Análisis de requerimientos de hardware funcionales de forma automática a través de árboles sintácticos

Ing. Alejandra Victoria Alcaraz[1],
Mtro. Gustavo Espejel Salazar[2]

***Resumen*—El proceso de validación juega un papel muy importante en empresas como Intel® y comienza con la definición de casos de prueba, estos se obtienen de forma manual por los ingenieros de validación a partir de los requerimientos funcionales, por lo tanto es una actividad susceptible a errores. De reducir el tiempo de análisis de requerimientos, automatizando el análisis y generando casos de prueba de manera automática se ahorraría mucho tiempo. Este artículo presenta la alternativa para el análisis automático de requerimientos por medio del diseño de una gramática libre de contexto que describe el lenguaje usado para definir los requerimientos funcionales de hardware. Esta gramática es capaz de generar ahora un árbol sintáctico a partir de requerimientos funcionales donde cada nodo representa un elemento importante del requerimiento.**

***Palabras clave*—Requerimientos, validación, casos de prueba, gramáticas libres de contexto.**

Introducción

La validación funcional de hardware se refiere a una etapa específica dentro del proceso de diseño y manufactura de un chip. Como lo define Mozhikunnath (2016), es el proceso en el cual el silicio es probado funcionalmente en una configuración de laboratorio. Esto se hace usando el chip real ensamblado en una plataforma de referencia junto con los demás componentes que son parte del sistema para el cual el chip fue diseñado. La meta es validar todos los casos de uso que un cliente puede eventualmente tener en una implementación real.

Los casos de uso o casos de prueba que se usan en la validación se obtienen de los requerimientos del sistema, Espejel (2019) define a la verificación como la comprobación de que el sistema cumple con los requisitos funcionales y no funcionales de su especificación. De esta manera se requiere de un proceso de análisis de los requerimientos que genere como salida los casos de prueba que serán ejecutados en la etapa de validación.

Comúnmente este proceso de análisis es llevado a cabo por los ingenieros de validación, en Intel® está comprobado que esta actividad es costosa. Sin embargo, a pesar de los costos se mantienen estas actividades porque se tiene presente que los planes de pruebas constituyen la actividad más crítica y fundamental en la preparación hacia la validación de post-silicio (Prabhat et al. 2017).

En este trabajo se busca probar que dentro de Intel® es posible automatizar el análisis de requerimientos funcionales a través de árboles sintácticos a fin de que se reduzca el periodo de validación y por lo tanto se reduzcan costos. Aunado a esto, se obtendrían los beneficios de eliminar los errores introducidos por el factor humano al automatizar este proceso.

Marco Teórico

Ingeniería de requerimientos

El proceso de ingeniería de requerimientos es una etapa temprana importante en el ciclo de vida de los sistemas de ingeniería, su meta es desarrollar y mantener un buen conjunto de requerimientos, su importancia estriba en el hecho de que el 35% de las fallas son introducidas en esta etapa del ciclo de vida de acuerdo con Siu et al. (2017).

Teoría de autómatas y lenguajes formales

De acuerdo con Málaga (2008), los lenguajes se definen como un conjunto de palabras formadas por símbolos de un alfabeto, las gramáticas permiten definir la estructura de un lenguaje, y los autómatas son los dispositivos teóricos que reciben cadenas de símbolos y determinan si pertenecen o no a un determinado lenguaje. Existen diferentes tipos de autómatas de acuerdo con su nivel de complejidad: los Autómatas Finitos son el grupo más sencillo, los Autómatas de Pila y los Autómatas Linealmente Acotados son del nivel intermedio, y las Máquinas de Turing son los autómatas más complejos.

[1] Ing. Alejandra Victoria Alcaraz es Estudiante de la Maestría Sistemas Inteligentes Multimedia en el Posgrado CIATEQ A.C. ale.victoria.alcaraz@gmail.com **(autor corresponsal)**
Ing. de validación en Intel Tecnología de México S.A. de C.V. alejandra.victoria.alcaraz@intel.com
[2] Mtro. Gustavo Espejel Salazar es Asesor de Tesis en Posgrado CIATEQ A.C. gespejels@gmail.com
Ing. en General Electric Company (GE). gustavo.espejel@ge.com

En la década de los años 50s el lingüista y pensador Noam Chomsky clasificó las gramáticas y los lenguajes que estas generan de acuerdo con una jerarquía de cuatro niveles como se muestra en la Figura 1. De manera sorprendente, es posible establecer una relación uno a uno entre los diferentes niveles de esta jerarquía de gramáticas y lenguajes, y los grupos de autómatas que podrán reconocer dichos lenguajes.

