Validación y Verificación de Software

# Principios de Model Checking

# Aplicaciones reales

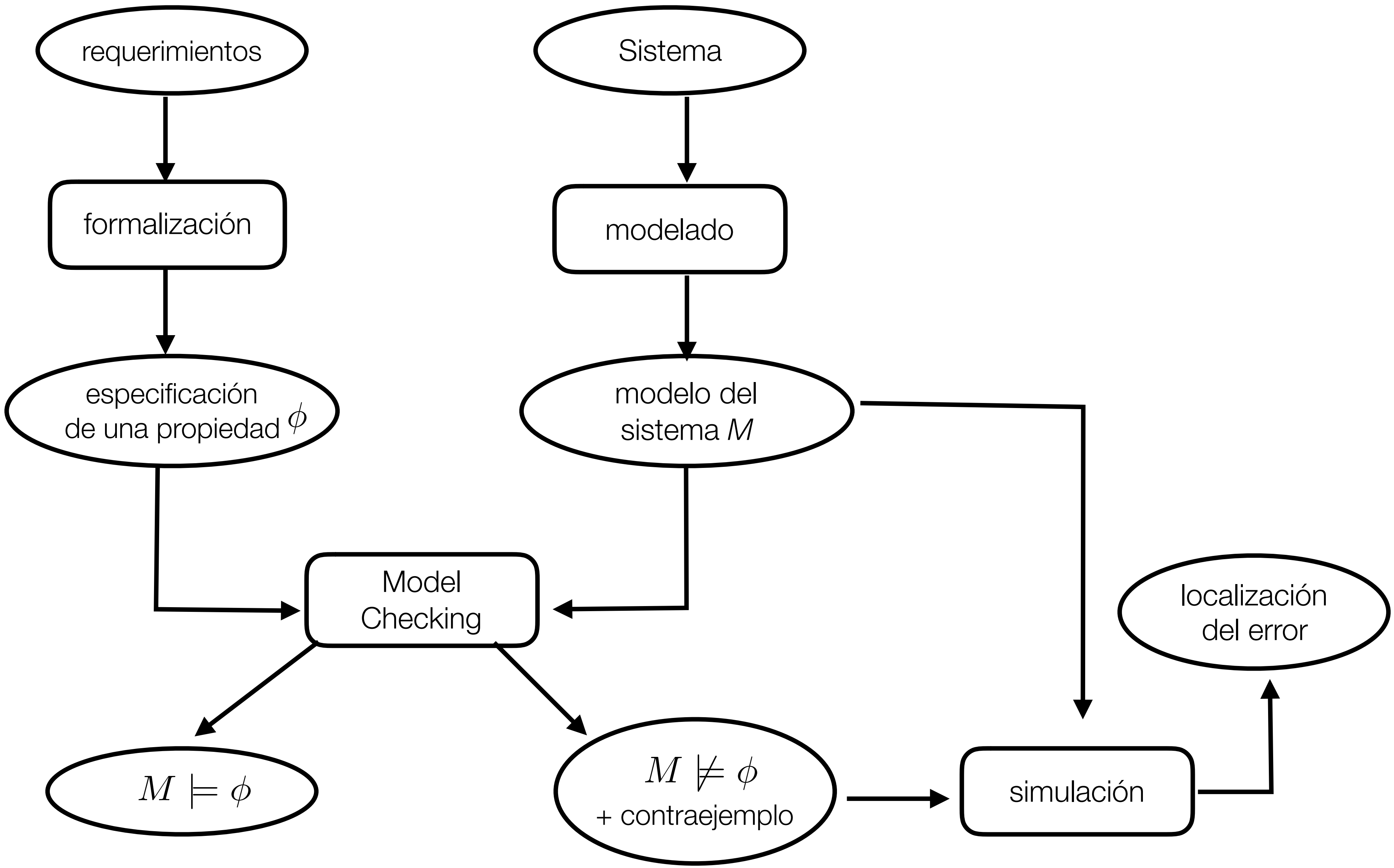
Deadlock detectado en sistema de reservas online de una aerolínea

5 errores detectados en controlador de Nave espacial Deep Space 1 (NASA)

Róterdam, Holanda. Model Checking reveló varios errores del software de control de un barrera que protege puerto contra inundaciones.

….

# Model Checking



# El problema de Model checking

Dado un modelo $M$ de un sistema (en algún lenguaje) y una propiedad (en alguna lógica) deseamos verificar automáticamente si es satisfecha por $\phi$, es decir, si $M \vDash \phi$.

En particular para el caso de LTL:

> sabemos que el lenguaje $\mathcal{L}(\phi)$ de una fórmula es el conjunto de todas las trazas donde ésta se hace verdadera, y que
>
> el comportamiento de un sistema $M$ (denotado $\mathcal{L}(M)$) está dado por el conjunto de todas las trazas que éste puede ejecutar.

# El problema de Model checking

Luego, $M \vDash \phi$ si y sólo si toda traza de $M$ satisface $\phi$, es decir:

$$M \models \phi \text{ si y sólo si } \mathcal{L}(M) \subseteq \mathcal{L}(\phi)$$

El problema de model checking se reduce entonces a validar esta inclusión de manera automática

¿Pero cómo?

