Construcción de un Compilador de Asertos de Programación Metódica Utilizando El Método De Autómatas Adaptativos

Diego Berolatti, Claudia Zapata

Agenda

- Introducción
- Marco Conceptual
- Descripción de la Solución
- Diseño e Implementación de los Autómatas

Introducción

- Siempre ha existido la necesidad de validar la codificación de un programa. Este proyecto tiene como objetivo la implementación de un compilador que, mediante notaciones matemáticas que especifican un programa, se genere las instrucciones de manera automática. El resultado de la compilación tiene como principal característica que sea formalmente correcto. Esto se da debido a la utilización de una metodología llamada derivación de programas la cual garantiza esa característica.

PUCP

Marco conceptual

- La lógica de Hoare es un sistema formal que proporciona una serie de reglas de inferencia para razonar sobre la corrección de programas imperativos con el rigor de la lógica matemática y su aplicación garantiza que el programa que las utilice cumpla con las especificaciones que este tenga a priori.
- El triple de Hoare {P} C {Q} formada por dos expresiones matemáticas (asertos) con un resultado de tipo booleano y un comando o instrucción. El aserto P, precondición, representa el estado del programa antes de ejecutar C. Q, postcondición, representa el estado del programa después de ejecutar C.

PUCP

Marco conceptual

- Una adaptación de estas reglas, llamada programación metódica, permite mediante la inclusión de métodos formales deducir el conjunto de instrucciones C.
- Esta forma de construir código a partir de asertos, llamada derivación, conserva las mismas cualidades que la lógica de Hoare debido a que siguen sus reglas.

PUCP

Descripción de la solución

- Este proyecto es la implementación de un compilador convencional de asertos de programación metódica utilizando el método de autómatas adaptativos en la representación de las propuestas de derivación aplicadas en la derivación de asertos. Cada una de ellas en función a las condiciones que existen para poder utilizarlas.

PUCP

Objetivos

- Establecer las reglas que el compilador va a ejecutar de manera automática para garantizar la aplicación correcta de la programación metódica.
- Implementar el analizador léxico, sintáctico y semántico de la notación matemática utilizada en la programación metódica para satisfacer las especificaciones que tenga cada programa generado por el compilador.
- Implementar un traductor de código intermedio que sea capaz de generar programas en el lenguaje de alto nivel para garantizar la "correctitud" formal de los programas y su aplicación en el desarrollo de software.

PUCP

Algoritmo

1. Se crea una dupla en función de dos asertos.
2. Se valida la equivalencia de los dos asertos.
3. Si son equivalentes se devuelve la lista de estados involucrados en la transformación de los asertos.
4. Si no son equivalentes y no han sido analizados antes se aplica el análisis de las submáquinas.
5. Se registra la dupla en la lista de duplas ya analizadas.
6. Se recibe cada resultado parcial en forma de lista de estados de cada submáquina.
7. Por cada uno de estos estados y en función de la dupla actual se genera una nueva dupla.
8. Por cada nueva dupla formada que no haya sido analizada se aplica el paso 2.
9. El algoritmo termina si no se cuenta con duplas que no hayan sido analizadas con anterioridad con resultado de error de capacidad de derivación.

PUCP

Regla 1

1. “Cuando en la postcondición tenemos una conjunción de igualdad entre solo dos variables podemos asignar una variable a otra”. Si en la postcondición nos encontramos que una ecuación, dentro de un conjunto de ecuaciones, es la igualdad entre dos variables entonces son propuestas de estado la asignación de una variable a otra. Cualquiera de ellas puede ser la asignada siempre y cuando se cumplan con las reglas de aplicación de la metodología.

Ejemplo:

{Post: … AND x=z}

Siendo propuestas validas z:=x y x:=z

PUCP

Implementación

- Esta configuración se aplica en la regla 1 la cual consta de los siguientes estados: el estado inicial (a0) es el encargado de determinar si existe una conjunción de igualdad entre solo dos variables; si existe se utiliza la transición t1 al estado b0, sino se toma la transición t3 al estado de salida a1. Luego, en el estado b0 se verifica si ambas variables son ligadas, de serlas se va al estado final b2 mediante la transición t2. Si solo una es ligada entonces se va al estado b1 por el estado t5. Debido a que este análisis se hace por cada ecuación, la transición t5 representa la transición de análisis entre ecuaciones de un aserto.

