Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Localización y modelado simultáneos mediante generación y actualización automática de controladores discretos

Tesis presentada para optar al título de
Licenciado en Ciencias de la Computación

Ivan Pasquini
LU: 141/09

Director: Leandro Ezequiel Nahabedian

Buenos Aires, 2016

LOCALIZACIÓN Y MODELADO SIMULTÁNEOS MEDIANTE GENERACIÓN Y ACTUALIZACIÓN AUTOMÁTICA DE CONTROLADORES DISCRETOS

En el área de robótica, el problema de la exploración consiste en recorrer, mediante un vehículo autónomo, una zona desconocida para obtener conocimiento sobre ella. La utilización de robots es muy importante para la cartografía o búsqueda y rescate en lugares que son peligrosos o inaccesibles para las personas. En el caso de la utilización de robots autónomos, se utiliza la técnica de localización y modelado simultáneo, para construir un mapa de la zona desconocida en la que se encuentra el robot, a la vez que estima su trayectoria al desplazarse dentro de la misma.

El objetivo de esta tesis es resolver el problema de localización y modelado simultáneo utilizando como modelo matemático los MTSs (Modal Transition Systems), los cuales son nociones abstractas de los LTSs (Labelled Transition Systems). Dado un conjunto de objetivos y un conjunto de suposiciones de dominio, los MTSs nos permiten determinar, mediante la síntesis de controladores, si los objetivos son imposibles de alcanzar, si podemos garantizar su cumplimiento, o si las suposiciones de dominio son insuficientes para decidir sobre los objetivos.

Para cumplir con el objetivo de la tesis, extenderemos la herramienta MTSA (Modal Transition System Analyser) para dar soporte a la exploración, y presentaremos una estrategia, que bajo ciertas condiciones, nos permita ir agregando información a nuestras suposiciones de dominio, hasta poder decidir si es o no posible garantizar el cumplimiento de los objetivos de la exploración.

Palabras claves: Exploración, Modelado, LTS, MTS, Síntesis de controladores.

AGRADECIMIENTOS

A mi familia, sin su soporte incondicional no me hubiera sido posible llegar a este punto.
A mis amigos, quienes me acompañaron durante todos estos años de carrera.
A mi director de tesis y colaboradores, Leandro Nahabedian, Nicolás D'Ippolito y Natalia Rodriguez, por su guía, consejo y tiempo.
Al jurado, por haberse dedicado a la lectura y corrección de esta tesis.

A mi familia

Índice general

1.. Introducción 1
1.1. Motivación 1
1.2. Resumen de la contribución 1
1.3. Esquema de la tesis 2

2.. Fundamentos teóricos 3
2.1. El mundo y la máquina 3
2.2. Labelled Transition Systems 3
2.3. Modal Transition System 5
2.4. Fluent Linear Temporal Logic 5
2.5. Generalised Reactivity (1) 6
2.6. Síntesis de controladores 7
2.6.1. LTS control problem 7
2.6.2. MTS control problem 8
2.7. Finite State Process 8

3.. Síntesis de controladores en entornos parcialmente conocidos 9
3.1. Problema 10
3.2. Algoritmo 11
3.3. Inputs 12
3.3.1. View 12
3.3.2. Model 12
3.3.3. Goal 12
3.4. Inicialización 12
3.5. Síntesis 13
3.5.1. Detalle 14
3.6. Estrategia 15
3.6.1. Optimista 15
3.6.2. Nueva acción 16
3.6.3. Optimista - Nueva acción 16
3.7. Ejecución 19
3.8. Ciclo de exploración 20

4.. Resultados 22
4.1. Laberinto de 25 posiciones 22
4.1.1. Especificación en MTSA 23
4.1.2. Estrategia Optimista 24
4.1.3. Estrategia Optimista - Nueva acción 26
4.1.4. Variantes 28
4.2. Zonas inseguras 29
4.2.1. Especificación en MTSA 29
4.2.2. Análisis de los modelos 30
4.3. Laberinto que baja 31

4.3.1. Especificación en MTSA 31
4.3.2. Análisis de los modelos 33
4.3.3. Resultado 34
4.4. Laberinto con puente 35
4.4.1. Especificación en MTSA 36
4.4.2. Estrategias 37
4.4.3. Análisis de los modelos 37
4.4.4. Análisis de la traza 37

5.. Conclusiones 39

6.. Apéndice 42

1. INTRODUCCIÓN

1.1. Motivación

En el área de robótica, el problema de la exploración consiste en recorrer, mediante un robot autónomo, una zona desconocida para obtener el conocimiento total de esta, o en caso de no poder reconocerla en su totalidad, maximizar el espacio explorado.

La utilización de robots es muy importante para la cartografía o búsqueda y rescate en lugares que son peligrosos o inaccesibles para las personas. Hay muchos ejemplos de exploración con robots autónomos, como las sondas espaciales no tripuladas que exploran lugares antes de que lleguen los astronautas, o robots que ingresan en estructuras colapsadas para crear una reconstrucción del entorno y reconocer exactamente el lugar en el que se encuentran las víctimas.

Los robots autónomos necesitan un mapa para poder operar en un entorno particular. Por esta razón, se utiliza la técnica de localización y modelado simultáneos, para construir un mapa de una zona desconocida en la que se encuentra el robot, a la vez que estima su trayectoria al desplazarse dentro de la misma.

Existen trabajos previos que logran generar un mapa mediante exploración, trabajando sobre entornos estáticos. Se utilizan diversas técnicas para el aprendizaje sobre el entorno, como por ejemplo redes neuronales [13], grafos [16] o múltiples robots [5]. Algunos trabajos dejan abierta la pregunta sobre como explorar entornos dinámicos [8] [14]. También existen trabajos previos que contemplan cambios en el entorno [17].

En este trabajo estamos interesados en decidir si es posible alcanzar un objetivo (posición) en el mapa evitando lo máximo posible que el robot explore la totalidad del mapa. Dicho problema fue explorado en [11] ampliando el algoritmo tradicional que otorga un camino hacia el objetivo para que soporte incertidumbre del área a recorrer.

1.2. Resumen de la contribución

Las contribuciones de esta tesis pueden verse desde un punto de vista teórico. Durante los últimos años, para resolver problemas de alcanzabilidad de un objetivo en un área, se han usado algoritmos basados en camino mínimo como Dijkstra, A*, entre otros. Sin embargo, este trabajo queda como evidencia de que nuevas estrategias pueden usarse. Decidimos utilizar algoritmos de síntesis de controladores para poder resolver el problema descrito.

Por otro lado, las técnicas existentes de síntesis de controladores nos permiten hallar una estrategia para el robot cuando el área a transitar es totalmente conocida. Para casos donde esta suposición del ambiente es muy fuerte, actualmente no existe forma de obtener un plan adecuado. La forma natural de modelar un área parcialmente conocida es mediante el modelado de un MTS (Modal Transition System), formalismo matemático que introduciremos en la sección 2.3.

Previo a nuestro trabajo, la síntesis de controladores para dichos ambientes genera una respuesta trivaluada: “Sin importar cómo se resuelvan las incertidumbres del ambiente, SIEMPRE obtendremos un plan para llegar al objetivo”, “Sin importar cómo se resuelvan las incertidumbres del ambiente, NUNCA obtendremos un plan para llegar al objetivo”

o “depende de cómo se resuelvan las incertidumbres”. Como contribución, extenderemos esta respuesta para que, en los casos favorables, además de asegurarnos la existencia, nos de el plan para llegar al objetivo.

En resumen, lo que queremos es poder dar una respuesta certera a si es posible garantizar el cumplimiento del objetivo. Para lograrlo necesitamos explorar para adquirir nueva información en caso de que la información que poseemos no sea suficiente para dar una respuesta. Extenderemos la herramienta MTSA (Modal Transition System Analyser) para dar soporte a la exploración y presentaremos una estrategia que nos permita dar una respuesta certera sobre la posibilidad de cumplimiento del objetivo.

1.3. Esquema de la tesis

A continuación, en el capítulo 2, introduciremos la teoría sobre la cual se sustenta nuestra solución al problema. En este capítulo definiremos formalmente la matemática necesarias que posteriormente vamos a utilizar. Luego, en el capítulo 3, introduciremos, formalmente, el problema a resolver. Presentaremos un algoritmo que lo resuelve y una posible estrategia de exploración que garantice encontrar una respuesta a este problema. El capítulo 4 muestra resultados de nuestra técnica. Para ello, mostraremos como se comporta nuestro algoritmo y nuestra estrategia frente a una serie de problemas que servirán para mostrar las características de nuestra implementación. Después, en el capítulo 5, daremos conclusiones de este trabajo. Analizaremos el algoritmo y la estrategia, y plantearemos que trabajos a futuro podrían hacerse a partir de esta tesis. Finalmente, en el capítulo 6 y a modo de apéndice, adjuntamos el manual de usuario para exploración en la herramienta MTSA basado en un ejemplo.

2. FUNDAMENTOS TEÓRICOS

2.1. El mundo y la máquina

El modelo del mundo y la máquina de Michael Jackson [6] es una abstracción que permite formular rigurosamente algunas nociones fundamentales de la ingeniería de requerimientos. En este modelo, los fenómenos son hechos, situaciones o eventos cuya existencia puede observarse, la máquina es una porción del sistema a desarrollar o modificar y el mundo es una porción de la realidad afectada por la máquina. El mundo y la máquina interactúan en la interfaz de la máquina. Un problema del mundo describe una parte del mundo real que nosotros queremos mejorar construyendo la solución como una máquina.

En este modelo, los requerimientos R son declaraciones prescriptivas (propiedades deseadas que pueden cumplirse o no) sobre el mundo expresadas en términos de fenómenos sobre la interfaz entre la máquina que queremos construir y el mundo en el cual viven los problemas que queremos resolver. Los requerimientos deben ser forzados por la máquina independientemente de cómo se comporte el mundo. Los problemas del mundo son capturados como declaraciones prescriptivas expresadas en término de fenómenos del mundo llamados objetivos G, y declaraciones descriptivas (propiedades sobre el sistema que se mantienen independientes a cómo se comporta el sistema) sobre lo que nosotros asumimos que es verdad en el mundo, suposiciones de dominio D. Las suposiciones de dominio podrían no mantenerse y deben ser satisfechas por el mundo.

La tarea clave de la ingeniería de requerimientos es comprender y documentar los objetivos y las características del entorno. Teniendo estos modelos se pueden formular un conjunto de requerimientos para la máquina de forma que en conjunto con el entorno satisfagan los objetivos. Más formalmente R, D $\vDash$ G. Esto puede ser formulado como un problema de síntesis [12]. Dados un conjunto de suposiciones del dominio y un conjunto de objetivos del sistema, construir automáticamente un modelo operacional de la máquina tal que compuesto con el modelo del entorno, los objetivos sean alcanzados.

Diremos que un evento es controlable si es controlable por la máquina. Un evento es observable, o no controlable, si es controlable por el mundo. Para cumplir los objetivos, el modelo operacional restringe la ocurrencia de eventos controlables basándose en la observación de los eventos que ya ocurrieron. Este problema se conoce como el problema de síntesis de controladores y está siendo estudiado exhaustivamente en varios aspectos de la ingeniería de los requerimientos.

2.2. Labelled Transition Systems

Los Labelled Transition Systems [7] (LTSs) son ampliamente utilizados para modelar y analizar el comportamiento de sistemas concurrentes y distribuidos. Son un sistema de transición de estados en el cual las transiciones son etiquetadas con acciones. El conjunto de acciones de un LTS se conoce como alfabeto comunicacional y constituye las interacciones que el sistema modelado puede tener con el entorno.

Definición 2.2.1. (Labelled Transition System) Sea *Estados* el conjunto universal de estados y *Acciones* el conjunto universal de etiquetas de acciones. Un Labelled Transition System es una tupla $E = (S_E, A_E, \Delta_E, s_0^E)$ en donde $S_E \subseteq Estados$ es un conjunto finito

de estados, $A_E \subseteq Acciones$ es un alfabeto finito, $\Delta_E \subseteq (S_E \times A_E \times S_E)$ es una relación de transición y $s_0^E \in S_E$ es el estado inicial.

Dado $(s, l, s') \subseteq \Delta_E$ decimos que l está habilitada desde s en E. Decimos que un LTS es determinístico si desde un estado, al realizar una acción, existe solamente un estado posible al que llegamos.

Sean M y E dos LTSs. La composición en paralelo $\|$ es un operador simétrico que crea un nuevo LTS $M \parallel E$, tal que sus estados son el producto cartesiano de los estados de M y E, sus acciones la unión de las acciones de M y E, y su función de transición sincroniza el movimiento de las dos máquinas en las acciones compartidas, mientras que se mueven independientemente en sus acciones propias.

Definición 2.2.2. (Composición en paralelo) Sean $M = (S_M, A_M, \Delta_M, s_0^M)$ y $N = (S_N, A_N, \Delta_N, s_0^N)$ dos LTSs. Composición en paralelo $\|$ es un operador simétrico tal que $M \parallel N$ es el LTS $P = (S_M \times S_N, A_M \cup A_N, \Delta_P, (s_0^M, s_0^N))$, donde Δ_P es la menor relación que satisface las siguientes reglas:

$$\frac{(s, l, s') \in \Delta_M}{((s,t), l, (s',t)) \in \Delta_P} \text{ con } l \in A_M \setminus A_N$$

$$\frac{(t, l, t') \in \Delta_N}{((s,t), l, (s,t')) \in \Delta_P} \text{ con } l \in A_N \setminus A_M$$

$$\frac{(s, l, s') \in \Delta_M, t, l, t') \in \Delta_N}{((s,t), l, (s',t')) \in \Delta_P} \text{ con } l \in A_M \cap A_N$$

Una traza es una secuencia de acciones ejecutadas en un LTS. Una traza puede ser finita o infinita.

Definición 2.2.3. (Traza) Sea $E = (S_E, A_E, \Delta_E, s_0^E)$ un LTS. Una secuencia $\pi = l_0, l_1, ...$ es una traza en E si existe una secuencia $s_0, l_0, s_1, l_1, ...$ en donde para todo $i \geq 0$ vale $(s_i, l_i, s_{i+1}) \in \Delta_E$.

La bisimulación es una relación de equivalencia entre LTSs.

Definición 2.2.4. (Bisimulación) Sea L el universo de todos los LTSs. Una relación binaria $R \subseteq L \times L$ es una bisimulación si y sólo si cuando $(P, Q) \in R$ entonces para cada acción $a \in A_P \cup A_Q$:

- $(P \xrightarrow{a} P') \Rightarrow (\exists Q' \cdot Q \xrightarrow{a} Q' \wedge (P', Q') \in R)$
- $(Q \xrightarrow{a} Q') \Rightarrow (\exists P' \cdot P \xrightarrow{a} P' \wedge (P', Q') \in R)$

La equivalencia entre LTSs se define mediante la noción de bisimilaridad.

Definición 2.2.5. (Bisimilaridad) Sea L el universo de todos los LTSs. Dos LTSs $P, Q \in L$ son bisimilares si y sólo si existe una bisimulación R tal que $(P, Q) \in R$.

2.3. Modal Transition System

Los Modal Transition System [9] (MTS), son nociones abstractas de los LTSs. Extienden a los LTSs ya que las transiciones que los MTSs poseen, pueden denotar eventos requeridos o bien posibles. Debido a esta particularidad de los MTSs, es que podemos modelar mediante este formalismo información parcial del mundo.

La definición es la misma que para LTSs, pero la función de transición se divide en dos para separar las acciones requeridas de las posibles. Las posibles incluyen a las requeridas.

Definición 2.3.1. (Modal Transition System) Sea *Estados* el conjunto universal de estados y *Acciones* el conjunto universal de etiquetas de acciones. Un Modal Transition System es una tupla $M = (S_M, A_M, \Delta_M^r, \Delta_M^p, s_0^M)$ en donde $S_M \subseteq Estados$ es un conjunto finito de estados, $A_M \subseteq Acciones$ es un alfabeto finito, $\Delta_M^r \subseteq \Delta_M^p \subseteq (S_M \times A_M \times S_M)$ son las relaciones de transición requeridas y posibles respectivamente, y $s_0^M \in S_M$ es el estado inicial.

Hay una relación de refinamiento entre los MTSs y los LTSs. Un LTS se puede ver como un MTS en donde la función de transición de las acciones posibles es igual que la función de transición de las acciones requeridas. Los LTSs que refinan un MTS son descripciones completas del comportamiento del sistema y se llaman implementaciones.

Definición 2.3.2. (Refinamiento de MTS) Sean $M = (S_M, A, \Delta_M^r, \Delta_M^p, s_0^M)$ y $N = (S_N, A, \Delta_N^r, \Delta_N^p, s_0^N)$ dos MTSs. La relación $H \subseteq S_M \times S_N$ es un refinamiento entre M y N si para todo $l \in A$, $(s_M, s_N) \in H$ se cumple:

- Si $(s_M, l, s'_M) \in \Delta_M^r$ entonces existe s'_N tal que $(s_N, l, s'_N) \in \Delta_N^r$ y $(s'_M, s'_N) \in H$.
- Si $(s_N, l, s'_N) \in \Delta_N^p$ entonces existe s'_M tal que $(s_M, l, s'_M) \in \Delta_M^p$ y $(s'_M, s'_N) \in H$.

Decimos que N refina a M si existe una relación de refinamiento H entre M y N tal que $(s_0^M, s_0^N) \in H$, denotada como $M \preceq N$.

Para facilitar el lenguaje, permitimos hablar de la relación inversa entre MTSs llamada implementación.

Definición 2.3.3. (Implementación de MTS) Un LTS L es una implementación de un MTS M si y sólo si $M \preceq L$.

Una implementación es deadlock free si todos sus estados tienen transiciones salientes. Un MTS es determinístico si ninguna de sus acciones conduce desde un estado a más de un estado.

2.4. Fluent Linear Temporal Logic

Linear Temporal Logic (LTL) es una logica ampliamente utilizada para describir requerimientos [15]. La motivación para utilizar LTL sobre fluents es que nos provee de un framework uniforme para especificar propiedades temporales de los estados en modelos basados en eventos [15]. FLTL [4] es una logica lineal temporal para predicar sobre fluents.

Definición 2.4.1. (Fluent) Sea *Acciones* el conjunto universal de etiquetas de acciones. Un fluent es es una tupla $fl = \langle I_{fl}, T_{fl}, Init_{fl} \rangle$ en donde $I_{fl} \subseteq Acciones$ son las acciones que inicializan, $T_{fl} \subseteq Acciones$ las que terminan, tales que $T_{fl} \cap I_{fl} = \emptyset$ y $Init_{fl} \in \{true, false\}$ indica el estado inicial del fluent fl. Notaremos a $\mathcal{F}$ al universo de todos los posibles fluents sobre *Acciones*.

La logica Fluent Linear Temporal Logic [4] (FLTL) nos permite razonar sobre fluents.

Definición 2.4.2. (Fluent Linear Temporal Logic) Sea *Acciones* el conjunto universal de etiquetas de acciones y Π el conjunto de posibles trazas infinitas sobre *Acciones*. Una fórmula FLTL se define inductivamente usando los conectores Booleanos standard y los operadores temporales **X** (Next) y **U** (strong until) de la siguiente forma:

$$\varphi ::= fl \mid \neg\varphi \mid \varphi \vee \psi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \psi$$

donde $fl \in \mathcal{F}$. Dada una traza, por cada posición en la traza podemos decidir si el fluent está activo o no.

Definición 2.4.3. (Satisfacción) Sea *Acciones* el conjunto universal de etiquetas de acciones, *Fluents* el conjunto de todos los posibles $\mathcal{F}$ sobre *Acciones*, y Π el conjunto de posibles trazas infinitas sobre *Acciones*. La traza $\pi \in \Pi$ satisface el fluent $Fl \in Fluents$ en la posición i si y sólo si vale alguna de las siguientes condiciones:

- $Init_{Fl} \wedge (\forall j \in \mathbb{N} \cdot 0 \leq j \leq i \rightarrow l_j \notin T_{Fl})$
- $\exists j \in \mathbb{N} \cdot (j \leq i \wedge l_j \in I_{Fl}) \wedge (\forall k \in \mathbb{N} \cdot j \leq k \leq i \rightarrow l_k \notin T_{Fl})$

Dada una traza infinita π, la satisfacción de una fórmula φ en la posición i, denotada $\pi, i \vDash \varphi$ se define segun la Figura 2.1.

Decimos que φ se sostiene en π si $\pi, 0 \vDash \varphi$. Una fórmula $\varphi \in FLTL$ se sostiene en un LTS E si se sostiene en cada traza infinita producida por E.

2.5. Generalised Reactivity (1)

Dada una secuencia infinita de estados, una fórmula GR(1) [1] denota cuales han ocurrido infinitamente. Más formalmente, $\phi = (A, B)$ en donde A y B son conjuntos de subconjuntos de estados en donde o bien alguno de los subconjuntos de A no contiene ningún estado que se visita infinitamente, o bien todos los subconjuntos de B contienen al menos un estado que se visita infinitamente.

$$\begin{array}{lcl}
\pi, i \models_d f & \triangleq & \pi, i \models_d f \\
\pi, i \models_d \neg\varphi & \triangleq & \neg(\pi, i \models_d \varphi) \\
\pi, i \models_d \varphi \vee \psi & \triangleq & (\pi, i \models_d \varphi) \vee (\pi, i \models_d \psi) \\
\pi, i \models_d \mathbf{X}\varphi & \triangleq & \pi, i+1 \models_d \varphi \\
\pi, i \models_d \varphi \mathbf{U} \psi & \triangleq & \exists j \geq i \cdot \pi, j \models_d \psi \quad \wedge \\
 & & \qquad \forall\, i \leq k < j \cdot \pi, k \models_d \varphi
\end{array}$$

Fig. 2.1: Semantics for the satisfaction operator.

