Algoritmo de Verificación para CTL
IIC3800

Pablo Barceló

¿Qué es el model checking de CTL?

Recordemos que CTL es la lógica dada por:

$$\phi, \phi' \ := \ a \ \mid \ \neg\phi \ \mid \ \phi \vee \phi' \ \mid \ \mathbf{EX}\phi \ \mid \ \mathbf{EG}\phi \ \mid \ \mathbf{E}(\phi \mathbf{U} \phi')$$

Queremos un algoritmo que verifique fórmulas de CTL sobre sistemas de transición.

Es decir, un algoritmo que dada una fórmula ϕ en CTL y un sistema de transición $\mathcal{M}$, compute el conjunto

$$\{e \mid (\mathcal{M}, e) \models \phi\}$$

Intuición sobre el algoritmo

El algoritmo funciona coloreando cada estado e con el conjunto $label(e)$ de subformulas de ϕ que son ciertas en e.

Inicialmente $label(e)$ es el conjunto de los $a \in \Sigma$ tal que ϕ menciona a la proposición a y $e \in P_a$.

Durante el paso i, se consideran las subfórmulas de ϕ con $i-1$ operadores anidados.

Una vez que una subfórmula es procesada, se agrega a $label(e)$ para cada estado e que la satisface.

Operadores anidados en una fórmula

Formalmente, el número de operadores anidados ($\#_o$) de una fórmula ϕ en CTL se define como sigue:

- Si $\phi = a$ entonces $\#_o(\phi) = 0$.
- Si $\phi = \neg\psi$ entonces $\#_o(\phi) = \#_o(\psi)$.
- Si $\phi = \psi \vee \psi'$ entonces $\#_o(\phi) = \max\{\#_o(\psi), \#_o(\psi')\}$.
- Si $\phi = \mathbf{EX}\psi$ entonces $\#_o(\phi) = \#_o(\psi) + 1$.
- Si $\phi = \mathbf{EG}\psi$ entonces $\#_o(\phi) = \#_o(\psi) + 1$.
- Si $\phi = \mathbf{E}(\psi \mathbf{U} \psi')$ entonces $\#_o(\phi) = \max\{\#_o(\psi), \#_o(\psi')\} + 1$.

Subfórmulas

Definimos el conjunto de subfórmulas (sf) de ϕ como sigue:

- Si $\phi = a$ entonces $\mathrm{sf}(\phi) = a$.
- Si $\phi = \neg\psi$ entonces $\mathrm{sf}(\phi) = \mathrm{sf}(\psi) \cup \{\neg\phi\}$.
- Si $\phi = \psi \vee \psi'$ entonces $\mathrm{sf}(\phi) = \mathrm{sf}(\psi) \cup \mathrm{sf}(\psi') \cup \{\psi \vee \psi'\}$.
- Si $\phi = \mathbf{EX}\psi$ entonces $\mathrm{sf}(\phi) = \mathrm{sf}(\psi) \cup \{\mathbf{EX}\psi\}$.
- Si $\phi = \mathbf{EG}\psi$ entonces $\mathrm{sf}(\phi) = \mathrm{sf}(\psi) \cup \{\mathbf{EG}\psi\}$.
- Si $\phi = \mathbf{E}(\psi \mathbf{U} \psi')$ entonces $\mathrm{sf}(\phi) = \mathrm{sf}(\psi) \cup \mathrm{sf}(\psi') \cup \{\mathbf{E}(\psi \mathbf{U} \psi')\}$.

Descripción del algoritmo

Sea ϕ una fórmula en CTL. El algoritmo trabaja en etapas $0 \ldots n$, donde $n = \#_o(\phi)$.

En etapa $i \in [0, n]$: Para cada subfórmula ψ de ϕ tal que $i = \#_o(\psi)$, y para cada e tal que $(\mathcal{M}, e) \models \psi$, se agrega ψ a $\mathit{label}(e)$.