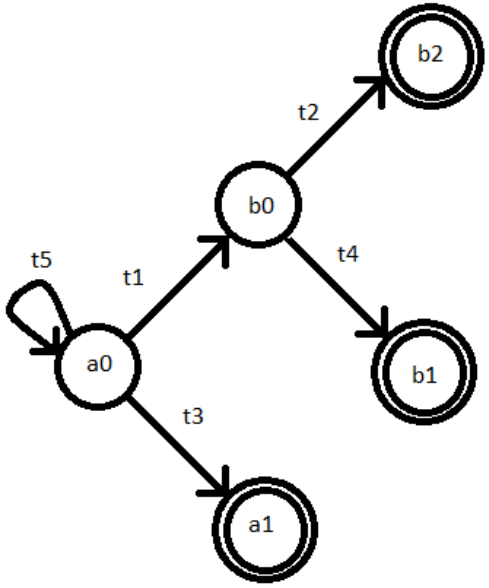


Fig. 1. Configuración del autómata del tipo 1.

Regla 2

2. “Si en la postcondición existe una única variable ligada, siendo las demás libres o constantes, habremos de efectuar una asignación sobre esta variable en función a todas las variables (variables de la precondicion y la postcondición)”. Si en la postcondición dentro de cada ecuación y conjunción existe solo una variable ligada, es entonces la asignación a esta variable ligada una propuesta de estado de derivación.

Ejemplo:

{Pre: x+y=T} {Post: z+y=T}

Siendo x una variable que no se encuentra en la postcondición, z una variable libre y T una constante entonces: y:= E(x,y,z) es una propuesta.

Regla 4

4. “Si la regla I no se puede aplicar en ninguna de las conjunciones se puede asignar la variable a una nueva variable".

Si por alguna razón la regla I o la regla de igualdad entre dos variables no se puede aplicar o no lleva a ningún resultado, se puede aplicar asignar la variable a una nueva variable creada.

Ejemplo:

{Post: x=y}

Asumiendo que no se puede aplicar la regla I entonces se propone hacer x:= z siendo z una variable nueva.

Regla 6

6. “Si en la postcondición existen conjunciones y ninguna es de igualdad entonces se puede establecer una asignación múltiple de todas las variables ligadas”. Si la postcondición está conformada por conjunciones y ninguna ecuación es de igualdad entonces se propone una asignación múltiple a cada variable ligada.

Ejemplo:

Si a y b son variables iniciales:

{Pre: a= 2*b*c + r AND r< 2*b}

{Post: a= b*c + r AND r< b}

Entonces se puede intentar hacer una asignación múltiple con ambos:

{Pre: a= 2*b*c+r AND r< 2*b}

<c,r>:=<E1,E2>

{Post: a= b*c+r AND r< b}

PUCP

Implementación

- Esta configuración se utiliza para aplicar las propuestas de derivación 2, 4 y 6 debido a que estas comparten la misma naturaleza.
- El estado inicial a0 se encarga de validar que se cumpla su condición la cual involucra todo el aserto. De no cumplirlo se va por la transición t2 al estado a1, en otro caso se traslada por la transición t1 al estado b1.

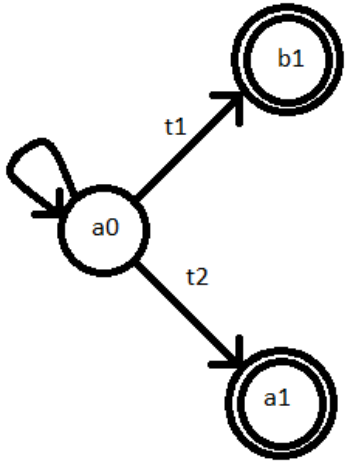


Fig. 2. Configuración del autómata del tipo 2.

Regla 3

3. “Si en la postcondición tenemos una conjunción de igualdad entre una variable ligada y una constante y esta constante se encuentra asignada a un variable en la precondición entonces se puede efectuar una asignación sobre esta variable con la variable de la precondición la cual tiene a la constante asignada". Si en la postcondición existe dentro de una ecuación una igualdad entre una variable ligada y una constante, es una propuesta la asignación a esa variable del valor de la constante, el cual sea igual al valor de otra variable en la precondición.

Ejemplo:

{Pre: x=X} {Post: y=X}

Debido a que la variable y es equivalente a la constante X en la postcondición y a que existe en la precondición una variable que es equivalente a esta constante, entonces y:=x seria la propuesta generada por esta regla.