Definición 2.5.1. (Generalised Reactivity (1)) Sea *Estados* el conjunto universal de estados. Dada una secuancia infinita de estados p, sean $inf(p)$ los estados que ocurren infinitamente en p. Sea $\phi_1, ..., \phi_n$ y $\Upsilon_1, ..., \Upsilon_m$ subconjuntos de *Estados*. Sea $gr((\phi_1, ..., \phi_n), (\Upsilon_1, ..., \Upsilon_m))$ el conjunto de infinitas secuencias p tales que o bien para algun i vale $inf(p) \cap \phi_i = \varnothing$ o para todo j vale $inf(p) \cap \Upsilon_j \neq \varnothing$.

2.6. Síntesis de controladores

Dados dos LTSs N y M, y $A_{N_u} \subseteq A_N$, decimos que M es un LTS Legal para N respecto a A_{N_u} si por cada estado en $E \parallel M$, realizar una acción perteneciente a A_{N_u} es igual a realizar la acción en N. Intuitivamente, en la composición una acción es deshabilitada si y sólo si también es deshabilitada en el LTS original. En otras palabras, M no restringe a N respecto a A_{N_u}.

Definición 2.6.1. (LTS Legal) Sean $M = (S_M, A_M, \Delta_M, s_0^M)$ y $N = (S_N, A_N, \Delta_N, s_0^N)$ dos LTSs y sea $A_{N_u} \in A_N$ un estado. Decimos que M es un LTS legal para N respecto a A_{N_u}, si para todo $(s_n, s_m) \in E \parallel M$ pasa que $\Delta_{E \parallel M}((s_n, s_m)) \cap A_{N_u} = \Delta_N(s_N) \cap A_{N_u}$.

La síntesis de controladores se encarga de producir automáticamente una máquina que restringe la ocurrencia de eventos controlables basándose en la observación de los eventos que ya ocurrieron. Cuando esta máquina es desplegada en un entorno adecuado garantiza la satisfacción de un conjunto de objetivos dado. La satisfacción de estos objetivos depende de la satisfacción de las asunciones sobre el entorno.

2.6.1. LTS control problem

Podemos describir la síntesis de controladores de la siguiente forma. Dados un LTS que describe el comportamiento del entorno, un conjunto de acciones controlables, un conjunto de fórmulas FLTL que representan las suposiciones del dominio y un conjunto de fórmulas FLTL que representan los objetivos del sistema, el problema de control de LTS [3] es encontrar un LTS que sólo restrinja la ocurrencia de acciones controlables y garantice que la composición en paralelo entre el entorno y el LTS es deadlock free y que si las suposiciones de dominio se satisfacen, entonces los objetivos del sistema también son satisfechos. Intuitivamente, al resolver un problema de síntesis, buscamos evitar alcanzar estados para los cuales no se puedan satisfacer los objetivos. Estos estados no deseables los vamos a llamar estados perdedores. Las zonas inseguras son conjuntos de estados perdedores.

Definición 2.6.2. (LTS control) Dado un modelo del domino en forma de un LTS determinístico $E = (S_E, A_E, \Delta_E, s_0^E)$, un conjunto de acciones controlables $A_c \subseteq A_E$ y una fórmula FLTL φ, la solución al LTS control problem $\varepsilon = \langle E, \varphi, A_c \rangle$ es un LTS $M = (S_M, A_M, \Delta_M, s_0^M)$ tal que $A_M = A_E$, y por cada estado en S_M todas las acciones en $A_M \setminus A_c$ estás habilitadas, $E \parallel M$ es deadlock free, y toda traza π en $E \parallel M$ garantiza $\pi \vDash \varphi$

El LTS Control Problem es decidible en tiempo exponencial y en caso de ser decidible, se encuentra la solución con la misma complejidad. Para esto es necesario que el modelo sea determinístico.

2.6.2. MTS control problem

El problema de la síntesis de controladores para MTSs [2] consiste en ver si todas, alguna o ninguna de sus implementaciones pueden ser controladas por un controlador LTS. Al responder esta pregunta, también estamos respondiendo la pregunta de si el problema es realizable, ya que si la respuesta es ninguno el problema no es realizable. Para decidir este problema, primero se intenta encontrar un controlador para la implementación del MTS que tiene todas las acciones posibles controlables. Si se lo encuentra, se tiene un controlador para todas las posibles implementaciones. Se hace lo mismo con el que tiene la mínima cantidad de acciones controlables y así se obtiene la respuesta.

Definición 2.6.3. (MTS control) Dado un modelo del domino en forma de un MTS $M = (S_M, A, \Delta_M^r, \Delta_M^p, s_0^M)$, un conjunto de acciones controlables $A_c \subseteq A_M$ y una fórmula FLTL φ, la solución al MTS control problem $\varepsilon = \langle M, \varphi, A_c \rangle$ es responder:

- **All** si para todo LTS I que sea implementación de M, el LTS control problem $\langle I, \varphi, A_c \rangle$ es realizable.
- **None** si para ningún LTS I que sea implementación de M, el LTS control problem $\langle I, \varphi, A_c \rangle$ es realizable.
- **Some** en otro caso.

2.7. Finite State Process

El Finite State Process (FSP) [10] es un lenguaje de especificación semántica bien definida en términos de LTSs que nos permite describirlos en forma concisa. El lenguaje que utilizaremos es una extensión de FSP que puede describir MTSs.

3. SÍNTESIS DE CONTROLADORES EN ENTORNOS PARCIALMENTE CONOCIDOS

Nuestro problema consiste en dado una posición inicial del sistema y un punto físico objetivo dentro de un área/habitación, es requerido trasladarnos al punto deseado. La particularidad del problema a resolver es la falta de información que tenemos sobre el área donde transitar. Diremos entonces que buscaremos soluciones al problema de navegación de robots cuando el entorno es parcialmente conocido. Tenemos un objetivo, pero no sabemos lo suficiente sobre el entorno como para decidir si es posible satisfacerlo. Debemos obtener el conocimiento necesario sobre el entorno para poder decidir sobre el objetivo. Por otro lado, el entorno puede ser tanto estático como dinámico. Será dinámico en caso de que mientras el robot intenta satisfacer su objetivo, existan componentes externos cuyas acciones modifican la configuración inicial del área a explorar. Para lograrlo, vamos a utilizar los sensores del robot, los cuales nos aportan información, y una estrategia que utilice el conocimiento adquirido para obtener conocimiento nuevo.

Cuando comenzamos la exploración, la información que tenemos sobre el mundo es parcial. Los MTSs nos permiten codificar la incertidumbre, ya que podemos distinguir la información que poseemos como transiciones requeridas y la información que podemos adquirir mediante exploración como transiciones posibles. El mundo real, el cual estamos explorando, puede representarse como un LTS (Labelled Transition System) que refina nuestro MTS inicial.

Además del LTS que representa al mundo real, otros LTSs pueden refinar al MTS que representa nuestro conocimiento. Estos LTSs representan a todas las posibles formas que puede llegar a tener el mundo real a partir del conocimiento que poseemos actualmente. En cada uno de estos LTSs, las acciones posibles o bien van a ser acciones requeridas, o bien no van a formar parte del LTS, y en caso de que sean acciones requeridas, puede variar la cantidad de veces que puede ejecutarse dicha acción. Por estas razones, la cantidad de LTSs que refinan a nuestro MTS puede llegar a ser infinita. Cada uno de estos LTSs está decidiendo sobre la incertidumbre del MTS que representa nuestro conocimiento actual.

Cuando exploramos, lo hacemos para poder realizar un objetivo. Mediante la síntesis de controladores, podemos intentar generar automáticamente una máquina, tal que al componerla en paralelo con el MTS que representa nuestro conocimiento del mundo, garantice el cumplimiento del objetivo.

Si para cada uno de los LTSs que refinan a nuestro MTS, el objetivo es alcanzable, sabemos que podemos garantizar el cumplimiento del objetivo, ya que el mundo puede representarse como un LTS que refina nuestro modelo. Cuando no es posible sintetizar un controlador que pueda garantizar que los objetivos se alcancen para ninguno de los LTSs que refinan a nuestro modelo, podemos afirmar que el objetivo no es realizable en el mundo real. Pero si para algunos de estos LTSs existe un controlador, y para otros no, no podemos afirmar nada sobre nuestro objetivo.

3.1. Problema

Una posible solución al problema de la síntesis de controladores en entornos parcialmente conocidos es un LTS x (ver sección 2.2) tal que $M \parallel x \models G$, en donde M es el MTS (ver sección 2.3) que modela el entorno y G es un fórmula en FLTL (ver sección 2.4) que describe cual es el sector objetivo.

El estado del arte en cuanto a síntesis de controladores sobre Modal Transition Systems, nos otorga una respuesta trivaluada como mostramos previamente en la sección 2.6.2. Sin embargo, actualmente no existe forma de obtener un LTS solución (x). Si bien una de nuestras contribuciones es generar automáticamente el LTS x, la complejidad de producir dicho output depende de la configuración inicial de nuestros inputs. Por lo tanto podemos detallar tres casos al momento inicial de la ejecución:

- *Es posible sintetizar el LTS x para cualquier implementación de M.* En otras palabras, sabemos que para toda incertidumbre que se encuentre en el área, sin importar qué encontremos al explorar esa zona, tenemos una estrategia que nos lleva al punto objetivo.
- *No es posible sintetizar el LTS x para ninguna implementación de M.* Esto es que sin importar como se resuelvan las incertidumbres, el sistema nunca va a poder alcanzar el objetivo.
- *Algunas implementaciones de M ofrecen buenas garantías para satisfacer el objetivo pero otras no.* Por lo tanto, no sabemos si es posible sintetizar un controlador para la implementación de M que representa al mundo real, el cual no conocemos.

En los dos primeros escenarios tenemos una respuesta a nuestro problema. De estar en el tercer escenario, queremos poder sintetizar el controlador en caso de ser posible, o tener garantías de que es imposible sintetizar dicho controlador. Para lograr nuestro objetivo debemos agregar información a M, para poder reducir su cantidad de posibles implementaciones, y que, de esta forma, su síntesis caiga en uno de los dos primeros escenarios.

La exploración nos permite adquirir conocimiento, ya que al observar mediante los sensores del robot lugares del entorno en los cuales no habíamos estado, podemos despejar la incertidumbre en nuestro modelo del entorno, representado por M. Vamos a realizar la exploración mediante un algoritmo, el cual se basa en ciclos de adquisición de conocimiento. Si nuestra información sobre el entorno es insuficiente, movemos al robot para que pueda aprender algo nuevo. Repetimos el ciclo hasta poder dar una respuesta.

Más formalmente podemos definir el problema síntesis de controladores en entornos parcialmente conocidos de la siguiente manera:

Definición 3.1.1. (Síntesis de controladores en entornos parcialmente conocidos) Sea G una fórmula en FLTL que representa los requerimientos del sistema y M es un MTS que representa las suposiciones del dominio conocidas en el momento inicial. Resolver el problema de síntesis de controladores en entornos parcialmente conocidos es decidir si existe un LTS x tal que $M' \| x \models G$ donde $M' \prec M$ que representa el conocimiento inicial del entorno más todo el conocimiento adquirido por la exploración.

3.2. Algoritmo

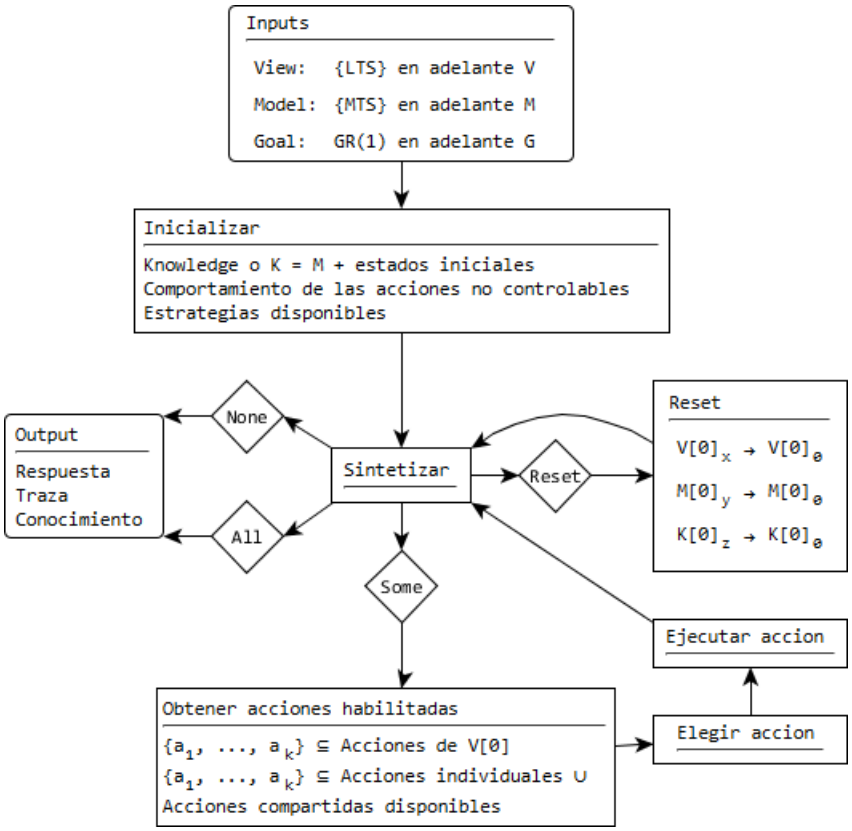


Fig. 3.1: Algoritmo de exploración.

Algorithm 1 Algoritmo de exploración

Require: {LTS} View, {MTS} Model, GR(1) Goal
Ensure: All si se puede satisfacer Goal en el Knowledge actual y None en caso contrario
  Knowledge := Model ∪ estados iniciales
  Respuesta := Sintetizar(Knowledge, Goal)
  while Respuesta == Some || Respuesta == Reset **do**
    if Respuesta == Some **then**
      Acciones habilitadas := {a_1, ..., a_k} tal que {a_1, ..., a_k} ⊆ Acciones individuales(Knowledge[0]) ∪ Acciones compartidas disponibles(Knowledge[0])
      Acción := Elegir(Knowledge, Acciones habilitadas)
      Ejecutar(View, Model, Knowledge, Acciones controlables, Acción)
    else
      Reiniciar estados iniciales(View, Model, Knowledge)
    end if
    Respuesta := Sintetizar(Knowledge, Goal)
  end while
  return Respuesta

3.3. Inputs

Para resolver el problema necesitamos poder observar al entorno a medida que nos movemos, saber con que información inicial contamos y tener un objetivo que cumplir.

3.3.1. View

View es una lista de LTSs. Representa el mundo sobre el cual se mueve el robot. El primer LTS de la lista representa el mapa del entorno, el cual regula como se mueve el robot por el mundo. Indica que acciones puede realizar en cada posición, y a que lugar lo llevan dichas acciones. Los otros LTSs representan el comportamiento de los agentes externos que interactúan con el entorno. Pueden bloquear y habilitar acciones en una determinada posición. El robot no tiene influencia sobre ellos. El robot solamente puede observar el estado actual del **View** mediante sus sensores. Sobre el mapa, solamente puede saber en qué posición está y que acciones puede ejecutar en dicha posición. Sobre los agentes externos, solamente puede observar su estado actual.

3.3.2. Model

Model es una lista de MTSs. Representa nuestro conocimiento inicial sobre el mundo. Por cada LTS en **View** hay un correspondiente MTS en **Model**. La única restricción para los MTSs de **Model**, es que puedan refinarse en sus correspondientes LTSs de **View**. Por lo tanto, como mínimo, cada MTS debe contar con un estado, y por cada acción en su correspondiente LTS, debe haber una acción posible en el MTS.

3.3.3. Goal

Goal es el objetivo del robot. Exploramos para poder decidir si es posible garantizar el cumplimiento de **Goal**. Está expresado con una fórmula $\mathrm{GR}(1)$.

3.4. Inicialización

Knowledge es una lista de MTSs que representa el conocimiento que vamos adquiriendo en cada iteración del algoritmo. En cada iteración, los MTSs de **Knowledge** son un refinamiento de los MTSs de la iteración anterior. Cada MTS de **Knowledge** se compone de dos grupos de estados: Los que están en *La nube* y los que representan el conocimiento.

La nube representa la incertidumbre. Es una copia del MTS de **Model**. Sus estados y acciones se mantienen inmutables en todas las iteraciones de **Knowledge**. El resto de los estados representan el conocimiento adquirido hasta el momento. Su cantidad de estados irá creciendo a medida que el robot adquiera información. En cada estado, las acciones que nunca fueron ejecutadas irán a un estado de *La nube*, mientras que las que ya fueron ejecutadas, irán a un estado del conocimiento.

En la implementación del algoritmo en la herramienta MTSA, hay dos cuestiones importantes de inicialización. Por un lado, el comportamiento de las acciones no controlables de los agentes externos. Actualmente existen dos patrones de comportamiento. Por defecto, los agentes externos elegirán una acción al azar. El otro patrón de comportamiento consiste en que se comporten de forma cíclica ejecutando las acciones que nosotros pasamos como input. En el futuro podrá extenderse la herramienta con más patrones de comportamiento.

La otra cuestión es la estrategia utilizada por el robot. En esta tesis presentamos una única estrategia que resuelve el problema, pero en el futuro puede extenderse la herramienta con estrategias diferentes.

3.5. Síntesis

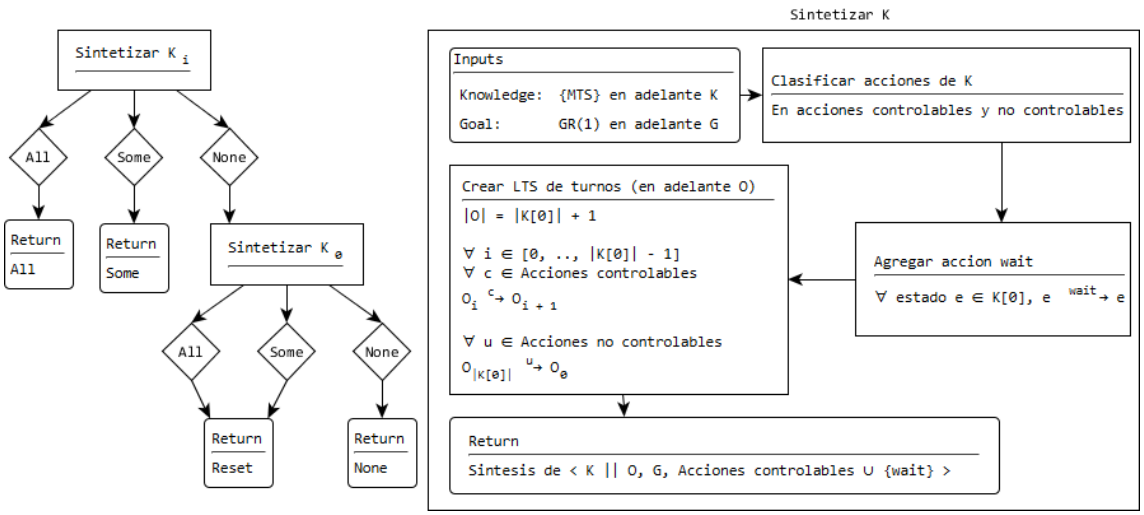


Fig. 3.2: Algoritmo de síntesis.

Algorithm 2 Algoritmo general de síntesis

Require: {MTS} Knowledge, GR(1) Goal

Ensure: Devuelve la respuesta al problema de control sobre Knowledge y Goal, o Reset si la respuesta es Some al volver la posición inicial de Knowledge al inicio

```
Knowledge en posición actual := Cambiar estado inicial a actual(Knowledge)
Respuesta := Problema de control(Knowledge en posición actual, Goal)
if Respuesta == None then
    Respuesta := Problema de control(Knowledge, Goal)
    if Respuesta == None then
        return None
    else
        return Reset
    end if
else
    return Respuesta
end if
```

Algorithm 3 Algoritmo del problema de control sobre MTSs

Require: {MTS} Knowledge, GR(1) Goal
Ensure: La respuesta decide la realizabilidad de Goal sobre las implementaciónes de Knowledge
  Acciones controlables := Obtener acciones controlables(Knowledge)
  Acciones no controlables := Obtener acciones no controlables(Knowledge)
  for Estado ∈ Estados(Knowledge[0]) **do**
    Agregar acción(Knowledge[0], WAIT, Estado, Estado)
  end for
  LTS de turnos := Crear LTS de turnos(Knowledge, Acciones controlables, Acciones no controlables)
  Respuesta := Síntesis(Knowledge || LTS de turnos, Goal, Acciones controlables ∪ {wait})
  return Respuesta

3.5.1. Detalle

Al principio de cada iteración, lo primero que hacemos es intentar dar una respuesta a la pregunta sobre si podemos garantizar el cumplimiento del objetivo. En otras palabras, queremos ver si podemos sintetizar un controlador que garantice el cumplimiento de **Goal**.

La síntesis la haremos sobre un MTS basado en **Knowledge**, pero con algunas modificaciones necesarias para modelar correctamente el problema. Primero cambiamos el estado inicial del primer MTS de **Knowledge**, el que representa el mapa, para que su estado inicial sea el estado en el que está situado actualmente el robot. Esto es para que el controlador garantice el objetivo desde nuestra posición actual.

Luego, al primer MTS de **Knowledge** (el mapa), le agregaremos en cada estado la acción controlable wait, que irá al mismo estado. Esto es para representar la posibilidad de que el robot espere en su posición actual un cambio en los agentes externos.

Por último, creamos un LTS de turnos que permita al robot moverse a cualquier estado antes de que los agentes externos realicen cambios. Esto es para evitar que los agentes externos esperen que el robot esté lejos para realizar una acción que beneficie al robot. El LTS de turnos permitirá que el robot se mueva una cantidad de veces igual a la cantidad de estados del primer MTS del **Knowledge** (el mapa) antes de que los agentes externos se muevan.

De esta forma, con la posibilidad de moverse y esperar en cualquier estado conocido, puede estar en el estado necesario para beneficiarse de las acciones de los agentes externos. Sintetizaremos el controlador para **Goal** sobre la composición en paralelo de los MTSs de **Knowledge**, incluyendo el MTS al cual le agregamos las acciones wait, con el LTS de turnos.