Gramáticas libres de contexto

Para nuestro propósito se diseñó una gramática libre de contexto, de acuerdo con Gómez (2015), estas son las gramáticas que típicamente se usan para definir los lenguajes de programación modernos ya que es posible diseñar un algoritmo que decida si un programa es sintácticamente válido, y para nuestro propósito nos es de utilidad una gramática que reconozca la sintaxis de un requerimiento funcional y nos entregue el requerimiento fraccionado en sus partes clave.

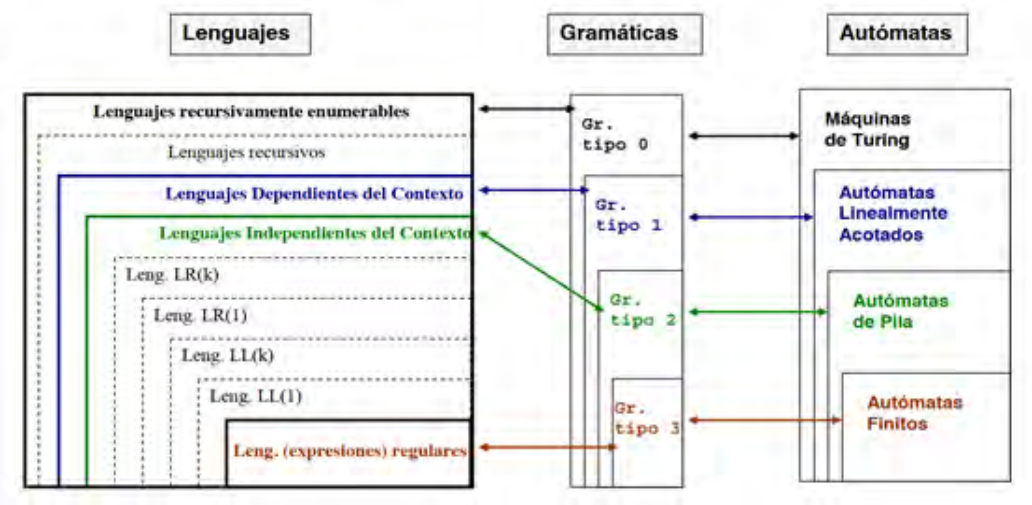

Figura 1. Jerarquía de gramáticas, lenguajes y autómatas (Málaga, 2008)

Compiladores

En la implementación de este analizador automático se usan las bases teóricas acerca de los compiladores, los cuales son programas que traducen de un lenguaje de alto nivel a uno de bajo nivel. (Málaga 2008)

Un compilador es un programa complejo en el que se puede establecer una división lógica en fases: análisis léxico, análisis sintáctico y análisis semántico. En el trabajo de Espejel (2019) la etapa del análisis semántico se utiliza para la generación automática de casos de prueba.

Análisis léxico

El analizador léxico, también conocido como scanner, lee los caracteres del programa fuente, uno a uno, desde el fichero de entrada y va formando grupos de caracteres con alguna relación entre sí (tokens). (Málaga 2008)

Análisis sintáctico

El analizador sintáctico, también llamado parser, recibe como entrada los tokens que genera el analizador léxico y comprueba si estos tokens van llegando en el orden correcto. Siempre que no se hayan producido errores, la salida teórica de esta fase del compilador será un árbol sintáctico. (Málaga 2008)

Descripción del Método

Reseña de las dificultades de la búsqueda

La principal dificultad al realizar esta investigación fue que los requerimientos no están escritos en un estilo estandarizado. De acuerdo con Scott et al. (2004), los requerimientos deficientes son citados frecuentemente como la principal causa de la falla de proyectos. Es común encontrar empresas que tienen problemas en el manejo de los requerimientos, algunos estudios de campo que se hicieron en compañías de Australia sobre las prácticas de ingeniería de requerimientos y las áreas de mejora concluyeron que los problemas fueron organizacionales y no técnicos por naturaleza, por ejemplo, administración de la documentación y manejo de incertidumbre (Martin S. et al. 2002).

