow$ DDD's

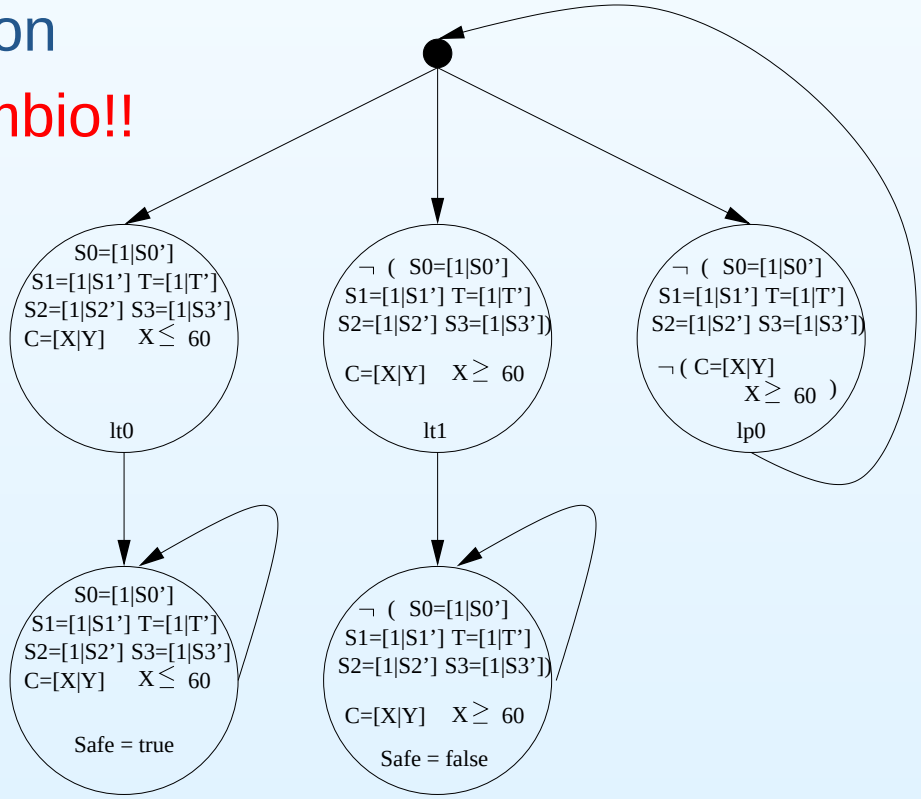
Complicaciones

- DDDs más difíciles de manejar que BDDs
- DDDs definidos para una clase reducida de sistemas de restricciones

Model checking simbólico para ccp

Estructura tccp

- Proposiciones atómicas: AP
- Estado tccp: $S \subseteq 2^{AP} \times 2^{L}$
- 5-tupla $M = (S, S_0, R, C, T)$, con
  1. S ← atención al cambio!!
  2. $S_0 \subseteq S$
  3. $R \subseteq S \times S$
  4. $C : S \to 2^{AP}$
  5. $T : S \to 2^{L}$



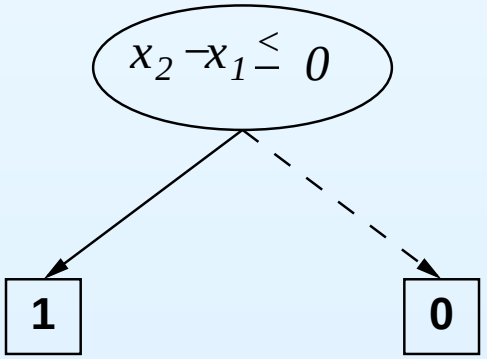
Model checking simbólico para ccp

Difference Decision Diagram

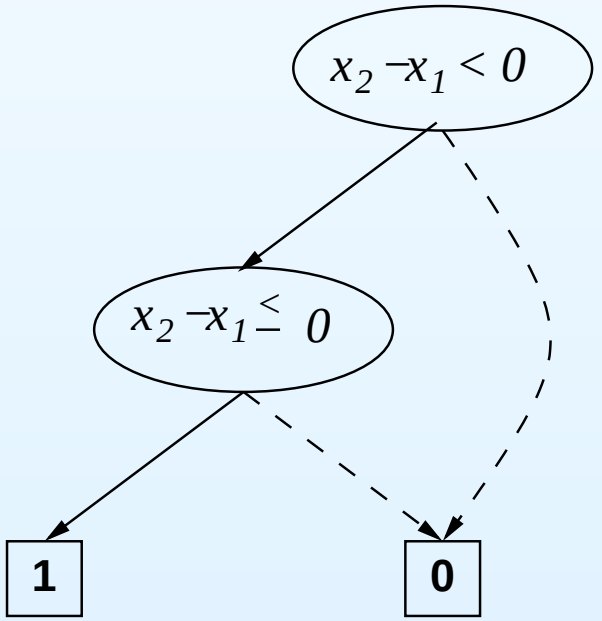
- lógica:

$$\phi ::= x - y \leq c \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists x.\phi$$

$x_2 - x_1 \leq 0$



$x_2 - x_1 = 0$



BDDs - *Model Checking* Simbólico

FIN