Si el resultado de la síntesis es que puede generarse un controlador para cualquier LTS que sea un refinamiento del MTS que armamos, significa que el robot puede cumplir el objetivo sin importar como sean las zonas inexploradas del entorno. El algoritmo termina, dando como resultado la garantía del cumplimiento de **Goal**.

En caso de que no pueda generarse un controlador para ningún LTS que sea un refinamiento del MTS que armamos, hay que volver a realizar la síntesis, pero sin realizar el cambio del estado inicial. Si nuevamente no puede generarse un controlador, significa que es imposible cumplir **Goal**, sin importar como sean las zonas inexploradas del entorno,

y el algoritmo termina confirmando la imposibilidad de cumplimiento de **Goal**. En caso contrario, significa que llegamos a un punto sin retorno, pero todavía hay zonas inexploradas que pueden ser alcanzadas desde el inicio, en las cuales podríamos o no cumplir el objetivo.

En caso de que nuestro robot tenga la capacidad de volver al inicio, o utilizar otro robot desde la zona inicial, podemos seguir explorando cambiando los estados iniciales de los primeros componentes de **View**, **Model** y **Knowledge** (los que representan al mapa) al estado inicial del comienzo de la exploración. Al hacer esto contamos con la información recolectada hasta el momento, la cual puede utilizar la estrategia para no volver a caer en el punto sin retorno.

Por último, si la síntesis puede generar un controlador para algunas de las implementaciones del MTS que armamos, pero para otras no, significa que no tenemos suficiente información sobre el entorno como para decidir sobre el objetivo, y necesitamos seguir explorando para refinar el **Knowledge**.

3.6. Estrategia

En caso de que sea necesario seguir explorando, necesitamos que la estrategia decida la próxima acción a ejecutar. El algoritmo está preparado para que, al utilizar múltiples estrategias, detecte si alguna entra en un ciclo infinito y la reemplace por otra.

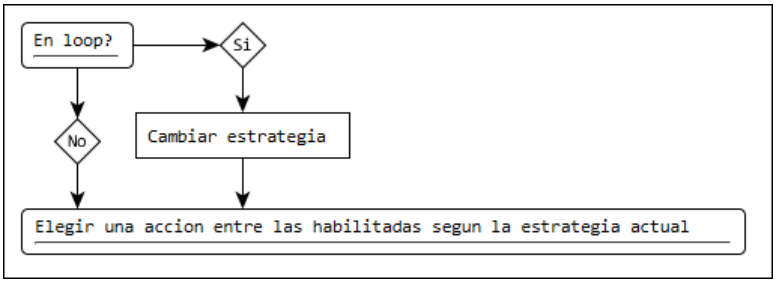


Fig. 3.3: Elección de estrategia

Algorithm 4 Algoritmo de elección de estrategia

Require: {MTS} Knowledge, Accion Acciones habilitadas
Ensure: Cambia la estrategia actual si entro en un ciclo
  if en un ciclo() **then**
    Cambiar estrategia()
  end if
  Elegir(Knowledge, Acciones habilitadas)

3.6.1. Optimista

Estamos por elegir una acción. Hacemos esto porque el intento de síntesis para generar un controlador que garantice **Goal** nos dio como resultado que existe un controlador para algunos refinamientos pero no para otros. En particular existe un controlador para el refinamiento más optimista, llamado controlador optimista. La estrategia Optimista

aprovecha este hecho, ya que por construcción, el controlador optimista evitará las zonas inseguras. Lo que hace la estrategia es elegir de entre las acciones posibles, una acción que esté en el estado inicial del controlador optimista. En cambio, la estrategia Optimista - Nueva acción lo utiliza para filtrar de entre las acciones posibles, cuáles son las acciones seguras, y así elegir mediante la estrategia Nueva acción una acción entre las seguras.

3.6.2. Nueva acción

La estrategia Nueva acción tiene como objetivo ejecutar acciones nuevas, o en otras palabras, que no hayan sido ejecutadas anteriormente en su estado asociado. Si siempre ejecutamos acciones nuevas, siempre adquiriremos conocimiento nuevo, y como el entorno a explorar es finito, terminaremos por reconocerlo completamente de seguir esta estrategia.

La estrategia se divide en varias etapas. Lo primero que debemos hacer es saber sí, entre las acciones disponibles, existe alguna que no haya sido ejecutada anteriormente desde el estado actual. Si existe, la elegiremos, porque es una acción nueva.

Si el estado actual es un estado completamente explorado, o en otras palabras, ya ejecutamos anteriormente todas las acciones disponibles desde este estado, debemos llegar a un estado no completamente explorado. Para hacerlo, buscamos si hay algún estado que cumpla las condiciones deseadas al que podamos volver. Esto significa que exista un estado con alguna acción controlable como posible, y que podamos llegar a él por un camino de acciones requeridas en el primer componente de **Knowledge** (el mapa).

Para construir el camino, sintetizaremos un controlador con el objetivo de llegar a ese estado desde nuestro estado actual. Para lograrlo construiremos un MTS a partir del primer componente de **Knowledge**. Debemos agregar una acción ganadora al estado al que queremos llegar. El objetivo será ejecutar dicha acción, por lo cual para lograrlo tenemos que llegar al estado deseado desde nuestro estado actual. El segundo paso es eliminar las acciones posibles, para que el camino solo esté compuesto por acciones requeridas, o en otras palabras, ya ejecutadas anteriormente. En caso de existir el controlador buscado, necesitamos que el camino comience por alguna de las acciones disponibles desde nuestro estado actual. En este caso, la elegimos, porque nos acerca a una acción nueva.

Si no existen estados con acciones controlables no ejecutadas anteriormente, lo que tenemos que hacer es llegar a un estado con una acción compartida no ejecutada anteriormente, y esperar a que los agentes externos nos permitan ejecutarla. Seguimos el mismo procedimiento de síntesis utilizado anteriormente. Si existe el camino, elegimos la acción que nos permita acercarnos, en caso contrario eligiéremos esperar, ya que es necesaria la interacción de algún agente externo para poder continuar explorando.

3.6.3. Optimista - Nueva acción

En esta tesis vamos a presentar únicamente la estrategia Optimista - Nueva acción, la cual no entra en ciclos. Dicha estrategia es la composición de la estrategia Optimista, la cual busca evitar caer en zonas inseguras utilizando la información proporcionada por **Model**, y la estrategia Nueva acción, la cual busca siempre adquirir conocimiento nuevo utilizando información proporcionada por **Knowledge**.

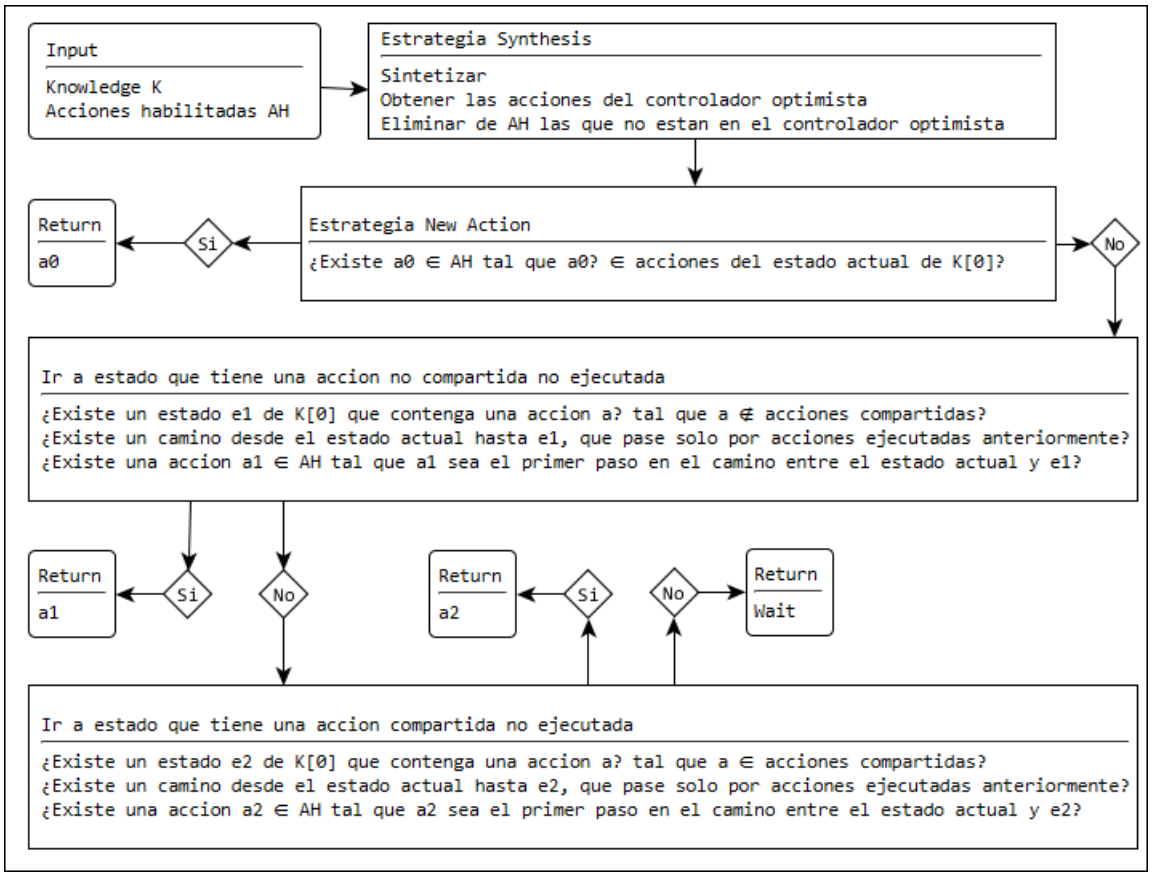


Fig. 3.4: Algoritmo de la estrategia Optimista - Nueva acción.

Algorithm 5 Algoritmo de la estrategia Optimista - Nueva acción

Require: MTS Knowledge, Accion Acciones habilitadas
Ensure: Una accion que pertenece a Acciones habilitadas o Wait

```
Controlador optimista := Sintetizar(Knowledge)
Acciones del controlador optimista := obtener acciones habilitadas(Controlador optimista)
Acciones habilitadas := Acciones habilitadas ∩ Acciones del controlador optimista
Acciones habilitadas no ejecutadas := Acciones habilitadas ∩ acciones posibles(estado actual(Knowledge))
if Acciones habilitadas no ejecutadas != ∅ then
  return Acciones habilitadas no ejecutadas[0]
else
  for Estado ∈ Estados(Knowledge) do
    if Estado == Estado inicial then
      continue
    end if
    Acciones posibles no compartidas := acciones posibles no compartidas(Estado)
    if Acciones posibles no compartidas != ∅ then
      Controlador al estado := sintetizar para encontrar camino(Knowledge, Estado)
      Acciones del controlador al estado := obtener acciones habilitadas(Controlador al estado)
      Acciones habilitadas que llevan al estado := Acciones habilitadas ∩ Acciones del controlador al estado
      if Acciones habilitadas que llevan al estado != ∅ then
        return Acciones habilitadas que llevan al estado[0]
      end if
    end if
  end for
  for Estado ∈ Estados(Knowledge) do
    if Estado == Estado inicial then
      continue
    end if
    Acciones posibles compartidas := acciones posibles compartidas(Estado)
    if Acciones posibles compartidas != ∅ then
      Controlador al estado := sintetizar para encontrar camino(Knowledge, Estado)
      Acciones del controlador al estado := obtener acciones habilitadas(Controlador al estado)
      Acciones habilitadas que llevan al estado := Acciones habilitadas ∩ Acciones del controlador al estado
      if Acciones habilitadas que llevan al estado != ∅ then
        return Acciones habilitadas que llevan al estado[0]
      end if
    end if
  end for
  return Wait
end if
```

3.7. Ejecución

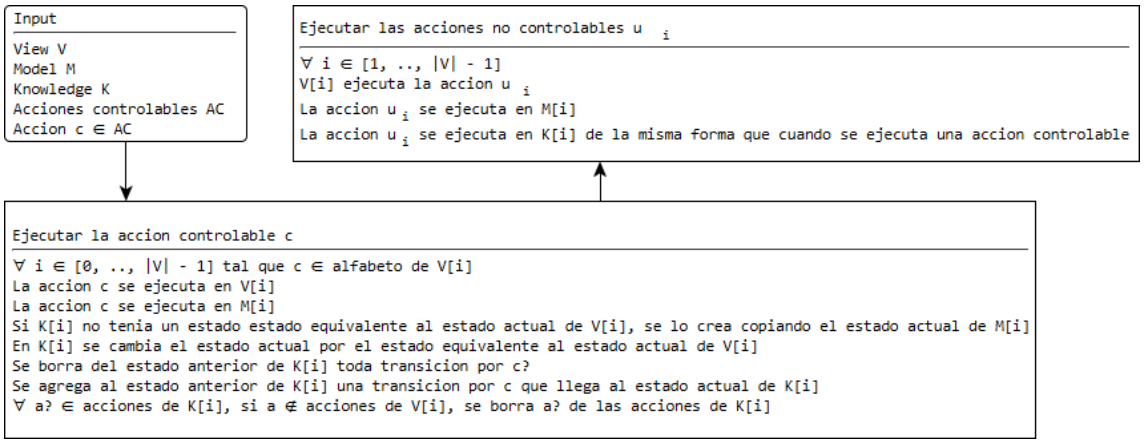


Fig. 3.5: Algoritmo de ejecución.

Algorithm 6 Algoritmo de ejecución

Require: {LTS} View, {MTS} Model, {MTS} Knowledge, {Accion} Acciones controlables, Accion Acción a ejecutar
  for i := 0; i < |View|; i++ **do**
    if Acción a ejecutar ∉ Acciones(View[i]) **then**
      continue
    end if
    ejecutar(Accion acción a ejecutar, View[i])
    ejecutar(Accion acción a ejecutar, Model[i])
    Índice del estado actual := Índice del estado actual(View[i])
    if Knowledge[i][Índice del estado actual] == null **then**
      Knowledge[i][Índice del estado actual] := Model[i][Índice del estado actual]
    end if
    Actualizar estado actual(Knowledge[i], Índice del estado actual)
    Eliminar transicion posible(Knowledge[i], Acción a ejecutar)
    Agregar transicion(Knowledge[i], Acción a ejecutar)
    for Accion posible in Acciones posibles(Knowledge[i]) **do**
      if Accion posible ∉ Acciones posibles(View[i]) **then**
        Eliminar accion(Knowledge[i], Accion posible)
      end if
    end for
  end for
  Ejecutar acciones no controlables(View, Model, Knowledge)

Al ejecutar la acción elegida, el robot puede, o no, cambiar de ubicación en el entorno. Esto implica un cambio en el estado actual del primer componente tanto de **View**, como de **Model** y **Knowledge**. El primer componente de **Knowledge** también puede cambiar su estructura, refinando el MTS, como consecuencia de la nueva información aportada por los sensores del robot en su nueva ubicación.

Si es la primera vez que el robot se encuentra en la ubicación actual, se va a agregar un nuevo estado en el primer componente de **Knowledge**, en el que todas sus acciones se dirigen hacia *La nube*. En caso de haber ejecutado una acción que se dirigía hacia *La nube*, ahora podemos observar a que ubicación en el entorno nos lleva, por lo cual vamos a poder transformar en nuestro modelo dicha acción en una acción requerida que va hacia el estado correspondiente a la ubicación actual.

Luego de ejecutar la acción elegida, hay que ejecutar las acciones ejecutadas por los agentes externos. Los cambios en los agentes externos afectan el estado actual de las restantes componentes de **View**, **Model** y **Knowledge**, y la estructura de las restantes componentes de **Knowledge**. El proceso se realiza en la forma descripta por la figura 3.5.

3.8. Ciclo de exploración

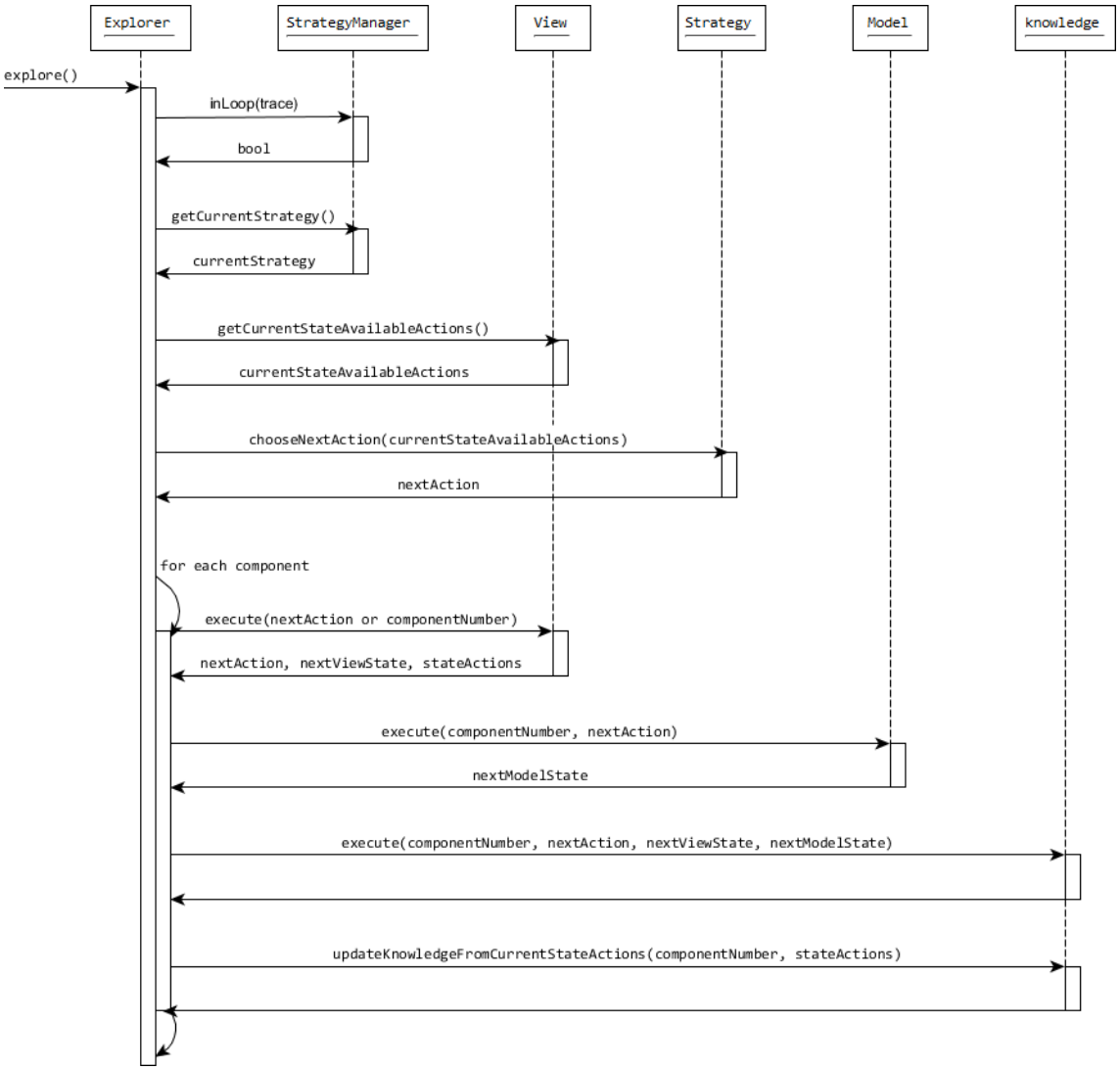


Fig. 3.6: Diagrama de secuencia.

La figura 3.6 nos muestra cómo interactúan los diferentes objetos durante un ciclo de exploración.

Lo primero que hacemos es comprobar que la estrategia actual no haya entrado en un ciclo, ya que de no visitar estados nuevos no puede aportar información nueva.

En caso de que la estrategia actual esté en un ciclo la reemplazamos por otra. Una vez que tenemos la estrategia adecuada, observamos mediante los sensores que acciones tenemos disponibles en la ubicación actual, con el estado actual de los agentes externos. Le pedimos a la estrategia que elija una acción entre las acciones disponibles.

Cuando tenemos definida la siguiente acción, necesitamos ejecutarla. Primero la ejecutamos en **View**. Al hacerlo podemos censar en qué ubicación nos encontramos y qué acciones tenemos disponibles.

Luego la ejecutamos en **Model**, para que su estado actual se corresponda al estado que modela al estado actual de **View**.

A continuación, ejecutamos la acción en **Knowledge**, y en caso de que la acción vaya a un nuevo estado, utilizamos como nuevo estado el estado actual de **Model** (En este nuevo estado todas las acciones se dirigen hacia *La nube*).

Por último, tenemos que agregar a **Knowledge** la nueva información obtenida, eliminando de su estado actual las acciones que no están disponibles, y transformando la acción que acabamos de ejecutar en caso de que originalmente estuviera dirigida hacia *La nube*.

Luego de ejecutar la acción elegida, hay que ejecutar las acciones elegidas por los agentes externos. Cada componente extra en nuestros modelos representa a un agente externo, por lo tanto, hay que realizarlo una vez por componente. El proceso es el mismo que al ejecutar la acción elegida por nosotros, la única diferencia es que nosotros no elegimos la acción a ejecutar. Al hacer esto reflejamos el cambio en los agentes externos durante de tiempo que nos toma ejecutar nuestra acción.

4. RESULTADOS

A continuación, vamos a plantear algunos casos de estudio para analizar en detalle el comportamiento del algoritmo.

4.1. Laberinto de 25 posiciones

El primer entorno que presentamos es un laberinto de 25 posiciones. En cada posición el robot puede tener hasta 4 acciones disponibles, las cuales son norte, sur, este y oeste. El entorno no tiene agentes externos. El objetivo de este caso de estudio es ver como el robot encuentra la salida en un laberinto complejo sin información previa.