Para este trabajo se tomó una muestra de requerimientos funcionales de los distintos componentes que conforman a un servidor en Intel® y se encontró un patrón en la mayoría de ellos que refleja que estos requerimientos tienen la siguiente forma: bajo ciertas condiciones, cuando ocurre alguno o varios eventos, entonces

se deben disparar uno o más flujos de acción; resultando así, que un requerimiento funcional tiene una estructura como la Ecuación **1**.

Ecuación 1. Estructura de un Requerimiento Funcional

$$REQUERIMIENTO\ FUNCIONAL \rightarrow Si\ CONDICIONES\ cuando\ EVENTOS, entonces\ RESULTADOS$$

Una vez encontrado este patrón en los requerimientos funcionales, la pregunta fue cómo procesar computacionalmente los requerimientos escritos en inglés para que se reconozcan los elementos de la estructura de manera automática.

Diseño de solución

Existen trabajos que se han hecho con el mismo objetivo de analizar requerimientos para generar automáticamente casos de prueba, uno de estos trabajos es la herramienta de ASSERT™ desarrollada dentro de General Electric, la cual acepta requerimientos formalizados para después analizarlos en base a un modelo de dominios construido a partir de un lenguaje ontológico (Siu et al, 2017). Otra herramienta es la desarrollada por Espejel (2019), la cual acepta requerimientos expresados en forma de diagramas de actividad UML (Lenguaje Unificado de Modelado), usando la sintaxis de PlantUML (herramienta de código abierto) y a partir de ahí genera un árbol sintáctico que sustituye grafos utilizados en la generación automática de casos de prueba a partir de diagramas UML.

Este trabajo logró combinar las propuestas usadas en los trabajos de (Siu et al, 2017) y Espejel (2019), y adaptarlas para ser compatibles con las necesidades del equipo de validación de servidores de Intel®, para diseñar una solución a nuestro problema de procesamiento de requerimientos. En Intel® los requerimientos son documentados en lenguaje inglés y el fin fue procesarlos a partir de este formato, por lo que se tomó la idea del trabajo de ASSERT™ (Siu et al. 2017) de estandarizar el formato para la escritura de requerimientos. Y para el procesamiento de requerimientos se tomó la idea del trabajo de Espejel (2019) sobre el uso de gramáticas y teoría de compiladores para procesar los archivos de entrada y generar un árbol sintáctico con los elementos clave de los requerimientos que permita construir los casos de prueba.

Resultados

Gramática de Requerimientos Funcionales

Se diseñó una gramática que permitiera describir los requerimientos funcionales que tienen la estructura mostrada en la Ecuación 1. Esta gramática, mostrada en la Figura 2, permite hacer un análisis sintáctico de un requerimiento en sus elementos clave por medio del analizador léxico y sintáctico.

```
REQUERIMIENTOS  => REQUERIMIENTO | REQUERIMIENTO REQUERIMIENTOS
REQUERIMIENTO   => FUNCIONAL
FUNCIONAL       => if and only if CONDICIONES when EVENTOS then RESULTADOS

CONDICIONES     => CONDICION | CONDICION and CONDICIONES | CONDICION or CONDICIONES
CONDICION       => COMPONENTE.REGISTRO = VALOR
COMPONENTE      => identificador
REGISTRO        => identificador
VALOR           => numero_entero

EVENTOS         => EVENTO | EVENTO and EVENTOS | EVENTO or EVENTOS
EVENTO          => MENSAJE VERB the DESTINO
MENSAJE         => identifier
DESTINO         => identifier

RESULTADOS      => RESULTADO | RESULTADO and RESULTADOS | RESULTADO or RESULTADOS
RESULTADO       => MENSAJE DEBER ACCION
DEBER           => must be | must not be
ACCION          => triggered | discarded | transmitted.
```

