DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Síntesis Dirigida de Controladores No Maximales para Requerimientos de tipo Non-Blocking

Matias Duran, Florencia Zanollo

Director: Sebastian Uchitel

12 de Mayo de 2021

Departamento de Computación - Universidad de Buenos Aires

Índice

Introducción

Objetivo

> "The good thing about computers is that they do what you tell them to do. The bad news is that they do what you tell them to do."
>
> – Ted Nelson

Objetivo

> "The good thing about computers is that they do what you tell them to do. The bad news is that they do what you tell them to do."
> – Ted Nelson

¿Es posible hacer que una computadora se diga a sí misma qué hacer?

Objetivo

"The good thing about computers is that they do what you tell them to do. The bad news is that they do what you tell them to do."
– Ted Nelson

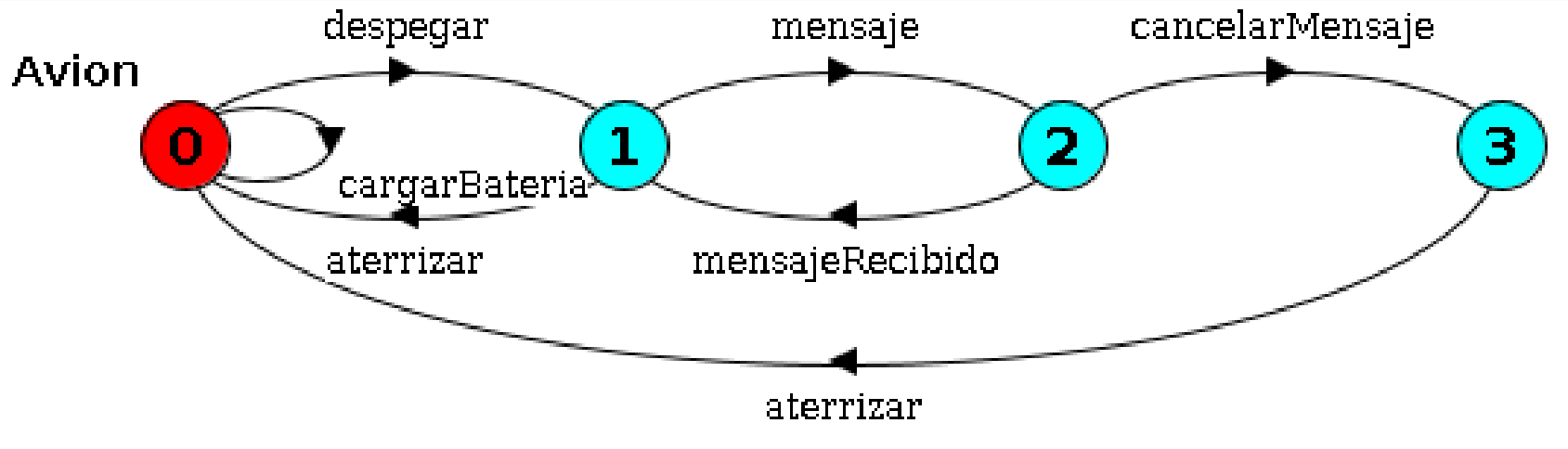
¿Es posible hacer que una computadora se diga a sí misma qué hacer?

Síntesis Automática de Controladores

Se le brinda a un programa las reglas y objetivos a cumplir, éste sintetiza una estrategia para ganar (si existe) conocida con el nombre de controlador.

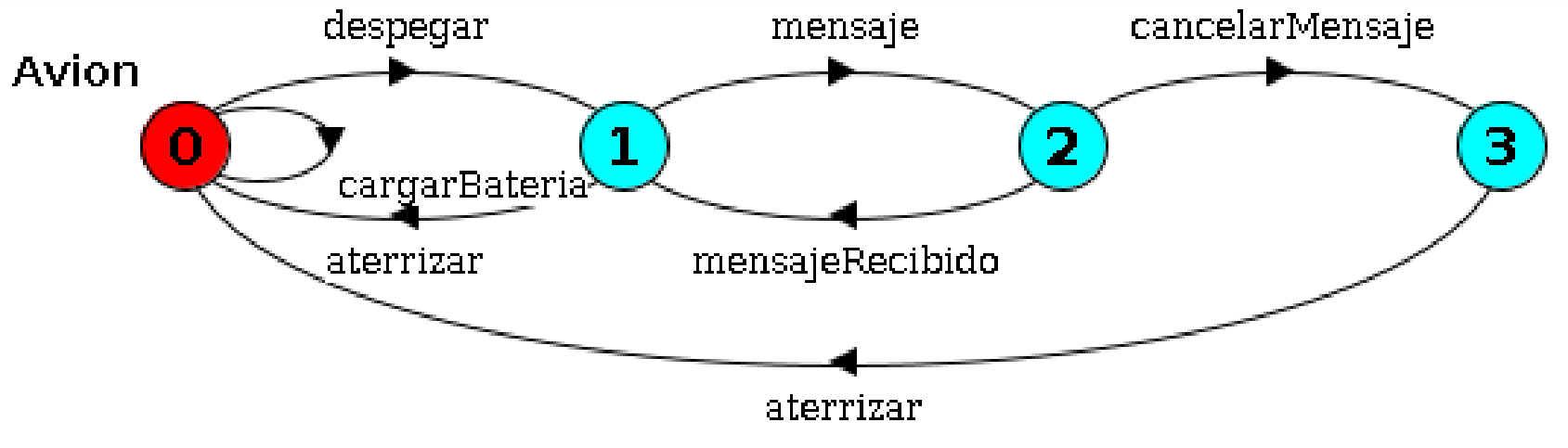
- Es una de las áreas que estudia problemas de síntesis.
- El problema es modelado usando autómatas finitos (o máquinas de estados finitos).

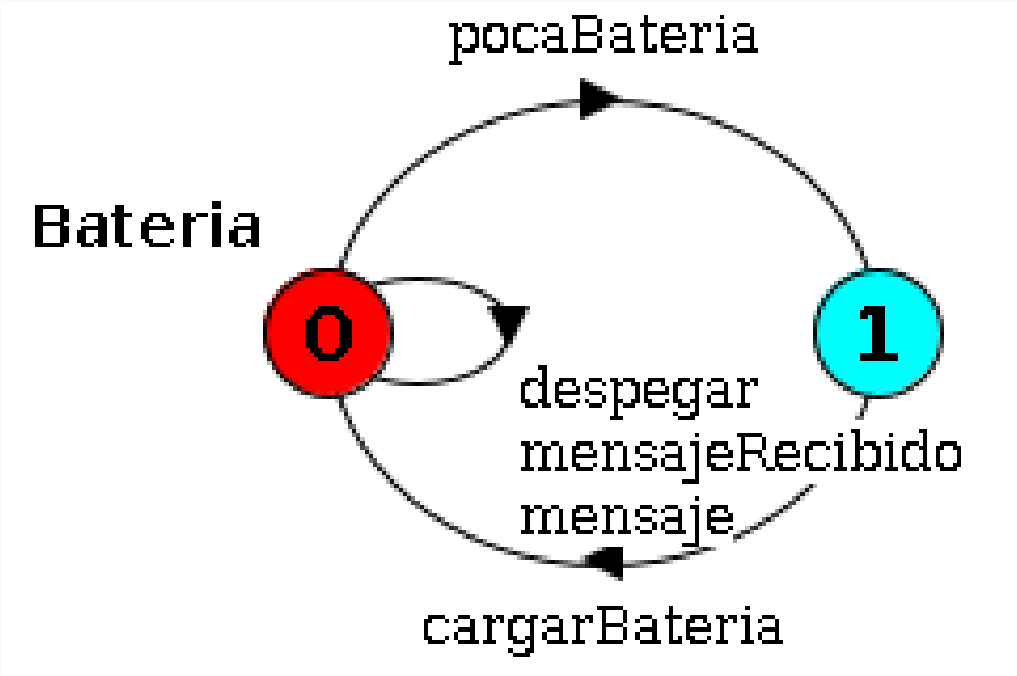
- Es una de las áreas que estudia problemas de síntesis.
- El problema es modelado usando autómatas finitos (o máquinas de estados finitos).
- Estos autómatas modelan la parte que nos interesa de la realidad.
- Se suele partir el modelo en pequeñas partes, más simples de abstraer, y luego se componen para formar el objeto de interés.