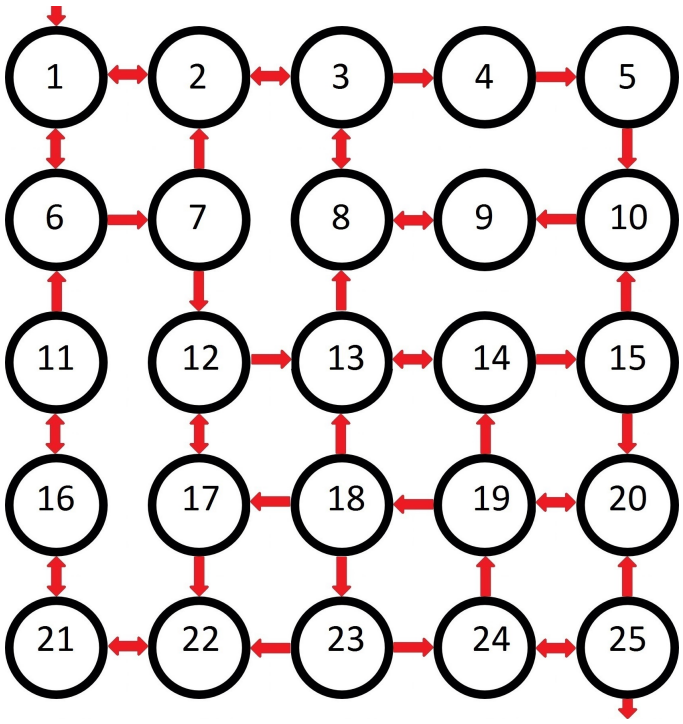


Fig. 4.1: Laberinto de 25 posiciones.

4.1.1. Especificación en MTSA

```
VIEW = P01,
P01  = (sur   -> P06 | este  -> P02),
P02  = (este  -> P03 | oeste -> P01),
P03  = (sur   -> P08 | este  -> P04 | oeste -> P02),
P04  = (este  -> P05),
P05  = (sur   -> P10),
P06  = (norte -> P01 | este  -> P07),
P07  = (norte -> P02 | sur   -> P12),
P08  = (norte -> P03 | este  -> P09),
P09  = (oeste -> P08),
P10  = (oeste -> P09),
P11  = (norte -> P06 | sur   -> P16),
P12  = (sur   -> P17 | este  -> P13),
P13  = (norte -> P08 | este  -> P14),
P14  = (este  -> P15 | oeste -> P13),
P15  = (norte -> P10 | sur   -> P20),
P16  = (norte -> P11 | sur   -> P21),
P17  = (norte -> P12 | sur   -> P22),
P18  = (norte -> P13 | sur   -> P23 | oeste -> P17),
P19  = (norte -> P14 | este  -> P20 | oeste -> P18),
P20  = (oeste -> P19),
P21  = (norte -> P16 | este  -> P22),
P22  = (oeste -> P21),
P23  = (este  -> P24 | oeste -> P22),
P24  = (norte -> P19 | este  -> P25),
P25  = (norte -> P20 | oeste -> P24 | salir -> P25).

MODEL = (norte? -> MODEL | sur?   -> MODEL | este? -> MODEL |
         oeste? -> MODEL | salir? -> MODEL).

set Controllable_25 = {norte, sur, este, oeste, salir}
fluent F_Salir      = <salir, Controllable_25\{salir}>
assert A_Salir      = F_Salir

controllerSpec GOAL_25 = {
    liveness     = {A_Salir}
    controllable = {Controllable_25}
}

exploration M25 = {
    environment = {VIEW},
    model       = {MODEL},
    goal        = {GOAL_25}
}
```

4.1.2. Estrategia Optimista

El robot logra salir del laberinto en 79 pasos, pasando en promedio 3.16 veces por cada posición.

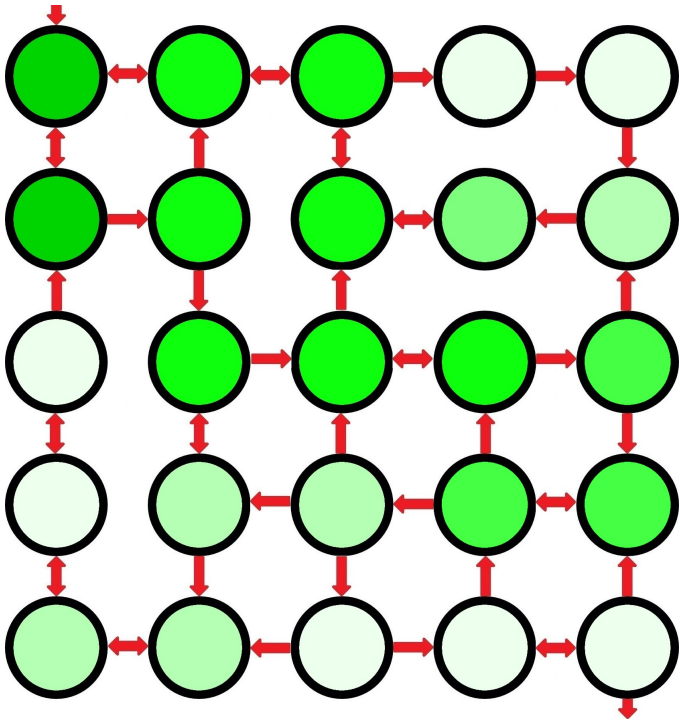


Fig. 4.2: Mapa de calor del robot en el laberinto de 25 posiciones utilizando la estrategia Optimista.

En el siguiente video `https://youtu.be/KakVlfLrIx4` podemos observar, paso a paso, como se mueve el robot a través del entorno mientras realiza la exploración.

Análisis de los modelos

Para lograr dar una respuesta sobre si es posible o no garantizar el cumplimiento del objetivo fue necesario explorar el entorno casi por completo. Únicamente siete acciones no fueron ejecutadas.

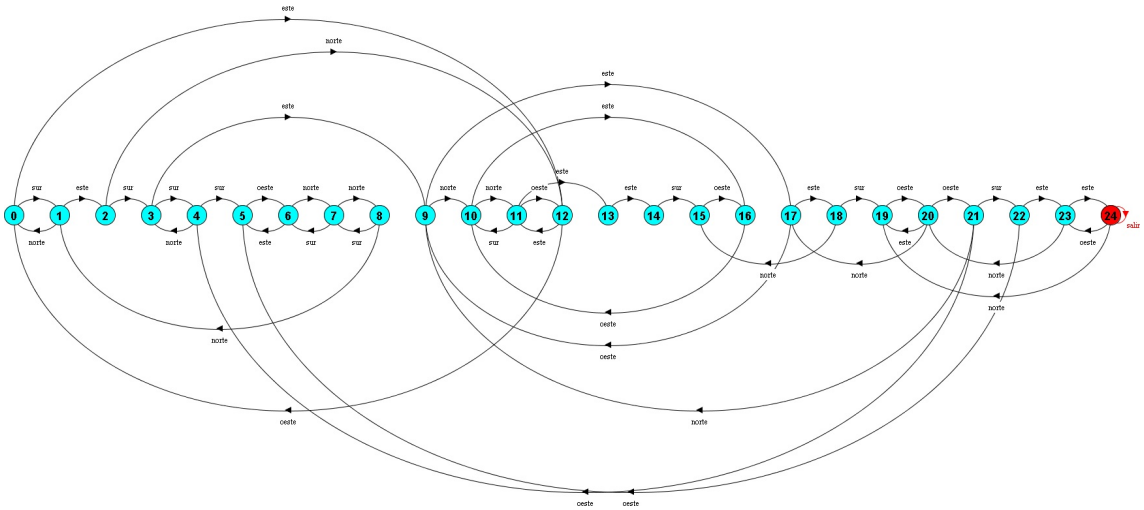


Fig. 4.3: LTS que representa al mapa del laberinto de 25 posiciones.

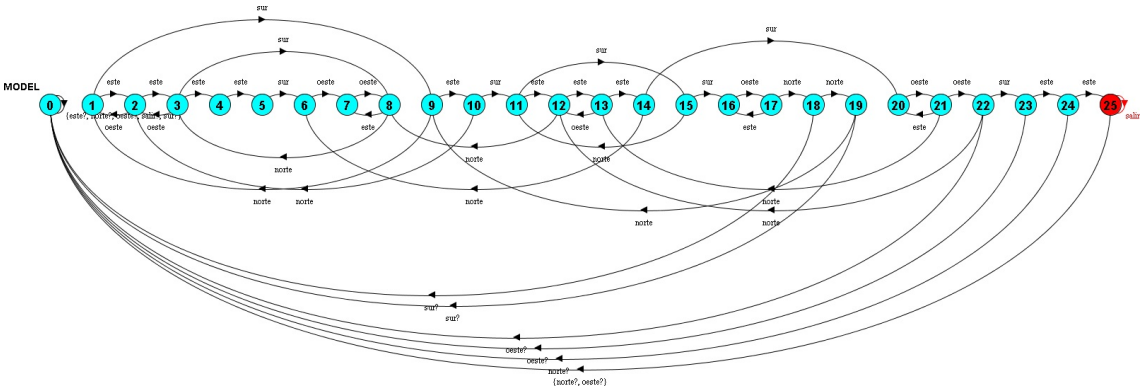


Fig. 4.4: MTS que representa al conocimiento adquirido sobre el laberinto de 25 posiciones utilizando la estrategia Optimista.

4.1.3. Estrategia Optimista - Nueva acción

En este caso de prueba, el modelo de conocimiento con el cual contamos al inicio de la exploración, no aporta información sobre zonas inseguras, por lo cual las estrategias Nueva acción y Optimista - Nueva acción obtienen exactamente el mismo resultado.

Análisis preliminar

El robot logra salir del laberinto en 116 pasos, pasando en promedio 4.64 veces por cada posición. Podemos observar a simple vista que las posiciones más visitadas forman parte del camino hacia la salida.

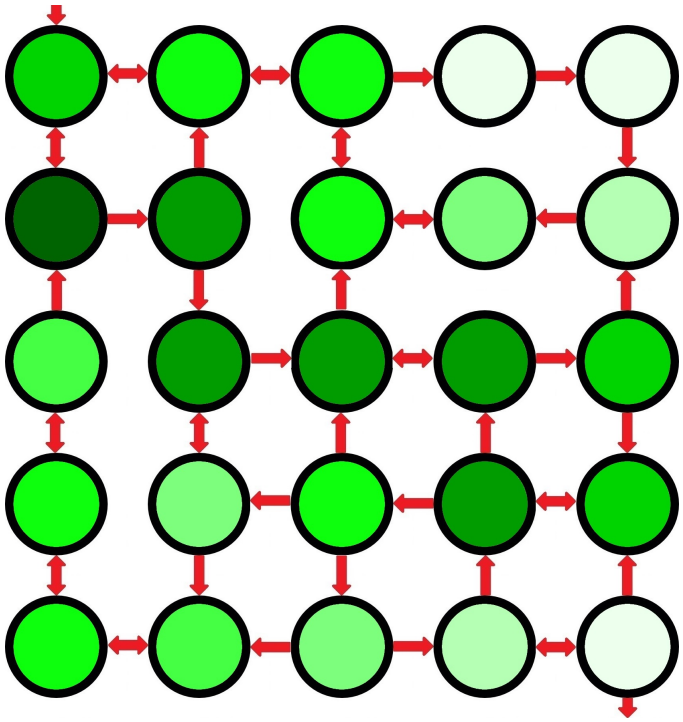


Fig. 4.5: Mapa de calor del robot en el laberinto de 25 posiciones utilizando la estrategia Optimista - Nueva acción.

En el siguiente video `https://youtu.be/8XwPYquCUB4` podemos observar, paso a paso, como se mueve el robot a través del entorno mientras realiza la exploración. El video nos permite ver de una forma más clara que regiones explora primero, y como va aprendiendo caminos que le permiten llegar a posiciones prometedoras para la exploración.

Análisis de los modelos

Para lograr dar una respuesta sobre si es posible o no garantizar el cumplimiento del objetivo fue necesario explorar el entorno casi por completo. Las únicas dos acciones no ejecutadas fueron norte y oeste desde la posición final, por lo cual en el modelo del conocimiento van hacia *La nube*.

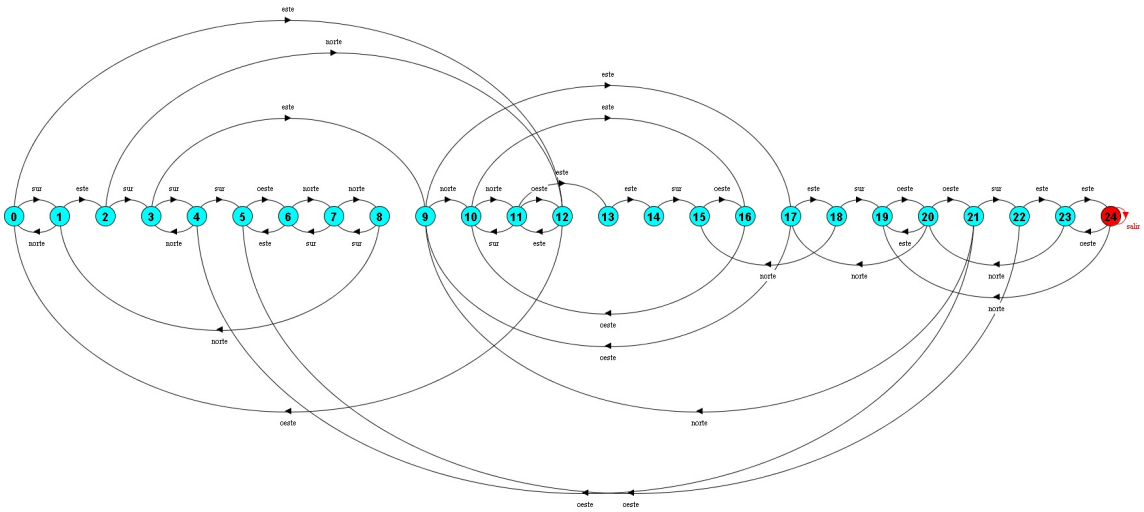


Fig. 4.6: LTS que representa al mapa del laberinto de 25 posiciones.

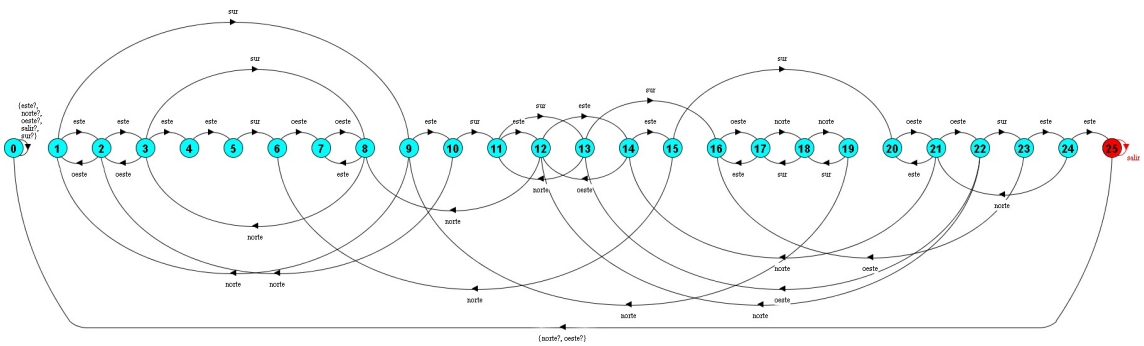


Fig. 4.7: MTS que representa al conocimiento adquirido sobre el laberinto de 25 posiciones utilizando la estrategia Optimista - Nueva acción.

4.1.4. Variantes

Si eliminamos la acción salir, el robot descubre que el laberinto no tiene salida en 122 pasos, formando un modelo de conocimiento de dos componentes conexas, en cual una componente conexa es bisimilar al LTS que representa al mapa, mientras que la otra componente conexa es *La nube*.

Si modificamos el mapa para que la acción salir se encuentre en P08 en vez de encontrarse en P25, el robot logra salir del laberinto en solamente 9 pasos, siendo su modelo del conocimiento mucho más reducido que el modelo del mapa.

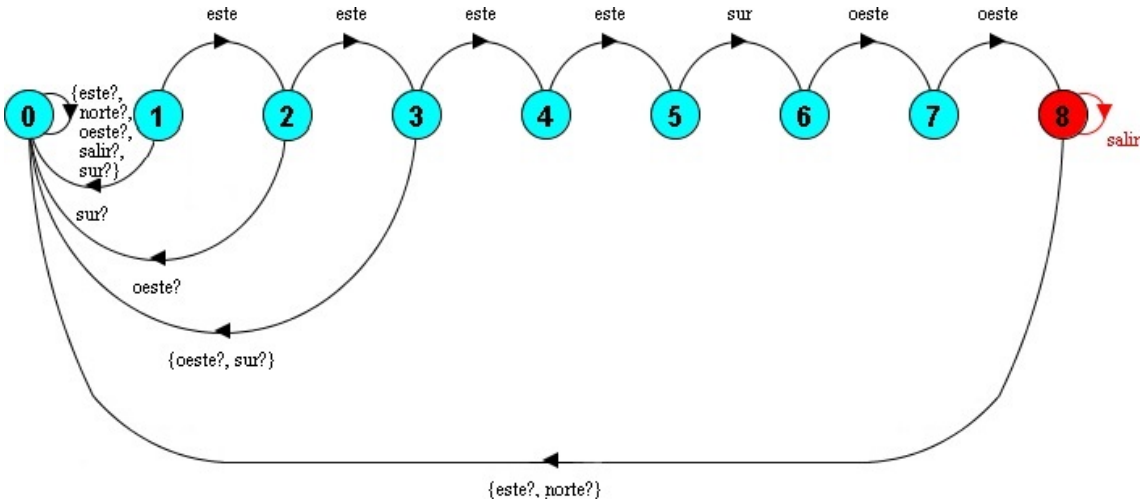


Fig. 4.8: MTS que representa al conocimiento adquirido sobre el laberinto de 25 posiciones con salida en la posición 8.

Tanto utilizando la estrategia Optimista como la estrategia Optimista - Nueva acción, el resultado es el mismo con ambas variantes.

4.2. Zonas inseguras

En este ejemplo veremos como la estrategia Optimista utiliza nuestro conocimiento previo para evitar las zonas inseguras.

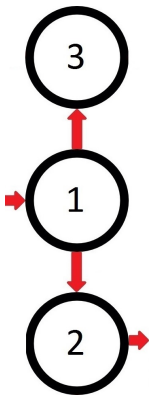


Fig. 4.9: Laberinto con zona insegura.

4.2.1. Especificación en MTSA

```
VIEW   = (sur    -> GANAR | norte -> PERDER),
GANAR  = (ganar  -> GANAR),
PERDER = (perder -> PERDER).

MODEL_CON_INFO = (sur?    -> MODEL_GANAR    | norte?  -> MODEL_PERDER  |
                  ganar?  -> MODEL_CON_INFO | perder? -> MODEL_CON_INFO),
MODEL_GANAR    = (sur?    -> MODEL_GANAR    | norte?  -> MODEL_GANAR   |
                  ganar?  -> MODEL_GANAR),
MODEL_PERDER   = (sur?    -> MODEL_PERDER   | norte?  -> MODEL_PERDER  |
                  perder? -> MODEL_PERDER).

set Controllable_UNSAFE = {norte, sur, ganar, perder}
fluent F_Ganar          = <ganar, Controllable_UNSAFE\{ganar}>
assert A_Ganar          = F_Ganar

controllerSpec GOAL_UNSAFE = {
    liveness     = {A_Ganar}
    controllable = {Controllable_UNSAFE}
}

exploration UNSAFE = {
    environment = {VIEW},
    model       = {MODEL_CON_INFO},
    goal        = {GOAL_UNSAFE}
}
```

4.2.2. Análisis de los modelos

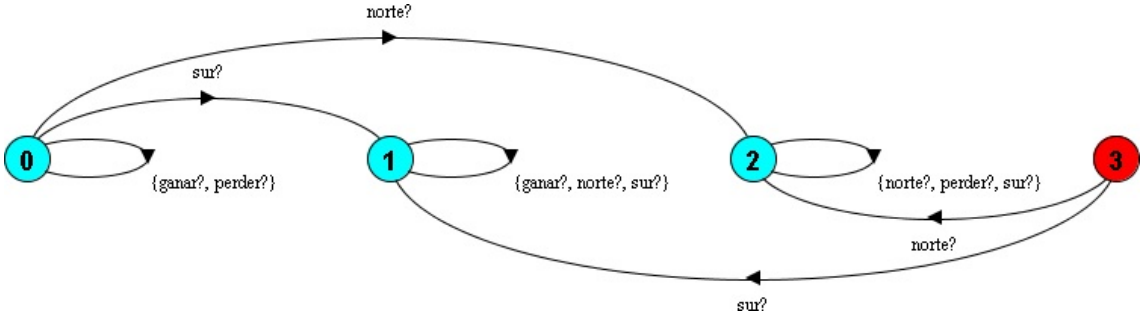


Fig. 4.10: MTS que representa al conocimiento inicial sobre el laberinto con zona insegura.

En la figura 4.10 podemos observar nuestro conocimiento sobre el entorno al iniciar la exploración. Los estados 0, 1 y 2 pertenecen a *La nube*, el estado 3 es el estado inicial.

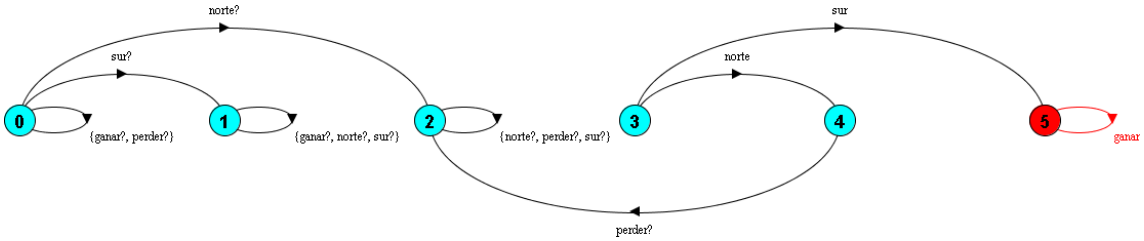


Fig. 4.11: MTS que representa al conocimiento adquirido sobre el laberinto con zona insegura, utilizando la estrategia Nueva Acción.

