Concretización a Perl de casos de prueba abstractos generados a partir de especificaciones Z

13 de noviembre de 2020

Autor:
Javier Bonet
javbonet@gmail.com

Director:
Dr. Maximiliano Cristiá
cristia@cifasis-conicet.gov.ar

Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Universidad Nacional de Rosario
Av. Pellegrini 250 - Planta baja - (S2000BTP)
Rosario - República Argentina
Teléfono: +54 - 341 - 4802649/52 - interno 112
http://fceia.unr.edu.ar

Índice general

1. Introducción **5**

2. Lenguaje de refinamiento **6**
2.1. Introducción 6

3. Testing funcional basado en especificaciones **8**
3.1. Especificaciones formales 8
3.2. Lenguaje de especificación Z 8
3.3. Testing funcional 10
3.3.1. Formalización del proceso de testing 10
3.3.2. Etapas del testing funcional 12
3.3.3. Ejemplo de testing 14

4. Estudio del estado del arte **16**

5. Fastest **18**
5.1. Diseño y detalles generales 18
5.2. Uso de Fastest 19
5.3. ATCAL 21
5.4. ATCAL - Gramática y descripción 21
5.4.1. Regla de refinamiento 22
5.4.1.1. @RRULE 22
5.4.1.2. preamble 22
5.4.1.3. datatypes 23
5.4.1.4. laws 23
5.4.1.5. refinement 23
5.4.1.6. biRefLaw 24
5.4.1.7. lawRefinement 24
5.4.1.8. lawReference 25
5.4.1.9. zExprs 25
5.4.1.10. epilogue 28
5.4.1.11. uut 28
5.5. Parser de ATCAL 29
5.5.1. ANTLR 29
5.5.2. Proceso de generación del parser 29

6. Proceso de refinamiento **31**
6.1. Aspectos generales 31
6.1.1. Generación de casos de prueba 31
6.1.2. Ejecución de refinamiento y descripción general 32
6.1.2.1. RefineCommand 34
6.1.2.2. TCaseRefineClient 34
6.1.2.3. TCaseRefineClientRunner 35
6.1.2.4. AtcalEvaluator 35
6.1.2.5. TypesEvaluator 35
6.1.2.6. RefinementLawEvaluator 35
6.2. Detalles 35
6.2.1. visitLaws 36
6.2.2. visitBiRefLaw 36
6.2.3. visitLawRefinement 36

6.2.4. visitLawReference . . . 36
6.2.5. ZExprEvaluator . . . 36
6.2.5.1. visitIdent . . . 36
6.2.5.2. visitNumLiteral . . . 37
6.2.5.3. visitStrLiteral . . . 37
6.2.5.4. visitAutoExpr . . . 37
6.2.5.5. visitElemExpr . . . 37
6.2.5.6. visitGroup . . . 37
6.2.5.7. visitProdProj . . . 37
6.2.5.8. visitProdCons . . . 37
6.2.5.9. visitSetDom . . . 37
6.2.5.10. visitSetDiff . . . 37
6.2.5.11. visitSetInter . . . 37
6.2.5.12. visitSetRan . . . 37
6.2.5.13. visitSetProj . . . 37
6.2.5.14. visitSetCons . . . 38
6.2.5.15. visitSetUnion . . . 38
6.2.5.16. visitNumMod . . . 38
6.2.5.17. visitNumMul . . . 38
6.2.5.18. visitNumPlus . . . 38
6.2.5.19. visitNumDiv . . . 38
6.2.5.20. visitNumMinus . . . 38
6.2.5.21. visitSetCard . . . 38
6.2.5.22. visitStrConcat . . . 38
6.2.6. visitRefinement . . . 38

7. Casos de estudio **40**
7.1. Sistema de bicicletas . . . 41
7.1.1. Especificación Z . . . 41
7.1.2. Casos Abstractos de prueba . . . 43
7.1.2.1. AddBike . . . 43
7.1.2.2. RemoveBike . . . 43
7.1.2.3. GetBikeManufacturer . . . 44
7.1.3. Implementación . . . 44
7.1.3.1. Implementación 1: array de contratos . . . 44
7.1.3.2. Implementación 2: contrato de records . . . 46
7.1.4. Descripción de leyes de refinamiento . . . 49
7.1.4.1. Array de contratos . . . 49
7.1.4.1.1. Identificador de regla . . . 49
7.1.4.1.2. Definición de tipos de datos . . . 49
7.1.4.1.3. Especificación de leyes de refinamiento . . . 50
7.1.4.1.4. Especificación de método a testear . . . 51
7.1.4.2. Contrato de records . . . 51
7.1.4.2.1. Identificador de regla . . . 51
7.1.4.2.2. Definición de tipos de datos . . . 51
7.1.4.2.3. Especificación de leyes de refinamiento . . . 52
7.1.4.2.4. Especificación de método a testear . . . 52
7.1.5. Casos concretos de prueba en Perl . . . 53
7.1.5.1. Implementación 1: array de contratos . . . 53
7.1.5.1.1. Operación AddBike . . . 53
7.1.5.1.2. Operación RemoveBike . . . 57
7.1.5.1.3. Operación GetBikeManufacturer . . . 58
7.1.5.2. Implementación 2: contrato de records . . . 59
7.1.5.2.1. Operación AddBike . . . 59
7.1.5.2.2. Operación RemoveBike . . . 63
7.1.5.2.3. Operación GetBikeManufacturer . . . 64

8. Conclusiones y trabajo futuro **66**
8.1. Conclusiones . . . 66
8.2. Trabajo futuro . . . 66

A. ATCAL - Gramática en BNF **68**

B. Relación de variables de especificación e implementación **71**
B.1. AddFriend_SP_2 . . . 71
B.2. AddFriend_SP_14 . . . 72

C. Registro civil **74**
C.1. Especificación Z . . . 74
C.2. Casos Abstractos de prueba . . . 77
C.2.1. CitizenRegistration . . . 77
C.2.2. ChangePersonDwelling . . . 78
C.2.3. AddChild . . . 80
C.2.4. CitizenDeath . . . 81
C.3. Implementación . . . 83
C.4. Leyes de refinamiento . . . 86
C.5. Casos concretos de prueba en Perl . . . 87
C.5.1. Operación CitizenRegistration . . . 87
C.5.2. Operación ChangePersonDwelling . . . 92
C.5.3. Operación AddChild . . . 95
C.5.4. Operación CitizenDeath . . . 97

D. Almacen de elementos **102**
D.1. Especificación Z . . . 102
D.2. Casos Abstractos de prueba . . . 103
D.2.1. AddInitialElement . . . 104
D.2.2. AddRelationToElement . . . 104
D.2.3. RemoveElement . . . 106
D.3. Implementación . . . 107
D.4. Leyes de refinamiento . . . 109
D.5. Casos concretos de prueba en Perl . . . 110
D.5.1. Operación AddInitialElement . . . 110
D.5.2. Operación AddRelationToElement . . . 111
D.5.3. Operación AddRelationToElement . . . 114

E. Rango etario **116**
E.1. Especificación Z . . . 116
E.2. Casos Abstractos de prueba . . . 117
E.2.1. Insert . . . 117
E.2.2. Update . . . 118
E.2.3. GetPeopleInRange . . . 119
E.2.4. Delete . . . 119
E.3. Implementación . . . 119
E.4. Leyes de refinamiento . . . 121
E.4.1. Identificador de regla . . . 121
E.4.2. Definición de tipos de datos . . . 121
E.4.3. Especificación de leyes de refinamiento . . . 122
E.4.4. Especificación de método a testear . . . 122
E.5. Casos concretos de prueba en Perl . . . 123
E.5.1. Operación Insert . . . 123
E.5.2. Operación Update . . . 125
E.5.3. Operación GetPeopleInRange . . . 126
E.5.4. Operación Delete . . . 127

F. Blog **128**
F.1. Especificación Z . . . 128
F.2. Casos Abstractos de prueba . . . 129
F.2.1. Insert . . . 129
F.2.2. Update . . . 130
F.2.3. Delete . . . 130
F.3. Implementación . . . 131
F.4. Leyes de refinamiento . . . 132

F.4.1. Identificador de regla . . . 132
F.4.2. Definición de tipos de datos . . . 132
F.4.3. Especificación de leyes de refinamiento . . . 133
F.4.4. Especificación de método a testear . . . 133
F.5. Casos concretos de prueba en Perl . . . 134
F.5.1. Operación Insert . . . 134
F.5.2. Operación Update . . . 136
F.5.3. Operación Delete . . . 137

G. Big Integer **139**
G.1. Especificación Z . . . 139
G.2. Casos Abstractos de prueba . . . 140
G.2.1. PlusOperation . . . 140
G.2.2. MinusOperation . . . 141
G.2.3. MultiplicationOperation . . . 141
G.2.4. DivisionOperation . . . 141
G.3. Implementación . . . 143
G.4. Leyes de refinamiento . . . 143
G.5. Casos concretos de prueba en Perl . . . 144
G.5.1. Operación PlusOperation . . . 144
G.5.2. Operaciones MinusOperation y MultiplicationOperation . . . 146
G.5.3. Operación DivisionOperation . . . 146

Capítulo 1

Introducción

La investigación entorno a la especificación de sistemas de software se ha acrecentado los últimos años y se debe a la importancia que tienen éstas tanto para la construcción como para el mantenimiento de sistemas. La característica que hace a las especificaciones un punto clave en el desarrollo de software es la de proveer las herramientas necesarias para formalizar los requerimientos funcionales, comúnmente expresados en lenguaje natural y, por lo tanto, muy propensos a contener ambigüedades.

La etapa de testing de software es otra de las áreas que se ve beneficiada por las especificaciones funcionales ya que, como se describe en [10], el testing basado en modelos (MBT la siglas de su traducción del inglés *Model-Based Testing*) permite la generación de casos de prueba a partir de una especificación del sistema que se desea testear. Para su tesina de grado Pablo Rodríguez Monetti [29] presenta una primera implementación de TTF (Test Template Framework) llamada FASTEST. Ésta herramienta es un framework de MBT que toma especificaciones en lenguaje Z como punto inicial del proceso. Esta implementación permite generar casos de prueba abstractos partiendo de una especificación Z.

En sus respectivas tesinas de grado Diego Ariel Hollmann [18] y Pablo Damián Coca [9] extienden FASTEST con módulos que permiten el refinamiento de casos de prueba abstractos, generados por FASTEST, a casos concretos en los lenguajes C [38] y Java [24], respectivamente. En su tesina Diego Ariel Hollmann presentó Test Case Refinement Language (TCRL de ahora en más), un lenguaje de refinamiento utilizado en ambos trabajos para establecer la correspondencia entre las variables de especificación y las de implementación.

En esta tesina se presenta el lenguaje de refinamiento ATCAL, creado por el actualmente graduado en Licenciatura en Ciencias de la Computación Cristian Rosa como parte de un trabajo realizado para un post-doctorado. A éste el autor hizo los agregados necesarios para completar su definición. En comparación con TCRL, se simplifican algunas de sus estructuras y se extiende su expresividad. Además, se implementa un nuevo módulo de FASTEST que refina los casos abstractos de prueba a casos concretos en Perl [28]. Junto con este desarrollo se presentan casos de estudio para ejemplificar el uso del lenguaje ATCAL como medio para describir los mapeos entre especificación funcional e implementación, además de los casos concretos generados, de modo tal que se hace posible realizar una comparación directa respecto a sus correspondientes casos abstractos.

En la figura 1.1 se puede apreciar la evolucionó FASTEST en relación a los módulos de refinamiento agregados desde su creación hasta la actualidad.

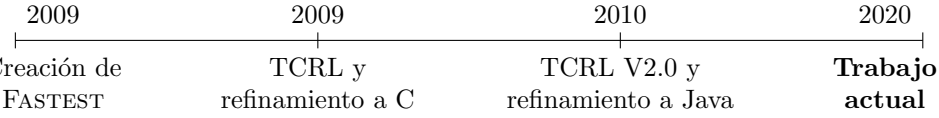

Figura 1.1: Evolución de FASTEST

Capítulo 2

Lenguaje de refinamiento

2.1. Introducción

A comienzos de los '70, en el ambiente de la ingeniería del software, se sabía que resultaba muy necesario hacer uso de los procesos de verificación entorno al desarrollo de software para qué, además de satisfacer los deseos de los clientes interesados, se logre que los productos finales cumplan con sus especificaciones. Si bien las técnicas de verificación estática, como el análisis de diagramas de diseño o de código fuente, fueron utilizadas en gran medida, la que predominó por encima de ellas fue el testing. El testing consiste en la ejecución de un determinado programa, bajo condiciones controladas, utilizando datos de entrada lo más reales posibles y analizando los resultados arrojados en la salida.

A fines de los '80, la utilidad de los métodos formales empleados como herramienta para especificar y diseñar sistemas de software era un tema ampliamente conocido. Además, se empezó a tener en cuenta la limitada utilidad de las especificaciones informales en el proceso de testing. En ese entonces, debido a que las especificaciones formales ya habían alcanzado un cierto grado de maduración y estabilidad, se empiezan a utilizar en conjunto con el testing de software y es así que se comienza a vislumbrar que las mismas pueden significar un gran aporte para dicho proceso.