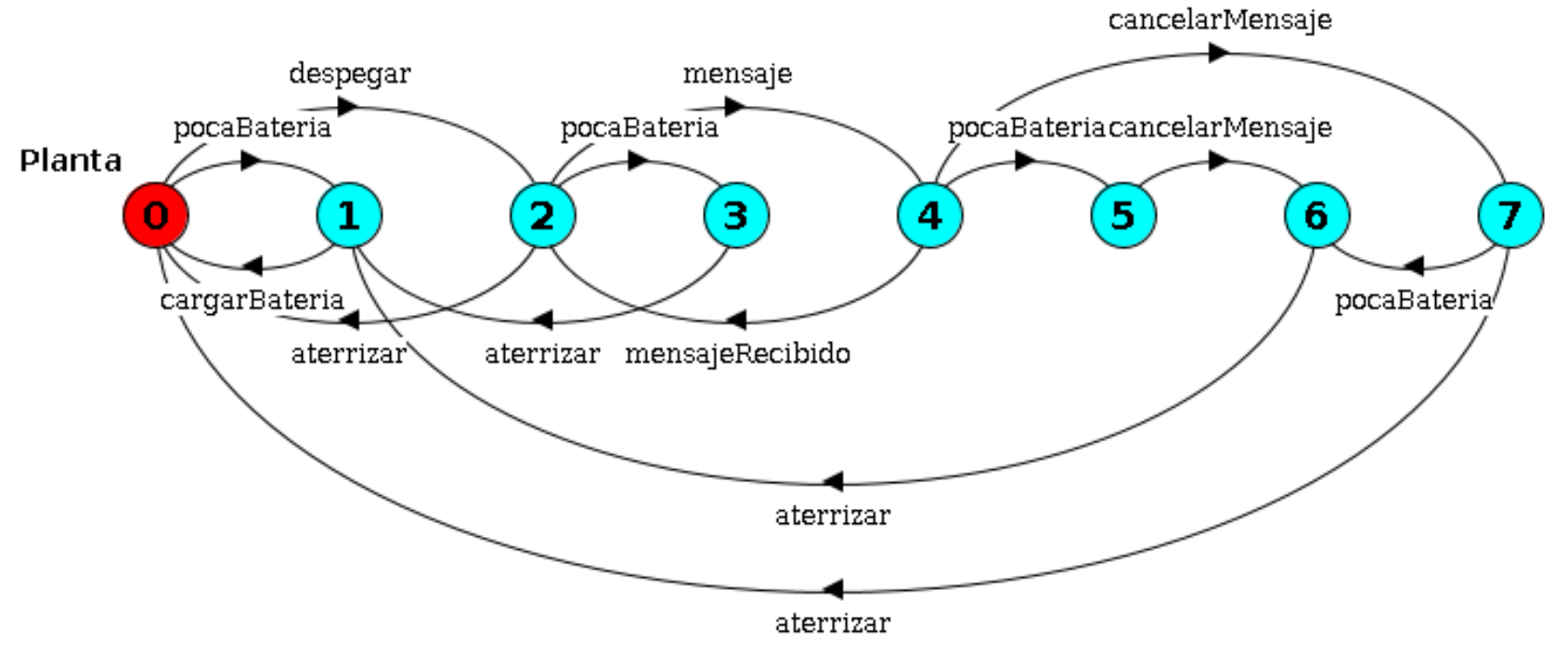
- Estados finitos, en este caso del 0 al 3 (4 estados).
- Estado inicial, en los ejemplos siempre será el 0.
- Acciones o eventos, en el ejemplo: despegar, aterrizar, cargarBateria, mensaje, mensajeRecibido, cancelarMensaje.

Componentes





Composición

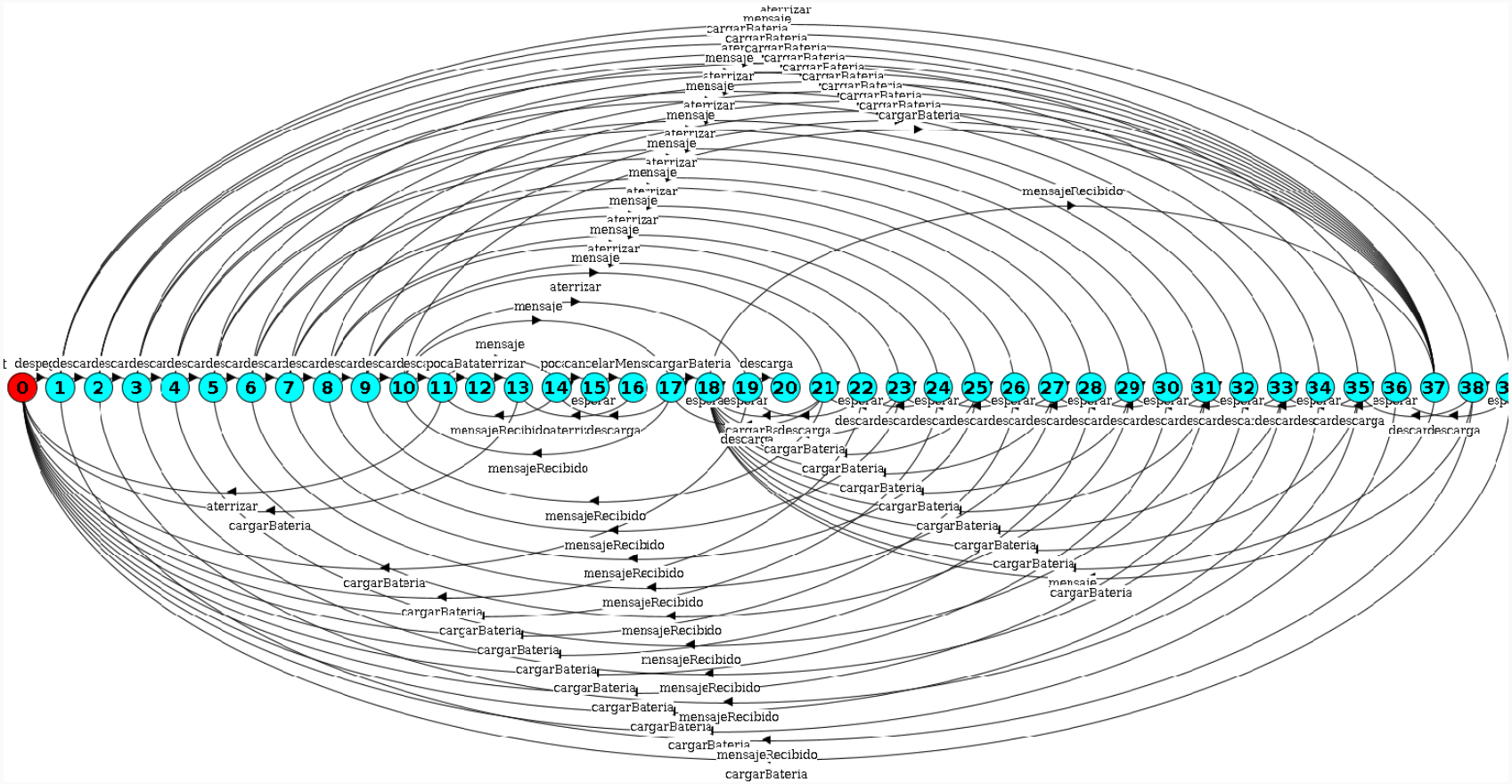


En nuestro caso el autómata *Batería* contiene la información de cómo actuar (qué acciones se pueden hacer) en caso de tener poca batería.

Al componer con el autómata *Avión* se restringen las acciones pero también hay un aumento de estados; ya que no es lo mismo estar en el aire con poca batería o completa.

Explosión de estados

La composición muchas veces explota en tamaño y termina siendo prohibitiva para algoritmos monolíticos (los que analizan la planta completa). Así queda el muy simple ejemplo anterior si la batería tiene un contador de carga:



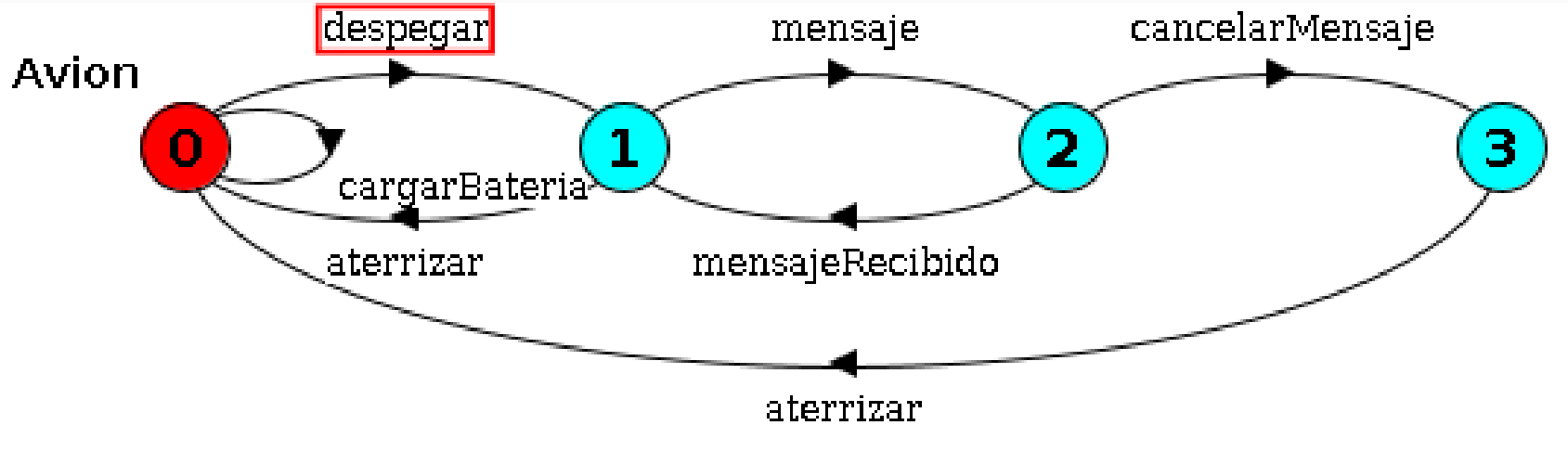
Idea general de este trabajo

Partiendo de las ideas presentadas en la tesis doctoral de Daniel Ciolek, queremos componer mientras se explora y terminar al tener una conclusión (con suerte antes de ver toda la composición).

La tesis doctoral incluye un algoritmo de exploración pero presenta algunos errores a la hora de clasificar los estados. Nuestro trabajo se centró en **diseñar un nuevo algoritmo** que solucione esos errores, y dar una **demostración** formal al respecto.