Si utilizamos la estrategia Nueva Acción, lo único que importa es elegir una acción nueva. En el caso de la figura 4.11, la estrategia eligió primero norte, llegando así a un estado sin salida. Fue necesario realizar un reset para luego elegir sur y llegar a un estado ganador.

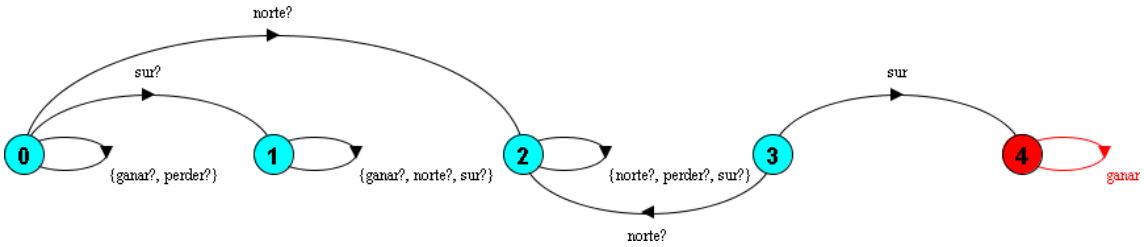


Fig. 4.12: MTS que representa al conocimiento adquirido sobre el laberinto con zona insegura, utilizando la estrategia Optimista - Nueva Acción.

En cambio, utilizando la estrategia Optimista u Optimista - Nueva Acción, por construcción, el controlador optimista evita las zonas inseguras. Al elegir la acción, se descarta norte, por lo cual el robot llega a un estado ganador sin la necesidad de realizar un reset, como puede observarse en la figura 4.12.

4.3. Laberinto que baja

En este ejemplo, el robot se puede mover libremente en forma horizontal, ya que si se mueve a izquierda o derecha, siempre puede regresar a la posición en la que estaba. Si el robot se mueve hacia abajo no puede volver a subir. La salida no se encuentra en el nivel inferior, por lo cual todos los estados del nivel inferior son estados perdedores.

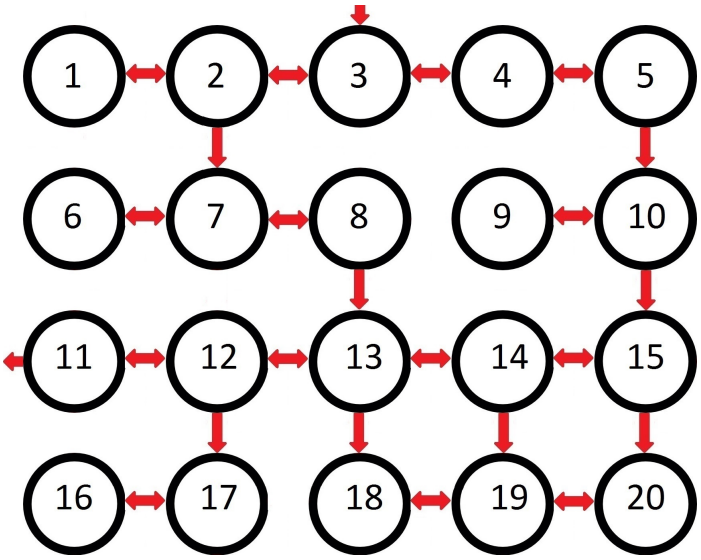


Fig. 4.13: Laberinto que baja.

4.3.1. Especificación en MTSA

```
VIEW = P03,
P01 = (este  -> P02),
P02 = (sur   -> P07 | este  -> P03 | oeste -> P01),
P03 = (este  -> P04 | oeste -> P02),
P04 = (este  -> P05 | oeste -> P03),
P05 = (sur   -> P10 | oeste -> P04),
P06 = (este  -> P07),
P07 = (este  -> P08 | oeste -> P06),
P08 = (sur   -> P13 | oeste -> P07),
P09 = (este  -> P10),
P10 = (sur   -> P15 | oeste -> P09),
P11 = (este  -> P12 | salir -> P11),
P12 = (sur   -> P17 | este  -> P13 | oeste -> P11),
P13 = (sur   -> P18 | este  -> P14 | oeste -> P12),
P14 = (sur   -> P19 | este  -> P15 | oeste -> P13),
P15 = (sur   -> P20 | oeste -> P14),
P16 = (este  -> P17),
P17 = (oeste -> P16),
P18 = (este  -> P19),
P19 = (este  -> P20 | oeste -> P18),
P20 = (oeste -> P19).
```

```
MODEL_CON_INFO = CENTRO,
CENTRO   = (sur?   -> CENTRO   | este?  -> ESTE_1 |
            oeste? -> OESTE_1 | salir? -> CENTRO),
ESTE_1   = (sur?   -> CENTRO   | este?  -> ESTE_2 |
            oeste  -> CENTRO   | salir? -> ESTE_1),
ESTE_2   = (sur?   -> CENTRO   | este?  -> ESTE_3 |
            oeste  -> ESTE_1   | salir? -> ESTE_2),
ESTE_3   = (sur?   -> CENTRO   | este?  -> ESTE_4 |
            oeste  -> ESTE_2   | salir? -> ESTE_3),
ESTE_4   = (sur?   -> CENTRO   | oeste  -> ESTE_3 |
            salir? -> ESTE_4),
OESTE_1  = (sur?   -> CENTRO   | este   -> CENTRO  |
            oeste? -> OESTE_2 | salir? -> OESTE_1),
OESTE_2  = (sur?   -> CENTRO   | este   -> OESTE_1 |
            oeste? -> OESTE_3 | salir? -> OESTE_2),
OESTE_3  = (sur?   -> CENTRO   | este   -> OESTE_2 |
            oeste? -> OESTE_4 | salir? -> OESTE_3),
OESTE_4  = (sur?   -> CENTRO   | este   -> OESTE_3 |
            salir? -> OESTE_4).

set Controllable_Down = {sur, este, oeste, salir}
fluent F_Salir        = <salir, Controllable_Down\{salir}>
assert A_Salir        = F_Salir

controllerSpec GOAL_Down = {
    liveness     = {A_Salir}
    controllable = {Controllable_Down}
}

exploration MDown = {
    environment = {VIEW},
    model       = {MODEL_CON_INFO},
    goal        = {GOAL_Down}
}
```

4.3.2. Análisis de los modelos

Sabemos que:

- Si podemos movernos una posición horizontalmente, también vamos a poder movernos una posición en la dirección contraria.
- Como máximo podemos movernos hasta cuatro veces en la misma dirección.
- Si bajamos, llegamos a una posición que cumple las dos condiciones antes mencionadas.

Vamos a utilizar como modelo del conocimiento inicial un MTS que represente estas tres condiciones.

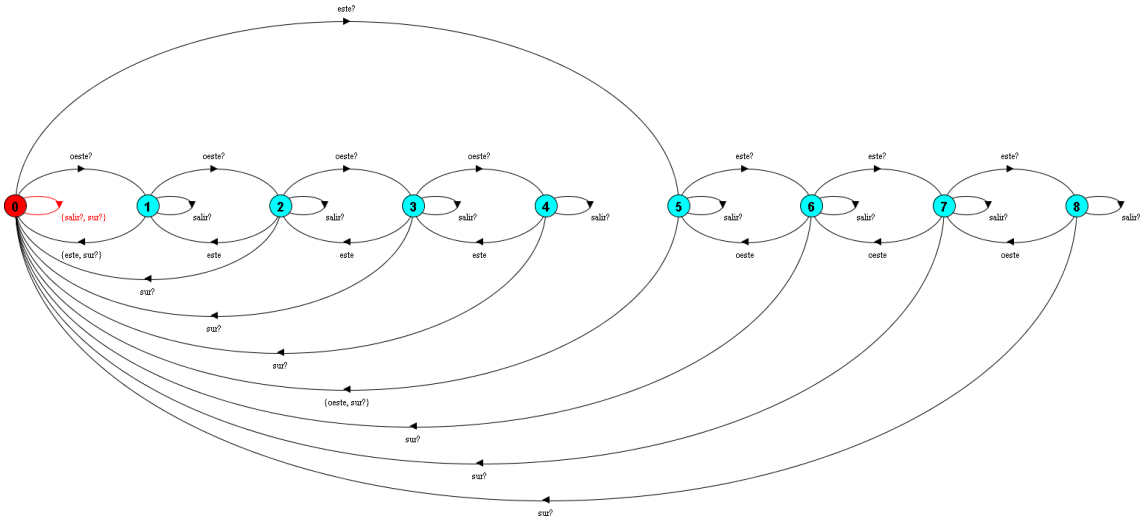


Fig. 4.14: MTS que representa al conocimiento inicial sobre el laberinto que baja.

El problema está en que no podemos modelar la siguiente condición:

- Si podemos movernos una posición horizontalmente, también vamos a poder movernos a **la misma** posición de la que partimos.

Por carecer de esta condición, no podemos saber que las acciones de movimiento horizontal son más seguras que la acción bajar, ya que la acción bajar nos puede llevar a un estado perdedor desde un estado ganador, cosa que no puede pasar con las acciones horizontales.

4.3.3. Resultado

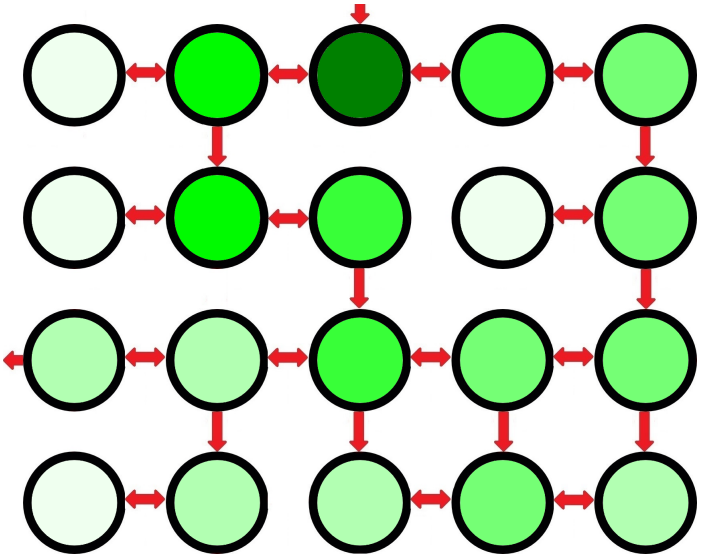


Fig. 4.15: Mapa de calor del robot en el laberinto que baja.

Utilizando la estrategia Nueva Acción o la estrategia Optimista - Nueva Acción, el robot logra salir del laberinto en 58 pasos, para lo cual fue necesario realizar un reset en 4 oportunidades. Al no poder distinguir los estados seguros de los que nos pueden llevar a estados perdedores, la estrategia elige arbitrariamente cualquier acción nueva, cayendo varias veces en estados perdedores.

Utilizando la estrategia Optimista, el robot logra salir del laberinto en 61 pasos, para lo cual fue necesario realizar un reset en 4 oportunidades.

4.4. Laberinto con puente

En este ejemplo, el robot puede moverse libremente entre todas las posiciones contiguas, pero entre las posiciones 7 y 8 hay un puente controlado por un agente externo, quien puede cerrar el paso.

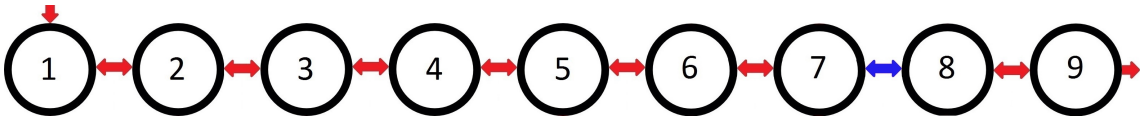


Fig. 4.16: Laberinto con puente.

Este ejemplo introduce un agente externo, por lo cual vamos a necesitar definir:

- Un LTS que represente el comportamiento del agente externo. Indica que acciones puede realizar el agente externo en cada momento.
- Un MTS que represente nuestro conocimiento inicial sobre el comportamiento del agente externo. El robot adquirirá conocimiento sobre el agente externo a medida que pueda observar sus acciones.
- Una suposición de dominio sobre el comportamiento del agente externo, la cual garantice que el agente externo no va a impedir que alcancemos nuestro objetivo.
- Aunque es opcional, vamos a utilizar una lista de acciones que ejecutará el agente externo cíclicamente. De esta forma, especificamos su comportamiento para este ejemplo.

  En caso de que hubiera más de un agente externo, deberíamos definir estos cuatro items para cada uno de ellos.

4.4.1. Especificación en MTSA

```
MAPA = M1,
M1 = (este   -> M2),
M2 = (este   -> M3 | oeste  -> M1),
M3 = (este   -> M4 | oeste  -> M2),
M4 = (este   -> M5 | oeste  -> M3),
M5 = (este   -> M6 | oeste  -> M4),
M6 = (este   -> M7 | oeste  -> M5),
M7 = (cruzar -> M8 | oeste  -> M6),
M8 = (este   -> M9 | cruzar -> M7),
M9 = (oeste  -> M8 | ganar  -> M9).

PUENTE = PUENTE_CERRADO,
PUENTE_CERRADO = (abrir  -> PUENTE_ABIERTO | cerrar -> PUENTE_CERRADO),
PUENTE_ABIERTO = (abrir  -> PUENTE_ABIERTO | cruzar -> PUENTE_ABIERTO |
                  cerrar -> PUENTE_CERRADO).

MTS_MAPA   = (este?   -> MTS_MAPA   | oeste?  -> MTS_MAPA   |
              cruzar? -> MTS_MAPA   | ganar?  -> MTS_MAPA).

MTS_PUENTE = (abrir?  -> MTS_PUENTE | cerrar? -> MTS_PUENTE |
              cruzar? -> MTS_PUENTE).

set Controllable_puente = {este, oeste, cruzar, ganar}

fluent F_Ganar = <ganar, Controllable_puente\{ganar}>
assert A_Ganar = F_Ganar

fluent F_PuedeCruzar = <abrir, cruzar>
assert PuedeCruzar   = F_PuedeCruzar

controllerSpec GOAL_PUENTE = {
    assumption   = {PuedeCruzar}
    liveness     = {A_Ganar}
    controllable = {Controllable_puente}
}

exploration PUENTE_CON_SALIDA = {
    environment         = {MAPA, PUENTE},
    model               = {MTS_MAPA, MTS_PUENTE},
    goal                = {GOAL_PUENTE},
    environment_actions = {{cerrar, cerrar, cerrar, cerrar, cerrar,
                            cerrar, cerrar, cerrar, cerrar, cerrar,
                            abrir, abrir, abrir}}
}
```

4.4.2. Estrategias

Este ejemplo cuenta con un agente externo. La estrategia Optimista no está preparada para estos casos, por lo cual entra en un ciclo que le impide adquirir nuevo conocimiento. Al utilizar las estrategias Nueva Acción u Optimista - Nueva Acción, el resultado es el mismo.

4.4.3. Análisis de los modelos

Como el robot observa todas las acciones del agente externo, al final de la exploración, el MTS que representa su comportamiento está completamente refinado.

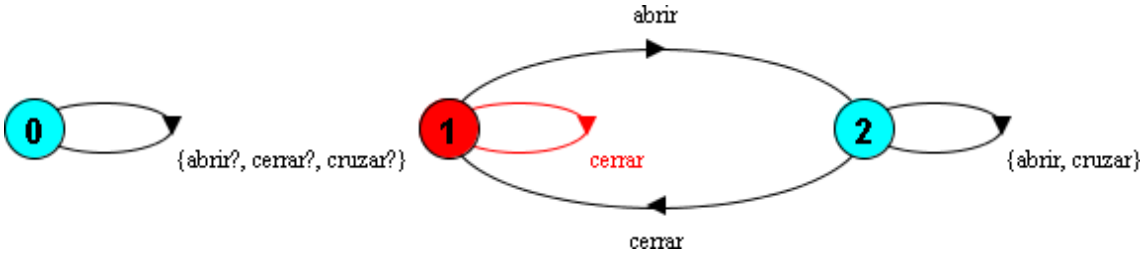


Fig. 4.17: MTS que representa al conocimiento adquirido sobre el agente externo del laberinto con puente.

En cuanto al mapa, la única acción no ejecutada es cruzar desde el estado 8 al estado 7.

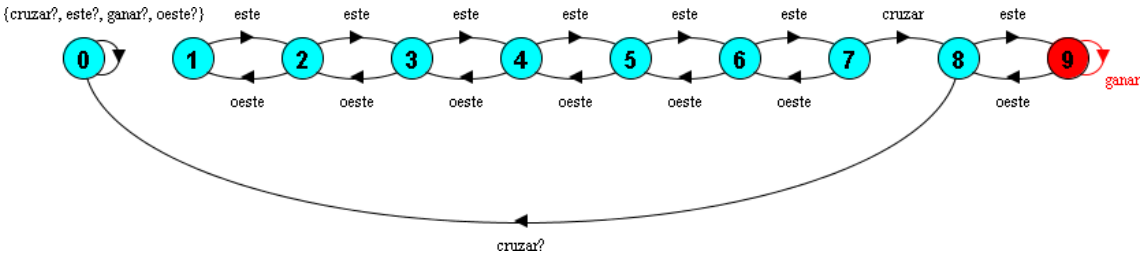


Fig. 4.18: MTS que representa al conocimiento adquirido sobre el laberinto con puente.

4.4.4. Análisis de la traza

El robot logra salir del laberinto en 30 pasos. En el siguiente video `https://youtu.be/wec0GPml9dc` podemos observar, movimiento a movimiento, como cambian los modelos de conocimiento que el robot tiene sobre el entorno, el cual en este caso es el camino que tiene un puente, y sobre el agente externo, que en este caso, es quien decide cuando el puente se abre o se cierra. El video nos permite ver de una forma más clara como las acciones nunca antes realizadas aportan información, y como el robot necesita volver a posiciones no completamente exploradas para incrementar su conocimiento.

Podemos observar que cuando el robot termina de explorar completamente los estados 1 a 7, va hasta el estado 7 a esperar que el agente externo abra el puente, en vez de esperarlo en su estado actual, ya que solamente desde el estado 7 puede beneficiarse de las acciones del agente externo.

El robot sabe que el agente externo va a abrir el puente por la suposición de dominio que tenemos. En caso de no tener esta suposición, en vez de esperar, el robot asumiría que no se puede garantizar el cumplimiento del objetivo, ya que el agente externo podría no abrir el puente nunca.

Robot	Estado	Agente
	1	
este	2	cerrar
este	3	cerrar
este	4	cerrar
este	5	cerrar
este	6	cerrar
este	7	cerrar
oeste	6	cerrar
oeste	5	cerrar
oeste	4	cerrar
oeste	3	cerrar
oeste	2	**abrir**
oeste	1	**abrir**
este	2	**abrir**
este	3	cerrar
este	4	cerrar
este	5	cerrar
este	6	cerrar
este	7	cerrar
WAIT	7	cerrar
WAIT	7	cerrar
WAIT	7	cerrar
WAIT	7	cerrar
WAIT	7	cerrar
WAIT	7	**abrir**
este	8	**abrir**
este	9	**abrir**
oeste	8	cerrar
este	9	cerrar
ganar	9	

Fig. 4.19: Traza del robot en el laberinto con puente.

5. CONCLUSIONES

En el final de esta tesis, hay cuatro temas interesantes para analizar.

El primer tema a analizar es la integración de la exploración a la herramienta MTSA, la cual se puede ver en detalle en el siguiente capítulo. Se extendió el lenguaje de Procesos de Estados Finitos (FSP) para que sea sencillo definir exploraciones, se agregaron botones para dar control total sobre la exploración y se agregó una nueva forma de visualización de los modelos, más flexible, mediante la pestaña Layout.

La herramienta nos permite visualizar la exploración paso a paso, tanto la traza de la exploración como la evolución de los modelos. También nos permite avanzar inmediatamente la cantidad de pasos necesarios, para luego ver en detalle una etapa más avanzada de la exploración. Es posible explorar siguiendo la estrategia de exploración, e intervenir manualmente en las decisiones en caso de ser necesario.

El segundo tema importante es el algoritmo de exploración. Resuelve el problema correctamente en caso de contar con una estrategia adecuada, y tiene la flexibilidad suficiente como para permitir las ventajas mencionadas en el párrafo anterior. Su mayor inconveniente es la complejidad temporal.

Por cada iteración del algoritmo es necesario realizar una síntesis de controladores para decidir si es necesario seguir explorando (dos síntesis en el caso de que sea necesario un reset), una síntesis para la estrategia Optimista, y un número variable de síntesis para la estrategia Nueva Acción en caso de que no haya acciones nuevas en el estado actual.

La síntesis de controladores es un proceso computacionalmente caro. Cada modelo que sintetizamos es muy similar al modelo sintetizado en la iteración anterior, sería interesante como trabajo a futuro poder utilizar la síntesis anterior para realizar la próxima en vez de volver a realizar el proceso desde el comienzo. Esto afectaría considerablemente el rendimiento del algoritmo en forma favorable.

El tercer tema interesante surge del ejemplo del laberinto que baja. En los MTSs, al utilizar una transición posible, perdemos la forma de especificar que podemos volver exactamente al mismo estado del cual sale la transición posible. Sería de bastante utilidad, como trabajo a futuro, extender los MTSs de alguna forma que permita especificar estos detalles.