Regla 5

5. "Si en la postcondición existe disyunción entre dos conjunciones se sugiere entonces por cada una establecer un conjunto de instrucciones alternativas". Si la postcondición está conformada por disyunciones de conjunciones entonces es una propuesta un conjunto de instrucciones alternativas donde para cada una se le proponga cada disyunción por separado.

Ejemplo:

Si la postcondición es:

{Post: (z = x OR z = y) AND…}

Entonces se propone,

[… →… para el aserto {z = x AND …} y

[]…→… para el aserto {z = y AND …}

Regla 7

7. “Si al generar un nuevo aserto este contiene conjunciones que no pueden ser garantizadas por la precondición entonces se puede utilizar la condición (la conjunción que no puede ser garantizada) como protección”. Es una forma de establecer las protecciones de las instrucciones alternativas, se establecen como condiciones para la aplicación de la instrucción, lo cual es a su vez, la nueva postcondición. La disyunción de todas representan el nuevo aserto generado por el conjunto de instrucciones alternativas.

Ejemplo:

{Pre: a= 2*b*c+r AND r< 2*b}

[r<b → c := 2*c

[] ? → …

{Post: a= b*c+r AND r< b}

Dado que el nuevo aserto de c:= 2*c es r<b este actúa también como protección de la instrucción. Siendo a un vector, x un entero y k un numero natural.

Para definir el autómata adaptativo encargado de la construcción de propuestas utilizaremos la siguiente analogía:

- Una frase es un aserto.
- Una palabra es una ecuación, siendo la frase y el aserto un conjunto de estos.
- Una letra es una variable, constante o relación dentro de una ecuación.

Implementación

- Esta configuración se utiliza para aplicar las propuestas de derivación 3, 5 y 7 debido a que estas comparten la misma naturaleza. El estado inicial a0 se encarga de validar que se cumpla con la existencia de una característica del conjunto de ecuaciones.
- Cada una de ellas sin excepción debe cumplirla, la transición t1 representa entonces el análisis de cada una de las ecuaciones. De no cumplirlo se va por la transición t5 al estado a1, en otro caso se traslada por la transición t2 al estado b0. Este estado valida que se cumpla con una condición en el aserto inicial o precondición. La transición t3 representa e análisis de cada uno de las ecuaciones de la precondición. Si la condición se cumple entonces se procede al estado b1 por la transición t4; sino se va al estado a1 por la transición t6.