Desde fines de los '80 a la actualidad, muchos autores fueron los que enfocaron sus esfuerzos en el estudio del testing de software, especialmente en el que utiliza las especificaciones formales como herramienta, ya que detectaron la importancia que esto tiene para la industria del software.

Teniendo en cuenta que el proceso de testing puede dividirse en una serie de pasos bien definidos, donde la mayoría son automatizables en gran medida, y considerando lo propuesto en los trabajos de Stocks [33], Hörcher y Peleska [19], y Stock y Carrington [32], se puede pensar que es posible construir una herramienta que permita automatizar o semi-automatizar el proceso de testing basado en especificaciones formales en el lenguaje Z.

En la actualidad, existe la implementación de una herramienta que aborda lo mencionado previamente, hasta la etapa de generación de casos abstractos de prueba. La misma se denominó FASTEST. Esta herramienta fue ideada por el director de esta tesina, Maximiliano Cristiá, y desarrollada en la tesis de grado de Pablo Rodríguez Monetti [11].

Tomando como base la implementación de FASTEST de ese entonces, Diego Ariel Hollmann, en su tesis de grado [18], desarrolló el lenguaje TCRL, que permite al usuario de la herramienta (de ahora en más *tester*) dar los detalles necesarios para establecer las reglas que indican cómo cada variable de especificación fue implementada. Luego, tomando los casos abstractos de prueba generados por FASTEST y utilizando dichas reglas, implementó un algoritmo capaz de traducir cada caso abstracto de prueba a uno concreto en el lenguaje de implementación C. Un año más tarde, Pablo Damián Coca mejoró algunos aspectos de TCRL, concibiendo así TCRL v2.0, y estableció un algoritmo que traduce cada caso de prueba abstracto generado por FASTEST a un caso de prueba concreto en el lenguaje de programación Java.

En esta tesina, se procede de una manera similar a la planteada por Pablo Damian Coca, sólo que en este caso se presenta como lenguaje de refinamiento a ATCAL que es una reformulación de TCRL v2.0. Haciendo uso de este lenguaje, se implementa un algoritmo cuyo objetivo es tomar los casos abstractos de prueba y generar para cada uno de ellos un caso concreto, es decir archivos ejecutables en el lenguaje elegido para implementar el

sistema especificado y el cual quiere someterse a testeo, en este caso Perl. La motivación de este trabajo es el hecho de que, actualmente, no se conoce ningún software que permita automatizar el proceso de traducción mencionado.

Por último, es importante destacar que la herramienta presentada en esta tesina no pretende ser de uso comercial, sino más bien un prototipo donde pueden verse aplicados los conceptos fundamentales previamente citados. Como trabajo futuro, agregando mejoras y completando ciertas funcionalidades, dicha herramienta puede comenzar a pensarse como un software orientado a la industria.

Capítulo 3

Testing funcional basado en especificaciones

3.1. Especificaciones formales

Al momento de afrontar el desarrollo de un sistema es de mucha utilidad contar con una descripción que contenga detalles acerca de las funcionalidades esperadas, con el fin de establecer metas claras. Estas descripciones también son conocidas como especificaciones y las mismas pueden presentarse tanto de manera informal como formal. Las especificaciones informales se caracterizan por hacer uso del lenguaje natural, mientras que las formales utilizan notación matemática, lo que permite describir el sistema de manera precisa y sin ambigüedades. Las principales ventajas de las especificaciones formales son:

- Proporcionan información detallada de los requisitos del software.
- Al plantearse con una notación matemática, pueden ser analizadas y se puede demostrar la consistencia y completitud de la especificación.
- Las especificaciones pueden usarse en el proceso de *testing* como una guía a la hora de generar casos de prueba para el sistema. Este punto será abordado más adelante.

La desventaja que presentan es que requieren un mínimo conocimiento técnico para hacer uso de las mismas. Es este el único punto donde las especificaciones informales se ven favorecidas.

Las especificaciones formales se enfocan en describir qué es lo que debe hacer el sistema, sin hacer mención sobre cómo hacerlo. De esta forma, se abstraen detalles de implementación que nada aportan al momento de comprender la funcionalidad del sistema en cuestión.

En la siguiente sección se presenta Z, un lenguaje de especificación que permite formalizar requerimientos de un sistema.

3.2. Lenguaje de especificación Z

Por el año 1980, **Jean Raymond Abrial** en conjunto con el grupo de investigación de la universidad de Oxford, presentaron Z [1], un lenguaje de especificación formal basado en la teoría de conjuntos y lógica de primer orden. También hace uso de construcciones denominadas esquemas para representar estados y operaciones. Estos esquemas contienen declaraciones de variables y predicados que establecen condiciones sobre esas variables, definiendo de esta forma, el estado del sistema.

Este lenguaje, en su forma más sencilla, puede representar máquinas de estado, pero también permite la representación de complejas jerarquías de máquinas de estado. Las máquinas de estado pueden definirse en dos etapas:

- **Definición del conjunto de estados:** en primer lugar se escribe un esquema donde se encuentran las variables que conforman el estado de la máquina. Es decir que se puede ver a un estado particular de la máquina como una tupla con valores específicos de esas variables. Por ejemplo, si una de las variables del estado pertenece a los números naturales, entonces la máquina tendrá una cantidad infinita de estados.

- **Definición de las operaciones:** una operación puede definirse como una serie de declaraciones de variables y propiedades sobre esas variables. Existen dos clases de operaciones:
  - Las que *producen una transición* en el estado: este tipo de operaciones indica cómo un estado de la máquina se transforma en otro.
  - Las que *no modifican el estado*: en este caso las operaciones son utilizadas para consultar el estado actual de la máquina.

Para que lo antedicho se entienda de una forma más clara, se presenta a continuación un ejemplo de la especificación de un sistema simplificado de registro de pacientes para un hospital: [1]

$[DNI, Name, Age]$

$Room ::= \mathbb{N}$

$Message ::= ok \mid alreadyRegisteredPatient$

Patient

$name : Name$
$age : Age$
$room : Room$

HospitalRegister

$patients : DNI \pfun Patient$

PatientRegistrationOk

$\Delta HospitalRegister$
$dni? : DNI$
$p? : Patient$
$output! : Message$

$dni? \notin \operatorname{dom} patients$
$patients' = patients \cup \{dni? \mapsto p?\}$
$output! = ok$

PatientPreviouslyRegistered

$\Xi HospitalRegister$
$dni? : DNI$
$output! : Message$

$dni? \in \operatorname{dom} patients$
$output! = alreadyRegisteredPatient$

$PatientRegistration == PatientRegistrationOk \lor PatientPreviouslyRegistered$

En primer lugar se definen los tipos de datos que se utilizarán en las dos etapas mencionadas previamente. Estos tipos pueden clasificarse en:

- **Básicos**: permiten abstraer tipos sobre los que no se quiere aportar ningún detalle como así también establecer sinónimos de otros tipos básicos o tipos predefinidos en Z.
- **Complejos**: se definen en base a los básicos y los provistos por Z.

[1] Es importante destacar que para pasos posteriores, como por ejemplo la generación de casos abstractos de prueba, la especificación provista a FASTEST será levemente modificada ya que no acepta sinónimos de tipos ni esquemas como tipos. Esto se debe a una faltante del programa.

En el ejemplo se puede observar cómo se proveen los tipos *DNI*, *Name* y *Age*, respecto a los cuales no se aporta ninguna información, aparte de la definición en sí. En cambio, si se observa el tipo *Room* se puede ver que se define como sinónimo de $\mathbb{N}$, con lo que se está restringiendo a que su implementación será de tipo numérico y con valores mayores a 0.

Luego, se presenta la definición de *Message*, un enumerado que puede tomar los valores *ok* y *alreadyRegisteredPatient*. Este tipo resulta útil para especificar la salida de la operación.

A continuación, se define el tipo complejo *Patient*. Como se mencionó anteriormente, en este caso se recurre a tipos básicos definidos previamente para construir otro utilizando un esquema que agrupa los datos relacionados a un paciente (o al menos los relevantes para el sistema).

Una vez que se han definido cada uno de los tipos, se procede con definiciones de nuevos esquemas, pero en este caso para definir tanto el estado del sistema como las operaciones. Por un lado, en la definición del estado, representado por el esquema *HospitalRegister*, se tiene que el mismo esta compuesto por una sola variable, *patients*, que es una función parcial. Esta función toma como entrada un DNI y retorna el paciente correspondiente a esa identificación. Por el otro, se presentan los esquemas que especifican cómo debe ser el comportamiento de la operación que se quiere describir, en este caso, *PatientRegistration*. El esquema *PatientRegistrationOk* establece que si la variable de entrada *dni*? no se encuentra en el dominio de *patients* (es decir, no es un paciente registrado en el sistema), entonces se agrega el mapeo $dni? \mapsto p?$ a *patients*, registrando de este modo al paciente. En el caso de *PatientPreviouslyRegistered*, al pertenecer *dni*? al dominio de *patients*, se sabe que el paciente ya fue cargado al sistema en algún momento y, por ende, no debe modificarse la variable de estado.

Los esquemas mencionados anteriormente representan operaciones parciales y es con la disyunción de estos que se define la operación total *PatientRegistration*.

Por convención, dentro de un esquema donde se especifica el comportamiento de una operación, las variables declaradas que terminan con *?* se consideran de entrada, en tanto que las que finalizan en *!* son de salida. En este ejemplo, *dni*? y *p*? son variables de entrada, mientras que *output*! es la variable de salida.

3.3. Testing funcional

Cuando un sistema se entrega al cliente, éste comienza a hacer uso de sus funcionalidades proveyendo al software los distintos valores de entradas que acepta y, en respuesta, el programa muestra la salida correspondiente. El proceso de *testing* busca simular el uso que darán los usuarios finales de la herramienta, sólo que en este caso no es el cliente quien observa la salida (la cual podría no ser la deseada), sino que es el tester quien analiza la misma y evalúa si es lo esperado o no. Debido a esto, el *testing* de software es una etapa crítica en todo el proceso de construcción del sistema y, si se es riguroso, puede significar un ahorro de tiempo importante.

Como las especificaciones formales definen detalles fundamentales de un sistema y el tester hace uso de las mismas para conocer su funcionalidad, se puede ver que guiar el proceso de *testing* en base a especificaciones formales hace que éste sea más simple, estructurado y riguroso que las técnicas estándar, como por ejemplo basarse en una especificación informal y generar un número acotado de casos de prueba, que se ven limitados por la creatividad y experiencia del tester.

Por otro lado, se considerará que un programa es correcto si verifica su especificación. Por esta razón, surge la necesidad de tener una especificación formal del programa que se quiere testear, ya que de lo contrario no se podrá corroborar, a ciencia cierta, si el resultado arrojado, en un determinado caso de prueba, es válido o no.

3.3.1. Formalización del proceso de testing

En el ámbito del *testing*, un programa se ve como una función que va del producto cartesiano de sus variables de entrada al producto cartesiano de sus variables de salida. Es decir:

$$P : ID \to OD$$

donde *ID* es el dominio de entrada (Input Domain) y *OD* es el dominio de salida (Output Domain). Los elementos de estos conjuntos son en realidad tuplas donde cada una de las

componentes es una de las variables de entrada/salida. Más específicamente:

$ID \triangleq [i_1 : A_1, \ldots, i_n : A_n]$
$OD \triangleq [o_1 : B_1, \ldots, o_m : B_m]$

donde:

- $A_1, \ldots, A_n$ son los tipos de las variables de entrada.
- $B_1, \ldots, B_m$ son los tipos de las variables de salida.

En base a esto, se presentan las siguientes definiciones:

Caso de prueba:

Un caso de prueba es un $\mathbf{x}$ tal que $x \in ID$. Por lo tanto, para testear P con este caso solo se debe calcular $\mathbf{P(x)}$.

Conjunto de prueba:

Un conjunto de prueba T es un conjunto conformado por casos de prueba tal que testear P con T, significa calcular $P(x)$ para cada $x \in T$.

Caso de prueba es exitoso:

Se dice que un caso de prueba es exitoso si detecta un error en el programa.

En relación a la última definición, es necesario destacar que para saber si determinado caso de prueba representa o no un error se debe recurrir a la especificación. Por lo tanto, surge el interrogante de saber cómo expresar los casos de prueba y el resultado de la ejecución de los mismos en el lenguaje de la especificación. Más adelante se presentarán algunos conceptos útiles que permitirán comprender cómo se relacionan el lenguaje abstracto (es decir, el lenguaje de especificación) con el concreto (es decir, el lenguaje de implementación).

Al igual que se consideró un programa como una función, se puede proceder de forma análoga con su correspondiente especificación Z. En este caso, si se considera que O_P es la especificación de P, entonces se puede definir una función parcial que va desde el espacio de estados de entrada IS (Input Space), al espacio de estados de salida OS (Output Space). Estos espacios se definen como:

$IS \triangleq [iv_1? : I_1, \ldots, iv_j? : I_j, s_1 : S_1, \ldots, s_l : S_l]$
$OS \triangleq [ov_1! : O_1, \ldots, ov_k! : O_k, s_1 : S_1, \ldots, s_l : S_l]$

donde:

- $iv_1, \ldots, iv_j$ son variables de entrada.
- $I_1, \ldots, I_j$ son los tipos de las variables de entrada.
- $ov_1, \ldots, ov_k$ son variables de salida.
- $O_1, \ldots, O_k$ son los tipos de las variables de salida.
- $s_1, \ldots, s_l$ son variables de estado.
- $S_1, \ldots, S_l$ son los tipos de las variables de estado.

