Algoritmo de verificación para LTL: Complejidad

¿Cuál es el orden del algoritmo anterior?

- Notación: $\|\mathcal{A}\|$ hace referencias al tamaño total de un autómata $\mathcal{A}$.

Antes de analizar el algoritmo necesitamos estudiar la complejidad del siguiente problema:

- Dado un autómata de Büchi $\mathcal{A}$, ¿cuánto tiempo toma determinar si $L_\omega(\mathcal{A}) = \emptyset$?

Algoritmo de verificación para LTL: Complejidad

¿Cuál es el orden del algoritmo anterior?

- Notación: $\|\mathcal{A}\|$ hace referencias al tamaño total de un autómata $\mathcal{A}$.

Antes de analizar el algoritmo necesitamos estudiar la complejidad del siguiente problema:

- Dado un autómata de Büchi $\mathcal{A}$, ¿cuánto tiempo toma determinar si $L_\omega(\mathcal{A}) = \emptyset$?

Respuesta: $O(\|\mathcal{A}\|)$.

Algoritmo de verificación para LTL: Complejidad

Ahora podemos estudiar la complejidad del algoritmo para verificar si $(\mathcal{M}, e) \models \mathbf{A}\varphi$:

Algoritmo de verificación para LTL: Complejidad

Ahora podemos estudiar la complejidad del algoritmo para verificar si $(\mathcal{M}, e) \models \mathbf{A}\varphi$:

- $\|\mathcal{A}_{\mathcal{M},e}\|$ es de tamaño $O(\|\mathcal{M}\|)$. ¿Por qué?

Algoritmo de verificación para LTL: Complejidad

Ahora podemos estudiar la complejidad del algoritmo para verificar si $(\mathcal{M}, e) \models \mathbf{A}\varphi$:

- $\|\mathcal{A}_{\mathcal{M},e}\|$ es de tamaño $O(\|\mathcal{M}\|)$. ¿Por qué?
- $\|\mathcal{A}_{\neg\varphi}\|$ es de tamaño $2^{O(\|\varphi\|)}$.

Algoritmo de verificación para LTL: Complejidad

Ahora podemos estudiar la complejidad del algoritmo para verificar si $(\mathcal{M}, e) \models \mathbf{A}\varphi$:

- $\|\mathcal{A}_{\mathcal{M},e}\|$ es de tamaño $O(\|\mathcal{M}\|)$. ¿Por qué?
- $\|\mathcal{A}_{\neg\varphi}\|$ es de tamaño $2^{O(\|\varphi\|)}$.
- $\|\mathcal{A}_{\mathcal{M},e} \times \mathcal{A}_{\neg\varphi}\|$ es de tamaño $\|\mathcal{M}\| \cdot 2^{O(\|\varphi\|)}$. ¿Por qué?

Algoritmo de verificación para LTL: Complejidad

Ahora podemos estudiar la complejidad del algoritmo para verificar si $(\mathcal{M}, e) \models \mathbf{A}\varphi$:

- $\|\mathcal{A}_{\mathcal{M},e}\|$ es de tamaño $O(\|\mathcal{M}\|)$. ¿Por qué?
- $\|\mathcal{A}_{\neg\varphi}\|$ es de tamaño $2^{O(\|\varphi\|)}$.
- $\|\mathcal{A}_{\mathcal{M},e} \times \mathcal{A}_{\neg\varphi}\|$ es de tamaño $\|\mathcal{M}\| \cdot 2^{O(\|\varphi\|)}$. ¿Por qué?
- Verificar si $L_\omega(\mathcal{A}_{\mathcal{M},e} \times \mathcal{A}_{\neg\varphi}) = \emptyset$ toma tiempo $O(\|\mathcal{A}_{\mathcal{M},e} \times \mathcal{A}_{\neg\varphi}\|)$.