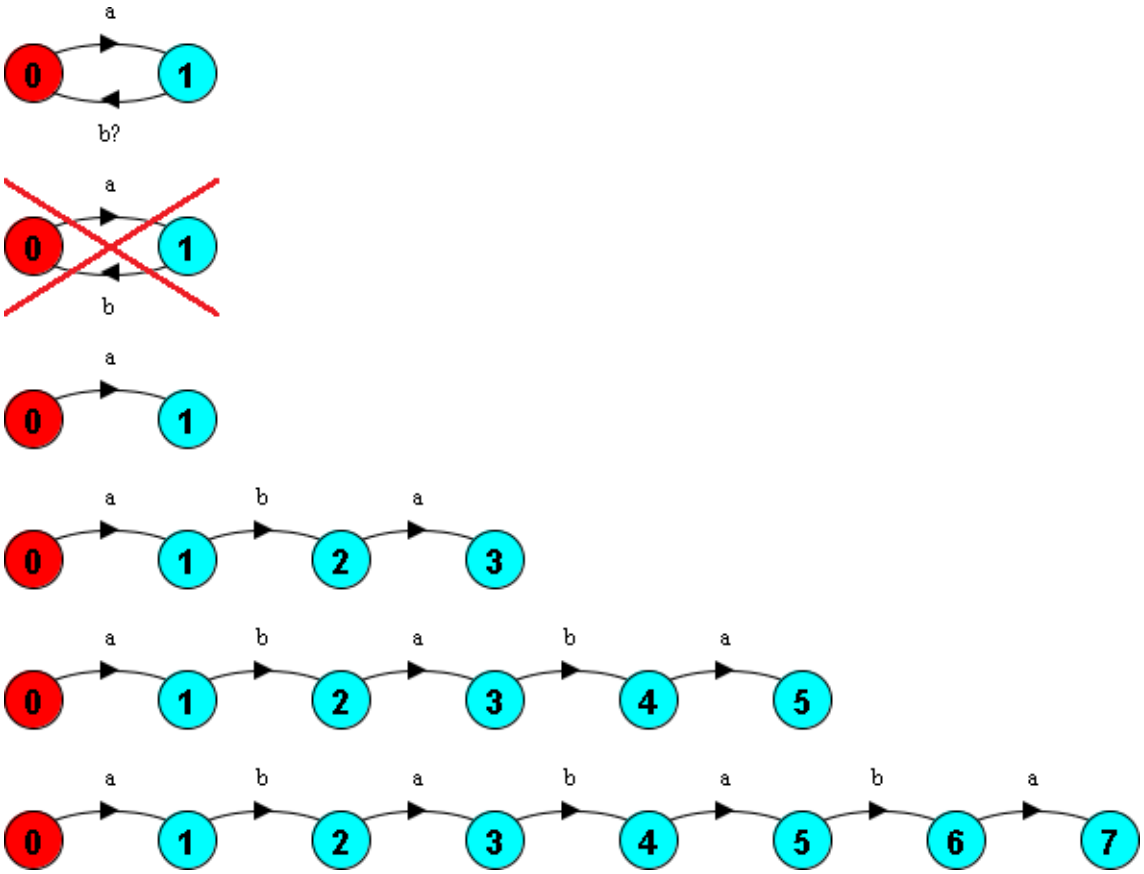


Fig. 5.1: Limitaciones del modelado con MTSs

Como se puede ver en la imagen, el MTS no se puede refinar al primero de los LTSs, sino que los LTSs que pueden ser refinados a partir del MTS son de la forma del segundo LTS en adelante.

Por último, el cuarto tema fundamental a tener en cuenta es la estrategia presentada. La estrategia resuelve el problema de salir de un laberinto, incluso con la participación de agentes externos que influyen sobre el entorno. Pero lo único que busca la estrategia es determinar si es posible o no cumplir el objetivo propuesto, como trabajo a futuro se puede mejorar en varios aspectos para reducir la cantidad de pasos necesarios para hallar una respuesta.

Por ejemplo, en caso de tener varias acciones disponibles, siempre las va a elegir basándose en el mismo orden arbitrario. En consecuencia, la exploración siempre da preferencia a una dirección por sobre las demás, lo cual puede ser perjudicial. Sería interesante definir la forma en la que se elige una acción en caso de que haya varias acciones que cumplen los requisitos de la estrategia, ya sea de forma aleatoria o mediante una heurística.

Algo similar ocurre cuando la estrategia necesita dirigirse hacia otro estado, por encontrarse en un estado completamente explorado. En la versión actual de la estrategia se intenta llegar a un estado elegido de forma arbitraria entre los que no fueron completamente explorados. Hay varias opciones que podrían llegar a ser más adecuadas, como

por ejemplo dirigirse hacia el estado más cercano a la posición actual, al más cercano al inicio para hacer una exploración de estilo BFS (Breadth First Search), o al más lejano para explorar en forma DFS (Depth First Search) o intentar heurísticas más complicadas, quizás utilizando la información sobre el entorno de la cual disponemos en el momento.

Además de las posibles optimizaciones sobre la estrategia presentada, sería interesante elaborar estrategias nuevas para resolver el problema, o estrategias adecuadas para circunstancias particulares, como no disponer de un modelo del entorno, o no poder distinguir inequívocamente las posiciones en el entorno.

En conclusión, este trabajo es un primer paso, el cual brinda las herramientas necesarias para comenzar a investigar el campo de la exploración mediante MTSs. Todavía queda mucho por hacer en varios aspectos diferentes, ya que es un tema nuevo, extenso e interesante.

6. APÉNDICE

Manual de usuario

Se explica detalladamente cómo utilizar la herramienta MTSA para la exploración. Mostramos cómo deben ser los inputs, cuales son las funcionalidades de la herramienta, como nos permite visualizar la exploración y avanzar en ella, y cuales son los output que nos brinda.

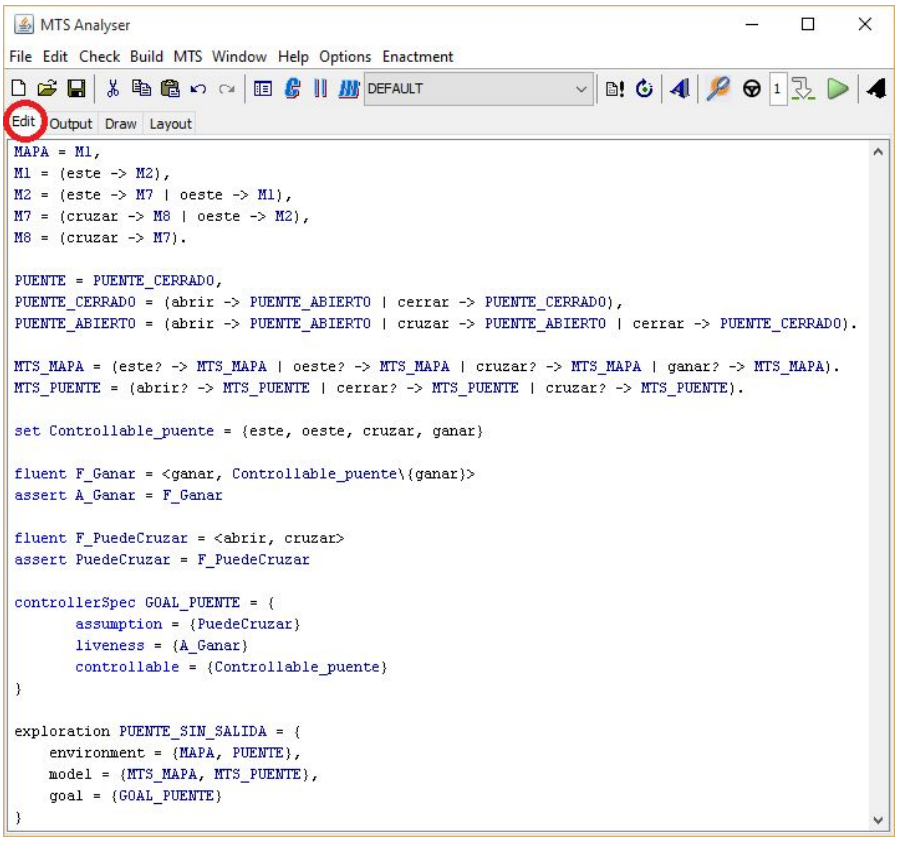
Ejemplo de exploración en MTSA

Mostramos con un ejemplo el seguimiento de una exploración en la herramienta MTSA. Comenzamos con un modelo, el cual incluye un agente externo, y mostramos las particularidades de la exploración hasta obtener un resultado. Vemos como van evolucionando los modelos, particularidades de la estrategia elegida y cómo visualizamos el paso a paso de la exploración.

Exploración en MTSA

Inputs

La exploración se define en la pestaña Edit.



El formato de entrada es el siguiente:

exploration NAME = {
environment = {VIEW, V_COMPONENT_1, ..., V_COMPONENT_N},
model = {MODEL, M_COMPONENT_1, ..., M_COMPONENT_N}},
goal = {GOAL}
}

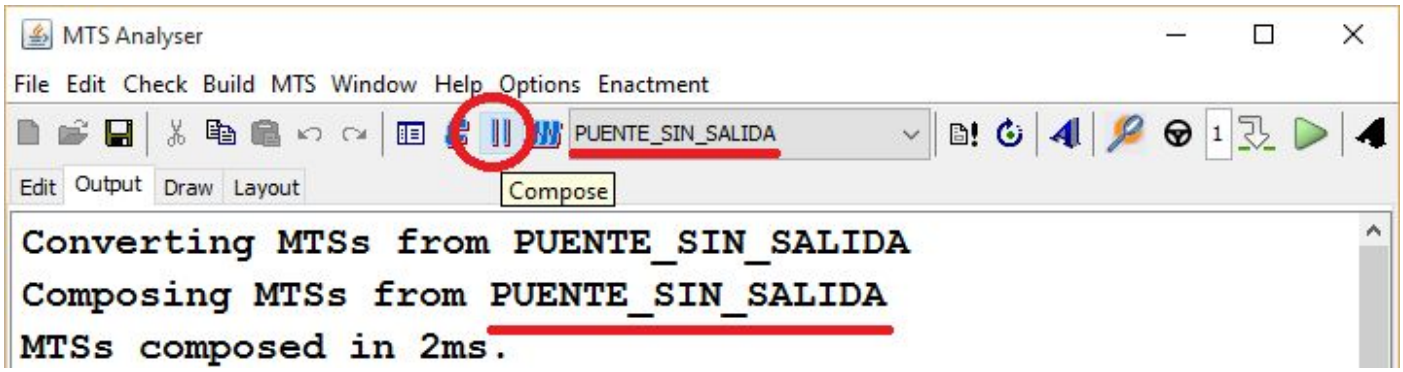
En donde

- NAME es el nombre de la exploración.
- VIEW es un LTS que representa la interacción entre el mundo y la interfaz del robot.
- V_COMPONENT son LTSs que representan a los agentes externos del mundo.
- MODEL es un MTS que contiene nuestra información inicial sobre el mundo.
- M_COMPONENT son MTSs que contienen nuestra información inicial sobre los agentes externos.
- GOAL es un controllerSpec que representa nuestro objetivo a satisfacer.

Además, definimos KNOWLEDGE como un MTS distinto para cada iteración, que contiene nuestra información sobre el mundo.
Opcionalmente, luego de goal, se puede agregar
environment_actions = {{a_1, a_2, ..., a_m}, {b_1, b_2, ..., b_m}, ..., {n_1, n_2, ..., n_m}}
De esta forma podemos indicarle al programa cual es el ciclo de acciones no controlables que ejecutará el ambiente para cada componente.
De no hacer uso de esta opción, el ambiente elegirá sus acciones de forma aleatoria.

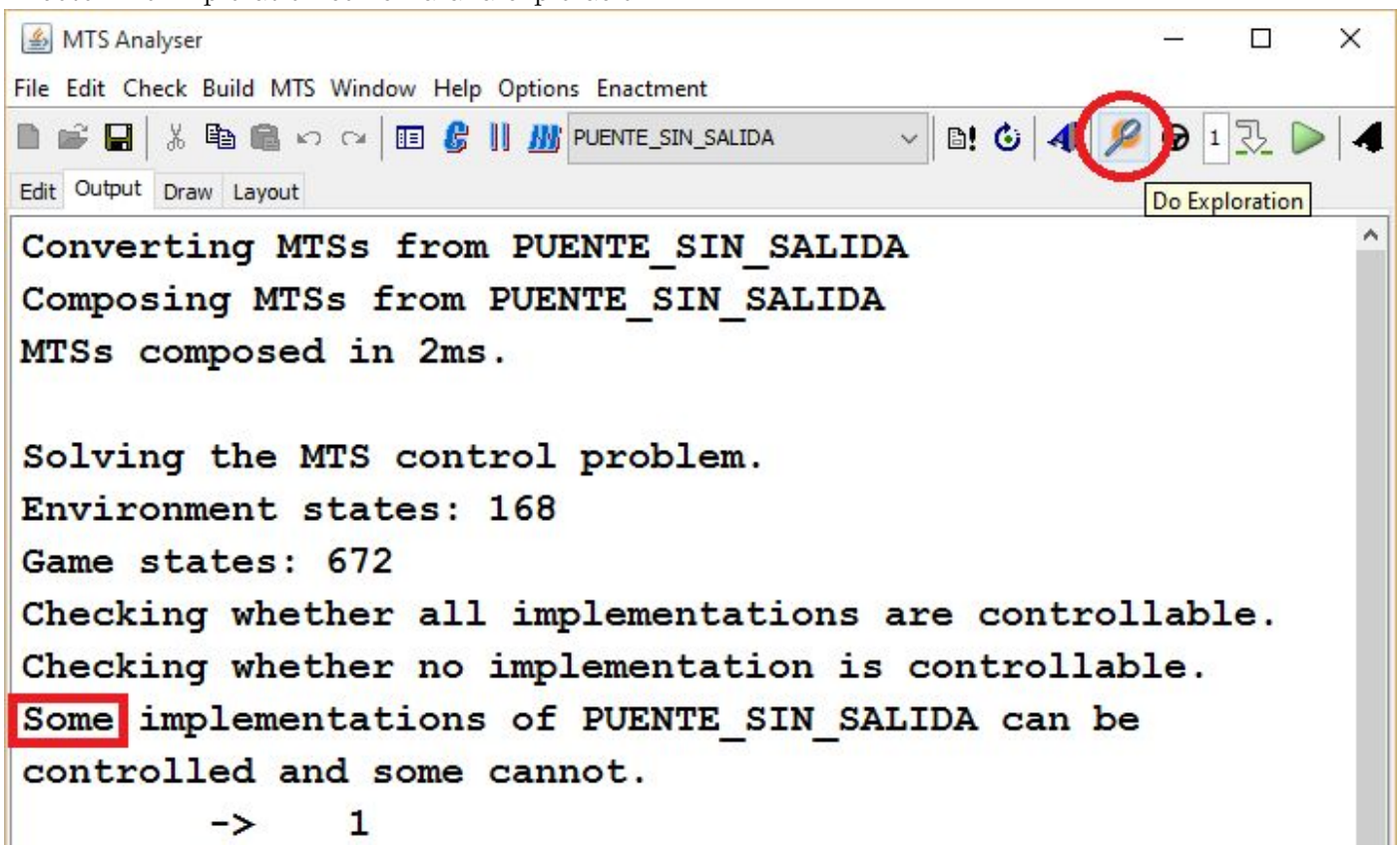
Inicio

El botón Compose compondrá la exploración.



Una vez compuesta, podemos seleccionarla por su nombre desde el combo.

El botón Do Exploration comenzará la exploración.



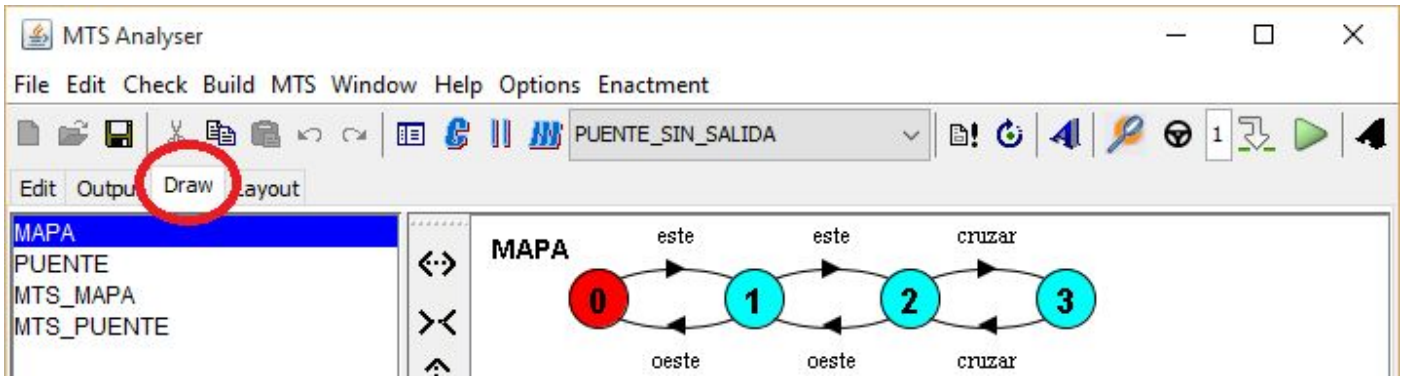
La exploración actual es la seleccionada en el combo.
En caso de no haber compuesto ninguna, se realizará la composición de la primera de la pestaña Edit.

Al comenzar la exploración, se nos informa qué podemos afirmar sobre el GOAL con nuestro conocimiento actual. Hay tres respuestas posibles:

- **All** nos dice que, sea cual sea la información que falte en nuestro KNOWLEDGE, podemos garantizar el cumplimiento del GOAL.
- **None** nos dice que, sea cual sea la información que falte en nuestro KNOWLEDGE, es imposible garantizar el cumplimiento del GOAL.
- **Some** nos dice que nuestro KNOWLEDGE no tiene la información suficiente como para decidir sobre la posibilidad de garantizar el cumplimiento del GOAL.

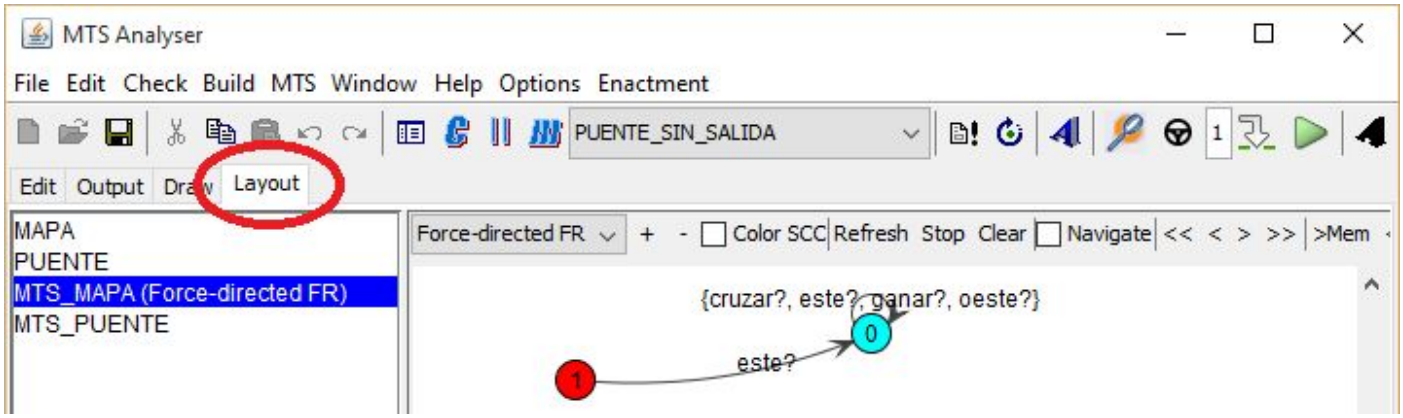
Visualización

En la pestaña Draw podemos ver los modelos que ingresamos.

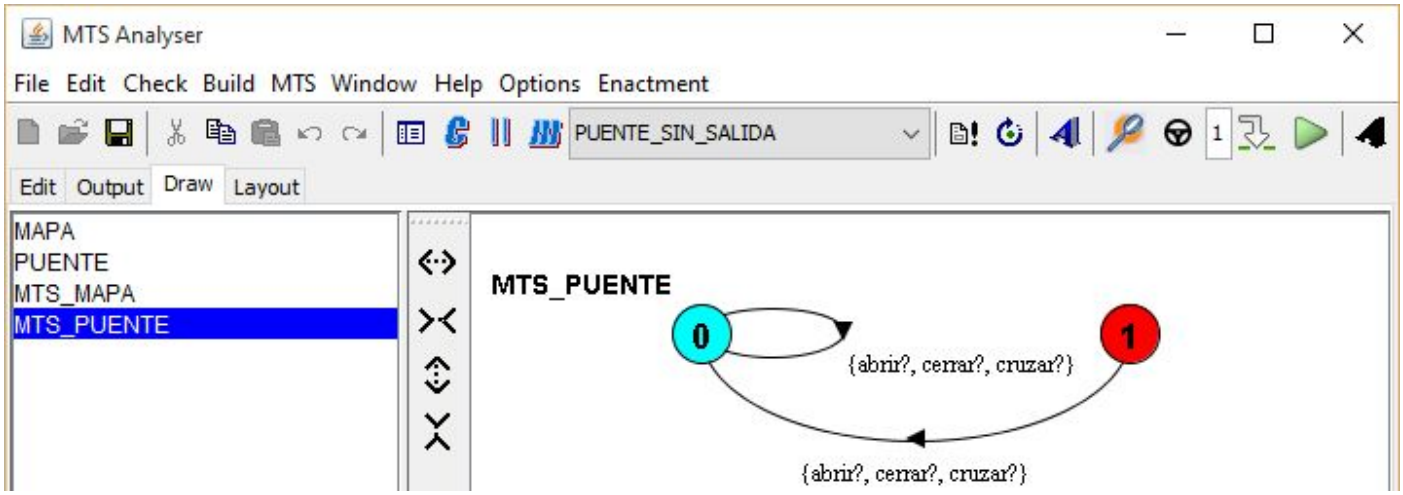


Los estados en rojo son los estados actuales.

