



Directed Controller Synthesis for Non-Maximal Blocking Requirements

Matias Duran, Florencia Zanollo

12 de Mayo de 2021

- Introducción
- Conocimiento previo
- On-the-fly
- Implementación y Tests
- Benchmark
- Conclusión

Introducción

“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

“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?

“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

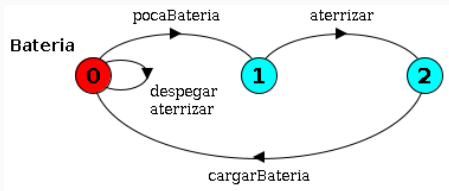
¿Es posible hacer que una computadora se diga a sí misma qué hacer?

Síntesis Automática de Controladores

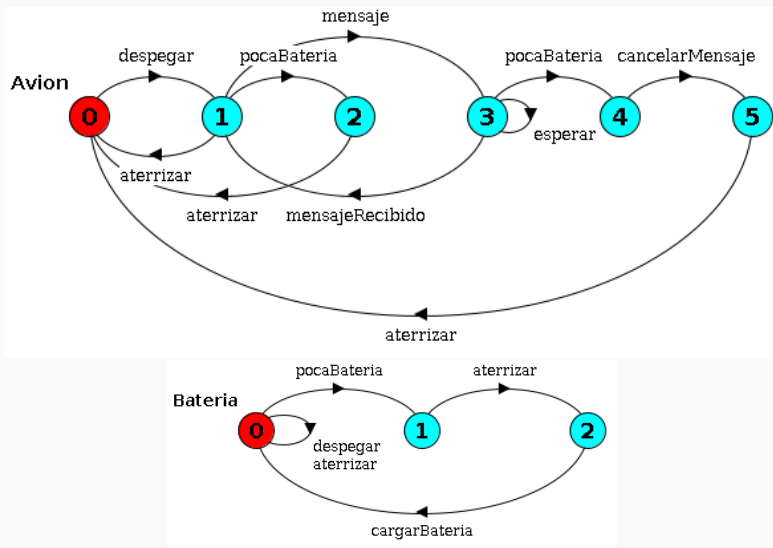
Se le brinda a un programa las reglas y objetivos a cumplir, éste sintetiza una estrategia para ganar (si existe) conocida con el nombre de controlador.

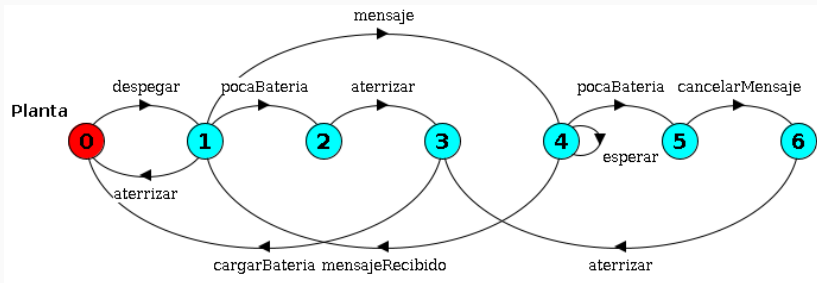
- Es una de las áreas que estudia problemas de síntesis.
- El problema es modelado usando autómatas finitos (o máquinas de estados finitos).

- 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.



detalles de los autómatas finitos (estado inicial, marcados (ver si va), etc)





ejemplo simple para explicar composición (mostrar planta completa) explicar por arriba cómo se compone o cuál es la idea (la planta refleja el comportamiento del todo) (reglas para sincronización entre las partes)

Partiendo de la versión presentada en la tesis doctoral de Daniel Ciolek, queremos componer mientras exploras y terminar al tener conclusión (con suerte antes de ver todo).

Dicha versión presentaba errores a la hora de clasificar los estados, nuestra intención es, no sólo arreglar los errores, sino dar una demostración formal al respecto.