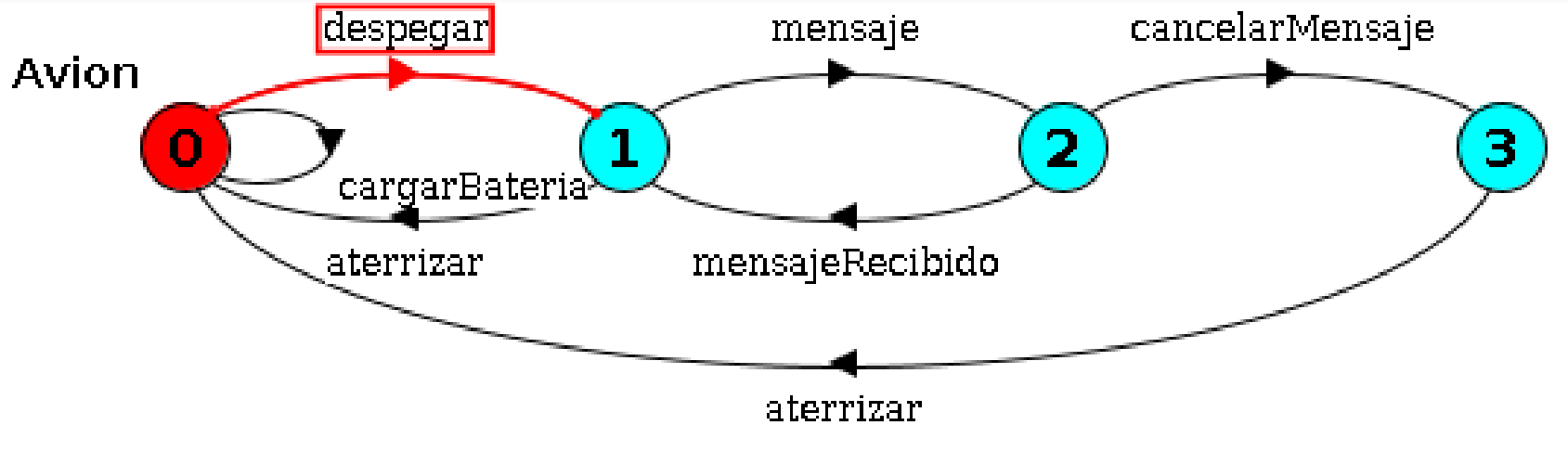
Conocimiento previo

Acciones, pasos y corridas (run)



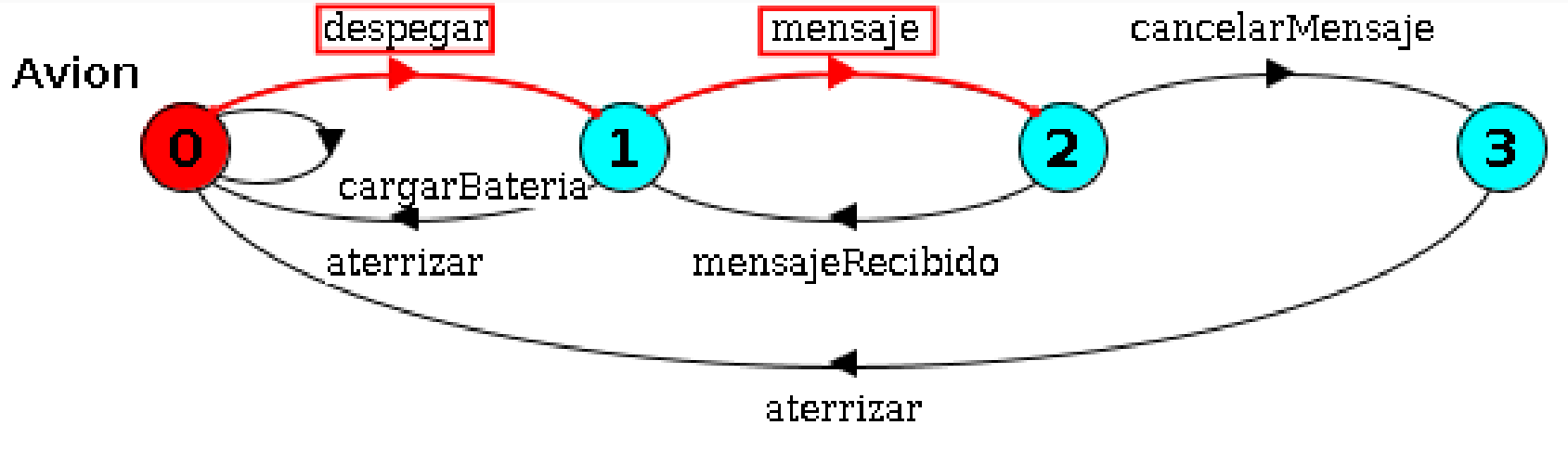
- Acción es una transición entre los estados.

Acciones, pasos y corridas (run)



- Acción es una transición entre los estados.
- Un paso es $t \xrightarrow{\ell}_T t'$, toma en cuenta el estado de partida y llegada.

Acciones, pasos y corridas (run)



- Acción es una transición entre los estados.
- Un paso es $t \xrightarrow{\ell}_T t'$, toma en cuenta el estado de partida y llegada.
- Una corrida (o traza) de una palabra $w = \ell_0, \ldots, \ell_k$ en T, es $t_0 \xrightarrow{w}_T t_{k+1}$, es decir, varios pasos.

Problema de control non-blocking

¿Cuál es la entrada de un problema de control?

- conjunto de autómatas (la composición de ellos es la planta completa que no queremos calcular)
- acciones controlables y no controlables
- estados marcados u objetivos (se quiere tener la *posibilidad* de visitar infinitas veces *al menos uno*)

¿Qué devuelve?

Una estrategia ganadora (llamada controlador) o afirmación de que no existe.

En nuestro ejemplo

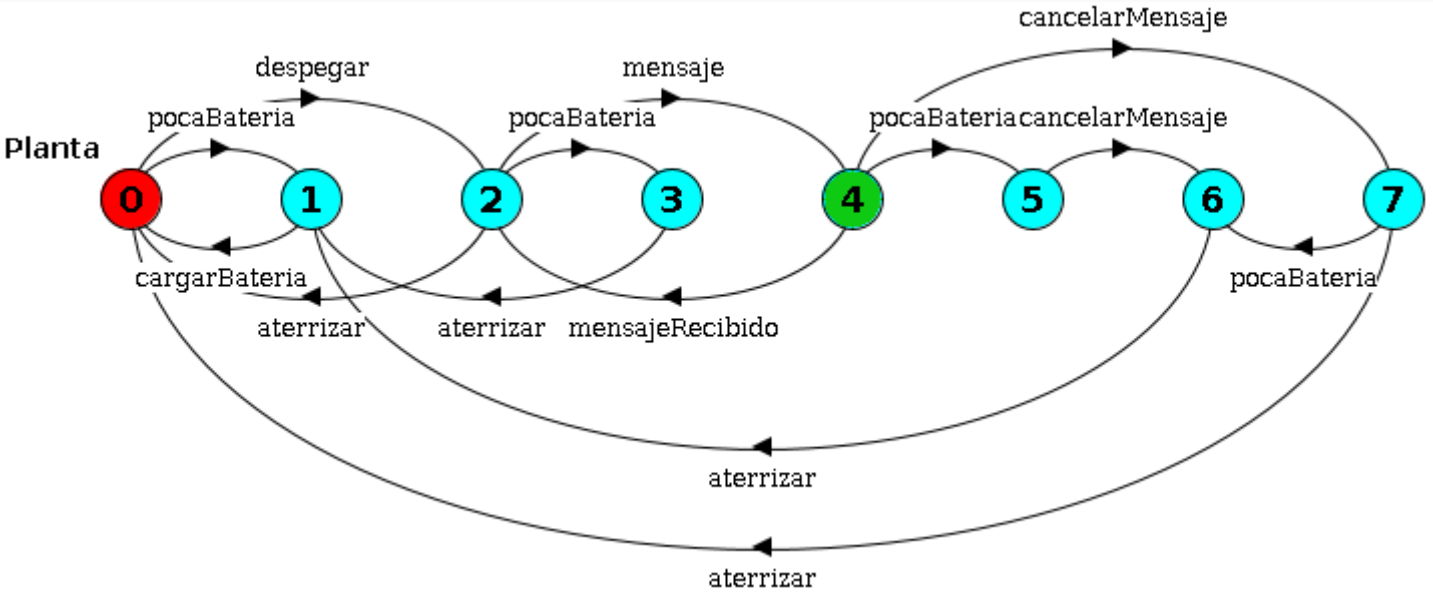
Conjunto de autómatas Avión, Batería.

Acciones controlables despegar, aterrizar, mensaje, cargarBateria, cancelarMensaje.

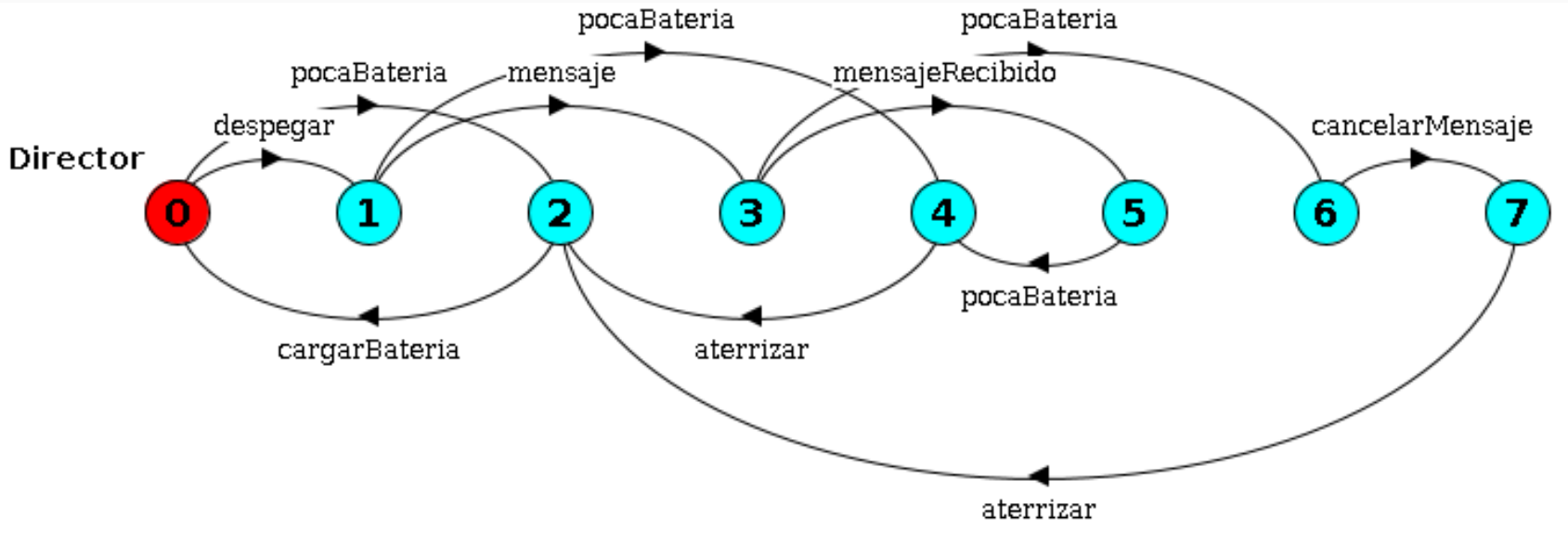
Acciones no-controlables mensajeRecibido, pocaBateria.