Algoritmo de verificación para LTL: Complejidad

Ahora podemos estudiar la complejidad del algoritmo para verificar si $(\mathcal{M}, e) \models \mathbf{A}\varphi$:

- $\|\mathcal{A}_{\mathcal{M},e}\|$ es de tamaño $O(\|\mathcal{M}\|)$. ¿Por qué?
- $\|\mathcal{A}_{\neg\varphi}\|$ es de tamaño $2^{O(\|\varphi\|)}$.
- $\|\mathcal{A}_{\mathcal{M},e} \times \mathcal{A}_{\neg\varphi}\|$ es de tamaño $\|\mathcal{M}\| \cdot 2^{O(\|\varphi\|)}$. ¿Por qué?
- Verificar si $L_\omega(\mathcal{A}_{\mathcal{M},e} \times \mathcal{A}_{\neg\varphi}) = \emptyset$ toma tiempo $O(\|\mathcal{A}_{\mathcal{M},e} \times \mathcal{A}_{\neg\varphi}\|)$.

Tiempo total: $\|\mathcal{M}\| \cdot 2^{O(\|\varphi\|)}$.

Verificación de fórmulas existenciales

El algoritmo que mostramos también puede ser usado para verificar si $(\mathcal{M}, e) \models \mathbf{E}\varphi$.

- Basta cambiar $\mathbf{E}\varphi$ por la fórmula equivalente $\neg\mathbf{A}\neg\varphi$.

El resultado es el siguiente algoritmo que verifica si $(\mathcal{M}, e) \models \mathbf{E}\varphi$:

Verificación de fórmulas existenciales

El algoritmo que mostramos también puede ser usado para verificar si $(\mathcal{M}, e) \models \mathbf{E}\varphi$.

- Basta cambiar $\mathbf{E}\varphi$ por la fórmula equivalente $\neg\mathbf{A}\neg\varphi$.

El resultado es el siguiente algoritmo que verifica si $(\mathcal{M}, e) \models \mathbf{E}\varphi$:

1. Construya el autómata $\mathcal{A}_{\mathcal{M},e}$;

Verificación de fórmulas existenciales

El algoritmo que mostramos también puede ser usado para verificar si $(\mathcal{M}, e) \models \mathbf{E}\varphi$.

- Basta cambiar $\mathbf{E}\varphi$ por la fórmula equivalente $\neg\mathbf{A}\neg\varphi$.

El resultado es el siguiente algoritmo que verifica si $(\mathcal{M}, e) \models \mathbf{E}\varphi$:

1. Construya el autómata $\mathcal{A}_{\mathcal{M},e}$;
2. Construya el autómata $\mathcal{A}_{\varphi}$;

Verificación de fórmulas existenciales

El algoritmo que mostramos también puede ser usado para verificar si $(\mathcal{M}, e) \models \mathbf{E}\varphi$.

- Basta cambiar $\mathbf{E}\varphi$ por la fórmula equivalente $\neg\mathbf{A}\neg\varphi$.

El resultado es el siguiente algoritmo que verifica si $(\mathcal{M}, e) \models \mathbf{E}\varphi$:

1. Construya el autómata $\mathcal{A}_{\mathcal{M},e}$;
2. Construya el autómata $\mathcal{A}_{\varphi}$;
3. Construya el autómata $\mathcal{A}_{\mathcal{M},e} \times \mathcal{A}_{\varphi}$ que acepta el lenguaje $L_\omega(\mathcal{A}_{\mathcal{M},e}) \cap L_\omega(\mathcal{A}_{\varphi})$;

Verificación de fórmulas existenciales

El algoritmo que mostramos también puede ser usado para verificar si $(\mathcal{M}, e) \models \mathbf{E}\varphi$.

- Basta cambiar $\mathbf{E}\varphi$ por la fórmula equivalente $\neg\mathbf{A}\neg\varphi$.