La pestaña Layout nos permite visualizar los modelos de una forma diferente, más flexible.

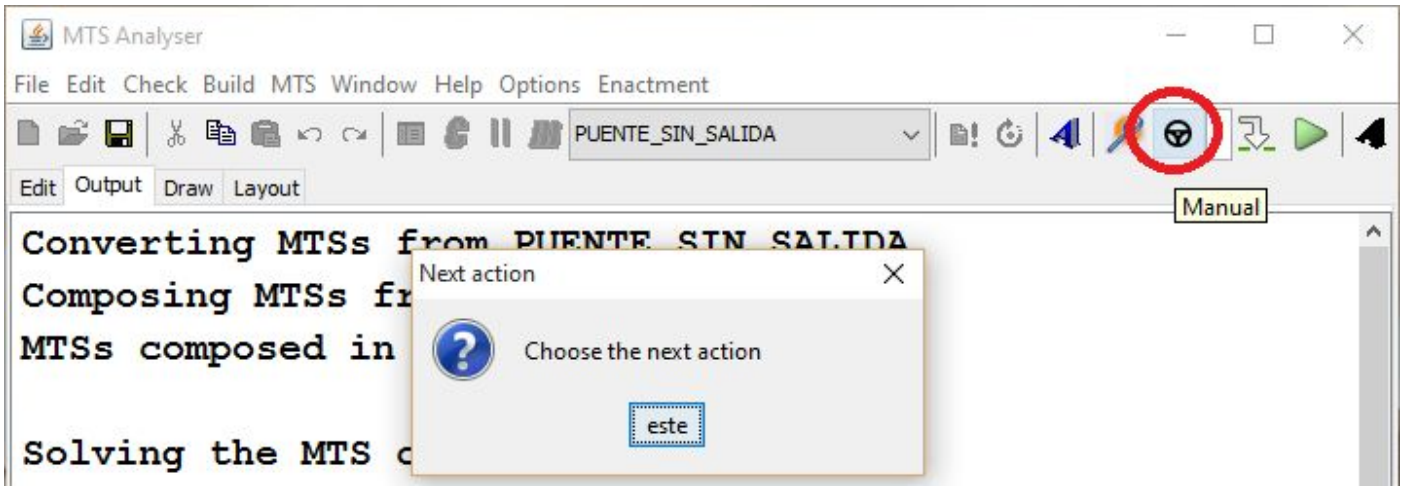


Ambas formas de visualización están integradas con la exploración y permiten ver el estado actual de todos los modelos, tanto los que representan el mundo real, como los que representan el conocimiento que vamos adquiriendo.



Exploración

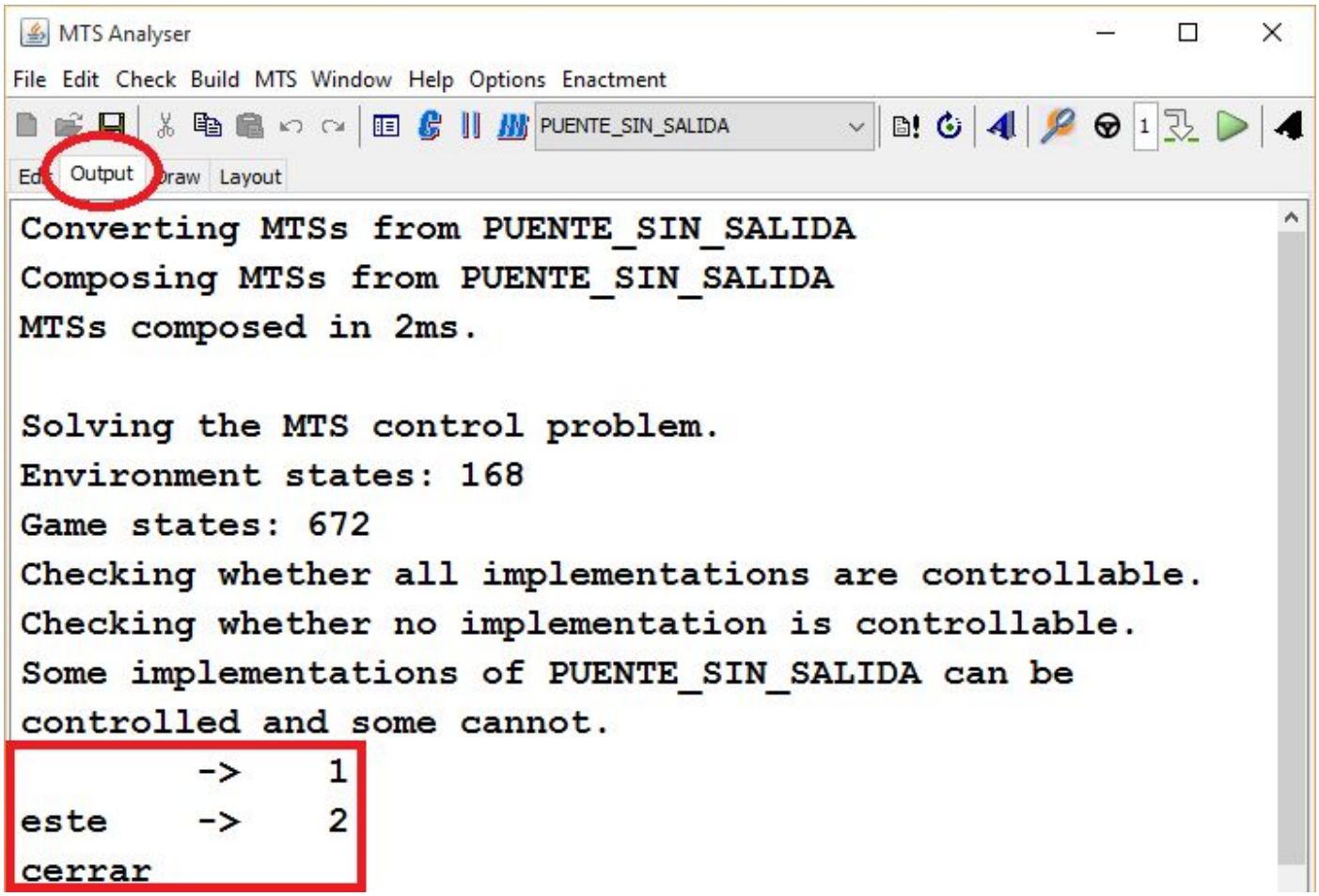
El botón Manual nos permitirá elegir la próxima acción entre las acciones disponibles.



El botón Step Over elige, automáticamente, la próxima acción de acuerdo a la estrategia actual.
Por ejemplo, si las acciones Norte, Sur y Oeste están disponibles, la estrategia decide cual es la acción más conveniente entre estas 3. Si la estrategia fuera elegir siempre la primer acción, se elegiría Norte.
Una vez seleccionada la acción, será ejecutada.

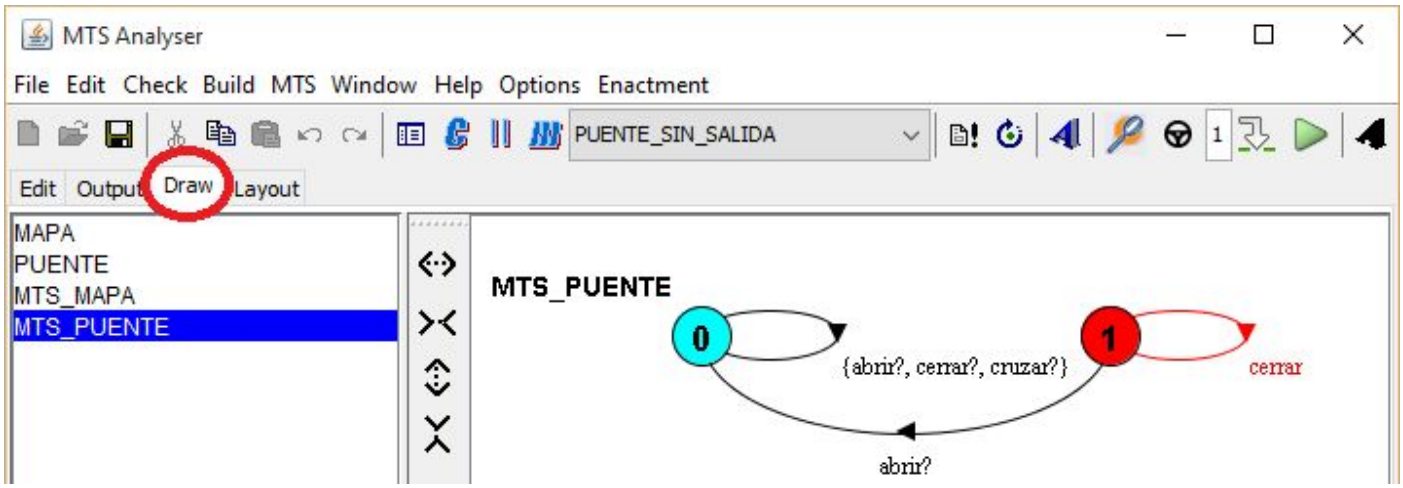
Este proceso se realiza la cantidad de veces que dice el textbox de su izquierda.
Por ejemplo, si el textbox dice 2, se elige y ejecuta una primera acción, para luego elegir y ejecutar otra acción entre las que están disponibles después de ejecutar la primera que elegimos.
Las acciones disponibles en un momento dado dependen de las acciones ejecutadas anteriormente.

Al ejecutar una acción, podemos ver la traza que genera.



Se muestra a qué estado llegamos al ejecutar la acción, y a que acciones no controlables ejecutan los agentes externos.

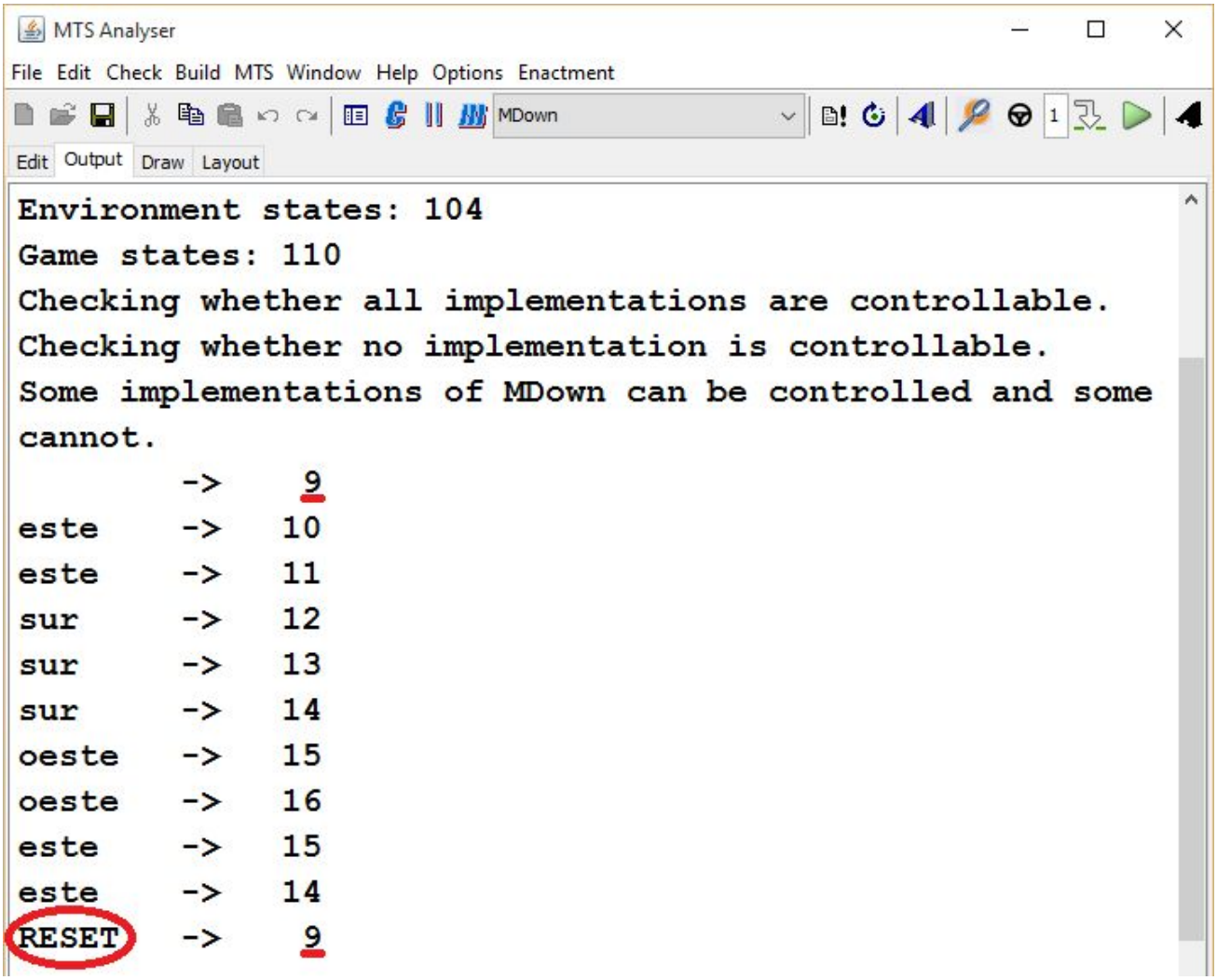
En la pestaña Draw podemos ver como las acciones ejecutadas afectan a los modelos.



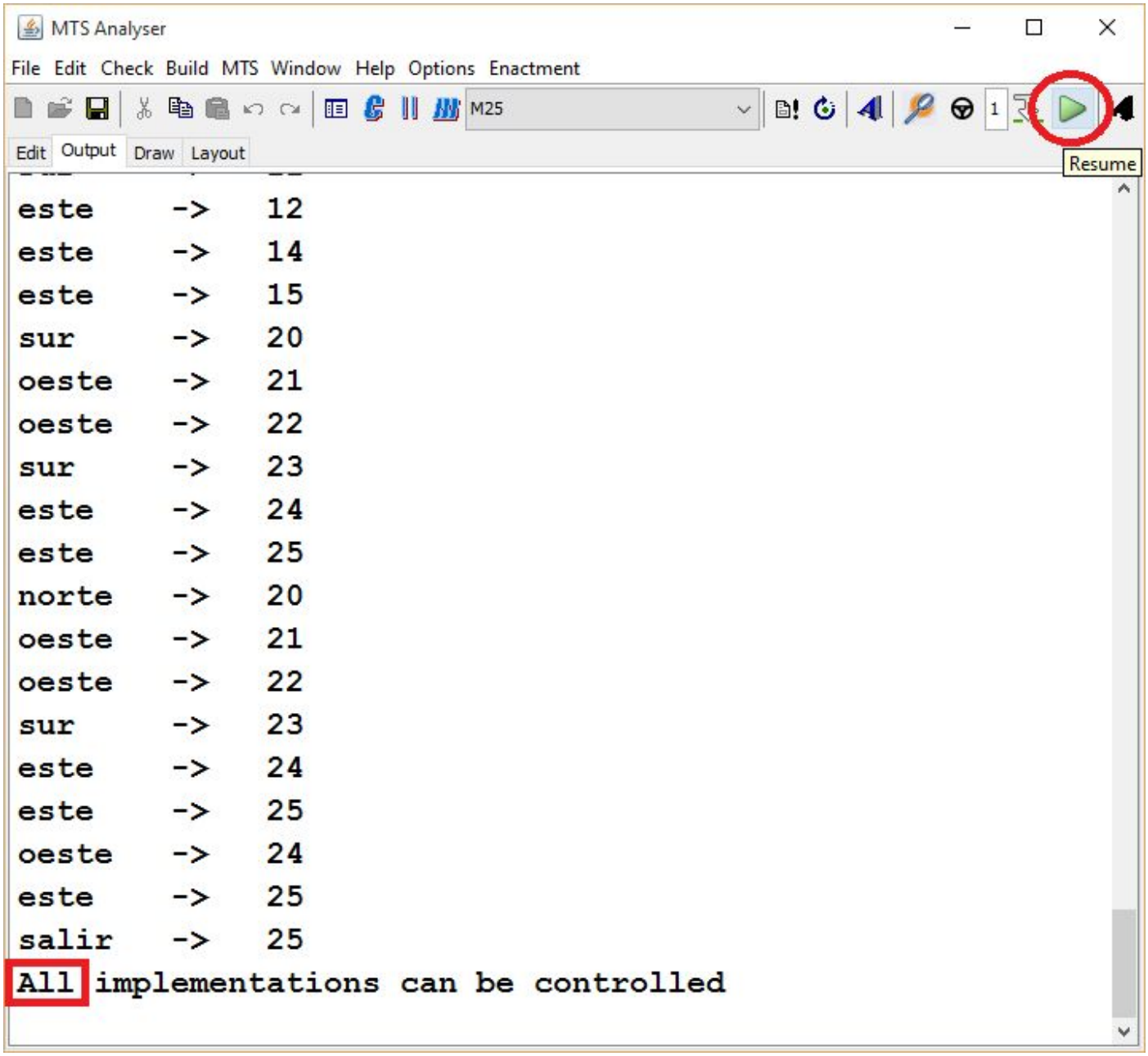
Si la estrategia detecta que para seguir incorporando conocimiento es necesario que el ambiente ejecute una acción no controlable que asumimos que va a ocurrir, puede esperar a que esa acción ocurra.

```
         ->    1
este     ->    2
cerrar
este     ->    3
cerrar
oeste    ->    2
cerrar
oeste    ->    1
cerrar
este     ->    2
cerrar
este     ->    3
cerrar
WAIT     ->    3
cerrar
```

Si detectamos que el camino elegido no nos permite seguir explorando, o en otras palabras, no es posible cumplir el objetivo desde donde estamos, pero podría ser posible cumplirlo si hubiéramos tomado otras decisiones, volveremos al inicio automáticamente.



El botón resume realiza tantos Step Over como sean necesarios hasta obtener una respuesta por **All** o **None**.



Ejemplo de exploración en MTSA

MTS Analyser

File Edit Check Build MTS Window Help Options Enactment

PUENTE_CON_SALIDA

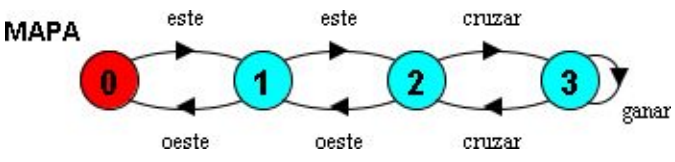
Edit Output Draw Layout

```
MAPA = M1,
M1 = (este -> M2),
M2 = (este -> M3 | oeste -> M1),
M3 = (cruzar -> M4 | oeste -> M2),
M4 = (cruzar -> M3 | ganar -> M4).

PUENTE = PUENTE_CERRADO,
PUENTE_CERRADO = (abrir -> PUENTE_ABIERTO | cerrar -> PUENTE_CERRADO),
PUENTE_ABIERTO = (abrir -> PUENTE_ABIERTO | cruzar -> PUENTE_ABIERTO | cerrar -> PUENTE_CERRADO).

MTS_MAPA = (este? -> MTS_MAPA | oeste? -> MTS_MAPA | cruzar? -> MTS_MAPA | ganar? -> MTS_MAPA).
MTS_PUENTE = (abrir? -> MTS_PUENTE | cerrar? -> MTS_PUENTE | cruzar? -> MTS_PUENTE).

set Controllable_puente = {este, oeste, cruzar, ganar}

fluent F_Ganar = <ganar, Controllable_puente\{ganar}>
assert A_Ganar = F_Ganar

fluent F_PuedeCruzar = <abrir, cruzar>
assert PuedeCruzar = F_PuedeCruzar

controllerSpec GOAL_PUENTE = {
       assumption = {PuedeCruzar}
       liveness = {A_Ganar}
       controllable = {Controllable_puente}
}

exploration PUENTE_CON_SALIDA = {
    environment = {MAPA, PUENTE},
    model = {MTS_MAPA, MTS_PUENTE},
    goal = {GOAL_PUENTE}
}
```

Input

Mapa

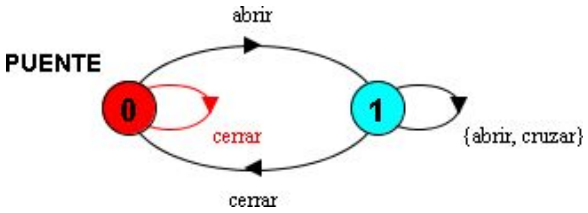
MAPA = M1,
M1 = (este -> M2),
M2 = (este -> M3 | oeste -> M1),
M3 = (cruzar -> M4 | oeste -> M2),
M4 = (cruzar -> M3 | ganar -> M4).



En este ejemplo comenzamos en la posición 0. Nuestro robot puede moverse hacia el este o hacia el oeste entre las posiciones 0 y 2. Las posiciones 2 y 3 están separadas por un puente.

Puente

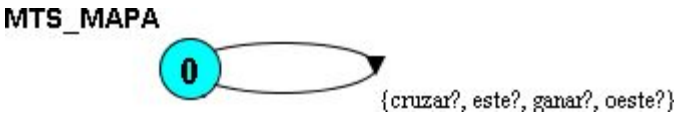
```
PUENTE = PUENTE_CERRADO,
PUENTE_CERRADO = (abrir -> PUENTE_ABIERTO
                 | cerrar -> PUENTE_CERRADO),
PUENTE_ABIERTO  = (abrir -> PUENTE_ABIERTO
                 | cruzar -> PUENTE_ABIERTO |
                 cerrar -> PUENTE_CERRADO).
```

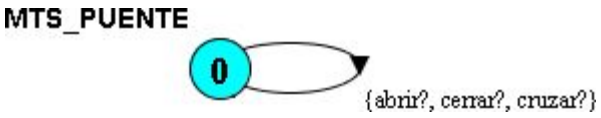


El puente comienza cerrado. Si se abre, se lo puede cruzar.