Figura 2. Gramática para requerimientos funcionales

Analizador léxico y sintáctico

Para la implementación de los analizadores léxicos se utilizan los Autómatas Finitos Determinísticos que fueron construidos a partir de las expresiones regulares mostradas en la Figura 3.

```
<?xml version='1.0' ?>
<regular_expressions>
    <integernumber ignore="no">[0-9][0-9]*</integernumber>
    <identifier ignore="no">([a-z]|[A-Z]|_)([a-z]|[A-Z]|[0-9]|_)*</identifier>
    <onedot ignore="no">.</onedot>
    <openbraket ignore="no">\[</openbraket>
    <closebraket ignore="no">\]</closebraket>
    <openparentesis ignore="no">\(</openparentesis>
    <closeparentesis ignore="no">\)</closeparentesis>
    <pipe ignore="no">\|</pipe>
    <inverteddiagonal ignore="no">\\</inverteddiagonal>
    <tilde ignore="no">~</tilde>
    <lessthan ignore="no">&lt;</lessthan>
    <greaterthan ignore="no">&gt;</greaterthan>
    <equal ignore="no">==</equal>
    <assign ignore="no">=</assign>
    <arrow ignore="no">-&gt;</arrow>
    <openquestionmark ignore="no">?</openquestionmark>
    <closequestionmark ignore="no">¿</closequestionmark>
    <arroba ignore="no">@</arroba>
    <plus ignore="no">+</plus>
    <minus ignore="no">-</minus>
</regular_expressions>
```

Figura 3. Expresiones regulares

El analizador sintáctico está implementado utilizando un algoritmo de back-tracking que hace uso de la gramática diseñada mostrada en la Figura 2 y recibe como entrada un archivo que contiene un requerimiento escrito en la forma estandarizada que definimos en la Ecuación 1.

En la Figura 4 se muestra el requerimiento de ejemplo que se utilizó para probar los analizadores léxico y sintáctico.

```
if and only if
    DPCCTLR.DPCTE=1
    and
    DPCCAPR.RPEFDPC=1
    and
    DPCECH.PTLPEBS=1
    and
    DPCCTLR.PTLPEBE=1
    and
    UEM.PT=0
    and
    DPCCTLR.DPCCC=1
when
    a_poisoned_TLP targets the egress_port
then
    DPC must be triggered
    and
    the_poisoned_TLP must be discarded
    and
    the_poisoned_TLP must not be transmitted
```

Figura 4. Ejemplo de requerimiento funcional

En la Figura 5 se muestran el árbol sintáctico obtenido al hacer uso de los analizadores utilizando la gramática diseñada y el requerimiento de ejemplo.

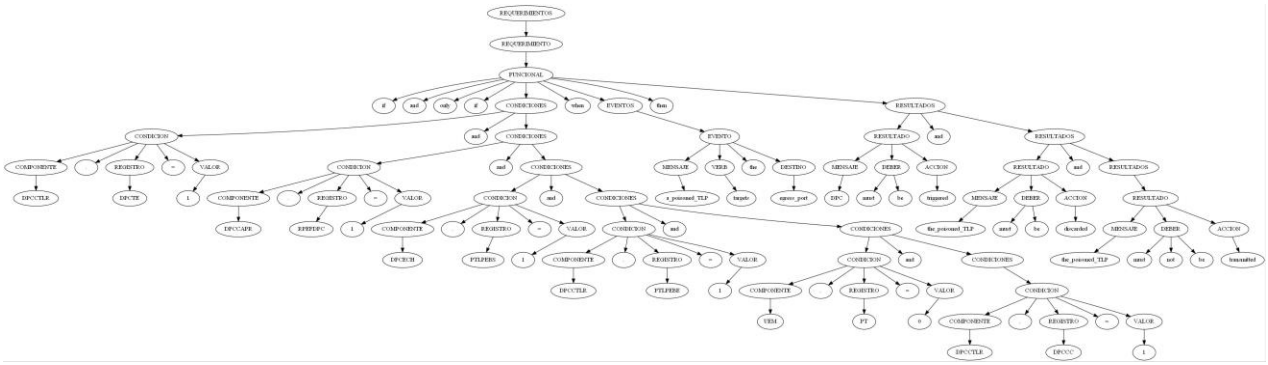

Figura 5. Árbol sintáctico

Se puede observar cómo se forma cada nodo con los elementos clave que están definidos en la gramática, en la Figura 6 que es un fragmento de la Figura 5, se puede apreciar mejor cómo se desglosan las condiciones que están presentes en el requerimiento.