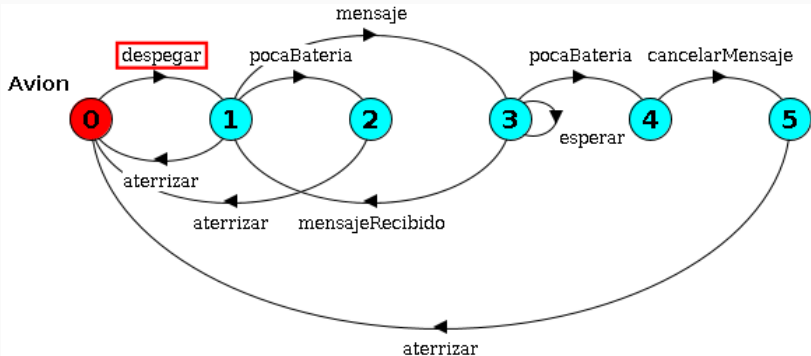
Conocimiento previo

¿Cuál es la entrada de un problema de control?

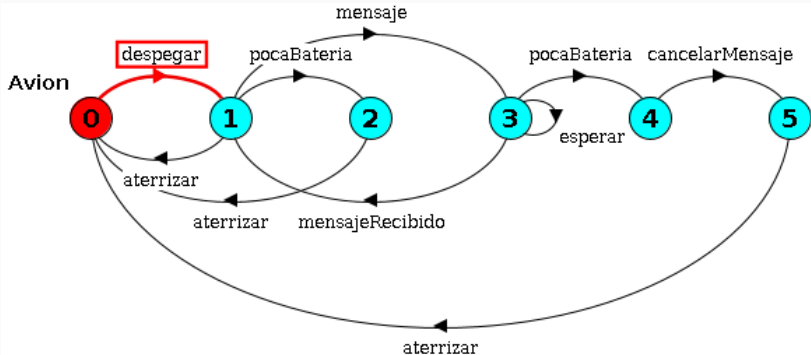
- conjunto de autómatas (la composición de ellos es la planta completa que no queremos calcular)
- acciones controlables y no controlables
- estados marcados u objetivos (se quiere tener la *posibilidad* de visitar infinitas veces *al menos uno*)

¿Qué devuelve?

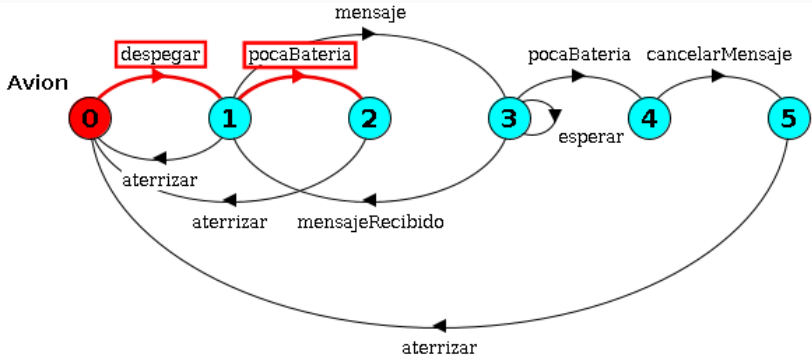
Una estrategia ganadora (llamada controlador) o afirmación de que no existe.



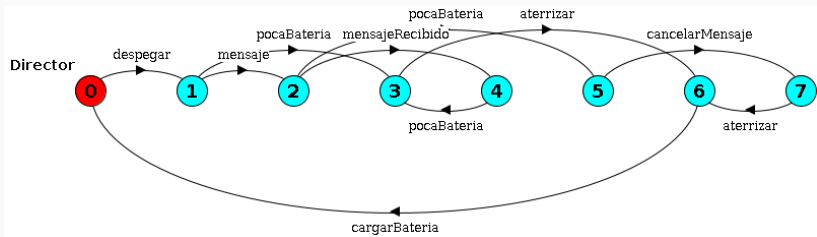
- Acción es una transición entre los estados.