Se dice que O_P es una función parcial ya que, en general, no todas las operaciones se especifican de manera total, o dicho de otra manera, puede darse el caso en que, para una determinada operación, se especifique su comportamiento para un subconjunto de todas las posibles combinaciones de variables de entrada y de estado. Por lo tanto, solo resulta útil testear el programa con los casos de prueba que acepta la especificación. Esto da lugar a presentar la definición de espacio válido de entrada (de ahora en más VIS, que proviene de sus siglas en ingles Valid Input Space). El VIS de una especificación O_P, es el subconjunto de IS que satisface la precondición de O_P, es decir:

$VIS_{O_P} \triangleq [IS \mid pre\ O_P]$

Tomando este conjunto, es posible definir O_P como una función total:

$O_P : VIS_{O_P} \rightarrow OS$

Para poder presentar de manera formal la noción de caso de prueba exitoso, es necesario definir una función que permita la traducción de elementos del VIS_{O_P} a elementos del ID_P, que concretice entradas abstractas, y otra que traduzca elementos de OD_P a OS_{O_P}, que abstraiga salidas concretas. Se define como sigue:

$concr_P^{O_P} : VIS_{O_P} \rightarrow ID_P$
$abs_P^{O_P} : OD_P \nrightarrow OS_{O_P}$

Por simplicidad, se llamará a las funciones del estilo de $concr_P^{O_P}$, funciones de refinamiento y a las del estilo de $abs_P^{O_P}$, funciones de abstracción.

En este momento, considerando que:

- P es un programa tal que $P : ID_P \rightarrow OD_P$.
- O_P la especificación Z de P tal que $O_P : VIS_{O_P} \rightarrow OS_P$.
- $t \in VIS_{O_P}$.
- $x = concr_P^{O_P}(t)$.

Con la información presentada hasta el momento es posible dar una definición mas precisa de caso de prueba exitoso:

Conjunto de prueba:

x es un caso de prueba exitoso para P si y solo sí
$O_P(t) \neq abs_P^{O_P}(P(x))$.

Dicho coloquialmente, la salida de ejecutar P con x como entrada no es lo que se esperaba según lo especificado.

En esta tesina se desarrolla un prototipo de herramienta que permite refinar casos de prueba abstractos, obtenidos a partir de una especificación Z, a casos de prueba concretos escritos en el lenguaje de programación Perl.

3.3.2. Etapas del testing funcional

Cuando se utiliza el testing funcional basado en especificaciones formales como medio para probar un programa P, teniendo su especificación O_P, pueden considerarse las cinco diferentes etapas mostradas en la Figura 3.1. Estas son:

1. **generación** de casos abstractos de prueba a partir de la especificación.
2. **refinamiento** del caso de prueba abstracto a uno concreto, es decir, en el lenguaje de implementación utilizado.
3. **ejecución** del caso concreto de prueba usando la implementación del sistema.
4. **abstracción** del resultado arrojado por el sistema.
5. **comparación** del resultado abstracto respecto a lo especificado.

Como se puede ver en la Figura 3.1, iniciando el proceso en O_P se puede obtener un caso de prueba abstracto t que será luego refinado al caso concreto x. Después, se ejecuta el sistema con x como entrada, arrojando $P(x)$ como resultado, que se abstrae a y, para permitir la comparación respecto a la especificación. En este punto ya es posible concluir si se encontró un error o no.

Si bien esta tesina se enfocará en la etapa dos, el autor considera relevante mencionar que, en la etapa 1, es de mucha utilidad recurrir a las *Tácticas de testing* que se presentan en [12]. A modo de ejemplo, a continuación se mencionan algunas de ellas:

- Forma Normal Disyuntiva (FND).

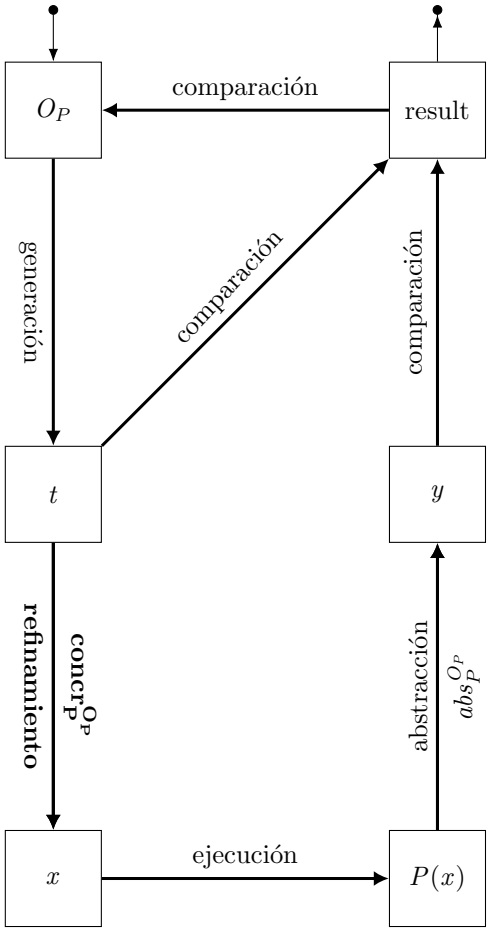

Figura 3.1: Etapas del proceso de testing funcional al usar una especificación

- Partición Estándar (PE).
- Propagación de. Sub Dominios (PSD)
- Mutación de Especificaciones (ME).

Las tácticas de testing se aplican al VIS generando así divisiones conocidas como clases de prueba y cada una de ellas representa una alternativa funcional según lo expresado por la especificación. Al generar estas clases, se pueden aplicar nuevamente las tácticas en ellas, generando nuevas subdivisiones. Esta iteración puede repetirse las veces que el tester considere necesarias para obtener la cantidad de casos de pruebas requeridos. De esta forma, las clases de prueba quedan relacionadas entre sí conformando lo que se conoce como **árbol de pruebas** [32].

3.3.3. Ejemplo de testing

En esta sección se presenta un ejemplo que muestra cómo, usando una de las tácticas de testing, se obtienen casos de prueba abstractos. La táctica es aplicada a la operación *PatientRegistration* de la especificación presentada en la Sección 3.2.

Debido a que la precondición de la operación es *true*, el *VIS* queda definido como sigue:

$$VIS_{PatientRegistration} \triangleq [patients : DNI \pfun Patient, dni? : DNI, p? : Patient]$$

Luego, al aplicar FND se obtienen las siguientes clases de prueba:

$$PatientRegistration_1^{FND} \triangleq [VIS_{PatientRegistration} \mid dni? \notin \operatorname{dom} patients]$$
$$PatientRegistration_2^{FND} \triangleq [VIS_{PatientRegistration} \mid dni? \in \operatorname{dom} patients]$$

Se debe tener en cuenta que, como fue mencionado previamente, en los siguientes ejemplos el esquema `Patient` es reemplazado por la tupla $Name \times Age \times \mathbb{N}$ ya que FASTEST, por una limitación, no soporta el uso de esquemas como tipos de datos.

Si bien, como fue mencionado con anterioridad, se pueden generar nuevas clases de prueba aplicando otras tácticas a las ya generadas, el autor considera que esto no es necesario por tratarse de un ejemplo. Luego, para cada clase de prueba se genera un caso abstracto de prueba:

$PatientRegistration_1^{FND}\_TCase$

$patients : DNI \pfun (Name \times Age \times \mathbb{N})$
$dni? : DNI$
$p? : Name \times Age \times \mathbb{N}$

$dni? \notin \operatorname{dom} patients$
$patients = \emptyset$
$dni? = DNIDniInput$
$p? = (NamePInput_1, AgePInput_2, 0)$

En este esquema se puede apreciar que en la sección superior se encuentran las declaraciones de las variables de estado y entrada, mientras que en la sección inferior se encuentran los predicados que definen los valores específicos para dichas variables. Por ejemplo, se puede ver que:

- `patients` toma el valor del conjunto vacío
- `dni?` es la cadena de caracteres *DNIDniInput*
- `p?` es la tupla de valores $(NamePInput_1, AgePInput_2, 0)$

$PatientRegistration_2^{FND}\_TCase$

$patients : DNI \pfun (Name \times Age \times \mathbb{N})$
$dni? : DNI$
$p? : Name \times Age \times \mathbb{N}$

$dni? \in \operatorname{dom} patients$
$patients = \{(DNIDniInput \mapsto (Name1108, Age1108, 1))\}$
$dni? = DNIDniInput$
$p? = (NamePInput_1, AgePInput_2, 0)$

En este caso se puede ver la misma declaración de variables, en tanto que en la sección los valores se mantienen salvo por la variable de estado `patients` que establece un mapeo entre *DNIDniInput* y la tupla de valores ($Name1108, Age1108, 1$)

Capítulo 4

Estudio del estado del arte

Desde la década de 1980, diversas investigaciones fueron realizadas dentro del área del testing que se centran en partir de especificaciones funcionales para generar los casos de prueba.

Entre los distintos trabajos investigados, algunos de ellos se basan en la creación de una envoltura entorno a la implementación del sistema, buscando así achicar la brecha semántica que existe entre el nivel de abstracción dado por la especificación y el de la implementación. Por otro lado, lo planteado en ciertos trabajos se centra en crear *scripts*, a partir de los casos abstractos de prueba y la implementación, para emular la ejecución de los casos de prueba utilizando el software implementado.

Un ejemplo del primer grupo es la arquitectura TorX presentada en [3]. Esta arquitectura permite relacionar distintas herramientas del ámbito de la generación y ejecución de pruebas para que, partiendo de una especificación formal escrita en LOTOS, Promela o SDL (soporta los tres lenguajes) y una implementación del sistema, se realicen pruebas sobre el mismo. También en este grupo se encuentra el caso de TGV [20] que, de manera aislada, genera casos abstractos de prueba. Estos casos de prueba representan tanto el comportamiento de las interacciones de entrada / salida entre el tester y la implementación que está siendo testeada como también propiedades acerca de ese comportamiento. Cabe destacar que es posible hacer uso de una herramienta especializada para transformar dichos casos abstractos de prueba en ejecutables, ya que esta tarea no fue considerada como parte de TGV.

Por otro lado, dentro del segundo grupo se encuentra el caso presentado en [5], donde se hace uso de la herramienta BZ-TT, que partiendo de una especificación en B, genera casos abstractos de prueba. Luego, utilizando la implementación[1] junto con un archivo que establece mapeos necesarios, permite la generación de *scripts* ejecutables en el lenguaje destino.

En [36], Jeremy Vanhecke, Xavier Devroey y Gilles Perrouin presentan AbsCon (Abstract test case Concretizer), un *plugin* que permite establecer un vínculo entre generadores de casos de pruebas abstractos y QTaste [30], una herramienta de escala industrial utilizada para ejecución de casos de prueba. Para este trabajo se utilizaron los casos abstractos de prueba generados por el *framework* VIBeS [15], que están compuestos por aseveraciones y acciones que realiza el sistema en cuestión. El proceso de concretización se divide en tres bloques para simplificar el mantenimiento:

- Interfaz de modelo del sistema: representa la interfaz mediante la cual se accede a los distintos componentes del sistema. Se define un archivo de código Python [26] que contiene métodos para acceder a dichos componentes.
- Mapeo entre aseveraciones/acciones y verificaciones/operaciones: cada aseveración se mapea a una verificación. De manera similar, cada acción tiene asociada una secuencia de operaciones. Vale la pena aclarar que las aseveraciones/acciones provienen de los casos abstractos y al mapearlas a verificaciones/operaciones se llevan al plano concreto.
- Mecanismo de mapeo de datos: el *framework* QTaste provee mecanismos para recuperar datos guardados en archivos CSV (*Comma Separated Values*). AbsCon hace uso de los mismos para tomar los valores almacenados y utilizarlos en la construcción de casos concretos.

[1] Esta implementación tiene la particularidad de contar con ciertas etiquetas que marcan sitios específicos del código donde luego se insertarán secuencias de invocaciones a operaciones

En “Atomic Action Refinement in Model Based Testing” [35], trabajo realizado en conjunto por Machiel Van Der Bijl y Arend Rensink, se presenta un *framework* que aplica el llamado refinamiento de acciones para obtener casos de prueba con un nivel suficiente de detalle como para que sean ejecutables. En este trabajo se muestran dos formas de generar casos concretos de prueba. Una de ellas sigue el camino de generar casos abstractos de prueba a partir de la especificación formal y luego refinar los mismos para obtener casos concretos (es decir, el mismo camino que se sigue en esta tesina), mientras que la otra sigue la estrategia de refinar el sistema y luego generar los casos concretos. Uno de los resultados de este trabajo demuestra que dichas opciones resultan equivalentes. Además, demuestran bajo qué condiciones el refinamiento de un conjunto completo de casos abstractos de prueba resulta en un conjunto completo de casos concretos de prueba.

