Universidad Nacional de Rosario

Facultad de ciencias exactas, ingeniería y agrimensura

Análisis de Lenguajes de Programación

Trabajo práctico Final

Alumno:
Trincheri Lucio

Febrero 2022

ALP - Informe EDSL ÇTL SAT-Solver"

Trincheri Lucio, LCC UNR FCEIA 28/02/2022

1. Introducción

La idea de este proyecto es presentar una herramienta "SAT-Solver" la cual dado un modelo de lógica temporal, específicamente uno de CTL (Computation tree logic), y diferentes formulas le permita al usuario conocer que estados del modelo satisfacen las fórmulas provistas. Esto sera útil para usuarios que deseen resolver ejercicios, verificar resultados o desarrollar sus propios problemas con sus respectivas soluciones, relacionados con CTL.

2. Compilación y uso

Para la compilación del programa, es necesario stack. Los pasos son:

- Clonar el repositorio: `https://github.com/LucioTrincheri/ALP-TPFinal`
- Ejecutar `stack setup`
- Dentro del directorio, ejecutar: `stack build`
- El programa se puede correr ejecutando: `stack exec SAT-exe "Nombre-de-archivo.ctl"`

3. Archivo argumento

El programa requiere un archivo como argumento, el cual contiene el modelo y las formulas a interpretar. A continuación se muestran los tipos de lineas que puede contener el archivo y ejemplos de las mismas. El archivo en cuestión deberá contener una o más de cada una de las siguientes lineas para su correcto funcionamiento:

3.1. Estados

Estas lineas indicarán los estados posibles del modelo. Siempre comenzaran con la palabra "STATE" al principio de la linea, seguida de una lista de estados. Un ejemplo de la notación es la siguiente:

STATES [s0, s1, s2, s3, s4, s5]

3.2. Transiciones

Estas lineas indicarán las transiciones entre estados del modelo. Siempre comenzaran con la palabra "TRANSITIONS" al principio de la linea, seguida de una lista de tuplas de estados. Un ejemplo de la notación es la siguiente:

TRANSITIONS [(s0, s1), (s0, s4), (s1, s2), (s2, s4), (s3, s2), (s3, s0)]

3.3. Valuaciones

Estas lineas indicarán las valuaciones de proposiciones del modelo. Es decir los estados en los cuales una valuación es verdadera. Siempre comenzaran con la palabra "VALUATIONS" al principio de la linea, seguida de una lista de tuplas de (valuación, estado). Un ejemplo de la notación es la siguiente:

VALUATIONS [(a, s1), (a, s2), (a, s3), (b, s2), (b, s4), (b, s5)]

3.4. Fórmulas

Estas lineas indicarán las formulas a evaluar con el modelo construido hasta el momento. Es cada formula solo utilizará el modelo construido hasta el punto que se lea la formula. Siempre comenzaran con la palabra "CTLEXP" al principio de la linea, seguida de una formula con operadores de lógica temporal y proposiciones. Un ejemplo de la notación es la siguiente:

CTLEXP AG AF a

La siguiente tabla contiene la equivalencia entre las operaciones de la lógica temporal y los operadores utilizados para representar las mismas en CTL. Los mismos fueron sacados de la representación dada en la página de wikipedia de CTL. En el ejemplo anterior, "a" es una proposición.

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

El orden de precedencia de los operadores es el siguiente, de menor a mayor precedencia. Los operadores dentro de la misma linea tienen la misma precedencia. Cabe aclarar que todos los operadores asocian a izquierda:

- &, |
- $->$
- El resto de operadores: !, AX, EX, AU, EU, AF, EF, AG, EG

4. Organización de archivos

A continuación se muestra el árbol del directorio. En las subsecciones más adelante se explicará la funcionalidad de cada uno de los archivos importantes.

```
.
├── app
│   └── Main.hs
├── src
│   ├── Common.hs
│   ├── Eval.hs
│   ├── Monads.hs
│   └── Parse.y
├── Ejemplos
│   ├── exampleStoplight.ctl
│   ├── testEvalErrorTransition.ctl
│   ├── testEvalErrorValuation.ctl
│   ├── testExit.ctl
│   ├── testParserErrorChar.ctl
│   ├── testParserErrorComm.ctl
│   ├── testParserErrorExp.ctl
│   ├── testParserErrorModel.ctl
│   └── testParserErrorModelMissing.ctl
├── README.md
├── Setup.hs
├── SAT.cabal
├── package.yaml
├── stack.yaml
└── stack.yaml.lock
```

4.1. Main.hs

El main es el encargado de, dado el archivo pasado como argumento, confirmar la presencia del mismo así como su correcta extensión. Una vez realizado esto, intenta parsear y luego evaluar cada una de las lineas del archivo. Ante un error en alguno de estos puntos, también realiza la impresión de mensajes de error por pantalla.

4.2. Common.hs

Aquí se definen todos los tipos de datos utilizados en los distintos archivos involucrados en el programa.

4.3. Eval.hs

El evaluador es la parte principal del proyecto, donde se implementa el algoritmo SAT-Solver en cuestión. Tomando de base el TP3, se hace uso de monadas para simplificar la implementación del estado y de los errores.

4.4. Monads.hs

Como explicado en la subsección del evaluador, se hizo uso de la monada "StateError" perteneciente al TP3, lo que facilita llevar el estado y el manejo de errores

4.5. Parse.y

El parser esta construido en base a la herramienta Happy, la cual es un generador de parsers para Haskell. Esta herramienta simplifico el desarrollo ya que el parser solo requiere pensar la gramática e implementarla en Happy, y el lexer fue fácil de construir al solo ser necesario representar como el usuario ingresa los datos.

4.6. Ejemplos

Esta carpeta incluye ejemplos tanto de modelos y formulas correctas (Ej exampleStoplight.ctl) así como de cada tipo de error posible en el programa (para probar mensajes de error),

5. Manejo de errores

El programa tiene un sistema de manejo de errores que se encarga de informar al usuario de los mismos para que los pueda solucionar. Estos errores se dividen en tres categorías.

5.1. Error de archivo

Los errores de este tipo informan al usuario de errores relacionados con el archivo argumento, tales como la ausencia del mismo o error de sufijo.

5.2. Error de parseo

Los errores de parseo indican errores dentro del archivo provisto. Estos errores pueden indicar caracteres inválidos en la formación de: comandos (Ej "STA<TES"), modelos (Ej "STATES [s1, s2, *]") o fórmulas (Ej "CTLEXP a & @b"). También puede informar de errores de mala formación de expresiones (pero con caracteres correctos).

5.3. Error de evaluación

El único error de evaluación posible se da en el caso que alguno de los estados pertenecientes a una transición o valuación ingresada por el usuario no exista. En este caso, el evaluador se encarga de informar al usuario de lo sucedido.

6. Bibliografía

Como bibliografía se utilizaron los apuntes provistos por la cátedra de ALP, página de wikipedia de CTL, los apuntes de la materia Lógica del 4to cuatrimestre y la documentación de Happy.