Métodos Formales con Pomela y Spin, VDM y TLA +

Jose David Barrita López[#1]

[#],*Universidad Tecnologica de la Mixteca*
Hujuapan de Leon, Oaxaca, Mexico
[1]davidbarrita.db@utm.gs.mx

***Abstract*— La presente investigación tiene como objetivo conocer los métodos formales como Pomela y Spin, VDM y TLA + donde se realice una comparativa entre ellos, y posteriormente seleccionar uno para realizar los siguientes ejemplos Contador, Registro, Directorio, *BirhdayBook* y entrega de tareas, actualmente los métodos formales son utilizados en empresas como Amazon, Microsoft, Tesla entre otros[1]. Se resalta la importancia de los métodos formales en la ingeniería de software y en específico en sistemas de altos impacto.**

***Keywords*— Metodos formales**

I. Introducción

Para abordar los métodos formales introduciendo el lenguaje Z, cuando se aborda la especificación se trata de dos partes importantes la herramienta que permita escribir la sintaxis y la comprobación por medio de teoría matemática tal caso pueden ser los teoremas aplicados sobre la teoría de conjuntos, dado que la mayoría de estos lenguajes tienen como fundamento la teoría de conjuntos, donde por medio del concepto de relación trabajan, claro ejemplo es el lenguaje *alloy.*A lo largo del ciclo de desarrollo de software hay dos partes medulares donde se aplican los métodos formales, la primera es la especificación de requerimientos y la otra en las pruebas de software, sin embargo como mencionan Jim Wood Cock, Peter Gorm Larsen y Juan Bicarregui, John Fitzegerld “Los métodos formales utilizan modelos matemáticos para el análisis y la verificación en cualquier parte del ciclo de vida del programa”[2]

II. Promela

Por sus siglas en ingles Protocol Meta Language promela es un lenguaje de modelado de verificación. Los modelos de promela se pueden verificar por medio de su herramienta SPIN, la cual fue desarrollada en el lenguaje de programación C.

El modelo promela consiste en:

- type *declarations*
- channel *declarations*
- variable declarations
- process declarations
- [init process]

Como se compone un proceso, se define mediante una definición de tipo de proceso, son capaces de ejecutarse simultáneamente, se comunica con otros procesos, utiliza variables globales, utilización de canales, con tenido de las variables locales.

```
/* Ejemplo de Hola, Promela de model for SPIN*/

active proctype Hello() {
printf("Hello process, my pid is: %d\n", _pid);
}
init {
int lastpid;
printf("init process, my pid is: %d\n", _pid);
lastpid = run Hello();
printf("last pid was: %d\n", lastpid);
}
```

***Figura 1**, Hola mudo promela*

Para entender el ejemplo se debe entender que hay dos funciones una función que dispara la ación y la otra que la ejecuta, si se piensa como programador en C, init es el método main, por lo tanto el resultado es mostrar primero el prinft, dentro de init, y después se llama a Hello.

Es importante mostrar que promela tiene un parecido con C, por lo tonta se abordaran ejemplos para hacer una comparativa y así tener un mayor un conocimiento sobre promela.

Ejemplo en C:
struct node{ int value; }

Ejemplo en promela:
typedef node{ int value; }

Condicional:
En c se tiene el siguiente ejemplo donde se tiene una operación ternaria.
a = (b > c) ? b : c;
En promela se especificaría de la siguiente manera:
if :: (b > c) -> a = b :: else -> a = c

Ciclo:
while(1){ if(a>b) break; else a++; }

do :: (a>b) -> break :: else -> a++ od

Ejemplo de contador en promela:
int ctr = 0;
int max = 0;

if::(0 < ctr && ctr < max)-> ctr = ctr++ ::else ctr = ctr

Otro ejemplo aportado de la clase CS 5219 es el siguiente:
byte count;
protype counter()
{
do
:: count = count + 1
::count = count - 1
::(count== 0)->break
od;
}

Spin es una herramienta que permite analizar la lógica de los sistemas denominados concurrentes.
cuando se trata de sistemas distribuidos, los lenguajes de especificación formal juegan un papel importante para su verificación, lo cual permite tener la certeza matemática.