Objetivo Enviar *mensaje*.

Un algoritmo monolítico debe hacer la composición de todos los autómatas (usualmente llamado **Planta**) antes de empezar



Controlador

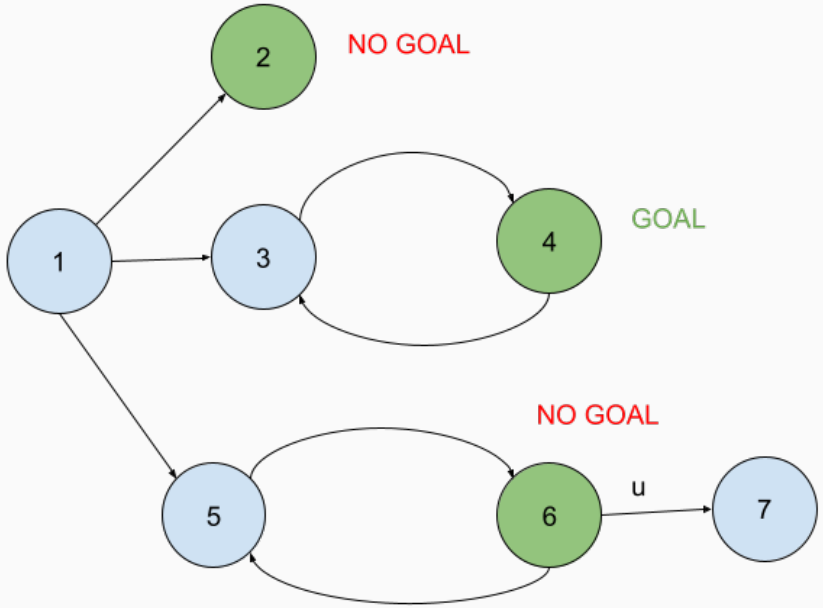


Es una restricción de la planta que cumple las reglas:

- Puede prohibir pasos controlables.
- Mantiene todas las acciones no-controlables.
- Todas las corridas posibles en la planta restringida tienen que poder extenderse para alcanzar algún estado marcado.
  - Incluso las que ya pasaron por estados marcados, es decir, se debe poder pasar infinitas veces por los marcados.

Estados ganadores y perdedores

Si existe un controlador comenzando desde un estado, es decir, existe estrategia ganadora a partir del estado, entonces lo consideramos estado ganador. Caso contrario, el estado es perdedor y debemos evitarlo.



Observación

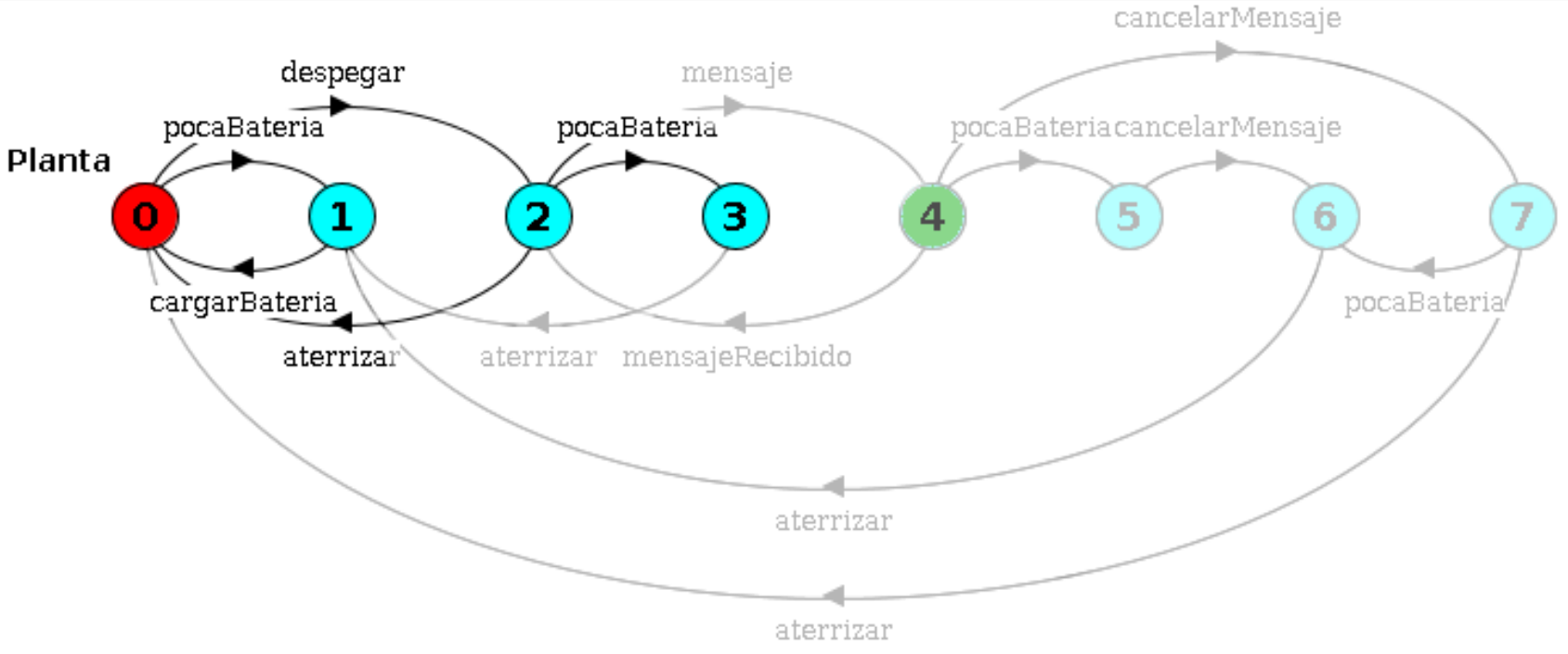
En particular nos interesa mucho si el estado inicial es ganador/perdedor, ya que eso nos dice si existe o no un controlador *empezando* desde ahí.

On-the-fly

Exploración parcial

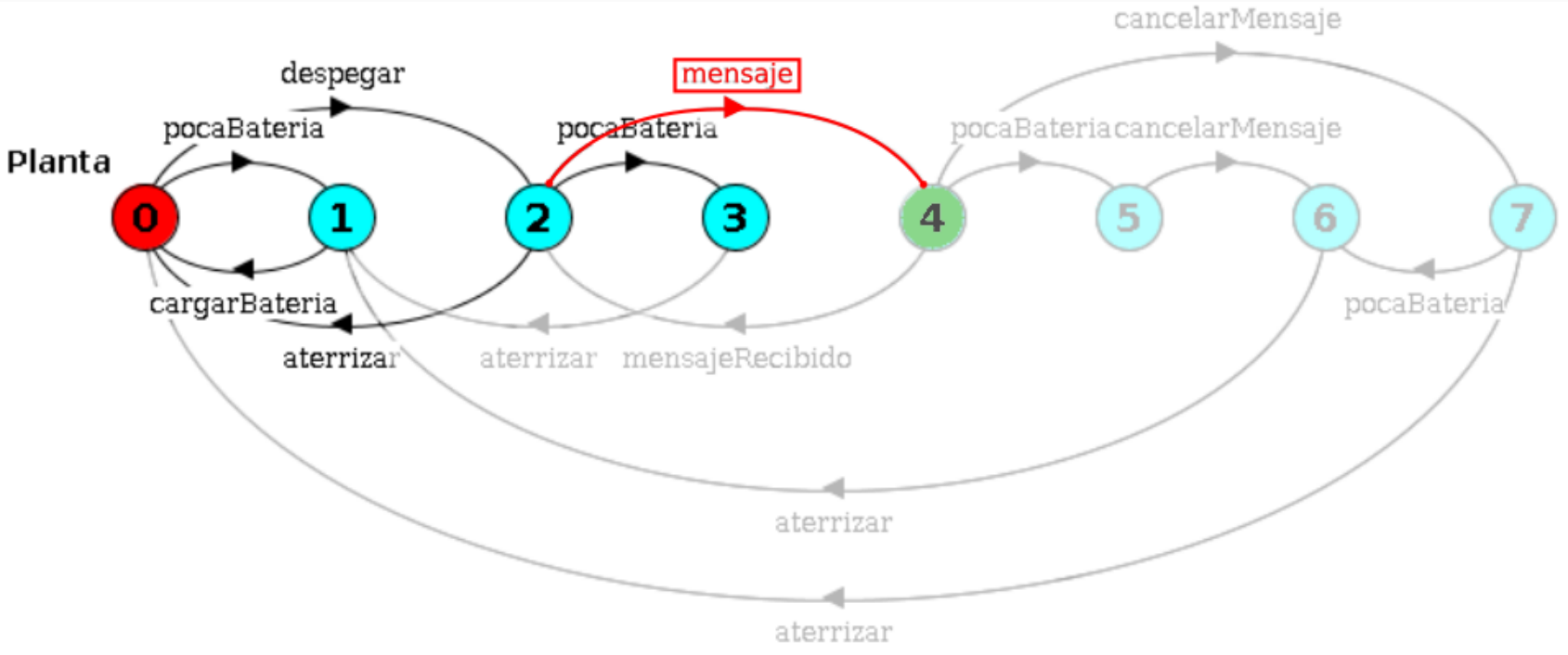
La idea es intentar sacar conclusiones a medida que se explora, y construye a la vez, la composición total.