# Cómo obtener el modelo de un sistema

Para poder definir $\mathcal{L}(M)$ necesitamos primero una manera razonable de definir $M$. Eso ya lo hemos hecho antes:

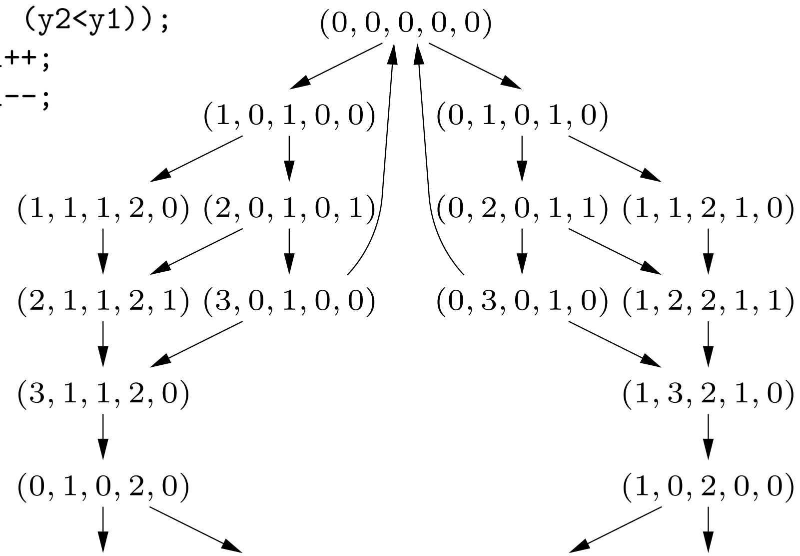
# Cómo obtener el modelo de un sistema

Para poder definir $\mathcal{L}(M)$ necesitamos primero una manera razonable de definir $M$. Eso ya lo hemos hecho antes:

```
  int y1 = 0;
  int y2 = 0;
  short in_critical = 0;

  active proctype process_1() {
    do
    :: true ->
0:        y1 = y2+1;
1:        ((y2==0) || (y1<=y2));
          in_critical++;
2:        in_critical--;
3:        y1 = 0;
    od
  }
```

```
  active proctype process_2() {
    do
    :: true ->
0:        y2 = y1+1;
1:        ((y1==0) || (y2<y1));
          in_critical++;
2:        in_critical--;
3:        y2 = 0;
    od
  }
```




Estructura del

$(pc_1, pc_2, \mathtt{y1}, \mathtt{y2}, \mathtt{in\_critical})$

# Cómo obtener el modelo de un sistema

El modelo del sistema define un sistema de transiciones.

Un **sistema de transiciones** es una estructura

$$M = (S, Act, s_0, \rightarrow, PA, v)$$

En el contexto de lógicas modales, esta estructura se denomina estructura de ***Kripke***

donde:

- $S$ : es un conjunto de **estados** donde $s_0 \in S$ es el **estado inicial**,
- $Act$ : es el conjunto de acciones,
- $\rightarrow \subseteq S \text{ x } Act \text{ x } S$ : es la relación de transición tal que $\forall s \in S : \exists s' \in S : s \rightarrow s'$
- $PA$ : conjunto de proposiciones atómicas.
- $v : S \rightarrow 2^{PA}$ : es la función de valoración tal que $v(s)$ es el conjunto de todas las proposiciones atómicas que son verdaderas en el estado $s$.

# Cómo obtener el modelo de un sistema

Ejemplo Sistema de transición:

$M_p = (\mathcal{S}, \mathcal{Act}, s_0, \longrightarrow, \mathcal{PA}, \nu)$

$\mathcal{S} = \{s_0, s_1, s_2, s_3\}$,

$\mathcal{Act}$={pay, beer, soda, get-beer, get-soda}

$\mathcal{PA}$={paid, drink}

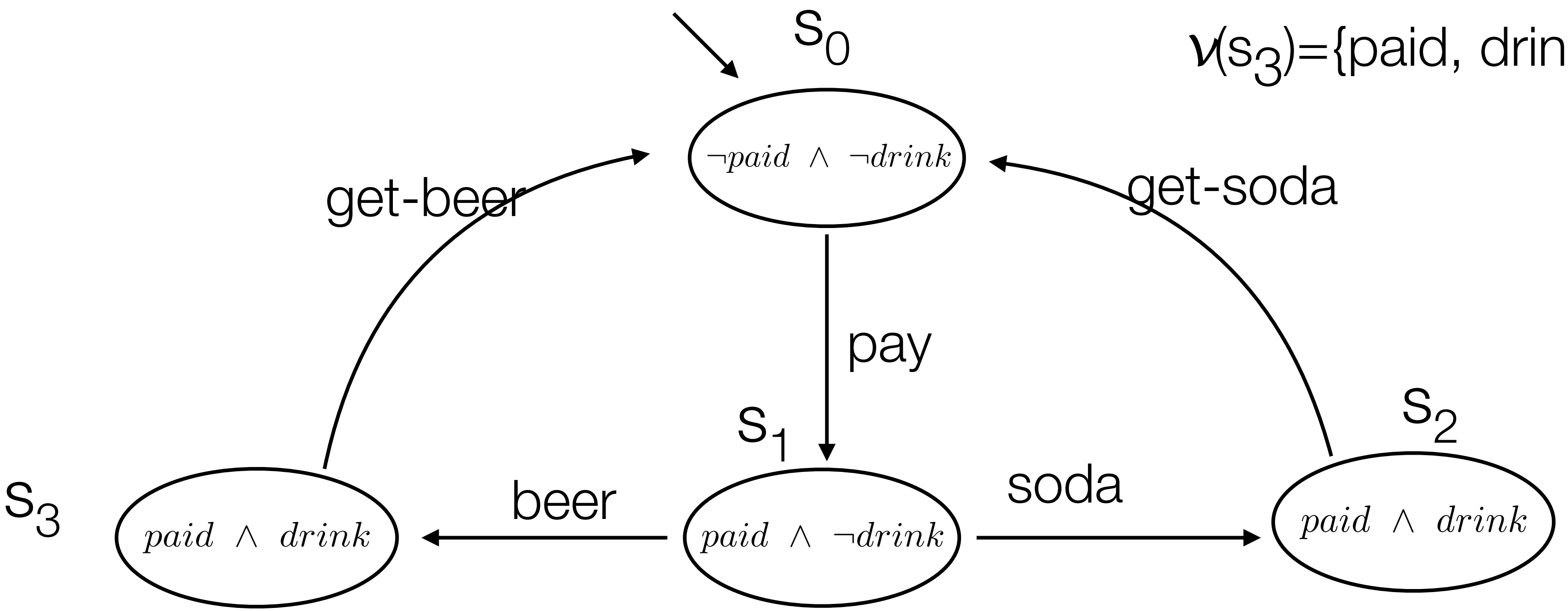
# Cómo obtener el modelo de un sistema

$\nu(s_0)=$

$\nu(s_1)=\{paid\}$

$\nu(s_2)=\{paid, drink\}$

$\nu(s_3)=\{paid, drink\}$



# Cómo obtener el modelo de un sistema

Una ejecución de *M:ST* es una secuencia:

$\rho = s_0 \alpha_1 s_1 \alpha_2 s_2 \ldots$ (infinita o que termina en un estado terminal)

tal que:

$$S_i \xrightarrow{\alpha_{i+1}} S_{i+1} \quad \textit{para todo } i >= 0$$

El comportamiento de *M* se define como:

$$\mathcal{L}(M) = \{\sigma \in \left(2^{PA}\right)^{\omega} \mid \exists \rho \text{ ejecución de } M : \forall i \geq 0 : \sigma(i) = w(\rho(i))\}$$

# Autómatas de Büchi

De la misma manera que los lenguajes regulares se manipulan a través de autómatas finitos que acepten los lenguajes a manipular, los lenguajes $\omega$-regulares pueden manipularse a través de los denominados autómatas de Büchi.

Un automata (no determinístico) de Büchi es una estructura

$$\mathcal{A} = (\Sigma, S, \delta, S_0, A)$$

- $\Sigma$ : Alfabeto (finito)
- $S$ : Conjunto finito de **estados** donde $s_0 \in S$ es el **estado inicial**,
- $\delta : \Sigma \times S \to 2^S$ es la función de transición,
- $A$ : Conjunto de estados de aceptación (**estados finales**),

# Aceptación de trazas en autómatas de Büchi

Dado un autómata de Buchi $\mathcal{A}$, decimos que una traza

$$\sigma = \sigma_0 \, \sigma_1 \, \sigma_2 \ldots \in \Sigma^\omega$$

es aceptada por $\mathcal{A}$ sii existe una secuencia infinita de estados de $\mathcal{A}$

$$q_0 q_1 q_2 \cdots$$

- $q_0$ es el estado inicial de $\mathcal{A}$,
- $q_i \xrightarrow{\sigma_i} q_{i+1}$ para todo i>=0,
- $q_i \in A$ para infinita cantidad de indices $i \in \mathbb{N}$ (existen infinitos estados de aceptación en $q_0 q_1 q_2 \cdots$ )

# Aceptación de trazas en autómatas de Büchi

Definimos como el lenguaje de $\mathcal{A}$ (Autómata de Büchi), notación $\mathcal{L}(\mathcal{A})$, al conjunto de todas las trazas (i.e., $\omega$-palabras) aceptadas por $\mathcal{A}$.

Los autómatas de Büchi aceptan exactamente todos los lenguajes $\omega$-regulares. Por consiguiente son más expresivos que LTL sobre el alfabeto $\Sigma$.

# Autómatas de Büchi como modelos de sistemas

Ya dijimos que un programa $P$ cuyo espacio de estado sea finito puede representarse con un sistema de transiciones finito $M_P = (S, s_0, \rightarrow, v)$.

$M_P$ puede verse como el autómata de Büchi $\mathcal{A}_P = (\Sigma, S, \delta, s_0, S)$, donde:

**todos** los estados son de aceptación

- $\Sigma = 2^{PA}$,
- $s_j \in \delta(B, s_i)$ sii $s_i \rightarrow s_j \wedge B = v(s_i)$

# Autómatas de Büchi como modelos de sistemas

Ya dijimos que un programa $P$ cuyo espacio de estado sea finito puede representarse con un sistema de transiciones finito $M_P = (S, s_0, \rightarrow, v)$.