El algoritmo finaliza en la etapa n de tal forma que:

$$\phi \in \mathit{label}(e) \iff (\mathcal{M}, e) \models \phi.$$

Descripción del algoritmo

La etapa 0 es trivial, ya que sólo consideramos fórmulas que son combinaciones Booleanas de proposiciones atómicas.

En la etapa i las fórmulas son combinaciones Booleanas de fórmulas de la forma $\mathbf{EX}\alpha$, $\mathbf{EG}\alpha$, y $\mathbf{E}(\alpha\mathbf{U}\alpha')$, donde $\#_o(\alpha), \#_o(\alpha') \leq i - 1$.

Por tanto, sólo debemos considerar estos tres casos por separado. (¿Por qué?)

Descripción del algoritmo: Caso **EX**α

Caso **EX**α: Intutivamente, debemos agregar esta fórmula a cada $\mathit{label}(e)$ para el cual existe un e' tal que $R(e, e')$ y $\alpha \in \mathit{label}(e')$. Utilizamos el siguiente algoritmo:

procedure CheckEXα
    $T := \{e \mid \alpha \in \mathit{label}(e)\}$;
    while $T \neq \emptyset$ **do**
        choose $e \in T$;
        $T := T \setminus \{e\}$;
        for all e' **such that** $R(e', e)$ **do**
            if **EX**$\alpha \notin \mathit{label}(e')$ **then**
                $\mathit{label}(e') := \mathit{label}(e') \cup \{\textbf{EX}\alpha\}$;
            end if;
        end for all;
    end while;
    return T;
end procedure

Descripción del algoritmo: Caso $\mathbf{E}(\alpha \mathbf{U} \alpha')$

Caso $\mathbf{E}(\alpha \mathbf{U} \alpha')$: Dados los estados que satisfacen α', buscamos hacia atrás todos aquellos estados desde los que hay un camino etiquetado en α que alcanza un estado etiquetado con α'.

Descripción del algoritmo: Caso $\mathbf{E}(\alpha \mathbf{U} \alpha')$

Utilizamos el siguiente algoritmo:

procedure $\mathrm{CheckE}\alpha\mathrm{U}\alpha'$
  $T := \{e \mid \alpha' \in \mathit{label}(e)\}$;
  for all $e \in T$ **do** $\mathit{label}(e) := \mathit{label}(e) \cup \{\mathbf{E}(\alpha \mathbf{U} \alpha')\}$;
  while $T \neq \emptyset$ **do**
    choose $e \in T$;
    $T := T \setminus \{e\}$;
    for all e' **such that** $R(e', e)$ **do**
      if $\mathbf{E}(\alpha \mathbf{U} \alpha') \notin \mathit{label}(e')$ **and** $\alpha \in \mathit{label}(e')$ **then**
        $\mathit{label}(e') := \mathit{label}(e') \cup \{\mathbf{E}(\alpha \mathbf{U} \alpha')\}$;
        $T = T \cup \{e'\}$;
      end if;
    end for all;
  end while
  return T;
end procedure

Descripción del algoritmo: Caso **EG**α

Caso **EG**α: Este caso es menos intuitivo, pero puede tratarse con el siguiente algoritmo:

procedure CheckEGα
  $T := \{e \mid \alpha \in \mathit{label}(e)\}$;
  for all $e \in T$ **do** $\mathit{label}(e) := \mathit{label}(e) \cup \{\mathbf{EG}\alpha\}$;
  while $T \neq \emptyset$ **do**
    choose $e \in T$;
    if there is no e' **such that** $R(e, e')$ **and** $\mathbf{EG}\alpha \in \mathit{label}(e')$ **then**
      $\mathit{label}(e) := \mathit{label}(e) \setminus \{\mathbf{EG}\alpha\}$;
      $T := T \setminus \{e\}$;
    end if;
  end while
  return T;
end procedure