Por ejemplo, en un momento dado podríamos tener explorado lo siguiente:



Exploramos de a una transición $e \xrightarrow{\ell} e'$ o paso.

Frontera optimista y pesimista



El paso $2 \overset{mensaje}{\rightarrow} 4$ está sin explorar. Como en principio no sabemos nada de él podríamos pensar que nos lleva a un estado ganador (frontera optimista) o perdedor (frontera pesimista).

Nueva definición de ganadores/perdedores

En la versión monolítica es más simple reconocer ganadores/perdedores porque se cuenta con toda la información de la planta; en on-the-fly debemos "imaginar" a dónde van a parar las transiciones no vistas.

Actualizamos estados sólo cuando estamos seguros, es decir, clasificamos ganadores sólo al pensar una frontera pesimista y perdedores en frontera optimista.

Observación

Los estados ganadores/perdedores según esta definición son ganadores/perdedores en la planta completa, es decir, una vez que clasificamos un estado nunca debemos reconsiderarlo.

Propagar información

Si estamos explorando una transición nueva $e \xrightarrow{\ell} e'$:

Propagar información

Si estamos explorando una transición nueva $e \xrightarrow{\ell} e'$:

Un estado s es un **nuevo** ganador/perdedor si: sin explorar la última transición no teníamos suficiente información y ahora sí.

Propagar información

Si estamos explorando una transición nueva $e \xrightarrow{\ell} e'$:

Un estado s es un **nuevo** ganador/perdedor si: sin explorar la última transición no teníamos suficiente información y ahora sí.

s solo tiene nueva información si **tiene un camino** hasta la nueva transición (s tiene que ser un antecesor de e).

Propagar información

Si estamos explorando una transición nueva $e \xrightarrow{\ell} e'$:

Un estado s es un **nuevo** ganador/perdedor si: sin explorar la última transición no teníamos suficiente información y ahora sí.

s solo tiene nueva información si **tiene un camino** hasta la nueva transición (s tiene que ser un antecesor de e).

Si no sabemos si e' gana o pierde, no tenemos información para propagar, porque **la transición nueva va a lo desconocido**.

Obtener nueva información (Ganador)

Como vimos, queremos trazas que visiten **infinitas** veces un estado marcado.

Obtener nueva información (Ganador)

Como vimos, queremos trazas que visiten **infinitas** veces un estado marcado. Una traza infinita en un autómata finito solo puede ocurrir si forma un **ciclo** (loop).

Obtener nueva información (Ganador)

Como vimos, queremos trazas que visiten **infinitas** veces un estado marcado. Una traza infinita en un autómata finito solo puede ocurrir si forma un **ciclo** (loop).

Para que exista la chance de ganar, ese ciclo debe tener un **estado marcado** o previamente **clasificado ganador**.

Obtener nueva información (Ganador)

Como vimos, queremos trazas que visiten **infinitas** veces un estado marcado. Una traza infinita en un autómata finito solo puede ocurrir si forma un **ciclo** (loop).

Para que exista la chance de ganar, ese ciclo debe tener un **estado marcado** o previamente **clasificado ganador**.

Al explorar una nueva transición $(e \xrightarrow{\ell} e')$ y en el caso que e' no esté clasificado, analizamos el ciclo **mas grande** que forma. Si no podemos ganar en él, no obtuvimos nuevos ganadores.

Obtener nueva información (Ganador)

Como vimos, queremos trazas que visiten **infinitas** veces un estado marcado. Una traza infinita en un autómata finito solo puede ocurrir si forma un **ciclo** (loop).

Para que exista la chance de ganar, ese ciclo debe tener un **estado marcado** o previamente **clasificado ganador**.

Al explorar una nueva transición $(e \xrightarrow{\ell} e')$ y en el caso que e' no esté clasificado, analizamos el ciclo **mas grande** que forma. Si no podemos ganar en él, no obtuvimos nuevos ganadores.

Para realmente ser ganador, el ciclo debe ser **controlablemente cerrado**. Intuición: Si una transición no controlable sale del ciclo, un controlador no puede mantenerse ahí dentro y ganar. Y si por no mantenerse adentro, va a un estado no explorado, **somos pesimistas** y asumimos que pierde.

Obtener nueva información (Perdedor)

Descubrimos nuevos estados perdedores cuando **seguro** sabemos que no forman trazas con infinitos estados marcados.

Obtener nueva información (Perdedor)

Descubrimos nuevos estados perdedores cuando **seguro** sabemos que no forman trazas con infinitos estados marcados.

Esto es obvio cuando el estado solo tiene la traza vacía. **No puede ir a ningún otro estado**.

Obtener nueva información (Perdedor)

Descubrimos nuevos estados perdedores cuando **seguro** sabemos que no forman trazas con infinitos estados marcados.

Esto es obvio cuando el estado solo tiene la traza vacía. **No puede ir a ningún otro estado**.

Más difícil de detectar son los **ciclos sin estados ganadores**.

Obtener nueva información (Perdedor)

Descubrimos nuevos estados perdedores cuando **seguro** sabemos que no forman trazas con infinitos estados marcados.

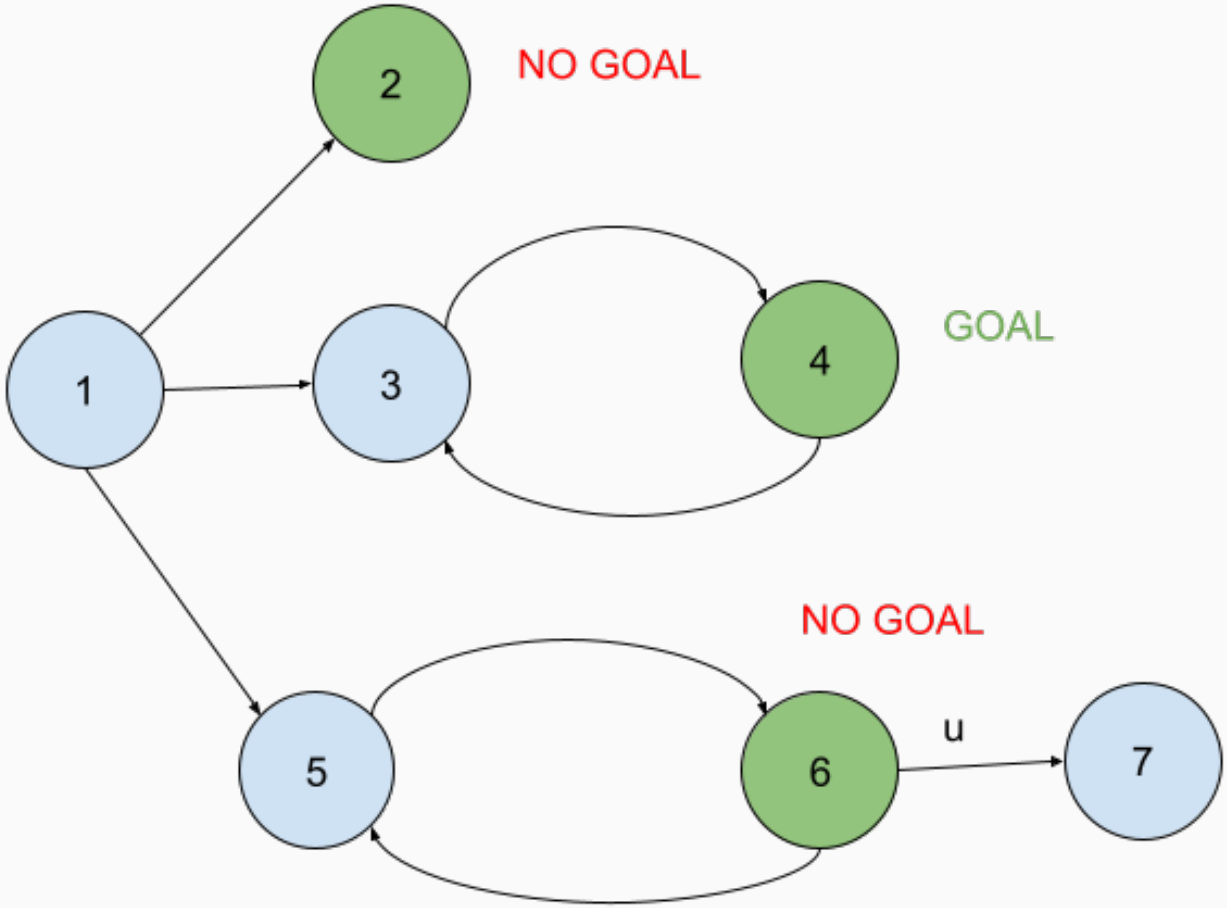
Esto es obvio cuando el estado solo tiene la traza vacía. **No puede ir a ningún otro estado**.

Más difícil de detectar son los **ciclos sin estados ganadores**.

Caso General

Si en todos los descendientes de un estado e, no hay ningún **ciclo ganador**, entonces e tiene que ser perdedor.
Tenemos que chequear **todos** los descendientes, porque siendo **optimistas**, un descendiente desconocido podría ser ganador.

Repaso del algoritmo con ejemplo previo



Resumen del nuevo algoritmo

- Clasificamos (ganador/perdedor) cuando estamos seguros.
- Propagamos a los antecesores afectados (si hay información respecto a e').
- Tratamos de obtener nueva información sólo al cerrar loops.

Observación:

Sólo hacemos nuevos cálculos cuando son realmente necesarios.

Nos interesa saber qué tan buena es la eficiencia del algoritmo, en tiempo de cómputo.