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



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



Es una restricción de la planta (la composición de todos los autómatas) que cumple las reglas:

- Puede prohibir pasos controlables.
- Debe mantener todos los no-controlables.
- Todas las corridas posibles en la planta restringida pasan por algún estado marcado.

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.

[Agregar ejemplos imágenes]

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.

Necesitamos un ciclo (loop) para ganar

a qué nos referimos con loop

La única forma de visitar infinitas veces un estado es que la corrida sea un ciclo.

[pictures pictures and more pictures]

On-the-fly

i.e. es lo visto hasta el momento

estados ganadores y perdedores ahora (antes era fácil, ahora tenemos que ver a dónde van)

def el concepto de “frontera” (en el sentido de transiciones existentes sin explorar) top, bottom

actualizamos estado solo cuando estamos seguros (ganador en bottom / perdedor en top) Obs: los estados ganadores/perdedores según esta definición son ganadores en la planta completa (WESy LES son monótonos).

exploramos de a una transición $e \xrightarrow{\ell} e'$. ejemplo de la tesis como animación?

Lema2 ganador/perdedor es antecesor explicar por qué o lema o ej de imagen, ver la mejor manera

Lema3 hay info para propagar si e' era conocido Y era ganador/perdedor (cc. no sabemos nada nuevo)

Lema4 como dijimos antes, necesitamos un loop para ganar, entonces sólo va a haber ganadores nuevos al encontrarnos dicho loop (i.e. el e' es uno que vimos Y no es ni ganador/perdedor porque sino cae en el caso anterior)

Lema5 si ya exploraste todo y no hay marcados por ningún lado entonces obvio que no podés llegar a ninguno [dibujo]

- 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:

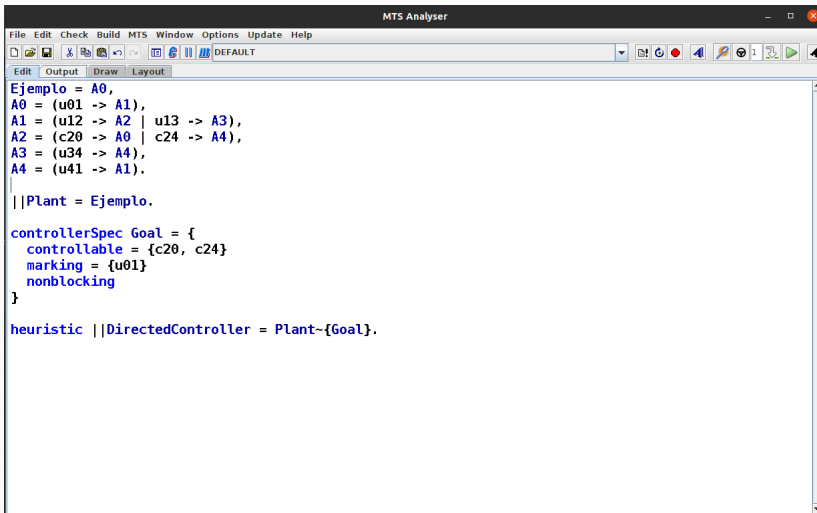
Sólo hacemos nuevos cálculos cuando son necesarios.

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

Implementación y Tests

- 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.

- 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.



The screenshot shows the MTS Analyser application window. The title bar reads "MTS Analyser". The menu bar includes "File", "Edit", "Check", "Build", "MTS", "Window", "Options", "Update", and "Help". The toolbar contains icons for file operations (open, save, print), editing (undo, redo, copy, paste), and analysis (run, stop, step through). The "Edit" tab is active, showing a text editor with the following code:

```
Ejemplo = A0,  
A0 = (u01 -> A1),  
A1 = (u12 -> A2 | u13 -> A3),  
A2 = (c20 -> A0 | c24 -> A4),  
A3 = (u34 -> A4),  
A4 = (u41 -> A1).  
  
||Plant = Ejemplo.  
  
controllerSpec Goal = {  
  controllable = {c20, c24}  
  marking = {u01}  
  nonblocking  
}  
  
heuristic ||DirectedController = Plant~{Goal}.
```