Vienna Development Method (VDM)

VDM es un método formal desarrollado por el laboratorio de IBM en Viena alrededor de la década de 1970, siendo este uno de los más antiguos, su funcionamiento es que a través de tipos construidos utilizando constructores se definen datos estructurados y colecciones como conjuntos, secuencias y mapeos a partir de valores básicos como booleanos y números, lo cual permitirá agregar precondiciones y poscondicciónes. Hay tres variantes de VDM, las cuales son: VDM-SL, VDM++, VDM-RT.

Se considera explicar como de VDM se puede pasar al lenguaje Java, por lo tanto:
En Java se declara una constante de la siguiente manera:
public static final int MAX 10;
En VDM se declararia de la siguiente manera:
MAX : 10
En caso de que la variable fuera vacia y de tipo entera seria de la siguiente manera:
temp : Z

Ahora se mostrará el ejemplo del contador, tomado del libro de *VDM to Java*

$$increment()$$

ext wr $temp : \mathbb{Z}$

pre $temp < MAX$

post $temp = \overline{temp} + 1$

Como se pueden observar las precondiciones y las precondiciones son fáciles de entender.

Comparativa entre PROMELA,VMD, TLA+

Para abordar estos se deben conocer que lenguaje actualmente es utilizado por grandes empresas, en el cual destaca TLA+ que empresas como amazon, microsoft, intel, compact lo utilizan para el desarrollo de sistemas distribuidos, al saber que las empresas más grandes de la nube lo están usando, nos da un principio de que lenguaje es el que tiene un mayor impacto en la actualidad, sin embargo los otros lenguajes permiten realizar la especificación, pero TLA+ es más eficiente y actualmente está disponible como extensión de visual studio. Por lo cual los ejercicios básicos se desarrollaran con dicho lenguaje.

TLA+

Es un lenguaje de especificación y verificación el cual funge como ayuda a los ingenieros a diseñar, especificar, razonar y verificar algoritmos de sistemas de software o hardware complejos de la vida real. Hay otros lenguajes de especificación que trabajan en conjunto con los lenguajes de programación ejemplo es Java Modeling Lenguage JML que es un lenguaje para verificar código en Java.

Se presenta un ejemplo básico:

En la imagen se puede observar la logica que se utiliza para hacer un semaforo.

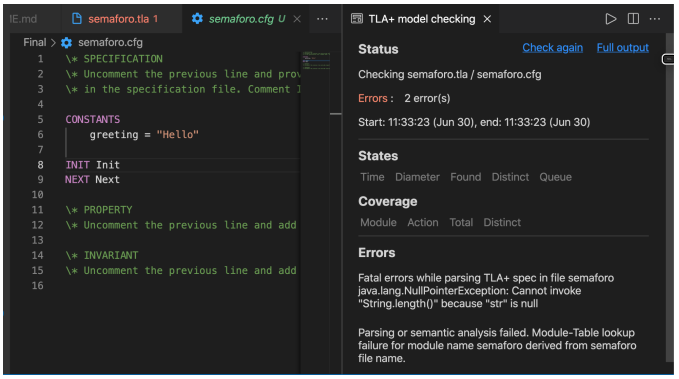
```
---- MOUDLE semaforo ----
VARIABLE color

TypeOK == color \in {"red", "yellow"}

Init == color = "red"

TurnGreen ==
    /\ color = "red"
    /\ color' = "green"
turnYellow ==
    /\ color = "green"
    /\ color' = "yellow"
turnRed ==
    /\ color = "yellow"
    /\ color' = "red"

Next ==
    \/ turnGreen
    \/ turnYellow
    \/ turnRed
```



El verificador encuentra dos errores el cual uno es la descripción, es el siguiente:

TLA+ spec in file semaforo:

java.lang.NullPointerException: Cannot invoke "String.length()" because "str" is null

Ejemplo contador:

```
---- MODULE contador ----
VARIABLE ctr
VARIABLE max

agregarUno(s) == IF 0 < ctr /\ ctr < max
                 THEN ctr = ctr + 1
                 ELSE ctr = ctr
quitarValor(x) == IF ctr < x
                  THEN ctr = ctr + x
                  ELSE ctr = ctr
```