En [4], un trabajo presentado por Sebastian Benz, se introduce AspectT, un lenguaje orientado a aspectos que permite la transformación de casos abstractos de prueba a *scripts* ejecutables. Estos tienen como objetivo el testeo de variadas características relacionadas al sistema que se quiere testear, y como dichos *scripts* pueden testear más de una de estas características, en este trabajo se hace uso de aspectos para encapsular cada una de esas características para luego poder reutilizarlas en los *scripts* generados.

Por otro lado, se puede destacar el conjunto de herramientas provisto en CADP (Construction and Analysis of Distributed Processes), un *framework* que ofrece distintos programas útiles para el diseño de sistemas concurrentes y distribuidos, entre otros. Una de las herramientas provistas es el compilador CAESAR que permite traducir una especificación funcional escrita en LOTOS a código en el lenguaje de programación C, con el fin de simular, verificar y testear un determinado software.

Por último, en [8], [6] y [7], Khusbu Bubna y Sujit Chakrabarti presentan la herramienta ACT (Abstract to Concrete Test) que, en base a un modelo Statecharts, logra generar casos de prueba concretos para el software Selenium RC [31] utilizando el *framework* de testing unitario JUnit [21]. Dichos casos de prueba no se obtienen de manera directa, sino que se debe seguir un proceso que, en líneas generales, se basa en las siguientes etapas:

1. Aplanar el modelo Statecharts.
2. Transformar el modelo Statecharts aplanado a un programa SMV [40], para luego ejecutarlo y obtener así los *test paths* [2].
3. Usar ejecución simbólica [22] para generar tanto valores de entrada abstractos como predicados de los *test path*.
4. Proveer los predicados de los *test paths* como entradas de un SMT Solver [14, 39] para obtener valores concretos de entrada.
5. En base a los *test paths* y usando los valores concretos de entrada se obtienen los casos de prueba abstractos.

Según las investigaciones realizadas por Pablo Coca en la sección 4.1 de su tesis de grado [9], y lo mencionado en los párrafos previos, se han desarrollado herramientas con el fin de automatizar la tarea de *testing* partiendo de especificaciones formales. Aún así, ninguna de ellas se basa en una especificación formal Z y genera casos de prueba concretos para el lenguaje de programación Perl. Contar con eso permitiría al tester probar los casos generados con una implementación Perl del sistema. Este último es el refinamiento presentado en esta tesina.

[2] Un *test path* es uno de los muchos caminos que puede recorrer el flujo del programa

Capítulo 5

Fastest

Como se mencionó a lo largo del Capítulo 3, los procesos de testing que sientan sus bases en las especificaciones formales permiten la detección de errores a nivel de implementación, sometiendo el sistema a pruebas y corroborando el resultado obtenido respaldándose en el comportamiento funcional especificado. En cierta medida, se puede decir que hacer uso de especificaciones permite formalizar el proceso de testing. Teniendo en vista este importante beneficio es que surge la necesidad de contar con una herramienta que permita simplificar algunos de los pasos del proceso de testing mostrado en la Figura 3.1. Por esta razón es que se crea FASTEST, una herramienta que tiene por finalidad generar casos abstractos de prueba a partir de una especificación Z, refinarlos, ejecutarlos utilizando la implementación del sistema y abstraer las salidas para permitir una comparación respecto a lo especificado.

La idea de crear esta herramienta surge del director de esta tesina, Maximiliano Cristiá, y actualmente se encuentra desarrollado un prototipo gracias al trabajo realizado en distintas tesinas de grado de la carrera "Licenciatura en Ciencias de la Computación"[1], perteneciente a la Universidad Nacional de Rosario[2]. Los graduados que desarrollaron sus tesinas alrededor del tema son:

- Pablo Rodríguez Monetti [29]
- Diego Ariel Hollmann [18]
- Pablo Coca [9]
- Pablo Albertengo [2]
- Joaquín Cuenca [13]
- Joaquín Mesuro [23]
- Julián Tomasi [34]

Como se mencionó previamente, en [18] Diego Ariel Hollmann agregó a FASTEST la funcionalidad de refinamiento de casos abstractos de prueba a casos concretos en el lenguaje de implementación C, utilizando el lenguaje de refinamiento TCRL para expresar las leyes que indican cómo realizar esa traducción. Luego, en [9], Pablo Coca hizo lo propio, añadiendo mejoras a TCRL, presentando de esta manera TCRL v2.0, y enfocando el refinamiento en el lenguaje de implementación Java. En esta tesina se procederá con un trabajo similar, pero en este caso presentando ATCAL, una reformulación de TCRL v2.0 que permite flexibilizar el sistema de tipos a utilizar en las leyes de refinamiento, y eligiendo Perl como lenguaje destino.

5.1. Diseño y detalles generales

Cuando se desarrolla un sistema medianamente complejo, es ideal diseñar previamente cómo será su arquitectura, ya que si se eligen correctamente los estilos arquitectónicos en los que se basará la estructura del software, se logrará un sistema flexible al momento de agregar componentes o modificar los ya existentes.

[1] http://www.fceia.unr.edu.ar/lcc/
[2] http://www.unr.edu.ar/

En el diseño de Fastest se ven involucrados tanto estilos arquitectónicos como patrones de diseño, logrando de esta forma el prototipo de una herramienta que se encuentra preparada para posibles cambios y el agregado de distintos módulos que permiten incluir nuevas funcionalidades. Entre los estilos arquitectónicos presentes en la herramienta se pueden destacar dos:

- *Cliente/Servidor*
- *Invocación Implícita*

Como el proceso de testing es una etapa en la que el consumo de recursos computacionales es elevado, el uso del estilo arquitectónico *Cliente/Servidor* permite aprovechar de manera eficiente los recursos provistos por una red de computadoras, ejecutando tareas en paralelo. También, posibilita la existencia de múltiples clientes de Fastest ejecutando al mismo tiempo.

La *Invocación Implícita* permite mantener desacoplados los componentes que realizan la llamada a un procedimiento de aquellos que conocen cómo realizar dicha tarea. Esto es posible debido a que un determinado componente puede anunciar uno o más eventos y otros componentes registrar su interés en los eventos que deseen, asociando procedimientos a cada uno de ellos. Entonces, cuando el sistema recibe el anuncio de un evento, invoca todos los procedimientos asociados al mismo, realizando las tareas que cada uno desempeña. De esta forma, al emplear *Invocación Implícita* en el diseño, se permite modificar componentes preexistentes o agregar nuevos sin que las demás interfaces se vean afectadas, facilitando en gran medida la evolución del sistema. Un claro ejemplo del beneficio que aporta el estilo es el componente de refinamiento presentado en esta tesina, que fue incluido en el sistema sin alterar el comportamiento de los demás componentes.

Por otro lado, si bien la elección del lenguaje utilizado para implementar Fastest podría haber sido, en principio, cualquier lenguaje de propósito general, en este caso, Pablo Rodriguez Monetti en [29], se inclinó por Java, principalmente por dos razones:

- Sigue un paradigma de programación orientada a objetos.
- Las herramientas que se presentan en el proyecto CZT [37] (Community Z Tools) fueron desarrolladas en Java. Estas herramientas incluyen soporte para editar, parsear, hacer chequeo de tipos (typechecking) e imprimir especificaciones Z en LaTeX, Unicode o formatos XML.

5.2. Uso de Fastest

Se utilizará el ejemplo de la sección 3.2 para mostrar el uso de Fastest en pos de generar casos de prueba abstractos a partir de una especificación funcional Z:

- Inicio de Fastest:

```
Fastest version 1.7, (C) 2019, Maximiliano Cristiá
Fastest>
```

- Cargar la especificación patientsRegistration.tex:

```
Fastest> loadspec patientsRegistration.tex
Loading specification..
Specification loaded.
Fastest>
```

  Se está suponiendo que el archivo que contiene la especificación se encuentra en el mismo directorio que el ejecutable de Fastest.

- Mostrar las operaciones cargadas:

```
Fastest> showloadedops
* PatientRegistrationOk
* PatientPreviouslyRegistered
* PatientRegistration
Fastest>
```

Estas operaciones son las incluidas en la especificación funcional.

- Selección de la operación que se desea testear. En la especificación en cuestión se dispone de PatientRegistration únicamente debido a que se trata de un ejemplo acotado:

```
Fastest> selop PatientRegistration
Fastest>
```

- Generación del árbol de pruebas:

```
Fastest> genalltt
Generating test tree for 'PatientRegistration' operation.
Fastest>
```

- Mostrar el árbol de pruebas en su estado actual:

```
Fastest> showtt
PatientRegistration_VIS
  |_______PatientRegistration_DNF_1
  |_______PatientRegistration_DNF_2
Fastest>
```

- Generación de casos de prueba abstractos:

```
Fastest> genalltca
PatientRegistration_DNF_1 test case generation -> SUCCESS.
PatientRegistration_DNF_2 test case generation -> SUCCESS.
Fastest>
```

En este punto se puede ver que se han generado exitosamente dos casos abstractos de prueba a partir de las particiones generadas en base a la táctica de testing conocida como Forma Normal Disyuntiva, de su traducción al inglés es que provienen las siglas DNF (Disjuntive Normal Form). Si ahora se visualiza el estado del árbol de pruebas se puede ver cómo cada caso abstracto se corresponde con una partición:

```
Fastest> showtt
PatientRegistration_VIS
  |_______PatientRegistration_DNF_1
  |   |_______PatientRegistration_DNF_1_TCASE
  |
  |_______PatientRegistration_DNF_2
      |_______PatientRegistration_DNF_2_TCASE
Fastest>
```

- Mostrar los esquemas de los casos abstractos de prueba:

```
Fastest> showsch -tca -u 2
\begin{schema}{PatientRegistration\_ DNF\_ 1\_ TCASE}\\
 patients : DNI \pfun ( Name \cross Age \cross \nat ) \\
 dni? : DNI \\
 p? : Name \cross Age \cross \nat
\where
 dni? \notin \dom patients \\
 patients =~\emptyset \\
 dni? = dNI3 \\
 p? = ( name1 , age2 , 0 )
\end{schema}


\begin{schema}{PatientRegistration\_ DNF\_ 2\_ TCASE}\\
 patients : DNI \pfun ( Name \cross Age \cross \nat ) \\
 dni? : DNI \\
 p? : Name \cross Age \cross \nat
\where
 dni? \in \dom patients \\
 patients = \{ ( dNI1 \mapsto ( name1 , age1 , 2 ) ) \} \\
 dni? = dNI1 \\
 p? = ( name3 , age4 , 0 )
\end{schema}
Fastest>
```

Se puede observar que los ejemplos aquí obtenidos son, en esencia, muy similares a los generados en la sección 3.3.3 aplicando de forma manual la táctica FND.

Más adelante, se mostrará cómo, tomando como base los casos de prueba abstractos que pueden generarse con FASTEST, se puede proceder a la etapa de refinamiento con el fin de obtener los casos de prueba concretos en el lenguaje de programación Perl.

5.3. ATCAL

ATCAL es un lenguaje de refinamiento creado por el actualmente graduado en Licenciatura en Ciencias de la Computación Cristian Rosa, como parte de un trabajo realizado para un post-doctorado y al que el autor hizo los agregados necesarios para completar su definición. Este lenguaje tiene sus bases en el lenguaje TCRL v2.0 presentado en [9], trabajo en el que Pablo Coca propone extender y mejorar el lenguaje TCRL, planteado por Diego Ariel Hollmann en [18]. Por su parte, ATCAL presenta como principal aporte la nueva sección avocada únicamente a la declaración de tipos de datos que se utilizarán en el refinamiento. La misma reemplaza a las estructuras de datos predefinidas en TCRL, cuya sintaxis carece de expresividad para la definición de nuevas estructuras. Si bien en esta sección se pueden encontrar algunas estructuras preestablecidas para la construcción de tipos básicos, como por ejemplo enumerados, registros y arreglos, también está disponible la estructura que permite la declaración de tipos de datos con contrato. Esta estructura hace posible definir tipos de datos con los siguientes elementos:

- Constructor: es el método utilizado para crear elementos del tipo de datos que se está definiendo.
- Setter: permite asignar valores a variables propias del tipo de dato.
- Getter: permite obtener el valor de las variables propias del tipo de dato.

Al igual que sus predecesores, TCRL y TCRL v2.0, ATCAL es un lenguaje que permite dar los detalles para describir cómo traducir los elementos de una especificación a su contraparte en el lenguaje de implementación elegido como destino. Al momento de diseñar este lenguaje, una de las metas que se planteó fue que tanto el lenguaje con el cual se especifica funcionalmente el sistema, como el utilizado para proveer la implementación del mismo, sean genéricos. Esto posibilita que en un futuro se agregue, a la implementación del proceso de refinamiento, soporte para distintos tipos de lenguajes de especificación y también nuevos lenguajes de implementación. De esta manera, se permite ampliar el espectro de aplicación del proceso de refinamiento a una mayor cantidad de escenarios, imposibles de representar en caso de haber planteado un diseño para un lenguaje de especificación e implementación específicos. Dicho esto, es importante destacar que para esta tesina se consideró que las especificaciones a partir de las cuales se generan los casos de prueba abstractos se escriben en el **lenguaje de especificación Z** y que el lenguaje de implementación en el que se expresarán los casos de prueba concretos es **Perl** [28].