$M_P$ puede verse como el autómata de Büchi $\mathcal{A}_P = (\Sigma, S, \delta, s_0, S)$, donde:

**todos** los estados son de aceptación

- $\Sigma = 2^{PA}$,
- $s_j \in \delta(B, s_i)$ sii $s_i \rightarrow s_j \wedge B = v(s_i)$

Esto es así porque nos interesan todas las ejecuciones posibles del sistema.

# Autómatas de Büchi como modelos de sistemas

Teorema:

$$\mathcal{L}(M_P) = \mathcal{L}(\mathcal{A}_P)$$

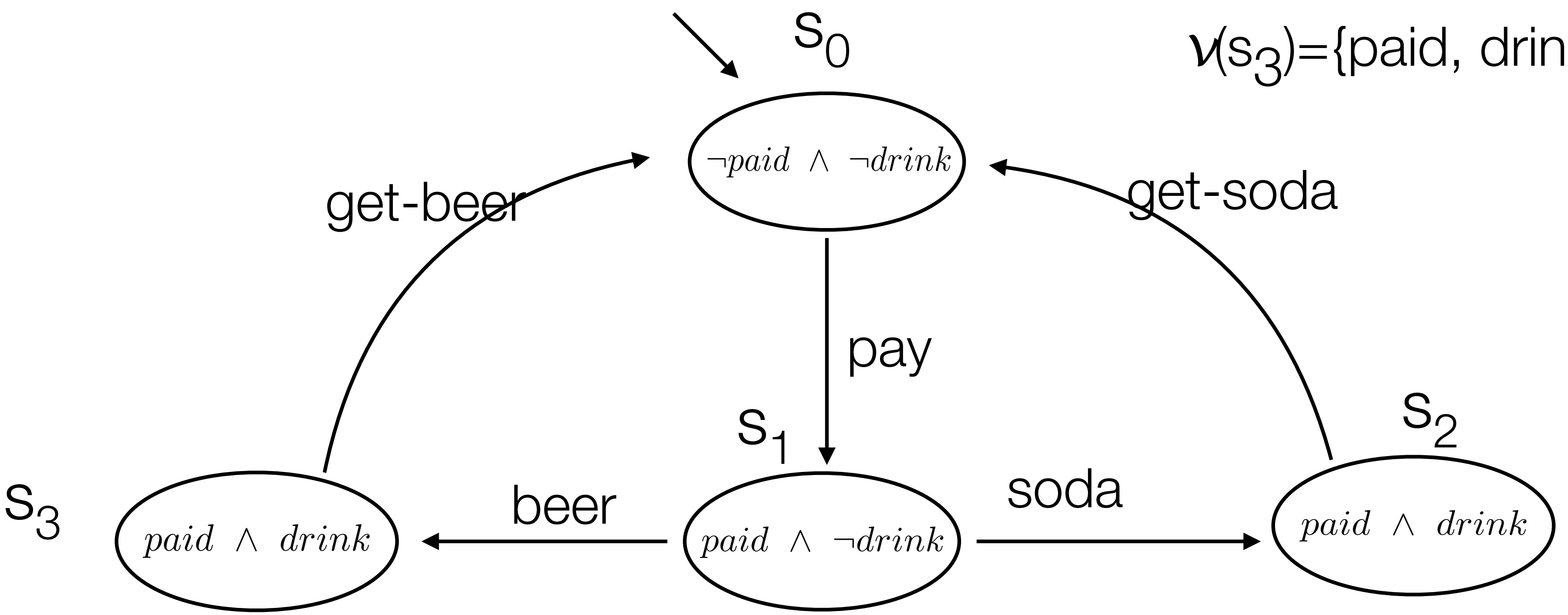
# Sistema de Transiciones

$\nu(s_0)=$

$\nu(s_1)=\{paid\}$

$\nu(s_2)=\{paid, drink\}$

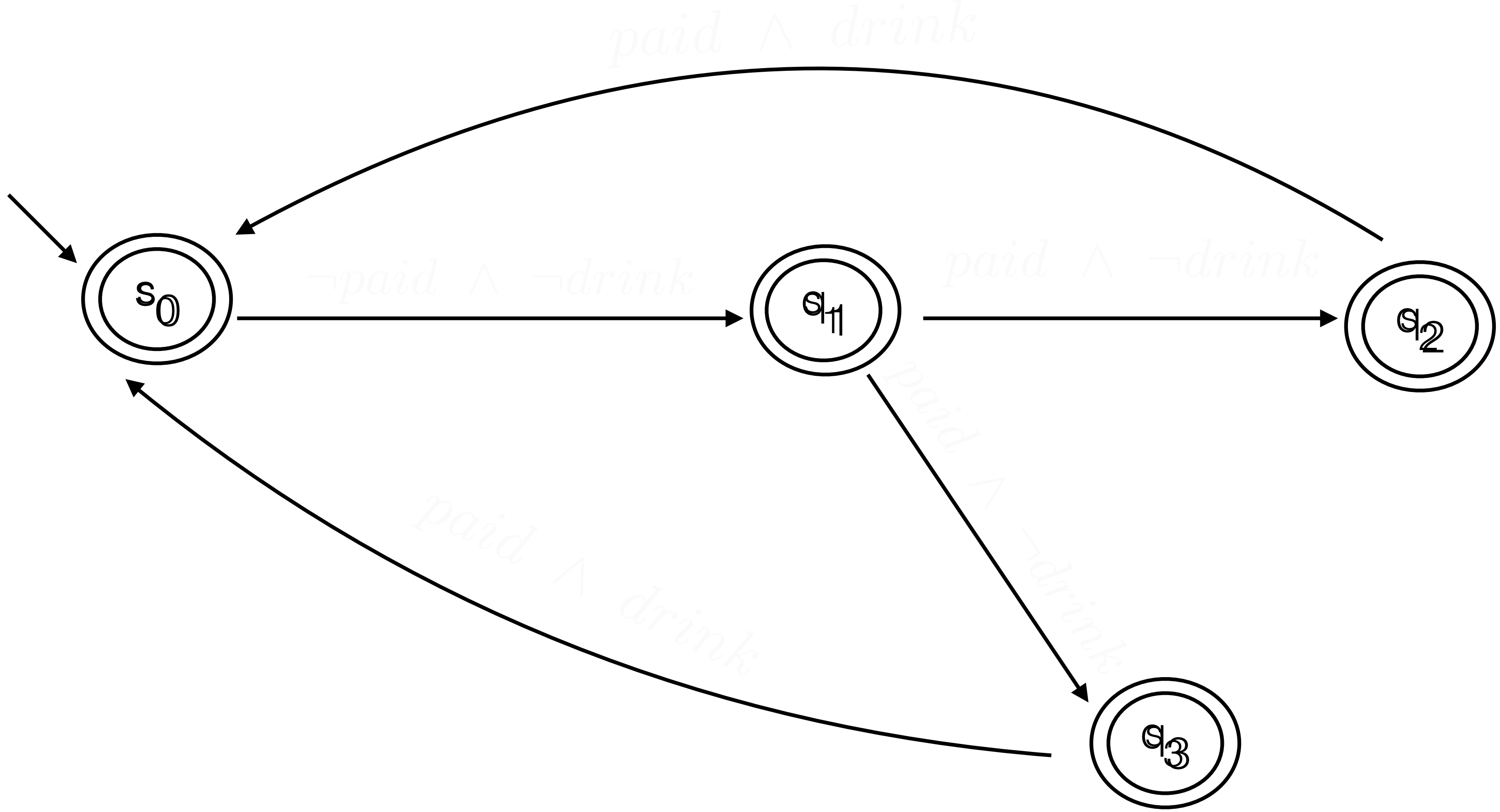
$\nu(s_3)=\{paid, drink\}$