MTS Analyser

File Edit Check Build MTS Window Options Update Help

Edit Output Draw Layout

```
Ejemplo = A0,  
A0 = (u01 -> A1),  
A1 = (u12 -> A2 | u13 -> A3),  
A2 = (c20 -> A0 | c24 -> A4),  
A3 = (u34 -> A4),  
A4 = (u41 -> A1).
```

||Plant = Ejemplo.

```
controllerSpec Goal = {  
  controllable = {c20, c24}  
  marking = {u01}  
  nonblocking  
}
```

heuristic ||DirectedController =

MTS Analyser

File Edit Check Build MTS Window Options Update Help

DirectedController

Edit Output Draw Layout

Plant:Ejemplo
||DirectedController

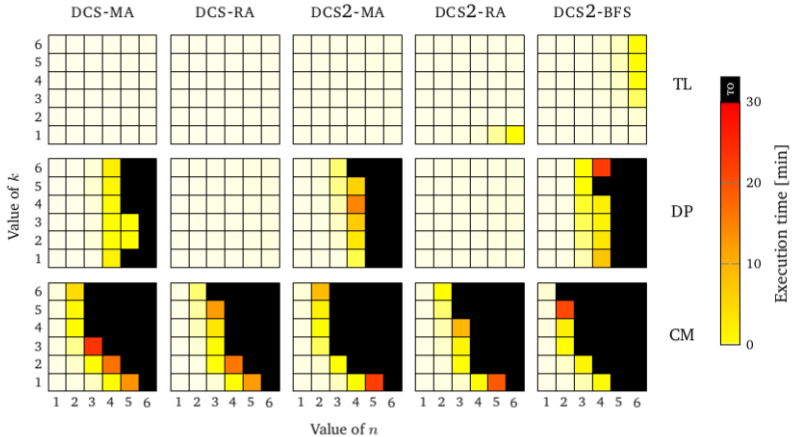
```
graph LR  
  0((0)) -- u01 --> 1((1))  
  1 -- u12 --> 2((2))  
  1 -- u13 --> 2  
  2 -- u13 --> 3((3))  
  2 -- c20 --> 0  
  3 -- u34 --> 4((4))  
  4 -- u41 --> 5((5))  
  3 -- u13 --> 5  
  5 -- u12 --> 2
```

Video de controlador animado?

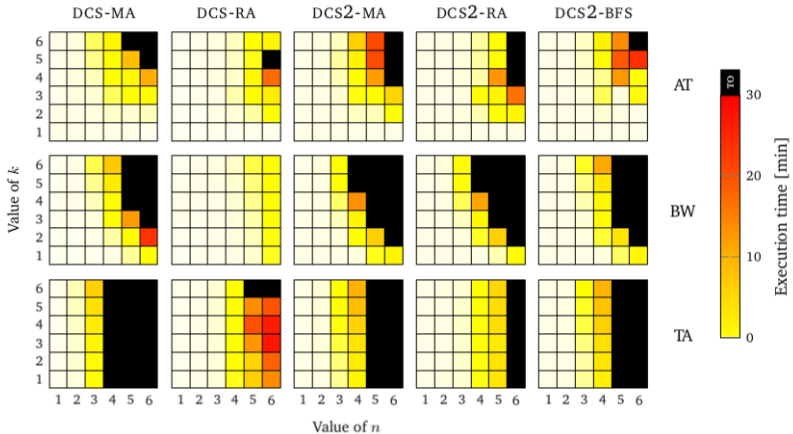
Benchmark

- 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



Air-Traffic Management, Bidding Workflow, Travel Agency

Conclusión

- Definición del problema

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

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

- 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

- 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

- 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.

- 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 pronto
(para ver GR1)