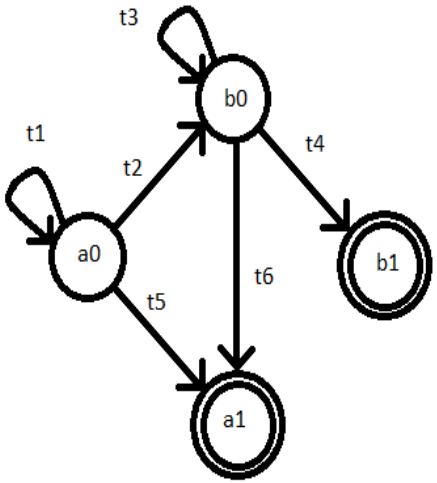
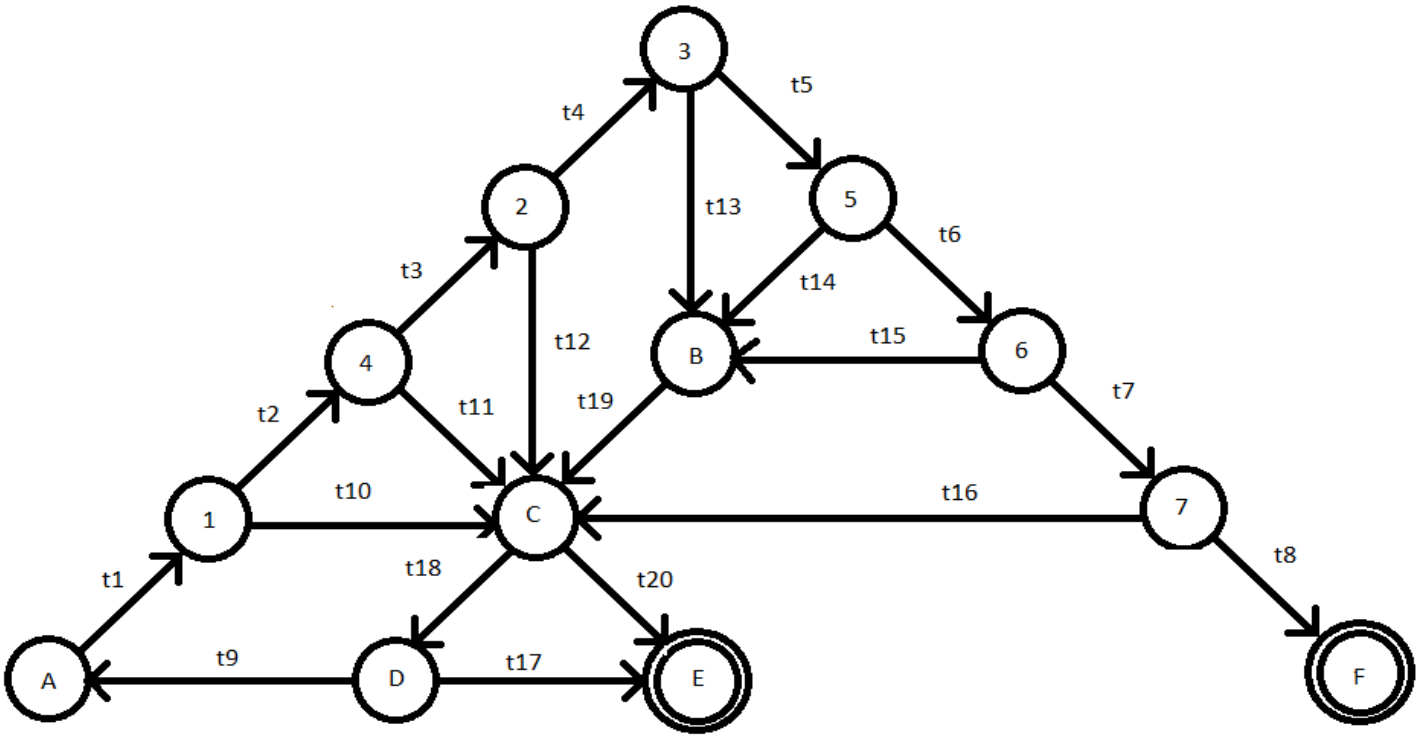


Fig. 3. Configuración del autómata del tipo 3.

Implementación

- De modo adicional para el propio funcionamiento de derivación se utilizó un autómata. El cual se encarga de la aplicación de las reglas de derivación.



Conclusiones

- La aplicación automática de procesos puede ser tarea sencilla para una persona que conozca las reglas de derivación. Sin embargo, pueden ser difíciles de ejecutar para un computador. En la mayoría de estos casos se optó por diseñar un algoritmo adaptado a estos procesos, en consecuencia se limite el alcance de aplicación de la regla.

PUCP

Conclusiones

- Se tuvieron que descartar propuestas de derivación debido a que su aplicación requería, en algunos casos, la generación de programas recursivos, aplicaciones en función a una estructura abstracta de datos específica y algoritmos de transformación por inducción. En otros casos, las reglas no podían ser generalizadas y eran aplicadas en función a sugerencias arbitrarias del autor.

PUCP

Conclusiones

- Era necesario incluir los vectores para enriquecer el lenguaje, pero debido a que las propuestas de derivación relacionadas a los vectores implicaban en su gran mayoría la inclusión de algoritmos np se optó por derivar de manera automática cada cuantificador sin utilizar las propuestas no implementadas. De esta manera se logró incluir la representación de vectores en el lenguaje cumpliendo con la expresividad necesaria para satisfacer los objetivos del mismo.

PUCP

Conclusiones

- En la implementación del traductor de código intermedio se aplicaron y adaptaron los conceptos relacionados a un autómata adaptativo. El algoritmo no solo es capaz de utilizar las propuestas de derivación de programas con éxito, sino que también soporta la inclusión de nuevas reglas ya que la formación de cada submáquina es independiente del funcionamiento del algoritmo.
- La expresividad del lenguaje, en algunos casos, excede la capacidad de traducción del compilador, sin embargo, en estos casos el compilador resuelve no dar resultado a dar uno erróneo.

PUCP

¡Muchas gracias!

¿Preguntas?

zapata.cmp@pucp.pe

PUCP