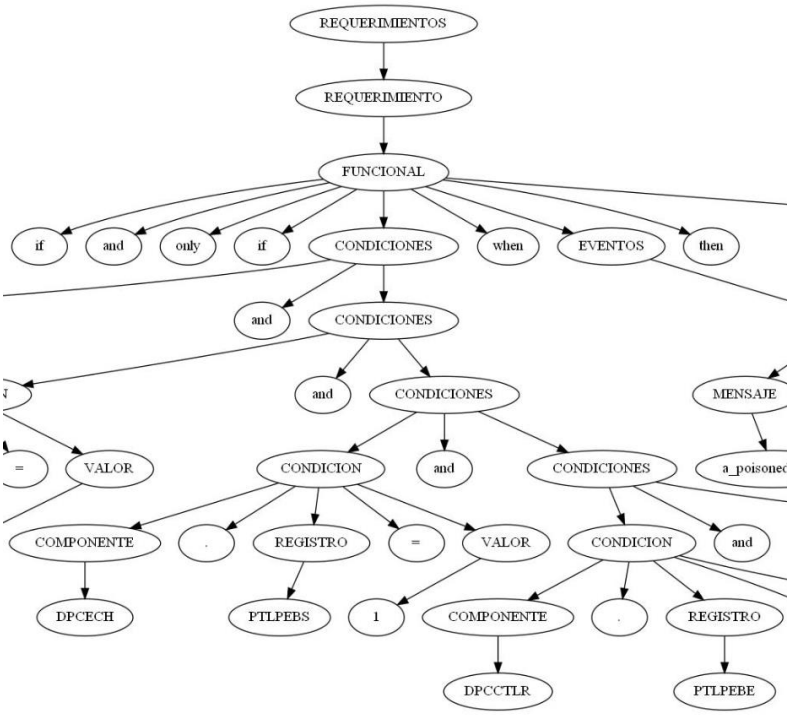

Figura 6. Enfoque en las condiciones

Comentarios Finales

Sería aquí el espacio para añadir los comentarios finales, que casi siempre incluyen un resumen de los resultados, las conclusiones, y las recomendaciones que hacen los autores para seguir el trabajo. Esta sección puede tener subsecciones.

Conclusiones

En este trabajo investigativo se estudió una alternativa para hacer el análisis de requerimientos funcionales, buscando una manera de automatizar el proceso que permitiera llevarlo a cabo en menor cantidad de tiempo y libre de errores humanos. Se encontró que por medio de una gramática libre de contexto, usando un analizador léxico y

sintáctico se puede obtener el árbol sintáctico del requerimiento. Actualmente estoy trabajando en la tesis de la maestría para generar casos de prueba a partir de este árbol sintáctico.

Referencias

Espejel Salazar G. "Desarrollo de infraestructura para facilitar el análisis y la verificación de diagramas de actividad identificando los caminos funcionales modelados" Querétaro, Querétaro 2019.

Gómez, D. y Pardo, Luis M. "Teoría de Autómatas y Lenguajes Formales" Santander: Universidad de Cantabria, 2015.

Málaga Jurado, E. "Teoría de autómatas y lenguajes formales". Cáceres: Universidad de Extremadura, 2008. Vol. 55. 1135-870-X/978-84-691-6345-0.

Martin S. et al. "Requirements Engineering Process Models in Practice" Melbourne: Deakin University, 2002. 0730025667

Mozhikunnath, R. "Verification, Validation, testing of ASIC/SOC designs – What are the differences" anysilicon.com *(*en línea*)*, CMC Mycrosystems, Septimbre 18, 2016, consultada por Internet el 19 de Septiembre de 2020. Dirección de internet: http://anysilicon.com/verification-validation-testing-asicsoc-designs-differences/.

Prabhat, Mishra, et al. "Post-silicon validation in the SoC Era: A tutorial introduction" IEEE Design & Test, 2017, Vol. 34. 2168-2356/2168-2364

Scott, W., Cook, S. y Kasser, J. "Development and Application of a Context-free Grammar for Requirements" Australia: In: SETE (2004 : Adelaide, S. Aust.). SETE 2004: Focussing on Project Success; Conference Proceedings; 8-10 November 2004. [Canberra, ACT]: Systems Engineering Society of Australia, 2004: [333]-[340]., 2004. 0957767285.

Siu, K., Moitra, A., Durling, M., Crapo, A., Li, M., Yu, H., Herencia-Zapana, H., Castillo-Effen, M., Sen, S., McMillan, C., Russell, D., Roy, S., y Manolios, P. (2017). "Flight critical software and systems development using ASSERT™". 2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC), 1–10. https://doi.org/10.1109/DASC.2017.8102059