Ejemplo Registro:

```
---- MODULE registro ----
VARIABLE usuarios = {}
VARIABLE adentro = {}
VARIABLE afuera = {}

registrarEntrada(nombre) == IF nombre \notin adentro
                               /\ nombre \notin afuera
                            THEN nombre = adentro
consultarAfuera == afuera

consultarAdentro == adentro
```

Ejemplo Directorio:

```
---- MODULE directorio ----
VARIABLE persona
VARIABLE telefono
DOMAIN <<"David">> = {9514044777}

agregarEntrada(nombre, telefono)
                    [<<nombre>>, telefono]
obtenerTelefonos == DOMAIN {}
borrarEntrada(nombre, telefono) ==  [<<nombre>>, telefono]
                                  / [<<nombre>>, telefono]
```

Ejemplo tareas:

```
---- MODULE tareas ----
VARIABLE nombreEstudiante = ""
VARIABLE codigoTarea = {}
VARIABLE fecha = ""
VARIABLE hora = ""

registrarTarea(nombreEstudiante) nombreEstudiante  == nombreEstudiante
codTarea(codigoTarea)   codigoTarea == codigoTarea
obtenerTarea(codigoTarea) codigoTarea \in codigoTarea
setTarea(codigoTarea,nombreEstudiante,fecha,hora) ==
    codigoTarea == codigoTarea /\ nombreEstudiante == nombreEstudiante
    fecha == fecha  /\ hora == hora
```

Conclusiones

El presente trabajo muestra, lo que es trabajar a un nivel básico con el método formal TLA+, con lleva que para aprender se requiere conocimientos de logica formal así como también de teoría de conjuntos por lo tanto es importante, antes de adentrarse a los métodos formales, por lo tanto es importante que para refuerzo académico, se cuente con una herramienta que brinda la accesibilidad de dicho conocimiento aplicando a la perspectiva de métodos formales para un impacto mayor a sistemas de mayor complejidad dado que son de alta importancia para la ingenieria de software porque empresas como amazon, ibm, microsoft, intel los utilizan para su verificación de sistemas tanto distribuidos, como computo en la nube, por lo tanto el compromiso se reitera para abstraer el fundamento matemático y de ahí partir para su correlación así como el conocimiento de la programación orientada a objetos y UML. Permitirá tener un impacto más profundo hacia el propósito de la implementación tangencial de sistemas básicos como complejos, para su correcta aplicación. En la parte de TLA+ es un lenguaje que está actualizado al día de hoy y cuenta con la herramienta moderna de validación que se puede en contrar simple descargado visual studio, el cual ya incluye el verificador.

Referencias

[1] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui y John Fitzgerald. 2009. Métodos formales: práctica y experiencia. Computación ACM. Surv. 41, 4, artículo 19 (octubre de 2009), 36 páginas. DOI: https://doi.org/10.1145/1592434.15924

[2] Barr, J. Amazon S3-Two Trillion Objects, 1.1 Million Requests per Second. Amazon Web Services Blog. March 2013; http://aws.typepad.com/aws/2013/04/amazon-s3-two-trillion-objects-11-million-requests-second.html

[3] Holloway, C. Michael. Why You Should Read Accident Reports. Presented at Software and Complex Electronic Hardware Standardization Conference, July 2005; http://klabs.org/richcontent/conferences/faa_nasa_2005/presentations/cmh-why-read-accident-reports.pdf

[4] Lamport, L. The TLA Home Page; http://research.microsoft.com/en-us/um/people/lamport/tla/tla.html

[5] Joshi, R., Lamport, L., et al. Checking Cache-Coherence Protocols With TLA+; http://research.microsoft.com/pubs/65162/fmsd.pdf

[6] Patterson, D., Fox, A., et al. The Berkeley/Stanford Recovery Oriented Computing Project; http://roc.cs.berkeley.edu/

[7] Zave, P. Using lightweight modeling to understand Chord; In ACM SIGCOMM Computer Communication Review volume 42 number 2, April 2012; http://www2.research.att.com/~pamela/chord.html