# Cómo obtener el modelo de un sistema

$$\mathcal{A}_P = (\Sigma, S, \delta, s_0, S)$$

$$\Sigma = 2^{\mathrm{PA}} \ (\mathrm{PA} = \{\mathrm{paid}, \mathrm{drink}\})$$



# Fórmulas LTL y Autómatas de Büchi

**Teorema**: Para toda fórmula LTL $\phi$, se puede construir un autómata de Büchi $\mathcal{A}_\phi$ tal que:

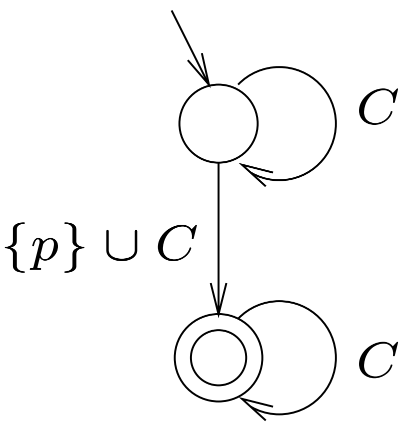
$$\mathcal{L}(\mathcal{A}_\phi) = \mathcal{L}(\phi)$$

La demostración de este teorema es compleja. Para formarse una idea de lo establecido por el teorema daremos algunos ejemplos:

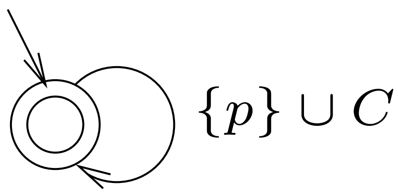
# Fórmulas LTL y Autómatas de Büchi

Ejemplos:

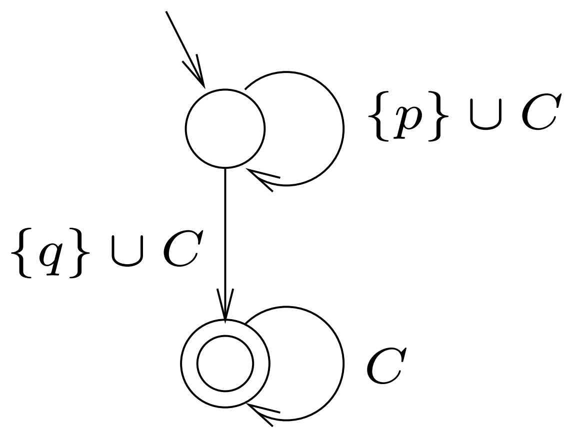
$\Diamond p$




$\Box p$




$p \, \mathtt{U} \, q$

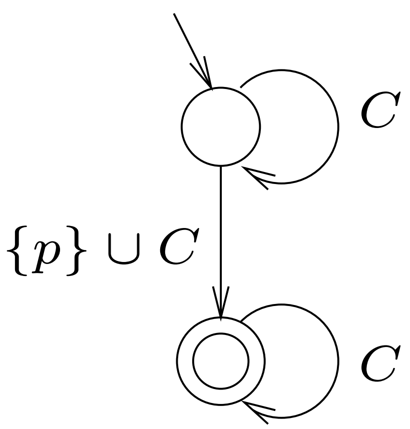


# Fórmulas LTL y Autómatas de Büchi

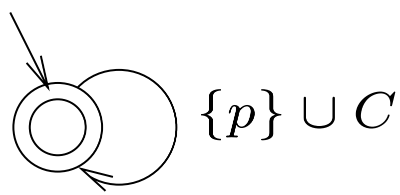
$C$ es cualquier subconjunto de $\mathcal{PA}$. Es decir, cada flecha de los dibujos representa muchas transiciones a la vez.

Ejemplos:

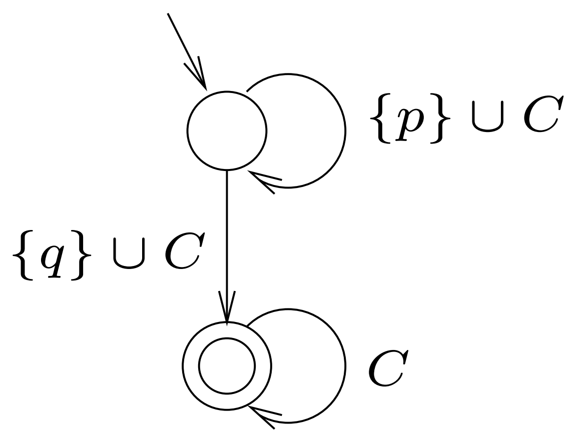
$\Diamond\, p$




$\Box\, p$




$p \;\mathtt{U}\; q$



# Manipulación de lenguajes ω-regulares

**Teorema**: Dados dos autómatas de Büchi $\mathcal{A}_1$ y $\mathcal{A}_2$ , se puede construir un autómata $\mathcal{A}_{\mathcal{A}_1 \cap \mathcal{A}_2}$ tal que:

$$\mathcal{L}(\mathcal{A}_{\mathcal{A}_1 \cap \mathcal{A}_2}) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$$

**Teorema**: Dado un autómada de Büchi $\mathcal{A}$ se puede construir un automata $\mathcal{A}^c$ tal que:

$$\mathcal{L}(\mathcal{A}^c) = \overline{\mathcal{L}(\mathcal{A})}$$

**Teorema**: Existe un algoritmo que permite decidir si el lenguaje ω-regular aceptado por un autómata de Büchi es vacío o no.

# Manipulación de lenguajes ω-regulares

**Teorema**: Dados dos autómatas de Büchi $\mathcal{A}_1$ y $\mathcal{A}_2$ , se puede construir un autómata $\mathcal{A}_{\mathcal{A}_1 \cap \mathcal{A}_2}$ tal que:

$$\mathcal{L}(\mathcal{A}_{\mathcal{A}_1 \cap \mathcal{A}_2}) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$$

**Teorema**: Dado un autómada de Büchi $\mathcal{A}$ se puede construir un automata $\mathcal{A}^c$ tal que:

$$\mathcal{L}(\mathcal{A}^c) = \overline{\mathcal{L}(\mathcal{A})}$$

El algoritmo es un doble DFS
con el fin de buscar componentes
fuertemente conexas que atrapen un
estado de aceptación.

**Teorema**: Existe un algoritmo que permite decidir si el lenguaje ω-regular aceptado por un autómata de Büchi es vacío o no.

# Model checking con fundamentos en la teoría de autómatas

Chequear que el programa $P$ de estados finitos satisfaga una propiedad temporal $\phi$

$$M_p \models \phi \quad \text{si y sólo si} \quad \mathcal{L}(M_P) \subseteq \mathcal{L}(\phi)$$

# Model checking con fundamentos en la teoría de autómatas

Chequear que el programa $P$ de estados finitos satisfaga una propiedad temporal $\phi$

$$M_p \models \phi \quad \text{si y sólo si} \quad \mathcal{L}(M_P) \subseteq \mathcal{L}(\phi)$$

$$\Downarrow$$

$$\mathcal{L}(\mathcal{A}_P) \subseteq \mathcal{L}(\mathcal{A}_\phi)$$

# Model checking con fundamentos en la teoría de autómatas

Chequear que el programa $P$ de estados finitos satisfaga una propiedad temporal $\phi$

$$M_p \models \phi \quad \text{si y sólo si} \quad \mathcal{L}(M_P) \subseteq \mathcal{L}(\phi)$$

$$\mathcal{L}(\mathcal{A}_P) \subseteq \mathcal{L}(\mathcal{A}_\phi)$$

$$\mathcal{L}(\mathcal{A}_P) \cap \overline{\mathcal{L}(\mathcal{A}_\phi)} = \emptyset$$