Implementación y Tests

Herramienta MTSA

- El LaFHIS, donde hicimos esta tesis, trabaja hace años en MTSA, una herramienta propia para resolver problemas de control con LTS.
- Implementamos el algoritmo en java, agregándolo a las capacidades de MTSA. Esto permitió ejecutar los tests y el benchmark de la próxima sección.

Test Driven Development

- Para ganar seguridad en nuestro código, y encontrar errores, fuimos armando una batería de tests.
- Con cada error encontrado en la implementación o la especificación del algoritmo, armábamos un nuevo test que detectara ese error, y luego lo arreglábamos.
- Quedaron 49 tests pequeños diseñados a mano para correr rápido y presentar las condiciones más problemáticas para nuestro algoritmo.
- Esta batería de tests fue añadida a las utilizadas por la herramienta como tests de regresión para alertar problemas por futuros cambios en el código.

Ejemplo de MTSA en uso

MTS Analyser

File Edit Check Build MTS Window Options Update Help

DEFAULT

Edit Output Draw Layout

```
set Controlables = {despegar, aterrizar, mensaje, cargarBateria, cancelarMensaje}
set NoControlables = {mensajeRecibido, pocaBateria}

Avion = EnTierra,
EnTierra = (despegar -> EnAire | cargarBateria -> EnTierra),
EnAire = (aterrizar -> EnTierra | mensaje -> EsperandoConfirmacion),
EsperandoConfirmacion = (mensajeRecibido -> EnAire | cancelarMensaje -> aterrizar -> EnTierra).

Bateria = Completa,
Completa = (despegar -> Completa | pocaBateria -> PocaBateria | mensaje -> Completa | mensajeRecibid
PocaBateria = (cargarBateria -> Completa).

||Planta = (Avion || Bateria).

controllerSpec Goal = {
  controllable = {Controlables}
  marking = {mensajeRecibido}
  nonblocking
}

heuristic ||Director = Planta~{Goal}.
```

Ejemplo de MTSA en uso

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

MTS Analyser

File Edit Check Build MTS Window Options Update Help

DEFAULT

Edit Output Draw Layout

```
set Controlables = {despegar, aterrizar, mensaje, cargarBateria, cancelarMensaje}
set NoControlables = {mensajeRecibido, pocaBateria}

Avion = EnTierra,
EnTierra = (despegar -> EnAire |
EnAire = (aterrizar -> EnTierra |
EsperandoConfirmacion = (mensajeR

Bateria = Completa,
Completa = (despegar -> Completa
PocaBateria = (cargarBateria -> C

||Planta = (Avion || Bateria).

controllerSpec Goal = {
  controllable = {Controlables}
  marking = {mensajeRecibido}
  nonblocking
}

heuristic ||Director = Planta~{Go
```

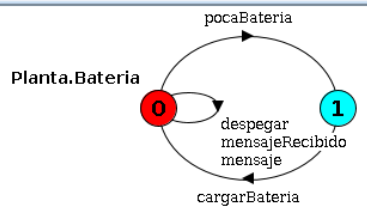
MTS Analyser

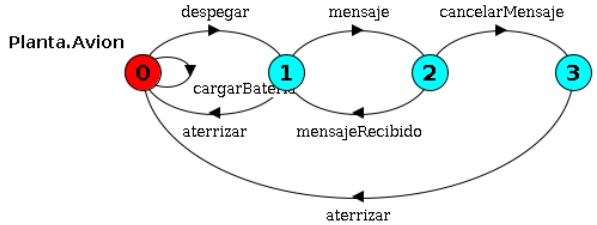
File Edit Check Build MTS Window Options Update Help