Modelos

```
MTS_MAPA = (este? -> MTS_MAPA | oeste? -> MTS_MAPA
           | cruzar? -> MTS_MAPA | ganar? -> MTS_MAPA).
MTS_PUENTE = (abrir? -> MTS_PUENTE | cerrar? -> MTS_PUENTE | cruzar? -> MTS_PUENTE).
```





Nuestros modelos iniciales sólo nos dicen qué acciones pueden estar disponibles en cada estado.

Objetivo

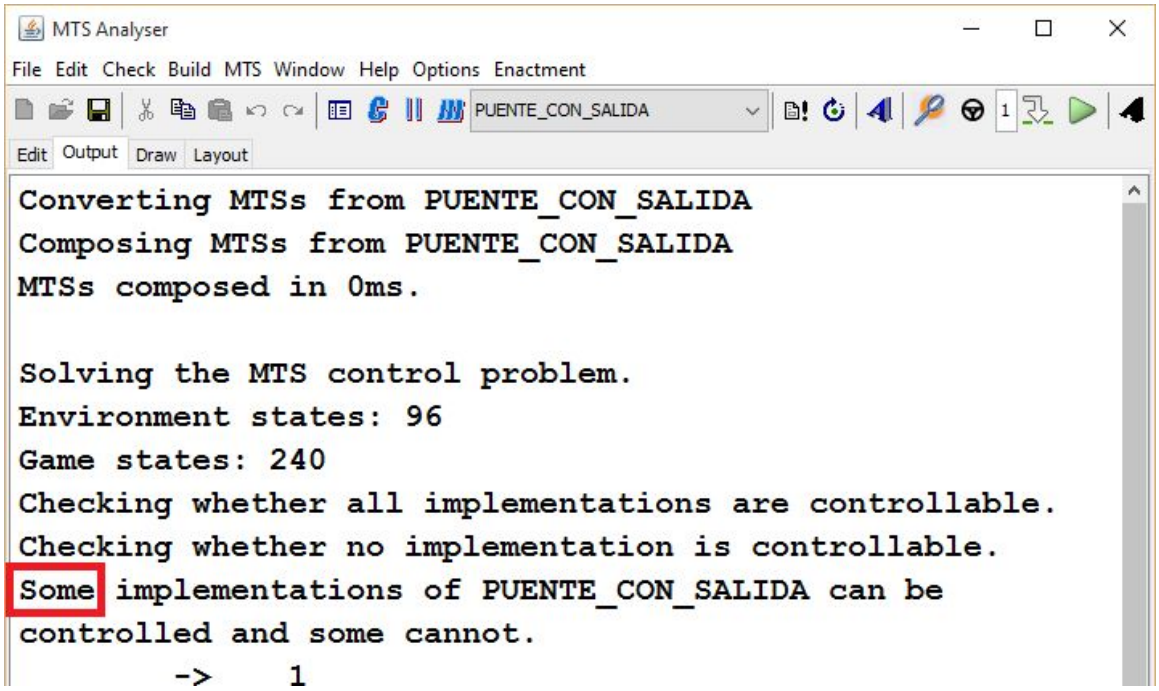
```
set Controllable_puente = {este, oeste, cruzar, ganar}

fluent F_Ganar = <ganar, Controllable_puente\{ganar}>
assert A_Ganar = F_Ganar

fluent F_PuedeCruzar = <abrir, cruzar> assert
PuedeCruzar = F_PuedeCruzar

controllerSpec GOAL_PUENTE = {
    assumption = {PuedeCruzar}
    liveness = {A_Ganar}
    controllable = {Controllable_puente}
}
```

Nuestro objetivo es ejecutar la acción ganar.
Asumimos que en algún momento se va a ejecutar la acción abrir.
Esto implica que el puente no va a quedar cerrado para siempre.
Para ganar, necesitamos movernos 3 veces hacia el este, pero la última vez hay que pasar por el puente.

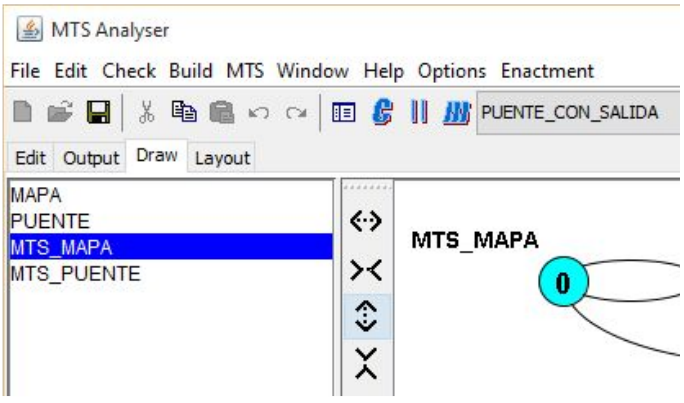
Estrategia y comportamiento del ambiente

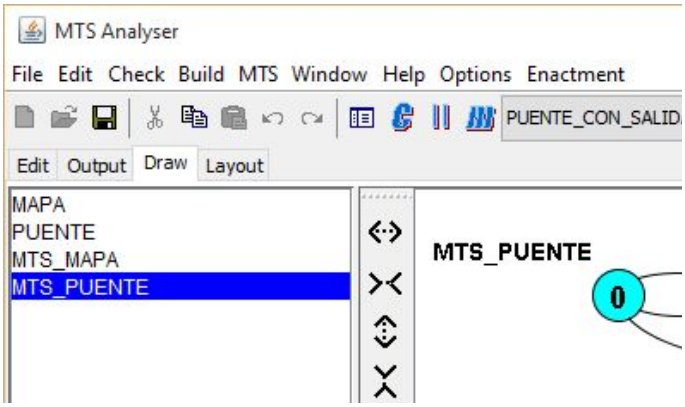
Nuestra estrategia consiste en ejecutar siempre acciones que sean nuevas para el estado en el cual nos encontramos.
Si desde el estado actual ya ejecutamos todas las acciones disponibles, elegimos una acción que nos lleve a un estado no explorado o esperamos a que el ambiente cambie y así nos permita seguir explorando.
El comportamiento del puente será cíclico, permaneciendo cerrado 11 turnos y abierto durante 3.

Exploración



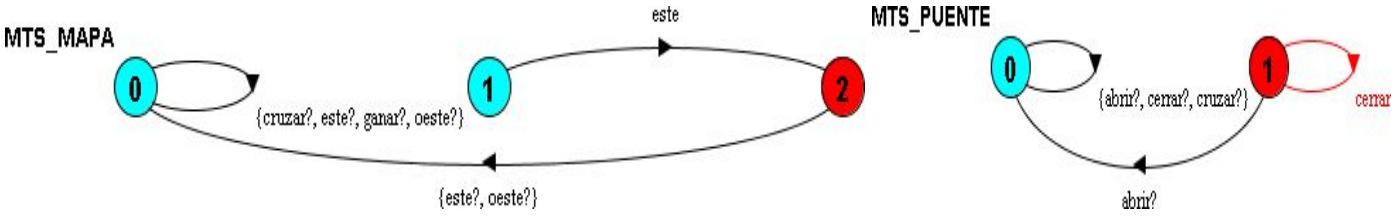
En un principio no tenemos la información suficiente para razonar sobre el objetivo.
Lo que sabemos sobre el robot es que, desde su estado actual, puede ejecutar la acción este. No sabemos a qué estado llegamos si ejecutamos dicha acción.
Sobre el puente sabemos que, desde su estado actual, podría tener disponibles las acciones abrir y cerrar y cruzar.





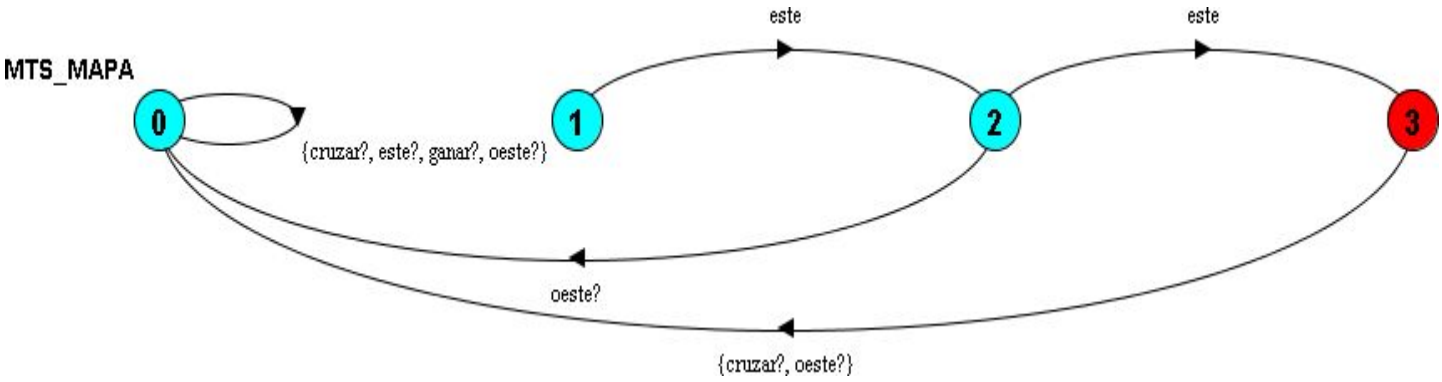
```
        ->    1
este    ->    2
cerrar
```

La estrategia elige la acción este. Al ejecutarla aprendemos que desde el estado inicial, por este, llegamos a un estado desde el cual podemos ejecutar las acciones este y oeste, sin saber a que estados nos llevan dichas acciones. El puente sigue cerrado. Aprendemos que desde el estado inicial del puente, por cerrar, llega al mismo estado.



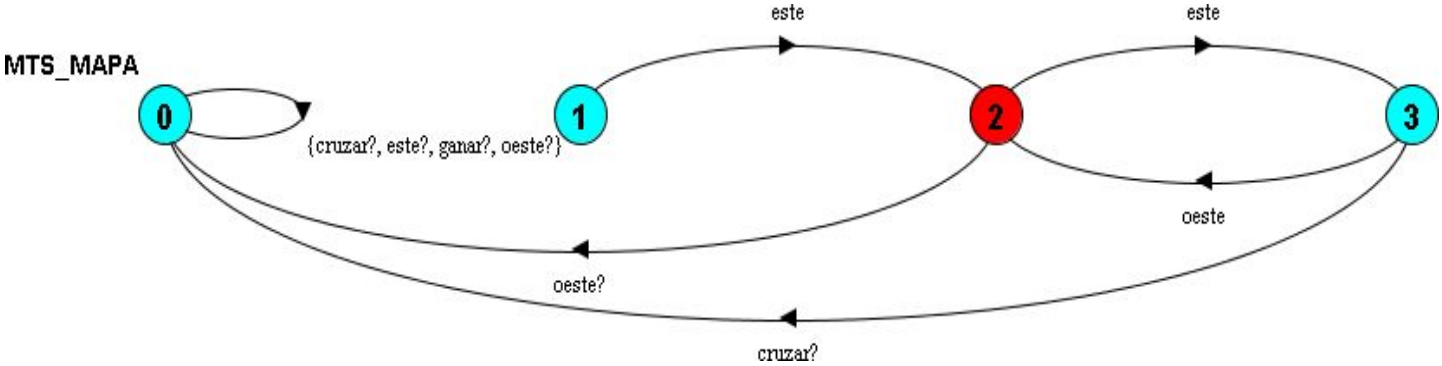

```
este    ->    3
cerrar
```

Ahora la estrategia tiene disponibles para elegir las acciones este y oeste. Ambas son acciones no ejecutadas desde el estado actual. Elige la acción este. Al ejecutarla llegamos a un nuevo estado desde el cual podemos ejecutar las acciones cruzar y oeste.
El puente sigue cerrado. Como ya había ejecutado la acción cerrar desde su estado actual no aprendemos nada nuevo.



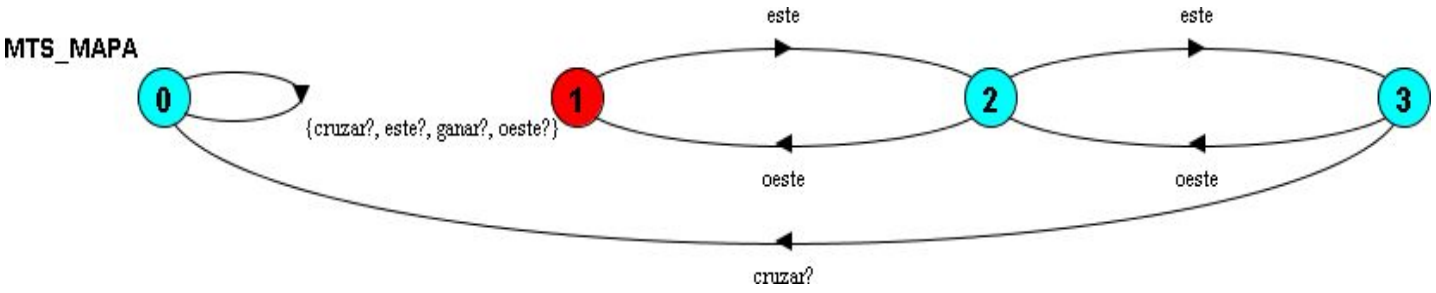

```
oeste   ->    2
cerrar
```

Cruzar es una acción compartida. Como no puede ser ejecutada desde el estado actual del puente, no está disponible.
Aprendemos que por oeste, desde el estado 3 volvemos al estado 2.
El puente sigue cerrado.



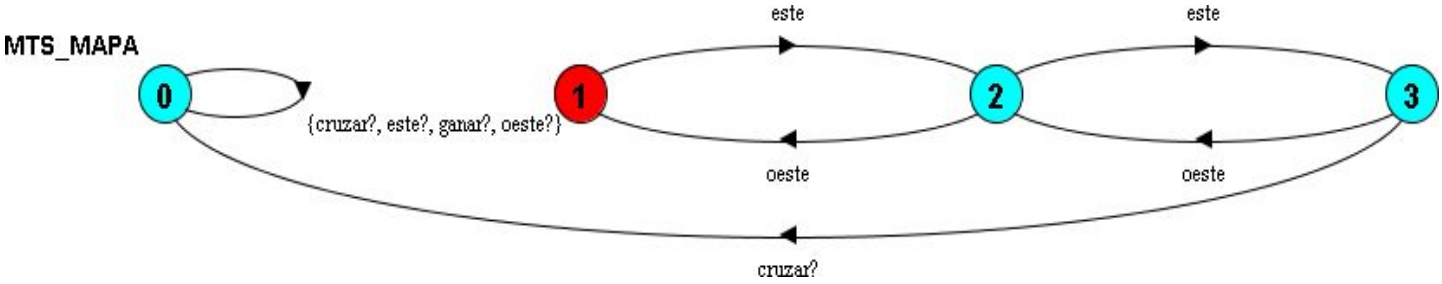
```
oeste    ->    1
cerrar
```

Las acciones este y oeste están disponibles. Como desde el estado actual ya habíamos ejecutado este, la estrategia elige oeste. Aprendemos que por oeste, desde el estado 2 volvemos al estado 1. El puente sigue cerrado.



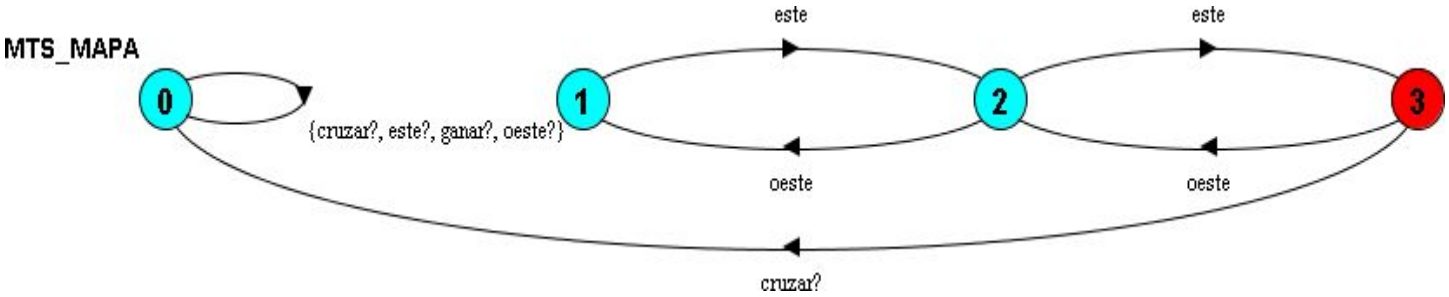

```
oeste    ->    1
cerrar
```

Las acciones este y oeste están disponibles. Como desde el estado actual ya habíamos ejecutado este, la estrategia elige oeste. Aprendemos que por oeste, desde el estado 2 volvemos al estado 1. El puente sigue cerrado.




```
este     ->    2
cerrar
este     ->    3
cerrar
```

Ahora no tenemos disponibles acciones que no hayan sido ejecutadas anteriormente desde el estado actual. La estrategia busca llegar a un estado que tenga acciones no ejecutadas, por eso ejecuta dos veces este para llegar al estado 3. El puente sigue cerrado.

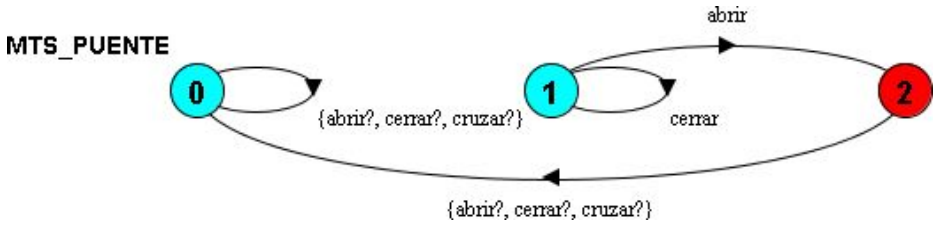


```
WAIT    ->    3
cerrar
```

La única acción que nos permite seguir adquiriendo conocimiento, cruzar, es una acción compartida. No podemos ejecutarla porque el puente no permite ejecutarla desde su estado actual, así que esperamos a que el puente cambie. El modelo de conocimiento del robot no cambia, ya que no se ejecutó ninguna acción. El puente sigue cerrado.

```
WAIT    ->    3
abrir
```

Luego de esperar durante 6 turnos, el puente se abre. Aprendemos que el puente, desde su estado inicial, por abrir llega a un estado que podría tener disponibles las acciones abrir y cerrar y cruzar.




```
cruzar  ->    4
abrir
cruzar  ->    3
abrir
cruzar  ->    4
cerrar
ganar   ->    4
cerrar
All implementations can be controlled
```

Cuando finalmente el puente se abre, el robot puede cruzar.
El robot logra llegar a un estado desde el cual se puede ejecutar la acción ganar.
La información obtenida es suficiente para garantizar el cumplimiento del objetivo.

Bibliografía

[1] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Saar. Synthesis of reactive(1) designs. *J. Comput. Syst. Sci.*, 78(3):911–938, May 2012.

[2] Nicolás D'ippolito, Victor Braberman, Nir Piterman, and Sebastián Uchitel. The modal transition system control problem. In *18th International Symposium on Formal Methods FM 2012*, volume 7436 of *Lecture Notes in Computer Science*, pages 155–170. Springer Verlag, 2012. FM 2012 - Paris, Francia, Agosto 2012.

[3] Nicolás D'ippolito, Victor Braberman, Nir Piterman, and Sebastián Uchitel. Synthesizing nonanomalous event-based controllers for liveness goals. *ACM Trans. Softw. Eng. Methodol.*, 22(1):9:1–9:36, March 2013.

[4] Dimitra Giannakopoulou and Jeff Magee. Fluent model checking for event-based systems. *SIGSOFT Softw. Eng. Notes*, 28(5):257–266, September 2003.

[5] C. M. Gifford, R. Webb, J. Bley, D. Leung, M. Calnon, J. Makarewicz, B. Banz, and A. Agah. Low-cost multi-robot exploration and mapping. In *2008 IEEE International Conference on Technologies for Practical Robot Applications*, pages 74–79, Nov 2008.

[6] Michael Jackson. The world and the machine. In *Proceedings of the 17th International Conference on Software Engineering*, ICSE '95, pages 283–292, New York, NY, USA, 1995. ACM.

[7] Robert M. Keller. Formal verification of parallel programs. *Commun. ACM*, 19(7):371–384, July 1976.

[8] Benjamin Kuipers and Yung-Tai Byun. Toward learning robots. In Walter Van de Velde, editor, *Toward Learning Robots*, chapter A Robot Exploration and Mapping Strategy Based on a Semantic Hierarchy of Spatial Representations, pages 47–63. MIT Press, Cambridge, MA, USA, 1993.

[9] K. G. Larsen and B. Thomsen. A modal process logic. In *Logic in Computer Science, 1988. LICS '88., Proceedings of the Third Annual Symposium on*, pages 203–210, 1988.

[10] Jeff Magee and Jeff Kramer. *Concurrency: State Models &Amp; Java Programs*. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[11] Nik A Melchior and Reid Simmons. Particle rrt for path planning with uncertainty. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1617–1624. IEEE, 2007.

[12] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '89, pages 179–190, New York, NY, USA, 1989. ACM.

[13] Jun Tani and Jun Yamamoto. On the dynamics of robot exploration learning. *Cogn. Syst. Res.*, 3(3):459–470, September 2002.

[14] S. Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

[15] Axel van Lamsweerde and Emmanuel Letier. Handling obstacles in goal-oriented requirements engineering. *IEEE Trans. Softw. Eng.*, 26(10):978–1005, October 2000.

[16] H. Wang, M. Jenkin, and P. Dymond. Enhancing exploration in graph-like worlds. In *Computer and Robot Vision, 2008. CRV '08. Canadian Conference on*, pages 53–60, May 2008.

[17] Wenhua Yang, Chang Xu, Yepang Liu, Chun Cao, Xiaoxing Ma, and Jian Lu. Verifying self-adaptive applications suffering uncertainty. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, pages 199–210, New York, NY, USA, 2014. ACM.