El resultado es el siguiente algoritmo que verifica si $(\mathcal{M}, e) \models \mathbf{E}\varphi$:

1. Construya el autómata $\mathcal{A}_{\mathcal{M},e}$;
2. Construya el autómata $\mathcal{A}_{\varphi}$;
3. Construya el autómata $\mathcal{A}_{\mathcal{M},e} \times \mathcal{A}_{\varphi}$ que acepta el lenguaje $L_\omega(\mathcal{A}_{\mathcal{M},e}) \cap L_\omega(\mathcal{A}_{\varphi})$;
4. Verifique si $L_\omega(\mathcal{A}_{\mathcal{M},e} \times \mathcal{A}_{\varphi}) \neq \emptyset$
   - Si es así retorne sí, y en caso contrario retorne no.

Verificación de LTL: Complejidad

El algoritmo que mostramos es exponencial.

- Una pregunta natural es si podemos construir un algoritmo más eficiente.

Vamos a contestar dos preguntas:

- ¿Es posible construir autómatas de tamaño polinomial para LTL?
- ¿Es posible utilizar otro enfoque (no necesariamente basado en autómatas) para verificar fórmulas en LTL en tiempo polinomial?

Verificación de LTL basada en autómatas

Primero vamos a mostrar que el enfoque basado en autómatas esencialmente no puede ser mejorado:

Teorema

Existe una clase de fórmulas $\{\varphi_n\}_{n\geq 1}$ *en LTL tal que:*

- φ_n *está definida sobre el alfabeto* $\Sigma_n = \{a_0, a_1, \ldots, a_n\}$;
- *para cada autómata de Büchi* $\mathcal{A}$ *que acepta exactamente los caminos* π *con alfabeto* 2^{Σ_n} *que satisfacen* φ_n, *se tiene que el número de estados de* $\mathcal{A}$ *es*

$$2^{\Omega(\frac{\|\varphi_n\|}{\log \|\varphi_n\|})}.$$

Enfoque basado en autómatas: Casos complicados

Vamos a demostrar el teorema.

- Esto nos va a dar una idea de que fórmulas son difíciles de evaluar usando autómatas.

Para cada $n \geq 1$, queremos definir una fórmula LTL sobre $\Sigma_n = \{a_0, a_1, \ldots, a_n\}$ que represente el siguiente lenguaje L_n:

L_n está formado por los caminos π con alfabeto 2^{Σ_n} tales que: Si dos estados en π están de acuerdo en las proposiciones a_1, $\ldots$, a_n, entonces están de acuerdo en a_0.

Enfoque basado en autómatas: Casos complicados

φ_n es definida como $\alpha_n \wedge \beta_n$, donde:

Enfoque basado en autómatas: Casos complicados

φ_n es definida como $\alpha_n \wedge \beta_n$, donde:

$$\alpha_n = \bigwedge_{I \subseteq [1,n]} \left(\mathbf{F}\left[\left(\bigwedge_{j \in I} a_j \right) \wedge \left(\bigwedge_{j \in [1,n] \setminus I} \neg a_j \right) \wedge a_0 \right] \rightarrow \mathbf{G}\left[\left(\left(\bigwedge_{j \in I} a_j \right) \wedge \left(\bigwedge_{j \in [1,n] \setminus I} \neg a_j \right) \right) \rightarrow a_0 \right] \right).$$

Enfoque basado en autómatas: Casos complicados

φ_n es definida como $\alpha_n \wedge \beta_n$, donde:

$$\alpha_n = \bigwedge_{I \subseteq [1,n]} \left(\mathbf{F}\left[\left(\bigwedge_{j \in I} a_j \right) \wedge \left(\bigwedge_{j \in [1,n] \setminus I} \neg a_j \right) \wedge a_0 \right] \rightarrow \mathbf{G}\left[\left(\left(\bigwedge_{j \in I} a_j \right) \wedge \left(\bigwedge_{j \in [1,n] \setminus I} \neg a_j \right) \right) \rightarrow a_0 \right] \right).$$

$$\beta_n = \bigwedge_{I \subseteq [1,n]} \left(\mathbf{F}\left[\left(\bigwedge_{j \in I} a_j \right) \wedge \left(\bigwedge_{j \in [1,n] \setminus I} \neg a_j \right) \wedge \neg a_0 \right] \rightarrow \mathbf{G}\left[\left(\left(\bigwedge_{j \in I} a_j \right) \wedge \left(\bigwedge_{j \in [1,n] \setminus I} \neg a_j \right) \right) \rightarrow \neg a_0 \right] \right).$$