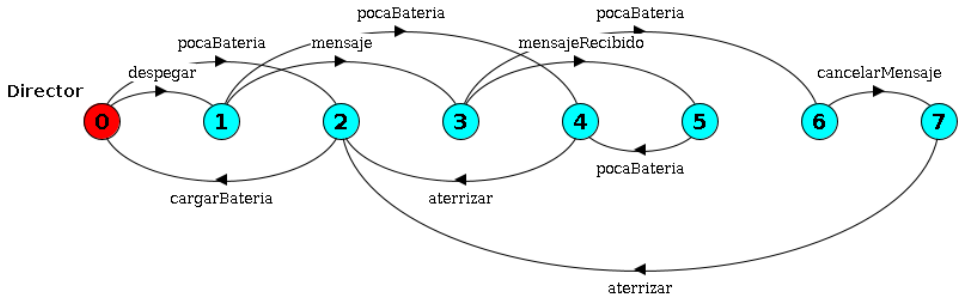
Director

Edit Output Draw Layout

Planta.Avion
Planta.Bateria
||Director





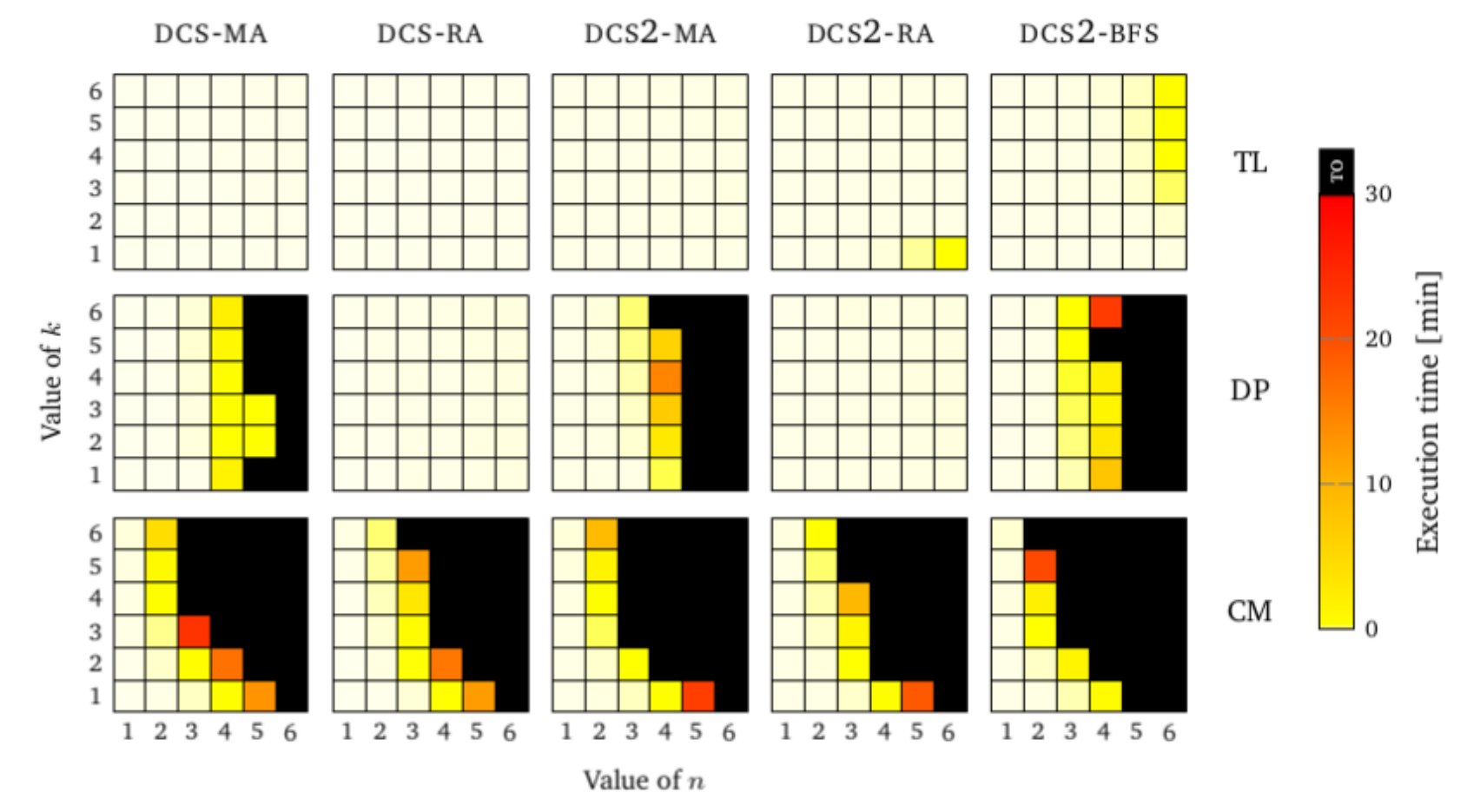


Benchmark

Benchmark

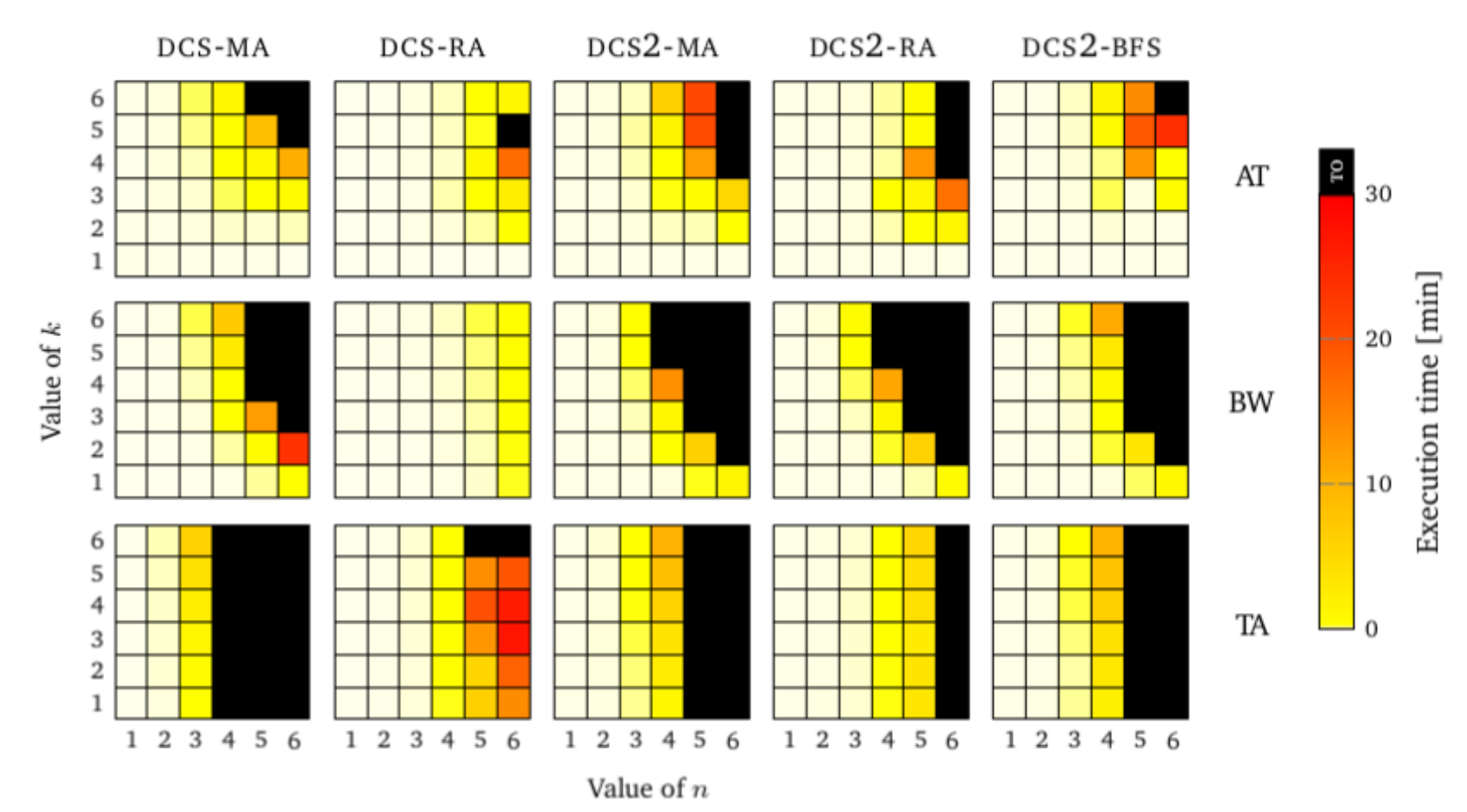
- Para realizar las pruebas de performance usamos el mismo conjunto de problemas utilizado para medir la versión anterior del algoritmo de exploración, recompilados por Daniel Ciolek en su tesis doctoral.
- Es un conjunto de seis tipos de problemas bastante clásicos, cada uno con dos partes parametrizables (n, k), en función de observar hasta qué tamaño de problema (n*k) soporta el algoritmo.
- DCS y DCS2 representan la versión anterior/nueva del algoritmo respectivamente.
- Utilizamos distintas estrategias a la hora de explorar, algunas más complejas (MA, RA) desarrolladas por Ciolek y una simple (BFS). La última fue agregada para obtener una idea de cuánto se puede mejorar cambiando la forma de explorar.

Performance (TL, DP, CM)



Transfer Line, Dinning Philosophers, Cat and Mouse

Performance (AT, BW, TA)



Air-Traffic Management, Bidding Workflow, Travel Agency

Conclusión

Qué vimos

- Definición del problema

Qué vimos

- Definición del problema
- Soluciones existentes y la idea “nueva” (exploración on-the-fly)

Qué vimos

- Definición del problema
- Soluciones existentes y la idea “nueva” (exploración on-the-fly)
- Nuevo algoritmo, parte de su demostración (corrección y completitud)

Qué vimos

- Definición del problema
- Soluciones existentes y la idea “nueva” (exploración on-the-fly)
- Nuevo algoritmo, parte de su demostración (corrección y completitud)
- Implementación en MTSA

Qué vimos

- Definición del problema
- Soluciones existentes y la idea “nueva” (exploración on-the-fly)
- Nuevo algoritmo, parte de su demostración (corrección y completitud)
- Implementación en MTSA
- Batería de tests, TDD

Qué vimos

- Definición del problema
- Soluciones existentes y la idea “nueva” (exploración on-the-fly)
- Nuevo algoritmo, parte de su demostración (corrección y completitud)
- Implementación en MTSA
- Batería de tests, TDD
- Benchmark y resultados versus la versión anterior. No se perdió eficiencia, teniendo en cuenta la confianza ganada en correctitud. En la tesis se encuentran además resultados de benchmark versus otras herramientas del estado del arte.

Qué vimos

- Definición del problema
- Soluciones existentes y la idea “nueva” (exploración on-the-fly)
- Nuevo algoritmo, parte de su demostración (corrección y completitud)
- Implementación en MTSA
- Batería de tests, TDD
- Benchmark y resultados versus la versión anterior. No se perdió eficiencia, teniendo en cuenta la confianza ganada en correctitud. En la tesis se encuentran además resultados de benchmark versus otras herramientas del estado del arte.

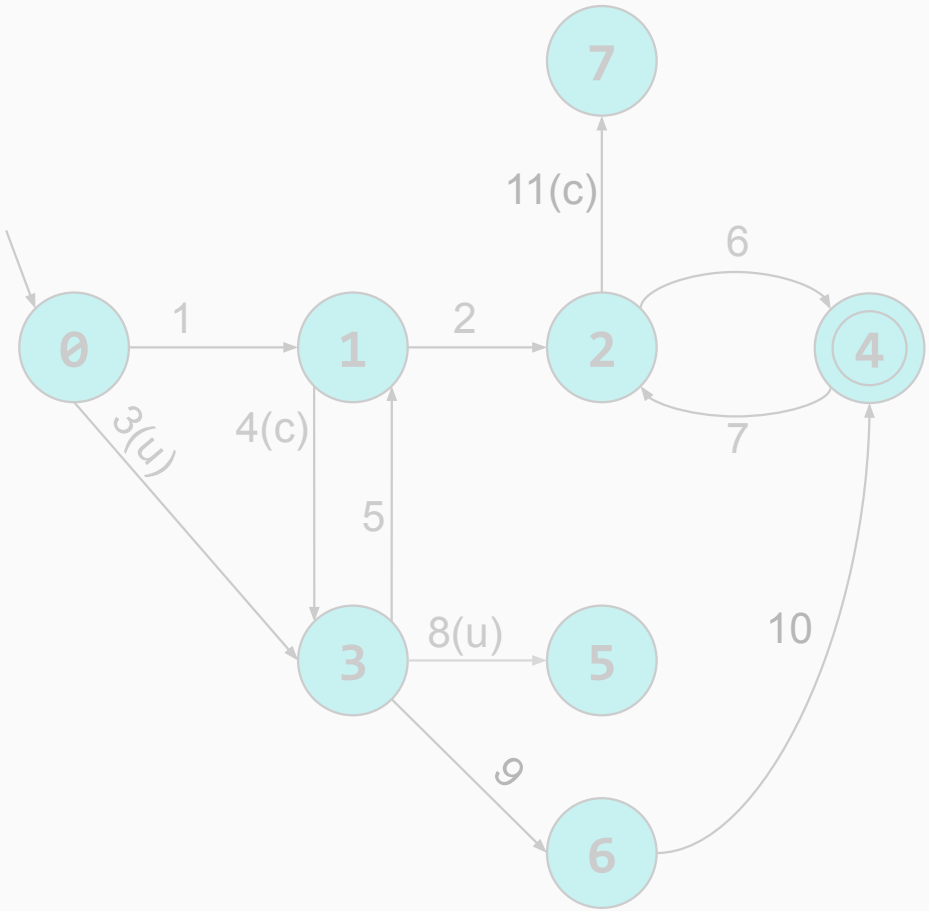
La idea de exploración on-the-fly, y gran parte de la estructura del algoritmo, se puede aplicar a otro tipo de problemas, ej: GR1 (próximamente).

¿Preguntas?

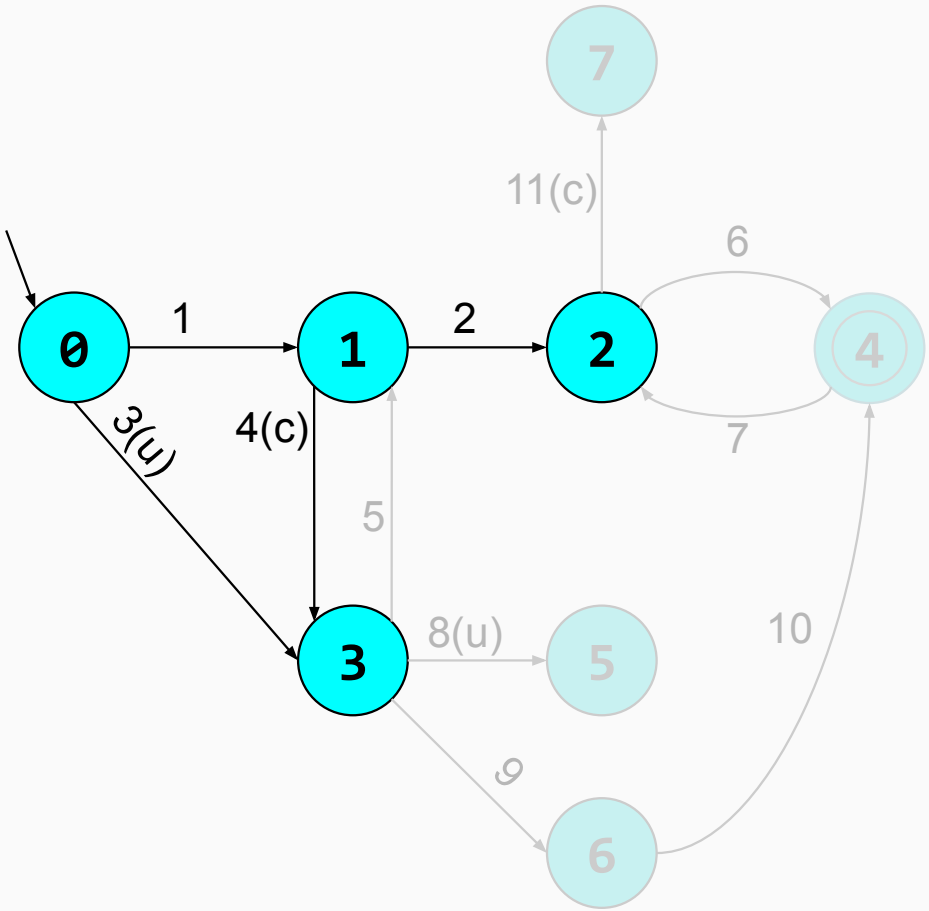
Gracias, vuelvan prontos

(para ver GR1)