Complejidad del algoritmo

Asuma que α y α' son fórmulas en CTL, y que ya hemos computado el conjunto $\mathit{label}(e)$, para cada estado e, de tal forma que:

$$\alpha \in \mathit{label}(e) \iff (\mathcal{M}, e) \models \alpha, \text{ y}$$
$$\alpha' \in \mathit{label}(e) \iff (\mathcal{M}, e) \models \alpha'.$$

¿En términos de $|\mathcal{M}| = (|E| + |R|)$, cuál es la complejidad del algoritmo que computa $\mathbf{EX}\alpha$ y $\mathbf{E}(\alpha \mathbf{U} \alpha')$?

¿Cuál es la complejidad del algoritmo que computa $\mathbf{EG}\alpha$? ¿Es esto un problema?

Algoritmo eficiente para $\mathbf{EG}\alpha$

Es fundamental que el algoritmo sea lineal en el tamaño de la estructura porque en la práctica los sistemas pueden ser extremadamente grandes.

¿Cómo podríamos mejorar el algoritmo para $\mathbf{EG}\alpha$? ¿Podemos obtener un algoritmo lineal en $|\mathcal{M}|$?

Vamos a tener que usar técnicas más avanzadas, en particular, del área de teoría de grafos.

Algoritmo eficiente para **EG**α

Una componente fuertemente conexa (CFC) de $\mathcal{M}$ es un subgrafo maximal $\mathcal{M}'$ de $\mathcal{M}$ tal que para cada e_1, e_2 en $\mathcal{M}'$ existe un camino de e_1 hasta e_2 completamente contenido en $\mathcal{M}'$.

Una CFC es no-trivial si contiene al menos un eje.

Sea $\mathcal{M}_\alpha$ obtenido desde $\mathcal{M}$ borrando todos los estados que no satisfacen α. Entonces,

Lemma

$(\mathcal{M}, e) \models \mathbf{EG}\alpha \Leftrightarrow$ *e está en* $\mathcal{M}_\alpha$ *y existe un camino en* $\mathcal{M}_\alpha$ *desde e a un nodo* e' *en una CFC no-trivial de* $\mathcal{M}_\alpha$.

Algoritmo eficiente para **EG**α

¿Cómo puede ocuparse este resultado para obtener un algoritmo lineal para **EG**α?

Obtenga las CFC no-triviales de $\mathcal{M}_\alpha$. Esto puede hacerse en tiempo $O(|\mathcal{M}|)$ (Tarjan).

Una vez que tenemos esto, encontramos los estados que pertenecen a las CFC de $\mathcal{M}_\alpha$, y usando el converso de R en $\mathcal{M}_\alpha$ determinamos los estados que pueden alcanzar estas CFCs.

Es fácil ver que el proceso es lineal.

Algoritmo eficiente para **EG**α

procedure ChecklinearEGα

$S' := \{e \mid \alpha \in \mathit{label}(e)\}$;
$SCC := \{C \mid C$ es una CFC no-trivial de $S'\}$;
$T := \bigcup_{C \in SCC} \{e \mid e \in C\}$;
for all $e \in T$ **do** $\mathit{label}(e) := \mathit{label}(e) \cup \{\mathbf{EG}\alpha\}$;
while $T \neq \emptyset$ **do**
choose $e \in T$;
$T := T \setminus \{e\}$;
for all $e' \in S'$ **such that** $R(e', e)$ **do**
if $\mathbf{EG}\alpha \notin \mathit{label}(e')$ **then**
$\mathit{label}(e') := \mathit{label}(e') \cup \{\mathbf{EG}\alpha\}$;
$T = T \cup \{e'\}$;
end if;
end for all;
end while
return T;

end procedure

Restricciones de honestidad (Fairness constraints)

En muchos casos, sólo interesa verificar la especificación a lo largo de caminos honestos.