Enfoque basado en autómatas: Casos complicados

φ_n es definida como $\alpha_n \wedge \beta_n$, donde:

$$\alpha_n = \bigwedge_{I \subseteq [1,n]} \left(\mathbf{F}\left[\left(\bigwedge_{j \in I} a_j \right) \wedge \left(\bigwedge_{j \in [1,n] \setminus I} \neg a_j \right) \wedge a_0 \right] \rightarrow \mathbf{G}\left[\left(\left(\bigwedge_{j \in I} a_j \right) \wedge \left(\bigwedge_{j \in [1,n] \setminus I} \neg a_j \right) \right) \rightarrow a_0 \right] \right).$$

$$\beta_n = \bigwedge_{I \subseteq [1,n]} \left(\mathbf{F}\left[\left(\bigwedge_{j \in I} a_j \right) \wedge \left(\bigwedge_{j \in [1,n] \setminus I} \neg a_j \right) \wedge \neg a_0 \right] \rightarrow \mathbf{G}\left[\left(\left(\bigwedge_{j \in I} a_j \right) \wedge \left(\bigwedge_{j \in [1,n] \setminus I} \neg a_j \right) \right) \rightarrow \neg a_0 \right] \right).$$

Tenemos que: $\|\varphi_n\|$ es $\Theta(n \cdot 2^n)$.

Enfoque basado en autómatas: Casos complicados

El siguiente lema es fundamental para la demostración:

Lema

Todo autómata de Büchi $\mathcal{A}$ tal que $L_\omega(\mathcal{A}) = L_n$ tiene al menos 2^{2^n} estados.

Demostración: Sea $\mathcal{A}$ un autómata de Büchi tal que $L_\omega(\mathcal{A}) = L_n$.

Definimos $\{b_0, \ldots, b_{2^n-1}\}$ como el conjunto potencia de $\{a_1, \ldots, a_n\}$.

Para cada $K \subseteq \{0, \ldots, 2^n - 1\}$, definimos una palabra finita $s_K = \sigma_0 \ldots \sigma_{2^n-1}$ de la siguiente forma:

$$\sigma_i = \begin{cases} b_i & i \in K \\ b_i \cup \{a_0\} & i \notin K \end{cases}$$

Enfoque basado en autómatas: Casos complicados

Para cada $K \subseteq \{0, \ldots, 2^n - 1\}$ se tiene que $(s_K)^\omega \in L_n$.

- Sea ρ_K una ejecución de $\mathcal{A}$ sobre $(s_K)^\omega$ que es aceptada, y sea $q_K = \rho_K(2^n)$.

Para todo $K, K' \subseteq \{0, \ldots, 2^n - 1\}$ tal que $K \neq K'$, se tiene que $s_{K'}(s_K)^\omega \notin L_n$.

- De esto se concluye que $q_K \neq q_{K'}$. ¿Por qué?

Por lo tanto: $\mathcal{A}$ tiene al menos 2^{2^n} estados ya que hay 2^{2^n} subconjuntos de $\{0, \ldots, 2^n - 1\}$.

Enfoque basado en autómatas: Casos complicados

Sabemos que $\|\varphi_n\|$ es $\Theta(n \cdot 2^n)$.

De esto y el lema anterior, se concluye que el número de estados de cualquier autómata de Büchi que representa a φ_n es:

$$2^{\Omega(\frac{\|\varphi_n\|}{\log \|\varphi_n\|})}.$$

¿Cómo se deduce esto?