Por otra parte, un detalle no menor a tener en consideración es que la implementación del proceso de refinamiento a Perl es sólo un prototipo y, si bien se busca representar un gran número de casos, no pretende abarcar la totalidad de ellos ya que se escapa del alcance de la actual tesina.

5.4. ATCAL - Gramática y descripción

En esta sección se utilizará la notación BNF para detallar las distintas partes que conforman la gramática de ATCAL. Además, se proveerá de una descripción para cada una de estas partes. La gramática en su totalidad puede encontrarse en el apéndice ATCAL - Gramática en BNF. Algunos detalles a tener en cuenta al momento de analizar la gramática son:

- Las palabras en mayúsculas son palabras reservadas de ATCAL.
- El lenguaje es case sensitive, es decir, distingue entre mayúsculas y minúsculas.
- Se utiliza el símbolo '?', no incluido en BNF, como herramienta adicional para la definición de la gramática. La semántica de este símbolo es que la expresión a su izquierda puede incluirse ó no. Más adelante se verá su utilidad.

5.4.1. Regla de refinamiento

Una regla de refinamiento tiene la siguiente estructura

⟨refinementRule⟩ ::= @RRULE ⟨id⟩
⟨preamble⟩?
⟨datatypes⟩?
⟨laws⟩
⟨plCode⟩?
⟨uut⟩
⟨epilogue⟩?

Los elementos gramaticales utilizados en `refinementRule` se detallan en las siguientes secciones.

5.4.1.1. @RRULE

@RRULE ⟨id⟩

Se usa para representar al identificador o nombre de la regla de refinamiento.

5.4.1.2. preamble

⟨preamble⟩ ::= @PREAMBLE (⟨plCode⟩ | ⟨id⟩.@PREAMBLE;)*

Se utiliza para proveer código fuente, escrito en el lenguaje de implementación destino, que debe ser cargado antes que el código generado por el refinamiento. En general, el código de esta sección se utiliza para:

- Incluir bibliotecas necesarias para el correcto funcionamiento del sistema.
- Declaración de variables.
- Definición de clases y/o módulos.

Como se puede ver, es posible incluir preámbulos de otras reglas de refinamiento. Un ejemplo de esto último se muestra a continuación:

```
1  @RRULE refinementRule1
2
3  @PREAMBLE
4      @CODESTART
5          #!/usr/bin/perl
6          use feature qw(say);
7          use Data::Dumper;
8      @CODEEND
9      . . .
10
11 @RRULE refinementRule2
12
13 @PREAMBLE
14     refinementRule1.@PREAMBLE;
15     @CODESTART
16         use bigint;
17     @CODEEND
18     . . .
```

5.4.1.3. datatypes

⟨datatypes⟩ ::= @DATATYPES ⟨typeDecs⟩$^+$
⟨typeDecs⟩ ::= DATATYPE ⟨id⟩ = ⟨type⟩;
⟨type⟩ ::= ⟨id⟩
| INT
| FLOAT
| STRING
| REFERENCE
| ARRAY ⟨type⟩(⟨number⟩)
| ENUM ⟨id⟩⟨args⟩
| RECORD ⟨id⟩ (⟨id⟩ : ⟨type⟩ (,⟨id⟩ : ⟨type⟩)*)
| (MODULE ⟨string⟩)? CONSTRUCTOR ⟨id⟩ ⟨contractMembers⟩
SETTER ⟨id⟩ ⟨contractMembers⟩
GETTER ⟨id⟩ ⟨contractMembers⟩

Haciendo uso del no terminal ⟨*datatypes*⟩ se define la sección donde se declararán tipos de datos que luego serán utilizados al escribir leyes de refinamiento. Si bien se puede prescindir de esta sección, ya que es posible escribir los tipos de datos al momento de dar las leyes de refinamiento, al definir todos los tipos en un único lugar simplifica la comprensión a la hora de interpretar las leyes y además permite la reutilización de los tipos definidos, lo cual no sería posible si se escribiese el tipo cada vez que se lo necesite.

5.4.1.4. laws

⟨laws⟩ ::= @LAWS (⟨law⟩;)*
⟨law⟩ ::= (⟨id⟩ :)? (⟨biRefLaw⟩ | ⟨lawRefinement⟩ | ⟨lawReference⟩)
⟨biRefLaw⟩ ::= ⟨id⟩ <==> ⟨refinement⟩(, ⟨refinement⟩)*
⟨lawRefinement⟩ ::= ⟨zExprs⟩ ==> ⟨refinement⟩(, ⟨refinement⟩)*
⟨lawReference⟩ ::= ⟨id⟩.(@LAWS | ⟨id⟩)

El no terminal ⟨*laws*⟩ es el que se utiliza para definir la sección donde se encuentran las leyes de refinamiento. Estas leyes permiten establecer la relación que vincula los elementos de la especificación con los de la implementación. Como se puede ver, existen tres tipos de leyes:

- ⟨*biRefLaw*⟩: establecen una traducción bidireccional entre los elementos de la especificación y los de la implementación.
- ⟨*lawRefinement*⟩: establecen cómo traducir los elementos de la especificación a los de la implementación.
- ⟨*lawReference*⟩: hacen uso de leyes definidas en reglas previamente cargadas.

En las siguientes secciones se incluirán más detalles sobre estas construcciones y su utilidad en la tarea de describir las traducciones necesarias para concretar con éxito el refinamiento.

5.4.1.5. refinement

Debido a que la semántica de los dos primeros tipos de leyes depende en gran parte del no terminal ⟨*refinement*⟩, se tratará este último con el fin de simplificar la explicación en las

posteriores secciones. La definición gramatical de dicho no terminal se expone a continuación:

$$
\begin{array}{rcl}
\langle \text{refinement} \rangle & ::= & \langle \text{lvalue} \rangle \text{ AS } \langle \text{type} \rangle \ \langle \text{constMapping} \rangle ? \langle \text{withRef} \rangle ? \\
\langle \text{lvalue} \rangle & ::= & \langle \text{id} \rangle \\
 & & \mid \ [\langle \text{number} \rangle ?] \\
 & & \mid \ .\langle \text{id} \rangle \\
 & & \\
\langle \text{withRef} \rangle & ::= & \text{WITH } [\langle \text{lawRefinement} \rangle \ (, \langle \text{lawRefinement} \rangle)^*] \\
\langle \text{constMapping} \rangle & ::= & \text{MAP } [\langle \text{constMap} \rangle \ (, \langle \text{constMap} \rangle)^*] \\
\langle \text{constMap} \rangle & ::= & \langle \text{id} \rangle \text{ -> } (\langle \text{id} \rangle \mid \langle \text{string} \rangle \mid \langle \text{number} \rangle)
\end{array}
$$

- $\langle lvalue \rangle$: permite indicar cuál es el elemento correspondiente a la implementación que se utilizará en la ley de refinamiento que se está procesando. Sus opciones son:
  - $\langle id \rangle$: utilizado para identificar una variable de implementación.
  - $[\langle number \rangle ?]$: se emplea para indicar que se refinará a un elemento de un arreglo previamente definido. Notar que para poder escribir este tipo de expresiones es necesario que esta ley se encuentre dentro de la definición de otra más general donde se especifique que el refinamiento involucra una variable de tipo arreglo. Hay dos opciones al utilizar este tipo de construcción:
    - [n]: para indicar que lo refinado se almacenará en la n-ésima posición del arreglo.
    - [ ]: para indicar que las posiciones se tomarán a medida que se vayan completando. Es decir, al procesar una variable de especificación que se refine como un arreglo, el primer elemento tomará la primera posición, el segundo la segunda posición y así sucesivamente.
  - $.\langle id \rangle$: para indicar que lo refinado se almacenará en el miembro de un record que se definió previamente. De manera similar al item anterior, esta expresión solo puede ser usada en caso que la misma se encuentre dentro de otra donde se especifique que el refinamiento involucra una variable de tipo registro.

5.4.1.6. biRefLaw

$$\langle \text{biRefLaw} \rangle \quad ::= \quad \langle \text{id} \rangle \ \text{<==>} \ \langle \text{refinement} \rangle (, \langle \text{refinement} \rangle)^*$$

Esta ley es utilizada cuando se quiere vincular de forma bidireccional un par de elementos donde uno de ellos pertenece a la especificación y el otro corresponde a la implementación. Al relacionar elementos de ambos dominios de este modo, se tiene la información necesaria para conocer la traducción de un elemento de la especificación en uno de la implementación y viceversa, lo que resulta útil en la etapa conocida como abstracción. Dicha etapa requiere que haya finalizado el proceso de refinamiento y se ejecute el caso concreto de prueba, y la misma consiste en abstraer el resultado de la ejecución del caso concreto para expresarlo en el lenguaje de especificación utilizado para describir el sistema.

Cabe destacar que el análisis y desarrollo de dicha etapa va más allá del alcance de esta tesina y, por tal motivo, se le dará poca relevancia en las demás secciones.

5.4.1.7. lawRefinement

$$\langle \text{lawRefinement} \rangle \quad ::= \quad \langle \text{zExprs} \rangle \ \text{==>} \ \langle \text{refinement} \rangle (, \langle \text{refinement} \rangle)^*$$

Mediante esta ley se darán los detalles de cómo realizar cada una de las traducciones. Para representar los elementos de la especificación se hará uso del no terminal $\langle zExprs \rangle$.

5.4.1.8. lawReference

$$\langle \text{lawReference} \rangle \quad ::= \quad \langle \text{id} \rangle.(\text{@LAWS} \mid \langle \text{id} \rangle)$$

Utilizando este tipo de leyes se pueden reutilizar leyes definidas en reglas de refinamiento cargadas anteriormente. Por ejemplo, si las leyes de refinamiento de la regla `R1` son como sigue:

```
@RRULE R1

@LAWS
    L1 : specVar1 ==> impVar1 AS type1;
    L2 : specVar2 ==> impVar2 AS type2;
    ...
```

entonces, en una regla `R2`, se pueden reutilizar las leyes de la regla `R1` del siguiente modo:

```
@RRULE R2

@LAWS
    ...
    R1.L1;
    ...
```

y en caso de querer incluir todas las reglas de `R1` se puede optar por

```
@RRULE R2

@LAWS
    ...
    R1.@LAWS;
    ...
```

que equivale a incluir las leyes `L1` y `L2` de `R1`.

Es importante destacar que para poder reutilizar leyes de otras reglas, éstas deben haber sido cargadas previamente en FASTEST utilizando el comando `loadrefrule`, de otro modo el proceso de refinamiento informará el correspondiente error.

5.4.1.9. zExprs

$$\begin{aligned}
\langle \text{zExprs} \rangle \quad ::= \quad & \langle \text{zExpr} \rangle \ (==> \ \langle \text{zExprs} \rangle)? \\
\langle \text{zExpr} \rangle \quad ::= \quad & \langle \text{id} \rangle \\
& \mid \langle \text{number} \rangle \\
& \mid \langle \text{string} \rangle \\
& \mid \langle \text{auto} \rangle \\
& \mid \langle \text{elem} \rangle \\
& \mid \langle \text{zExpr} \rangle.\langle \text{tupProj} \rangle \\
& \mid < \langle \text{zExpr} \rangle (, \langle \text{zExpr} \rangle)^* > \\
& \mid \langle \text{zExpr} \rangle.\langle \text{dom} \rangle \\
& \mid \langle \text{zExpr} \rangle.\langle \text{ran} \rangle \\
& \mid \langle \text{zExpr} \rangle.\langle \text{proj} \rangle \\
& \mid \langle \text{zExpr} \rangle \ \langle \text{inter} \rangle \ \langle \text{zExpr} \rangle \\
& \mid \langle \text{zExpr} \rangle \ \langle \text{union} \rangle \ \langle \text{zExpr} \rangle \\
& \mid \langle \text{zExpr} \rangle \ \langle \text{diff} \rangle \ \langle \text{zExpr} \rangle \\
& \mid \{ \langle \text{zExpr} \rangle (, \langle \text{zExpr} \rangle)^* \} \\
& \mid \langle \text{zExpr} \rangle.\langle \text{card} \rangle \\
& \mid \langle \text{zExpr} \rangle \ \langle \text{mul} \rangle \ \langle \text{zExpr} \rangle \\
& \mid \langle \text{zExpr} \rangle \ \langle \text{div} \rangle \ \langle \text{zExpr} \rangle \\
& \mid \langle \text{zExpr} \rangle \ \langle \text{mod} \rangle \ \langle \text{zExpr} \rangle \\
& \mid \langle \text{zExpr} \rangle \ \langle \text{plus} \rangle \ \langle \text{zExpr} \rangle \\
& \mid \langle \text{zExpr} \rangle \ \langle \text{minus} \rangle \ \langle \text{zExpr} \rangle \\
& \mid \langle \text{zExpr} \rangle \ ++ \ \langle \text{zExpr} \rangle \\
& \mid (\langle \text{zExpr} \rangle)
\end{aligned}$$

Este conjunto de reglas gramaticales permite construir una amplia variedad de expresiones para representar elementos de la especificación. Estas expresiones forman parte de las leyes de refinamiento indicando sobre qué elemento de la especificación aplica cada ley. A continuación se detalla el uso de cada elemento gramatical:

- $\langle id \rangle$: se utiliza para representar identificadores.
- $\langle number \rangle$: se utiliza para representar números.
- $\langle string \rangle$: se utiliza para representar cadenas de caracteres.
- $\langle auto \rangle$: se utiliza para indicar valores que deben auto-completarse a nivel de implementación. Esta ley será utilizada en casos que el especificador haya decidido omitir detalles sobre determinadas partes de la implementación o, simplemente, cuando una estructura a nivel de implementación cuente con más elementos que a nivel de especificación. Por ejemplo, supóngase un sistema que se utilizará para almacenar la relación de una persona con sus familiares. Lo que sigue es parte de la especificación:

  $[DNI, NAME, AGE]$

  Family
  $\ldots$
  $relatives : DNI \pfun Person$

  Person
  $name : NAME$
  $age : AGE$

  Supongamos además, que en la implementación se tiene en cuenta un campo alfanumérico para cada persona usado internamente como identificación. Al escribir las leyes de refinamiento esto puede expresarse así:

```
@RRULE refinementRuleName

@DATATYPES
    DATATYPE PersonImpl = RECORD PersonImpl (nameImpl : STRING, ageImpl :
         INT, serialId : STRING);
    DATATYPE TUPLE = RECORD TUPLE (first : INT, second: PersonImpl);
     ...
 ...
@LAWS
    relatives ==> relativesImpl AS ARRAY TUPLE (20) WITH [
        relatives.@DOM ==> .first AS INT,
        relatives.@RAN ==> .second AS PersonImpl WITH [
            name ==> nameImpl AS STRING,
            age ==> ageImpl AS INT,
            @AUTOFILL ==> serialId AS STRING
        ]
    ]
 ...
```

  De este modo, cada elemento del rango de `relatives` se asociará con una variable del tipo `PersonImpl`. Vemos que se establece una relación directa entre los dos miembros del tipo `Person` y sus correspondientes en el tipo de la implementación. Además, como en la implementación se agrega un tercer miembro, cuyo rol no fue especificado, al escribir las leyes de refinamiento se indica que debe auto-completarse.