# Model checking con fundamentos en la teoría de autómatas

**Problema**: Complementar un autómata de Büchi es computacionalmente muy caro (se produce una explosión exponencial).

$$\mathcal{L}(\mathcal{A}_P) \cap \overline{\mathcal{L}(\mathcal{A}_\phi)} = \emptyset$$

# Model checking con fundamentos en la teoría de autómatas

**Problema**: Complementar un autómata de Büchi es computacionalmente muy caro (se produce una explosión exponencial).

$$\mathcal{L}(\mathcal{A}_P) \cap \overline{\mathcal{L}(\mathcal{A}_\phi)} = \emptyset$$

$$\Downarrow$$

$$\overline{\mathcal{L}(\mathcal{A}_\phi)} = \mathcal{L}(\mathcal{A}_{\neg\phi})$$

# Model checking con fundamentos en la teoría de autómatas

**Problema**: Complementar un autómata de Büchi es computacionalmente muy caro (se produce una explosión exponencial).

$$\mathcal{L}(\mathcal{A}_P) \cap \overline{\mathcal{L}(\mathcal{A}_\phi)} = \emptyset$$

$$\Downarrow$$

$$\overline{\mathcal{L}(\mathcal{A}_\phi)} = \mathcal{L}(\mathcal{A}_{\neg\phi})$$

$$\Downarrow$$

$$\mathcal{L}(\mathcal{A}_P) \cap \mathcal{L}(\mathcal{A}_{\neg\phi}) = \emptyset$$

# El algoritmo de model checking

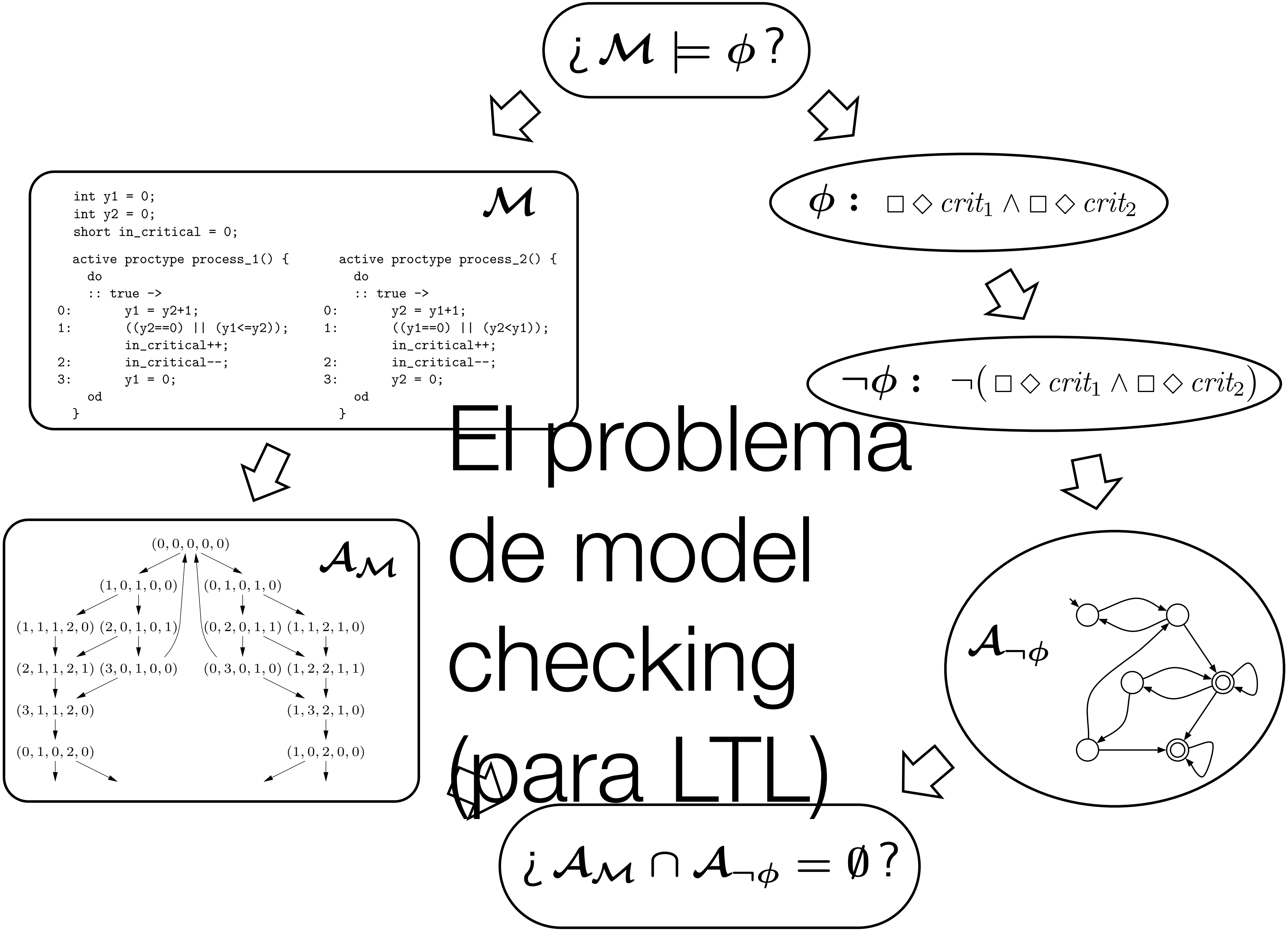
1. Construir el autómata de Büchi $\mathcal{A}_P$
2. Construir el autómata de Büchi $\mathcal{A}_{\neg\phi}$
3. Construir el autómata de Büchi $\mathcal{A}_{\mathcal{A}_P \cap \mathcal{A}_{\neg\phi}}$
4. Comprobar si $\mathcal{L}(\mathcal{A}_{\mathcal{A}_P \cap \mathcal{A}_{\neg\phi}})$ es vacío