- Ejemplo: En un protocolo de comunicación sólo nos interesan aquellos caminos en los cuales todo mensaje enviado es eventualmente recibido.

En CTL no es posible expresar este tipo de restricciones.

Para tratarlas debemos cambiar la semántica.

Caminos honestos

Un sistema de transición honesto $\mathcal{M}_F$ extiende un sistema de transición $\mathcal{M} = (E, R, (P_a)_{a \in \Sigma})$ con un conjunto $F = \{P_1, \ldots, P_k\}$, tal que para cada $i \leq k$, $P_i \subseteq E$.

Sea π un camino en $\mathcal{M}$. Defina

$$inf(\pi) = \{ e \mid \ e \text{ aparece infinitas veces en } \pi \}.$$

Decimos que π es honesto en $\mathcal{M}_F$ si para cada $i \leq k$, $P_i \cap \inf(\pi) \neq \emptyset$.

Semántica para restricciones de honestidad

Sea $\mathcal{M}_F$ un sistema de transición honesto y e un estado de $\mathcal{M}_F$. Luego,

- $(\mathcal{M}_F, e) \models_F a$ si y sólo si existe camino honesto desde e y $e \in P_a$.
- $(\mathcal{M}_F, e) \models_F \neg\phi$ si y sólo si $(\mathcal{M}_F, e) \not\models_F \phi$.
- $(\mathcal{M}_F, e) \models_F \phi \vee \phi'$ si y sólo si $(\mathcal{M}_F, e) \models_F \phi$ o $(\mathcal{M}_F, e) \models_F \phi'$.
- $(\mathcal{M}_F, e) \models_F \mathbf{E}\psi$ si y sólo si existe camino honesto π desde e tal que $(\mathcal{M}_F, \pi) \models_F \psi$.

Verificación de CTL con restricciones de honestidad

Podemos extender el algoritmo antes presentado para considerar restricciones de honestidad.

Para ello debemos primero tratar con el caso $\mathbf{EG}\phi$.

Realizamos una modificación del procedimiento que utilizaba las CFCs visto anteriormente.

Verificación con restricciones de honestidad: Caso $\mathbf{EG}\phi$

Una CFC de $\mathcal{M}_F = (E, R, (P_a)_{a\in\Sigma}, F)$ es una CFC de $(E, R, (P_a)_{a\in\Sigma})$.

Decimos que una CFC es honesta si para todo $i \leq k$,

$$\text{CFC} \cap P_i \neq \emptyset.$$

Sea $\mathcal{M}_\alpha$ el sistema de transición obtenido desde $(E, R, (P_a)_{a\in\Sigma})$ borrando todos los estado que no satisfacen ϕ. Entonces,

Lemma

$(\mathcal{M}_F, e) \models_F \mathbf{EG}\alpha \Leftrightarrow$ *e está en* $\mathcal{M}_\alpha$ *y existe un camino en* $\mathcal{M}_\alpha$ *desde e a un nodo* e' *en una CFC no-trivial y honesta de* $\mathcal{M}_\alpha$.

Verficación con restricciones de honestidad: Otros casos

Primero computamos el conjunto de todos los nodos tales que $(\mathcal{M}_F, e) \models_F \mathbf{EG}(a \vee \neg a)$. Para cada uno de éstos agregamos la nueva proposición `fair` a $\mathit{label}(e)$.

Luego, el algoritmo se realiza como el de model checking para CTL con las siguientes variaciones:

- Para chequear a, chequeamos $a \wedge \texttt{fair}$.
- Para chequear $\mathbf{EX}\phi$ basta chequear $\mathbf{EX}(\phi \wedge \texttt{fair})$.
- Para chequear $\mathbf{E}(\phi \mathbf{U} \phi')$ basta chequear $\mathbf{E}(\phi \mathbf{U}(\phi' \wedge \texttt{fair}))$.

La complejidad del algoritmo es $O(|\phi| \cdot (|E| + |R|) \cdot |F|)$.