  Los valores de auto-completado se definen en código y quedan a decisión del implementador.

- $\langle elem \rangle$: se utiliza para representar de manera genérica un elemento de un conjunto específico C, tal que $C : \mathbb{P}(\mathbb{P}\, X)$ o $C : \mathbb{P}(\, Y_1 \times Y_2 \times \ldots Y_n)$, para tipos cualesquiera X, Y_1, $Y_2, \ldots Y_n$. Por ejemplo, dada la variable de especificación. *relationGroups* : $\mathbb{P}(\mathbb{P}(DNI \times DNI \times RelationType))$, donde *DNI* es un tipo de datos básico y *RelationType* es un enumerado cuyos elementos son *BloodLinked* e *InLaw*. Si se quisiese refinar *relationGroups* como un array de arrays de ternas, si no existiese $\langle elem \rangle$, lo único que se puede especificar es:

```
@RRULE refinementRuleName

@DATATYPES
    DATATYPE Relation = RECORD Relation (dni1: INT, dni2: INT, type:
        RelationType);
    DATATYPE RelationsArray = ARRAY Relation (15);
    DATATYPE RelationGroupsArray = ARRAY RelationsArray (10);
    DATATYPE RelationType = ENUM RelationType (BloodLinkedImpl, InLawImpl
        );
     ...
 ...
@LAWS
    relationGroups ==> relationGroupsImpl AS RelationGroupsArray;
 ...
```

En cambio, utilizando este elemento gramatical se logra describir el refinamiento deseado sin inconvenientes:

```
@RRULE refinementRuleName

@DATATYPES
    DATATYPE Relation = RECORD Relation (dni1: INT, dni2: INT, type:
        RelationType);
    DATATYPE RelationsArray = ARRAY Relation (15);
    DATATYPE RelationGroupsArray = ARRAY RelationsArray (10);
    DATATYPE RelationType = ENUM RelationType (BloodLinked, InLaw);
     ...
 ...
@LAWS
    relationGroups ==> relationGroupsImpl AS RelationGroupsArray WITH [
        @ELEM ==> relGroupElem AS RelationArray WITH [
            @ELEM ==> relationElem AS Relation WITH [
                @ELEM.#1 ==> .dni1 AS INT,
                @ELEM.#2 ==> .dni2 AS INT,
                @ELEM.#3 ==> .type AS RelationType MAP [
                    BloodLinked -> BloodLinkedImpl,
                    InLaw -> InLawImpl
                ]
            ]
        ]
    ]
 ...
```

donde *relGroupElem* es una variable auxiliar utilizada para almacenar el valor de un elemento de *relationGroups*. Un detalle importante que no se debe pasar por alto es el papel que juega $@ELEM$ en cada nivel. Se puede observar que en las líneas 12 y 13, como el *scope* superior se refina como un arreglo, cada ocurrencia de $@ELEM$ representa de manera genérica a los elementos del arreglo del *scope* superior. Por otro lado, en las líneas 14, 15 y 16, como en el *scope* superior se indica que $@ELEM$ se refina como un registro, entonces $@ELEM.\#N$ hace referencia al elemento N-ésimo de dicho registro. En la Sección E.4, correspondiente al caso de estudio Rango etario, se puede ver otro ejemplo de uso de $@ELEM$.

- $\langle zExprs \rangle.\langle tupProj \rangle$: se utiliza para acceder a un miembro de una tupla. En el item anterior se puede ver su uso en expresiones como $@ELEM.\#1$.
- $< \langle zExprs \rangle (, \langle zExprs \rangle)^* >$: permite la construcción de tuplas constantes.
- $\langle zExprs \rangle.\langle dom \rangle$: $e.@DOM$ se utiliza para obtener el dominio e, por lo tanto el tipo de e debe ser de la forma $\mathbb{P}(X \times Y)$, para cualquier par de tipos X e Y.
- $\langle zExprs \rangle.\langle ran \rangle$: $e.@RAN$ se utiliza para obtener el rango e, por lo tanto el tipo de e debe ser de la forma $\mathbb{P}(X \times Y)$, para cualquier par de tipos X e Y.
- $\langle zExprs \rangle.\langle proj \rangle$: p.@n es utilizado para obtener el conjunto n-ésimo de p, por lo tanto p debe ser del tipo $\mathbb{P}(X_1 \times X_2 \times \cdots \times X_m)$, donde $n \leq m$.
- $\langle zExprs \rangle \langle inter \rangle \langle zExprs \rangle$: permite representar la intersección de dos conjuntos.
- $\langle zExprs \rangle \langle union \rangle \langle zExprs \rangle$: permite representar la unión de dos conjuntos.

- ⟨*zExprs*⟩⟨*diff*⟩⟨*zExprs*⟩: permite representar la diferencia de dos conjuntos.
- { ⟨*zExprs*⟩(, ⟨*zExprs*⟩)* }: permite la construcción de conjuntos por extensión.
- ⟨*zExprs*⟩.⟨*card*⟩: *setExpr*.@*CARD* se utiliza para representar la cardinalidad de *setExpr*, con lo cual se debe cumplir que $setExpr : \mathbb{P}\, X$ para algún tipo X.
- ⟨*zExprs*⟩⟨*mul*⟩⟨*zExprs*⟩: permite representar la multiplicación entre dos números.
- ⟨*zExprs*⟩⟨*div*⟩⟨*zExprs*⟩: permite representar la división entre dos números.
- ⟨*zExprs*⟩⟨*mod*⟩⟨*zExprs*⟩: permite representar el módulo de un número respecto a otro.
- ⟨*zExprs*⟩⟨*plus*⟩⟨*zExprs*⟩: permite representar la suma entre dos números.
- ⟨*zExprs*⟩⟨*minus*⟩⟨*zExprs*⟩: permite representar la resta entre dos números.
- ⟨*zExprs*⟩ ++ ⟨*zExprs*⟩: permite representar la concatenación de dos cadenas de texto.
- (⟨*zExprs*⟩): se utiliza para agrupar expresiones.

5.4.1.10. epilogue

⟨epilogue⟩ ::= @EPILOGUE (⟨plCode⟩ | ⟨id⟩.@EPILOGUE;)$^+$

Se utiliza para proveer código fuente, escrito en el lenguaje de implementación destino, que debe ser cargado luego del código generado por el refinamiento. En general, el código de esta sección se utiliza para:

- Liberar memoria de variables utilizadas durante el refinamiento.
- Ejecutar funciones luego de finalizar el código generado por el refinamiento.

Como puede verse, de manera similar a la regla gramatical ⟨*preamble*⟩, es posible incluir epílogos de otras reglas de refinamiento.

5.4.1.11. uut

⟨uut⟩ ::= @UUT ⟨id⟩ ⟨args⟩;
| ⟨id⟩ <== @UUT ⟨id⟩ ⟨args⟩ AS ⟨type⟩;

La regla de esta sección se utilizará para indicar cuál es la unidad que se desea testear, de esto surge la sigla UUT, que proviene de su traducción al ingles Unit Under Test. Estas unidades, en general, son funciones o métodos pertenecientes a la implementación del sistema. Por ejemplo, como en esta tesina se desarrolla el refinamiento de casos abstractos de prueba a casos concretos en Perl, las unidades sometidas a prueba son métodos.

Como puede verse, hay dos formas de especificar qué unidad se someterá a pruebas. A continuación se explica cada una de ellas:

- @*UUT* ⟨*id*⟩ ⟨*args*⟩: en este caso se indica que se debe someter a pruebas al método identificado por ⟨*id*⟩ y con ⟨*args*⟩ como argumentos.
- ⟨*id*⟩ <== @*UUT* ⟨*id*⟩ ⟨*args*⟩ *AS* ⟨*type*⟩: este otro caso se comporta de manera similar al anterior, salvo que se indica que el resultado del método tiene ⟨*type*⟩ como tipo de salida y debe almacenarse en la variable identificada por la primer ocurrencia de ⟨*id*⟩[3].

[3]La segunda de las ocurrencias (la que se encuentra a la derecha del terminal @*UUT*) es la que identifica al nombre del método

5.5. Parser de ATCAL

5.5.1. ANTLR

ANTLR (ANother Tool for Language Recognition)[27] es un poderoso generador de parsers para lectura, procesamiento, ejecución o traducción de texto estructurado y/o archivos binarios. En particular, uno de sus usos consiste en la generación de parsers de un lenguaje partiendo de una gramática que lo describa. A partir de este punto se puede, por ejemplo, implementar un intérprete de dicho lenguaje.

En esta tesina, ANTLR se utilizó para generar el parser del lenguaje de refinamiento ATCAL en base a la gramática detallada en el apéndice ATCAL - Gramática en BNF. Es importante destacar que las gramáticas ingresadas a ANTLR como entrada tienen una sintaxis muy similar a BNF (Backus-Naur Form) [41], salvo por algunas diferencias. Dicha notación ofrece los siguientes elementos para construir gramáticas:

- **:** se utiliza para la definición. La expresión de la izquierda se define como la expresión de la derecha.
- | sirve para expresar diferentes alternativas, de las cuales sólo una puede elegirse.
- * permite indicar que la expresión a la izquierda del * puede darse 0 o más veces.
- + permite indicar que la expresión a la izquierda del + puede darse una o más veces.
- - permite indicar que la expresión a la derecha del - no esta permitida.
- () es útil para agrupar expresiones.
- **?** indica que la expresión a su izquierda puede estar ó no.

Una descripción en detalle de todos los elementos disponibles para la construcción de gramáticas en ANTLR puede encontrarse en [17]. Además, se sigue la convención de escribir las reglas del *lexer* con mayúsculas y las correspondientes al parser, en minúsculas.

Existen dos mecanismos diferentes para recorrer el AST (Abstract Syntax Tree) obtenido a partir de un parser generado por ANTLR:

- Mecanismo con *listeners*
- Mecanismo con *visitors*

Si bien con ambos mecanismos se pueden obtener los mismos resultados, hay ciertas diferencias que vale la pena mencionar:

- Los métodos del mecanismo que utiliza *listeners* son llamados automáticamente por el objeto *walker* provisto por ANTLR, mientras que al utilizar *visitors* es el programador quien debe llamar explícitamente a cada uno de los métodos *visit* sobre sus correspondientes elementos. El olvidar alguno de estos llamados provoca que dicho elemento y sus sub-elementos en el AST no sean visitados.
- Los métodos *listeners* no pueden devolver ningún valor, en cambio, los *visitors* pueden devolver valores de cualquier tipo.

En esta tesina se decidió hacer uso del mecanismo que involucra a los *visitors* ya que permite tener un control específico de los nodos procesados y definir así un comportamiento según las necesidades de cada caso. Además, dicho mecanismo sigue la estructura del patrón de diseño homónimo [16, Capítulo 5], lo cual se adecua a los lineamientos generales utilizados hasta el momento en el diseño de Fastest.

5.5.2. Proceso de generación del parser

Maven [25] es una herramienta que permite construir y gestionar proyectos Java de una manera simple y uniforme, de tal modo que no se tenga que invertir demasiado tiempo en temas relacionados a la gestión general del proyecto y así poder enfocar la atención en el diseño e implementación del mismo.