Bonus track: Detalles de la exploración



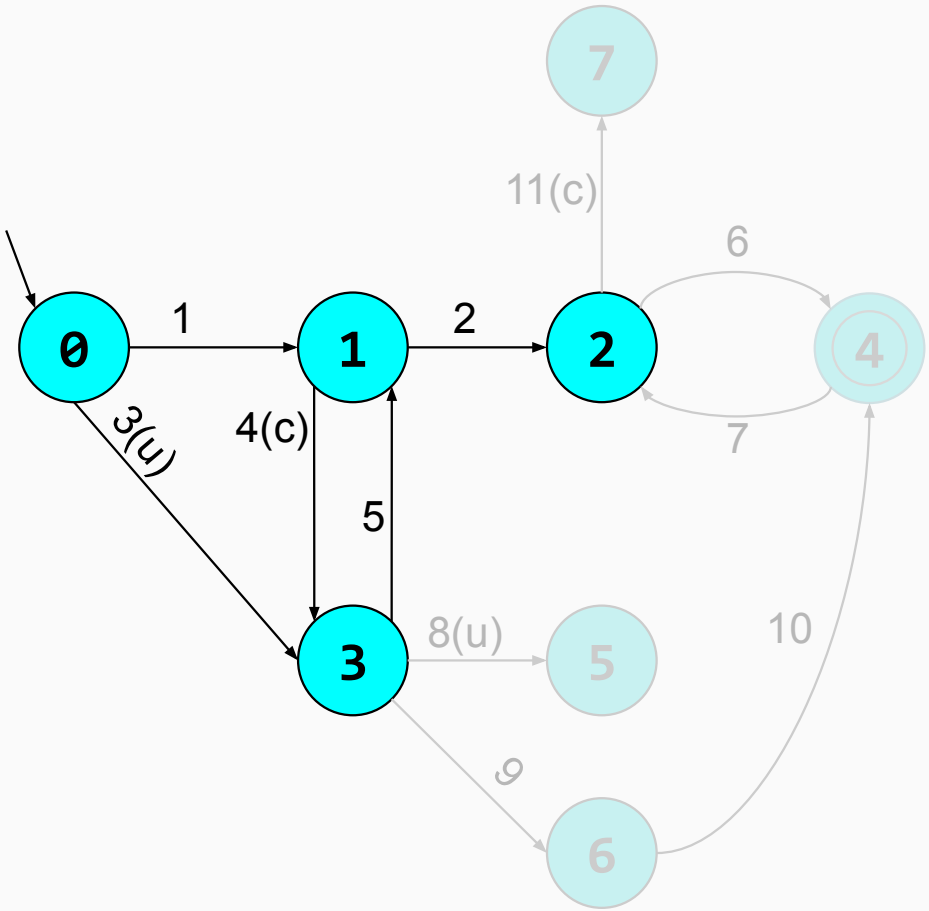
Bonus track: Detalles de la exploración



Empieza por el estado **0** y mira los pasos:

- $0 \xrightarrow{1} 1$
- $1 \xrightarrow{2} 2$
- $0 \xrightarrow{3(u)} 3$
- $1 \xrightarrow{4(c)} 3$

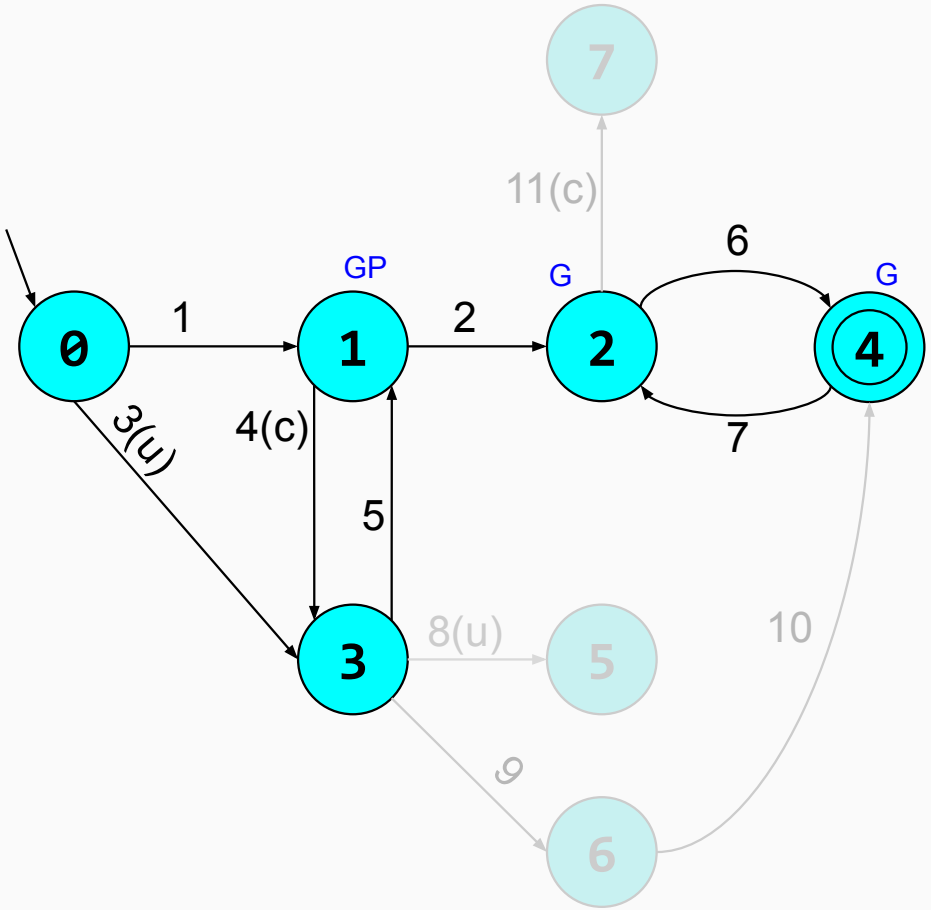
Bonus track: Detalles de la exploración



Explora el paso $3 \xrightarrow{5} 1$

Cierra loop pero todavía no hay estado marcado (u objetivo).

Bonus track: Detalles de la exploración

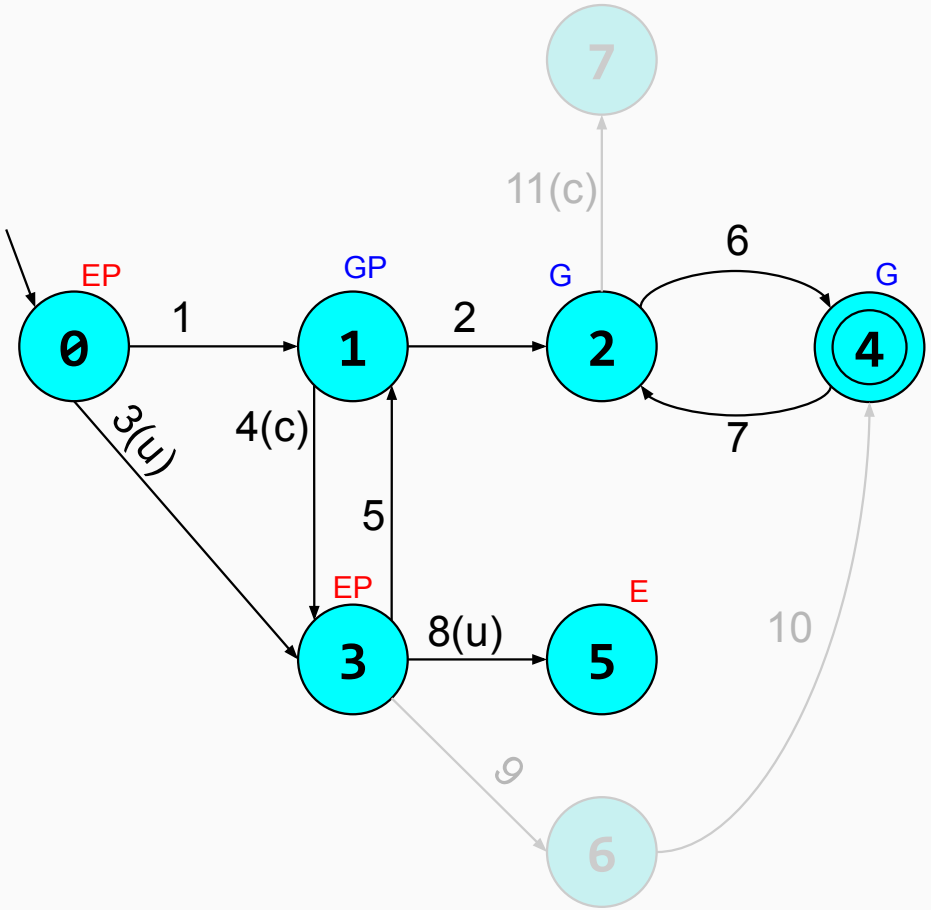


Mira los pasos:

- $2 \xrightarrow{6} 4$
- $4 \xrightarrow{7} 2$

Cierra loop que incluye el estado marcado **4**, encontramos una parte ganadora, propagamos esa información.

Bonus track: Detalles de la exploración



Explora 3 $\xrightarrow{8}$ 5, encuentra que **5** es perdedor y propaga la información. Llega al inicial entonces concluye que no hay controlador.