Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

SAT-Solver

Un evaluador para modelos de lógica temporal

Autor:
Natalia Mellino

Diciembre 2021

Contenidos

1 Motivación **2**
1.1 ¿Por qué un DSL para Lógica Temporal? 2

2 Introducción **2**
2.1 Un repaso de CTL 2
2.1.1 Sintaxis y Semántica 2
2.1.2 El Algoritmo SAT para *model-checking* 3
2.2 Notación 4

3 Gramática **4**

4 Resolución **4**

5 Instalación y Uso **4**

6 Trabajo a Futuro **4**

7 Referencias **4**

1 Motivación

1.1 ¿Por qué un DSL para Lógica Temporal?

La principal motivación de la realización de este proyecto, no es más ni menos que poder proveer una herramienta que pueda ser útil al momento de aprender y poner en práctica los conocimientos adquiridos de la teoría de Lógica Temporal.

Al momento de realizar ejercicios o querer hacer alguna verificación sobre estos modelos lógicos, nos encontramos con que no siempre podemos realizar estas tareas de manera rápida y simple como deseamos. Es por ello, que mi intención es poder facilitar en la medida que sea posible a cualquier persona que se encuentre en alguna de las situaciones mencionadas anteriormente.

2 Introducción

2.1 Un repaso de CTL

El objetivo de esta sección no es dar una explicación exhaustiva de la teoría de lógica temporal, sino más bien, refrescar los conceptos que sean necesarios para un mayor entendimiento de cómo funciona este proyecto. Se asume que el lector posee un cierto entendimiento de los conceptos básicos de Lógica Temporal.

Si bien en la Lógica Temporal hay muchas ramas y tópicos, en este proyecto nos centramos en lo que se conoce como CTL, o más bien, Computation Tree Logic. Este es un tipo de lógica temporal en el que el se modela el tiempo como una estructura de árbol en donde el futuro no está determinado, sino que hay distintos caminos en el futuro donde cualquiera de ellos puede ser el camino que se vaya a realizar.

2.1.1 Sintaxis y Semántica

La **sintaxis** de CTL se puede definir de forma inductiva de la siguiente manera:

$$\phi ::= \top \mid \bot \mid p \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \forall \bigcirc \phi \mid \exists \bigcirc \phi \mid$$
$$\forall[\phi \cup \psi] \mid \exists[\phi \cup \psi] \mid \forall\Box\phi \mid \exists\Box\phi \mid \forall\Diamond\phi \mid \exists\Diamond\phi$$

Un **Modelo** o **Sistema de transiciones** está formado por una tupla: $(S, \rightarrow, I, L)$ donde:

- S es un conjunto finito de estados.
- I es un conjunto de estados iniciales ($I \subseteq S$).
- $\rightarrow \subseteq S \times S$ es una relación de transición entre estados
- $L : S \rightarrow \mathcal{P}(AT)$ una función de etiquetado, donde a cada estado se le asigna un conjunto de proposiciones atómicas.

Ahora recordemos cómo era la **semántica** de estos operadores. Definimos la relación $\vDash$ por inducción en ϕ:

- $M, s \nvDash \bot$
- $M, s \vDash p_i \iff p_i \in L(s)$
- $M, s \vDash \neg\phi \iff M, s \nvDash \phi$

- $M, s \vDash \phi \wedge \psi \iff M, s \vDash \phi$ y $M, s \vDash \psi$
- $M, s \vDash \forall \bigcirc \phi \iff$ para todo s' tal que $s \to s'$ se cumple que $M, s \vDash \phi$.
- $M, s \vDash \exists \bigcirc \phi \iff$ para algún s' tal que $s \to s'$ se cumple que $M, s \vDash \phi$.
- $M, s \vDash \forall[\phi \cup \psi] \iff$ para cada traza $s_0 \to s_1 \to ...$ con $s_0 = s$ existe $j \in \mathbb{N}$ tal que:
  - $M, s_j \vDash \psi$
  - $M, s_i \vDash \phi$ para todo $i < j$
- $M, s \vDash \exists[\phi \cup \psi] \iff$ para alguna traza $s_0 \to s_1 \to ...$ con $s_0 = s$ existe $j \in \mathbb{N}$ tal que:
  - $M, s_j \vDash \psi$
  - $M, s_i \vDash \phi$ para todo $i < j$

Luego tenemos los **operadores derivados**:

- $\forall\Diamond\phi = \forall[\top \cup \phi]$
- $\exists\Diamond\phi = \exists[\top \cup \phi]$
- $\forall\Box\phi = \neg\exists\Diamond\neg\phi$
- $\exists\Box\phi = \neg\forall\Diamond\neg\phi$

Para los operadores $\to, \top$ se omitió su semántica ya que las mismas se derivan a partir de los otros operadores ya existentes. Esto en realidad vale también para varios de los operadores que definimos, en las próximas secciones veremos como esto nos sirve para facilitar la implementación del evaluador.

2.1.2 El Algoritmo SAT para *model-checking*

Recordemos rápidamente de qué se trataba el algoritmo SAT: dado un modelo y una fórmula CTL, devuelve el conjunto de estados del modelo que satisface dicha fórmula. Esto es, en esencia, el objetivo de este proyecto. Para una fórmula y un modelo, nuestro algoritmo se comporta de la siguiente manera:

- $sat(\bot) = \emptyset$
- $sat(p_i) = \{s \in S \mid P_i \in L(s)\}$
- $sat(\neg\phi) = S - sat(\phi)$
- $sat(\phi \wedge \psi) = sat(\phi) \cap sat(\psi)$
- $sat(\phi \vee \psi) = sat(\phi) \cup sat(\psi)$
- $sat(\exists \bigcirc \phi) = pre_{\exists}(\phi)$
- $sat(\forall \bigcirc \phi) = pre_{\forall}(\phi)$
- $sat(\exists[\phi \cup \psi]) = exUntil(sat(\phi), sat(\psi))$
- $sat(\forall[\phi \cup \psi]) = forallUntil(sat(\phi), sat(\psi))$
- $sat(\forall\Diamond\phi) = inev(sat(\phi))$
- $sat(\exists\Diamond\phi) = exUntil(S, sat(\phi))$

Donde:

$preExists(Y) = \{s \in S \mid \exists s' : s \to s' \wedge s' \in Y\}$
$preAll(Y) = \{s \in S \mid \forall s' : s \to s', s' \in Y\}$

```
exUntil(X, Y):
  while (Y != Y u (X n preExists(Y))) do:
    Y <- Y u (X n preExists(Y))
  return Y

forallUntil(X, Y):
  while (Y != Y u (X n preAll(Y))) do:
    Y <- Y u (X n preAll(Y))
  return Y

inev(Y):
  while (Y != Y u preAll(Y)) do:
    Y <- Y u preAll(Y)
  return Y
```

2.2 Notación

Operador	Traducción
$\top$	TOP
$\bot$	BT
$\neg\phi$	$!\phi$
$\phi \wedge \psi$	ϕ & ψ
$\phi \vee \psi$	$\phi \mid \psi$
$\phi \to \psi$	ϕ -> ψ
$\forall \bigcirc \phi$	AX ϕ
$\exists \bigcirc \phi$	EX ϕ
$\forall[\phi \cup \psi]$	AU ϕ
$\exists[\phi \cup \psi]$	EU ϕ
$\forall\Box\phi$	AG ϕ
$\exists\Box\phi$	EG ϕ
$\forall\Diamond\phi$	AF ϕ
$\exists\Diamond\phi$	EF ϕ

3 Gramática

4 Resolución

5 Instalación y Uso

6 Trabajo a Futuro

7 Referencias