Entre los *plugins* disponibles de Maven se encuentra `antlr4-maven-plugin`, que permite hacer uso de las herramientas brindadas por ANTLR para generar parsers a partir de una

determinada gramática. Dicho *plugin* posee una gran variedad de opciones de configuración para que el implementador haga la personalización que satisfaga sus requerimientos. Dentro de esta amplia variedad de opciones, se hará mención de dos de las utilizadas para la generación del parser de ATCAL ya que son las más relevantes:

- `no-listener`: establece que no deben generarse módulos para soportar los métodos *listener*. En caso de no especificarse esta opción, dichos módulos se generan por defecto.
- `visitor`: indica que deben generarse módulos para soportar los métodos *visitor*. Por defecto éstos módulos no se generan.

Habiendo definido la gramática, en el archivo `Atcal.g4`, tal como se detalla en el apéndice ATCAL - Gramática en BNF, por línea de comandos, se ejecuta como último paso el comando `mvn generate-sources`. Esto genera los archivos que conforman la estructura del parser a nivel de Java:

- `AtcalParser.java`: contiene los métodos que parsean reglas de refinamiento escritas en ATCAL.
- `AtcalLexer.java`: permite la construcción de un lexer, elemento necesario para obtener el parser.
- `AtcalVisitor.java`: define la interfaz que expone todos los métodos que se utilizarán para recorrer el AST generado por el parser.
- `AtcalBaseVisitor.java`: clase que da una implementación por defecto de los métodos en la interfaz definida por `AtcalVisitor.java`. Para dar la interpretación a las reglas de refinamiento se escribirán clases que extiendan a `AtcalBaseVisitor.java` como se explicará más adelante.
- `Atcal.tokens` y `AtcalLexer.tokens`: estos son archivos auxiliares utilizados en `AtcalParser.java` y `AtcalLexer.java`.

Capítulo 6

Proceso de refinamiento

En este capítulo se detalla el proceso de refinamiento a nivel de Java. En principio se abordará el proceso desde un punto de vista general, abstrayendo lo más posible la información innecesaria, ya que el objetivo es mostrar al lector el esquema superficial del proceso. Luego, se optará por un enfoque que permita comprender el refinamiento con un mayor nivel de detalle.

Es importante aclarar que el desarrollo de dicho proceso lo inició Cristian Rosa en un trabajo para su post-doctorado y el mismo quedó incompleto. Partiendo de esta base se reformuló y adaptó el código preexistente con el fin de completar el proceso de refinamiento planteado como objetivo en esta tesina.

6.1. Aspectos generales

6.1.1. Generación de casos de prueba

En primer lugar, es necesario generar los casos de prueba abstractos que serán luego traducidos a casos de prueba concretos. Para simplificar la explicación, a continuación se presenta una especificación Z que será usada como ejemplo en la construcción de dichos casos:

$[DNI, NAME]$

Friends

$$people : DNI \pfun (NAME \times \mathbb{N})$$

AddFriendOk

$$\Delta Friends$$
$$dni? : DNI$$
$$person? : NAME \times \mathbb{N}$$
$$dni? \notin \mathrm{dom}\, people$$
$$people' = people \cup \{dni? \mapsto person?\}$$

IsAlreadyAFriend

$$\Xi Friends$$
$$dni? : DNI$$
$$dni? \in \mathrm{dom}\, people$$

$AddFriend == AddFriendOk \lor IsAlreadyAFriend$

Supongamos que esta especificación se guardó en el archivo `friends.tex` dentro del directorio `/home/user/desktop`. En base a esto, a continuación se detallan los comandos FASTEST utilizados para generar los casos de prueba abstractos:

- `loadspec /home/user/desktop/friends.tex`: carga la especificación que se encuentra en el archivo `friends.tex` dentro del directorio `/home/user/desktop/`.
- `selop AddFriend`: selecciona la operación `AddFriend` para que se considere al momento de generar el árbol de pruebas.
- `addtactic AddFriend SP \cup people \cup {dni? \mapsto person?}`: agrega la táctica **SP** (**S**tandard **P**artition), indicando que:
  - Tiene efecto en la operación `AddFriend`
  - Debe aplicarse sobre el operador binario `\cup`
  - La expresión involucrada es `people \cup {dni? \mapsto person?}`
- `genalltt`: genera el árbol de pruebas. Este comando tiene en cuenta las operaciones seleccionadas previamente utilizando el comando `selop`.
- `genalltca`: genera los casos de prueba abstractos para todas las pruebas del árbol de pruebas generados previamente.
- `loadrefrule /home/user/desktop/friends.atcal`: carga la regla de refinamiento que se encuentra en el archivo `friends.atcal` dentro del directorio `/home/user/desktop/`.

6.1.2. Ejecución de refinamiento y descripción general

Como último paso para generar los casos de prueba concretos se debe ejecutar el comando `refine`, que tiene la siguiente estructura:

```
refine AddFriend_VIS to test friends.pl implemented in perl with friends
```

donde:

- `AddFriend_VIS` es el sub-árbol de pruebas que contiene todos los casos de prueba abstractos generados que se desean refinar.
- `friends.pl` contiene la implementación del sistema a testear escrito en código Perl.
- `perl` es el lenguage de implementación destino en el que se expresarán los casos concretos.
- `friends` es el nombre de la regla de refinamiento que se desea utilizar para generar los casos de prueba. Este nombre se debe corresponder con el asignado, usando la marca `@RRULE`, en el archivo `friends.atcal`, cargado previamente con el comando `loadrefrule`.

Al ejecutar este comando se inicia el proceso de refinamiento para cada uno de los casos de prueba abstractos generados usando `genalltca`, utilizando la regla de refinamiento `friends` previamente cargada con el comando `loadrefrule`, cuyo contenido es[1]

```
1  @RRULE friends
2
3  @DATATYPES
4    DATATYPE Person = RECORD Person (name : STRING, age: INT);
5    DATATYPE Pair = RECORD Pair (dni : STRING, person: Person);
6
7  @LAWS
8    people ==> peopleArray AS ARRAY Pair (5) WITH [
9      people.@DOM ==> .dni AS STRING,
10     people.@RAN ==> .person AS Person WITH [
11       people.@RAN.#1 ==> .name AS STRING,
12       people.@RAN.#2 ==> .age AS INT
13     ]
14   ];
15
16   dni? ==> dni AS STRING;
17
18   person? ==> person AS Person WITH [
```

[1]Notar que en este caso no se definió código para las sección `@EPILOGUE` ni la que se encuentra entre `@LAWS` y `@UUT`, éstas, en general, contendrán código útil, pero en este caso no son necesarias

```
    person?.#1 ==> .name AS STRING,
    person?.#2 ==> .age AS INT
  ];

@UUT
addFriend(dni, person);
```

Una vez finalizado el refinamiento, se obtienen los casos de concretos de prueba en Perl. A continuación se presenta un caso abstracto de prueba y su correspondiente concretización para que el lector pueda ver con facilidad el mapeo de los valores entre ambos.

- Caso abstracto AddFriend_SP_4_TCASE

AddFriend_SP_4_TCASE

AddFriend_SP_4

$person? = (nAME2 \mapsto 0)$
$people = \{(dNI2 \mapsto (nAME1 \mapsto 1))\}$
$dni? = dNI1$

- Caso concreto AddFriend_SP_4_CTCASE

```
#!/usr/bin/perl
use feature qw(say);
use YAML::XS;
use Data::Dumper;

sub __fastest_dump {
  open F, '>', 'state.yml';
  print F Dumper ( @_ );
  close F;
}

sub addFriend {
    # Definición de método que se quiere testear
}

@peopleArray = ();
$person_name0 = "nAME1";
$person_age0 = 1;
$peopleArray_person0 = {"name" => $person_name0, "age" => $person_age0};
$peopleArray_dni0 = "dNI2";
$peopleArray[0] = {"dni" => $peopleArray_dni0, "person" =>
    $peopleArray_person0};
$dni = "dNI1";
$person_name0 = "nAME2";
$person_age0 = 0;
$person = {"name" => $person_name0, "age" => $person_age0};
addFriend($dni,$person);
$state->{'peopleArray'} =  \@peopleArray;
$state->{'dni'} = $dni;
$state->{'person'} =  \$person;
__fastest_dump($state);
```

Como se mencionó previamente, la definición de una regla de refinamiento permite establecer las leyes sobre cómo traducir las variables de los casos de prueba abstractos a variables de implementación en los casos concretos. De este modo, observando lo declarado en la regla `friends` y ambos casos de prueba (el abstracto y el concreto), se pueden destacar los siguientes puntos:

- En la regla de refinamiento se establece que `people` se refina como el arreglo de *records* `peopleArray`, donde cada componente del dominio toma el lugar del miembro `dni` de tipo *string* y cada componente del rango se corresponde con el miembro `person` de tipo `Person`. Para este último componente, se declara que está formado por dos elementos, donde el primero se mapea con `name` de tipo *string* y el segundo con `age` de tipo *int*, ambos pertenecientes al *record* `person`. Por lo tanto, como en el caso abstracto `AddFriend_SP_4_TCASE` `people` contiene únicamente el mapeo entre $dNI2$ y la tupla $(nAME1, 1)$ entonces en el caso concreto escrito en código Perl se puede ver que el arreglo `peopleArray` tiene un solo elemento de tipo *record* cuyo miembro `dni` es $dNI2$ y el miembro `person` es otro *record* donde `name` es $nAME1$ y `age` es 1.

- Como en la regla de refinamiento se declara que `dni?` refina a la variable de implementación `dni` de tipo *string* y en el caso abstracto el valor de `dni?` es `dNI1` se puede observar que en el caso concreto la variable `dni` toma el valor $dNI1$.
- Para la variable de especificación `person?` se indica que refina a la variable de implementación `person` siguiendo las mismas leyes que los elementos del rango `people`, entonces como en `AddFriend_SP_4_TCASE` `person?` es la tupla $(nAME2, 0)$ se puede observar que, en el caso concreto, `person` es un *record* donde `name` es $nAME2$ y `age` es 0.

En el anexo B se encuentran otros dos ejemplos donde se puede apreciar cómo, mediante el vínculo establecido en la regla de refinamiento `friends`, las variables de los casos abstractos se relacionan con las de los casos concretos.

Como se puede observar, se generó un programa en Perl que puede ser utilizado para testear la implementación de la operación AddFriend (en este caso, el método de nombre `addFriend`).

Otro punto a destacar es el prefijo que contienen los ejemplos:

```
#!/usr/bin/perl
use feature qw(say);
use YAML::XS;
use Data::Dumper;

sub __fastest_dump {
  open F, '>', 'state.yml';
  print F Dumper ( @_ );
  close F;
}
```

Éste, al ser necesario en los distintos casos concretos de prueba generados por el proceso de refinamiento, es incluido por defecto por el algoritmo que implementa dicho proceso. De esta manera, todo preámbulo que se escriba en la regla de refinamiento será prefijado con este bloque de código.

La implementación del proceso de refinamiento es el eje central de esta tesina y, por lo tanto, en él se enfocará toda la atención.

El proceso toma como punto de partida los casos abstractos de prueba generados a partir de una especificación Z y una regla de refinamiento. Se recorren los casos abstractos y para cada uno de ellos se genera, utilizando la información provista en la regla de refinamiento, un caso concreto en Perl. A continuación se introducen los módulos que conforman el proceso.

6.1.2.1. RefineCommand

Al ejecutarse el comando `refine`, el control de la ejecución pasa a la clase `RefineCommand`. Esta clase parsea los parámetros que ingresó el usuario y genera un evento `tCaseRefineRequested` por cada uno de los casos abstractos de prueba, pasándoles los parámetros obtenidos. Dichos eventos son captados por el sistema y derivados a la clase `TCaseRefineClient` que es la encargada de su procesamiento.

6.1.2.2. TCaseRefineClient

Esta clase es la que da inicio al refinamiento de un caso abstracto de prueba específico, delegando la tarea en una instancia de la clase `TCaseRefineClientRunner` que, por cuestiones de eficiencia, se ejecuta en un hilo aparte. A esta instancia se le indican:

- El caso abstracto de prueba a procesar.
- La operación en base a la cual se generó el caso abstracto de prueba.
- El lenguaje de implementación al que se quiere refinar dicho caso.
- La regla de refinamiento a utilizar.
- Ubicación del archivo que contiene la implementación del sistema.

6.1.2.3. TCaseRefineClientRunner

Cuando se le cede el control de ejecución, esta clase se encarga de generar un evaluador ATCAL (`AtcalEvaluator`) y luego llamar al método `visitRefinementRule` para que dé comienzo al algoritmo que se encarga de procesar el caso abstracto de prueba según la regla de refinamiento indicada. Una vez concretado el refinamiento, se emite un evento informando que el caso abstracto de prueba ha sido refinado.

6.1.2.4. AtcalEvaluator

El objetivo de esta clase es armar un caso de prueba concreto a partir de uno abstracto y una regla de refinamiento. Para lograrlo recorre la regla de refinamiento procesando las leyes incluidas. Por cada ocurrencia de elementos de la especificación Z (a partir de la cual se generó el caso de prueba abstracto que se procesa) que se encuentre en estas leyes, se consultará el caso abstracto de prueba para obtener el valor específico correspondiente. Debido a que para ciertos elementos de la regla de refinamiento se requiere un tratamiento complejo, se decidió separar el algoritmo encargado en una clase aparte. A continuación se establece la relación entre clase y elemento procesado por dicha clase:

- `AtcalEvaluator`:
  - Nombre de la regla indicado por el token `@RRULE`
  - Código del lenguaje de implementación destino incluido en las secciones
    - `preamble`
    - `PLCODE`
    - `epilogue`
- `TypesEvaluator`: se encarga del procesamiento de las leyes de la sección `datatypes`, que permite definir los tipos de datos que se utilizarán en las demás leyes.
- `RefinementLawEvaluator`: procesa las leyes incluidas en la sección `laws`, donde se establece cómo traducir las variables de entrada y de estado, declaradas en la especificación, a variables del lenguaje de implementación.