Verificación de LTL: Complejidad

Que el enfoque basado en autómatas no pueda ser mejorado **no** implica que no pueda haber otro enfoque más eficiente para la verificación de LTL.

Ahora la teoría de complejidad nos va a dar una mano.

- Vamos a *demostrar* que no existe un enfoque eficiente para la verificación de LTL.

Verificación de LTL: Complejidad

Teorema

Tanto el problema de verificar si $(\mathcal{M}, e) \models \mathbf{A}\varphi$ *como el de verificar si* $(\mathcal{M}, e) \models \mathbf{E}\varphi$ *son NP-hard.*

Vamos a demostrar el teorema.

- Esto nos va a dar una idea de que fórmulas LTL son difíciles de evaluar en general.

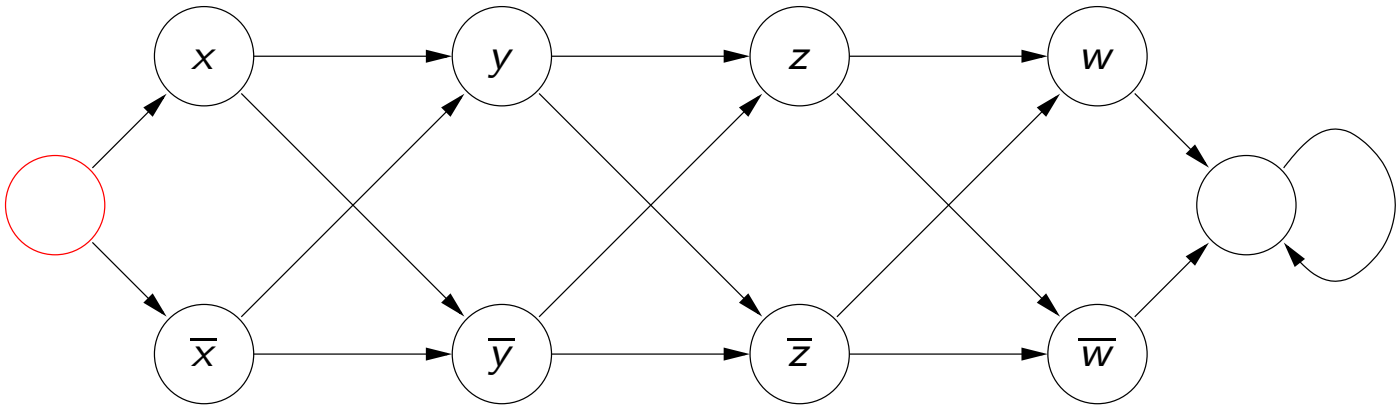
Verificación de LTL: Demostración de NP-hardness

Vamos a reducir SAT al problema de verificar si $(\mathcal{M}, e) \models \mathbf{E}\varphi$.

Vamos a ver como funciona la reducción para la fórmula proposicional $\alpha = (x \vee y \vee \neg z) \wedge (\neg x \vee z \vee w)$.

- Es fácil generalizar la reducción.

En la reducción se usa el siguiente sistema de transición $\mathcal{M}$:



Verificación de LTL: Demostración de NP-hardness

En el sistema de transición $\mathcal{M}$:

- llamamos e al estado de color rojo,
- $\overline{x}$ es usado para representar $\neg x$ (lo mismo sucede para $\overline{y}$, $\overline{z}$, $\overline{w}$).

La fórmula LTL φ es definida de la siguiente forma:

$$(\mathbf{F}x \vee \mathbf{F}y \vee \mathbf{F}\overline{z}) \wedge (\mathbf{F}\overline{x} \vee \mathbf{F}z \vee \mathbf{F}w).$$

Se tiene que: α es satisfacible si y sólo si $(\mathcal{M}, e) \models \mathbf{E}\varphi$.