# El algoritmo de model checking

1. Construir el autómata de Büchi $\mathcal{A}_P$
2. Construir el autómata de Büchi $\mathcal{A}_{\neg\phi}$
3. Construir el autómata de Büchi $\mathcal{A}_{\mathcal{A}_P \cap \mathcal{A}_{\neg\phi}}$
4. Comprobar si $\mathcal{L}(\mathcal{A}_{\mathcal{A}_P \cap \mathcal{A}_{\neg\phi}})$ es vacío

Un aspecto muy importante del model checking (sino el más importante) es la obtención de un contraejemplo en caso de que la propiedad no sea verdadera.

# El problema de model checking (para LTL)



# Herramientas de Model Checking

# El model checker SPIN

Desarrollado en AT&T / Bell Labs.

Principalmente desarrollado por Gerard Holzmann

Bibliografía:

G. Holzmann. The Spin Model Checker. Addisson Wesley. 2004.

www.spinroot.com

# El model checker SPIN

Permite distintos tipos de verificaciones:

- Propiedades en LTL
- Aserciones dentro del modelo
- Deadlocks
- Progreso
- Permite verificar bajo weak fairness

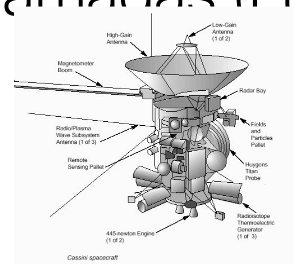
# El model checker SPIN (cont.)

Spin se ha utilizado en múltiples ocaciones, y en particular, directamente en la industria (¡se implementó en la industrial!).

Además es utilizado en la academia para aplicaciones reales (subcontratos/proyectos por parte de empresas).

**Ejemplos:**

- Verificación de protocolos embebidos en automotores (Bosch)
- Verificación del dique de emergencia climática en Rotterdam.
- Software para el procesamiento de llamadas (Lucent Tech.)



# Otros model checkers

Software model checkers:

SLAM (C) - Terminator (C) - Space Invaders (C) - Blast (C) - CBMC (C y  C++) - Java PathFinder - Bandera/Bogor (Java) - MoonWalker (.Net)

Más:

CADP - HyTech - IF - Design/CPN - Rapture - MRMC - E⊢MC$^2$ - mCRL2 - ...

# Otros model checkers

Verificación automática de
device drivers de Windows

Software model checkers:

SLAM (C) - Terminator (C) - Space Invaders (C) - Blast (C) - CBMC (C y  C++) - Java PathFinder - Bandera/Bogor (Java) - MoonWalker (.Net)

Más:

CADP - HyTech - IF - Design/CPN - Rapture - MRMC - E$\vdash$MC$^2$ - mCRL2 - ...

# Otros model checkers

Lograron verifcar
completamente de manera
automática más de la mita
del kernel de Linux

Verificación automática de
device drivers de Windows

Software model checkers:

SLAM (C) - Terminator (C) - Space Invaders (C) - Blast (C) - CBMC (C y C++) - Java PathFinder - Bandera/Bogor (Java) - MoonWalker (.Net)

Más:

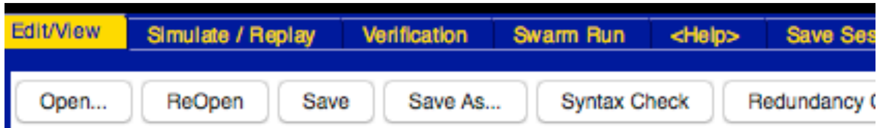
CADP - HyTech - IF - Design/CPN - Rapture - MRMC - $E \vdash MC^2$ - mCRL2 - ...

# Bibliografía

Cap 2 (Sección 2.1), Cap 4 (Sección 4.3.1,4.3.2) Principles of Model Checking, Christel Baier and Joost-Pieter Katoen

www.spinroot.com (Spin/Promela web site)

# DEMO




```
byte  y1 =0;
byte y2 =0;
int crit1=0;
int crit2=0;
short in_critical =0;

ltl p0 {([]<>crit1) && ([]<>crit2)}
        //ltl pr1 {[] (!crit1 || !crit2)}
proctype p1(){
        do
        ::true ->
                y1 =y2+1;
                ((y2==0)||(y1<=y2));
                in_critical++;
                crit1=1;
                in_critical--;
                crit1=0;
                y1=0;
        od
}

proctype p2(){
        do
        ::true ->
                y2 =y1+1;
                ((y1==0)||(y2<y1));
                in_critical++;
                crit2=1;
                in_critical--;
                crit2=0;
                y2=0;
        od
}

init{
        run p1();
        run p2();
}
```