6.1.2.5. TypesEvaluator

Cada una de las declaraciones de tipos de datos utilizados en las leyes de refinamiento se lleva a cabo en esta clase, que puede considerarse simplemente una intermediaria, ya que, `TypeEvaluator` es donde se encuentra el algoritmo que conoce cómo procesar dichas declaraciones.

6.1.2.6. RefinementLawEvaluator

Los puntos clave del proceso de refinamiento se encuentran en esta clase. La misma, contiene los detalles sobre cómo debe interpretarse cada una de las leyes que establecen el vínculo entre lo abstracto y lo concreto, es decir, las que describen cómo pasar de las variables de entrada y de estado de la especificación, a sus correspondientes variables de implementación.

6.2. Detalles

La sección actual está enfocada en profundizar los detalles del proceso de refinamiento. Con esto se busca describir cómo el algoritmo encargado de analizar y procesar una regla de refinamiento junto con un caso abstracto de prueba, produce código válido en el lenguaje de implementación elegido como destino. El contenido de esta sección se divide en subsecciones que describen el comportamiento particular de los distintos bloques y métodos que, en conjunto, conforman el algoritmo en su totalidad.

6.2.1. visitLaws

En este método, se recorren las leyes que componen a la sección `laws` (ver sección 6.1.2.4) con el objetivo de derivar cada una de ellas al método que le corresponde. Para tomar esta decisión, cada ley se clasifica según su tipo. Si se trata de una ley que establece una traducción bidireccional, su procesamiento será delegado en `visitBiRefLaw`. De otro modo, si la ley indica cómo traducir elementos de la especificación a elementos de la implementación, se procesará en el método `visitLawRefinement`. Por último si la ley establece el uso de leyes provenientes de otras reglas previamente definidas entonces será procesada por el método `visitLawReference`.

6.2.2. visitBiRefLaw

Las leyes procesadas por este método tienen el siguiente formato

```
ID <==> refinement
```

donde `ID` es el nombre de una variable Z y `refinement` indica cómo se refina dicha variable.

Este método sigue un comportamiento similar al que se explicará en la sección 6.2.3. La principal diferencia se establece por el carácter bidireccional del tipo de leyes procesadas por el método, ya que, por esta razón, es necesario guardar la relación entre el elemento de la especificación y el de la implementación.

6.2.3. visitLawRefinement

Las leyes procesadas por este método tienen el siguiente formato

```
zExprs ==> refinement
```

donde `zExprs` permite construir expresiones Z y `refinement` indica cómo se refinan dichas expresiones.

Cada expresión Z contenida en `zExprs`, es tratada por el evaluador de expresiones (`ZExprEvaluator`) para recuperar el correspondiente valor concreto del caso abstracto de prueba que se está refinando. Una vez obtenido el valor para cada expresión, se procede a llamar, por cada expresión Z, al método `visitRefinement`. En cada una de estas llamadas, se le hace saber al método cuál es el valor concreto de la expresión Z que está procesando. De esta manera se disponibiliza la información necesaria para generar código en el lenguaje de implementación destino.

6.2.4. visitLawReference

Las leyes procesadas por este método tienen el siguiente formato

```
ID . ( @LAWS | ID )
```

Como se puede ver, el tipo de leyes que procesa el método puede tomar dos tipos:

- `RuleName.@LAWS`: si se encuentra una ley de este tipo lo que se hace es procesar todas las leyes correspondientes a regla de nombre `RuleName` que debe haberse cargado previamente.
- `RuleName.LawID`: en este otro caso, se procede a procesar la ley `LawID` de la regla `RuleName`, que también debe haberse cargado previamente.

6.2.5. ZExprEvaluator

Tal como se explicó previamente, esta clase es la encargada de evaluar expresiones Z con el fin de obtener el valor concreto de dicha expresión a partir de un caso abstracto de prueba determinado. Cada método que la conforma conoce cual es el tratamiento a aplicar para el tipo específico de expresión que le corresponde. En las siguientes secciones se mencionan dichos métodos.

6.2.5.1. visitIdent

Se encarga de tratar expresiones que representan. identificadores Z

6.2.5.2. visitNumLiteral

Se encarga de tratar expresiones que representan números.

6.2.5.3. visitStrLiteral

Se encarga de tratar expresiones que representan cadenas de caracteres.

6.2.5.4. visitAutoExpr

Se encarga de tratar expresiones de la forma `@AUTOFILL`, utilizadas para indicar que la variable que se está refinando debe completarse con valores por defecto. Dichos valores son definidos en código.

6.2.5.5. visitElemExpr

Se encarga de tratar expresiones de la forma `@ELEM`, utilizadas para hacer referencia a un elemento de forma genérica. Este tipo de expresiones solo deben usarse dentro de una cláusula `WITH` y la expresión Z, a la cual corresponde dicha cláusula, tiene alguno de los siguientes tipos:

- $\mathbb{P}(\mathbb{P}\, X)$ para algún tipo X dado, en el caso que el operador se utilice en alguna de las siguientes formas:
  - `@ELEM`, es decir, sin ningún operador a su derecha
  - `@ELEM.@DOM`
  - `@ELEM.@RAN`
- $X_1 \times X_2 \times \cdots \times X_n$ para cualesquiera $X_1, \ldots, X_n$, en el caso que el operador se utilice como `@ELEM.#NRO`

6.2.5.6. visitGroup

Se encarga de tratar expresiones rodeadas por paréntesis, es decir `( zExpr )`.

6.2.5.7. visitProdProj

Se encarga de tratar expresiones de la forma `zExpr.#NRO`, donde `NRO` es un número.

6.2.5.8. visitProdCons

Se encarga de tratar expresiones de la forma `<zExpr`$_1$`, zExpr`$_2$`, ...>`, donde `zExpr`$_1$ y `zExpr`$_2$ son expresiones Z. Los puntos suspensivos indican que se pueden seguir incluyendo expresiones Z, siempre y cuando sea una cantidad finita de veces.

6.2.5.9. visitSetDom

Se encarga de tratar expresiones de la forma `zExpr.@DOM`.

6.2.5.10. visitSetDiff

Se encarga de tratar expresiones de la forma `zExpr`$_1 \setminus$ `zExpr`$_2$.

6.2.5.11. visitSetInter

Se encarga de tratar expresiones de la forma `zExpr`$_1 \cap$ `zExpr`$_2$.

6.2.5.12. visitSetRan

Se encarga de tratar expresiones de la forma `zExpr.@RAN`.

6.2.5.13. visitSetProj

Se encarga de tratar expresiones de la forma `zExpr.@NRO`, donde `NRO` es un número.

6.2.5.14. visitSetCons

Se encarga de tratar expresiones de la forma {`zExpr1 , zExpr2, ...`}, donde `zExpr1` y `zExpr2` son expresiones Z. Los puntos suspensivos indican que se pueden seguir incluyendo expresiones Z, siempre y cuando sea una cantidad finita de veces.

6.2.5.15. visitSetUnion

Se encarga de tratar expresiones de la forma `zExpr1` $\vee$ `zExpr2`.

6.2.5.16. visitNumMod

Se encarga de tratar expresiones de la forma `zExpr1 % zExpr2`.

6.2.5.17. visitNumMul

Se encarga de tratar expresiones de la forma `zExpr1 * zExpr2`.

6.2.5.18. visitNumPlus

Se encarga de tratar expresiones de la forma `zExpr1 + zExpr2`.

6.2.5.19. visitNumDiv

Se encarga de tratar expresiones de la forma `zExpr1 / zExpr2`.

6.2.5.20. visitNumMinus

Se encarga de tratar expresiones de la forma `zExpr1 - zExpr2`.

6.2.5.21. visitSetCard

Se encarga de tratar expresiones de la forma `zExpr.@CARD`, utilizado para referirse a la cardinalidad de `zExpr`.

6.2.5.22. visitStrConcat

Se encarga de tratar expresiones de la forma `zExpr1 ++ zExpr2`.

6.2.6. visitRefinement

Si bien el proceso de refinamiento esta constituido por muchos métodos, cada uno de ellos con un objetivo específico, en esta sección se abordará la descripción de los métodos que, a interpretación del autor, conforman una pieza central en el algoritmo.

El método `visitRefinement` es el encargado de decidir de qué tipo de refinamiento se trata y en base a esto llamar al método correspondiente, que es el que conoce todos los detalles y particularidades del caso puntual que se está refinando. El tipo de refinamiento tiene una relación directa con el tipo de dato presente en la ley que se procesa, que es utilizado para indicar, a nivel del lenguaje de implementación, el tipo de dato que tendrá la variable. Como ejemplo, supóngase que se tiene la siguiente ley:

```
age ==> ageImpl AS INT
```

en este caso, el tipo de dato presente es `INT` y lo que indica es que la variable de implementación `ageImpl`, que refina a la variable de especificación `age`, se implementa como un entero. Entre los tipos de refinamiento presentados a continuación se puede ver que la ley del ejemplo anterior corresponde al segundo caso:

- Refinamiento a `STRING`: la variable se implementa como una cadena de caracteres.
- Refinamiento a `INT` ó `FLOAT`: la variable se implementa como un número entero ó flotante. Ambos casos caen en la misma categoría ya que siguen el mismo algoritmo.
- Refinamiento a `ARRAY`: la variable se implementa como un arreglo.
- Refinamiento a `ENUM`: la variable se implementa como un enumerado.

- Refinamiento a `RECORD`: la variable se implementa como un registro.
- Refinamiento a `CONTRACT`: la variable se implementa como un contrato.
- Refinamiento a `REFERENCE`: la variable se implementa como una referencia.

Con el objetivo de aislar los diferentes tratamientos que cada caso requiere, se decidió desarrollar un método para cada uno de ellos. Los mismos son los siguientes:

- `refineAsString`
- `refineAsIntOrFloat`
- `refineAsArray`
- `refineAsEnum`
- `refineAsRecord`
- `refineAsContract`
- `refineAsReference`

Capítulo 7

Casos de estudio

En este capítulo se expondrá uno de los casos de estudio a través del cual se pretende dar a conocer los detalles de los distintos pasos del proceso de refinamiento a Perl. Luego, a partir del apéndice C se pueden encontrar los demás casos de estudio considerados en esta tesina. El caso de estudio presentado en esta sección se estructurará en varias secciones:

- **Especificación Z** del caso de estudio en sí. Ésta contiene las operaciones para las cuales se generarán los casos abstractos de prueba.
- **Casos abstractos de prueba** generados por FASTEST para cada operación de la especificación. Es posible que se incluyan solo algunos de los casos generados ya que en ciertos casos de estudio se generan muchos y mostrarlos todos no ofrece un aporte significativo.
- **Una o más implementaciones en Perl** del sistema especificado en Z. El uso de más de una implementación tiene como fin mostrar la expresividad del lenguaje ATCAL para representar distintos refinamientos a partir de una misma especificación.
- **La descripción de las leyes de refinamiento** incluidas en cada archivo de refinamiento asociado a cada implementación. La descripción de cada archivo de refinamiento se compone de:
  - Identificador de regla.
  - Definición de tipos de datos.
  - Especificación de leyes de refinamiento.
  - Especificación del método a testear.
- **Casos concretos de prueba en Perl** generados por el proceso de refinamiento desarrollado en esta tesina. En cada una de las implementaciones, salvo previa aclaración, solo el primer caso concreto de prueba generado se incluirá por completo, en los demás se suprimirán ciertas secciones que se repiten, sin aportar detalles de interés. El código fuente del sistema a testear es un claro ejemplo de fragmento que será omitido luego del primer caso concreto.

En el cuadro 7.1 se presenta un resumen de los casos de estudio abordados en esta tesina con el fin de ofrecer al lector una visión general sobre todos los casos estudiados. Las columnas de este cuadro consisten en:

- Nombre del caso de estudio.
- # oper. Z: cantidad de operaciones Z que contiene la especificación del caso de estudio.
- # ATC: cantidad de casos abstractos de prueba. La sigla proviene de Abstract Test Case.
- # imp: cantidad de implementaciones del sistema asociado al caso de estudio.
- # reglas de ref.: cantidad de reglas de refinamiento.
- # CTC: cantidad de casos concretos de prueba. La sigla proviene de Concrete Test Case.
- Apéndice donde se encuentra incluido el caso de estudio.

Nombre	# oper. Z	# ATC	# imp.	# reglas de ref.	# CTC	Apéndice
Sistema de bicicletas	3	11	2	2	11	-
Registro civil	4	30	1	4	30	Registro civil
Almacen de elementos	3	25	1	3	25	Almacen de elementos
Rango etario	4	19	1	4	19	Rango etario
Blog	3	13	1	3	13	Blog
Big Integer	4	44	1	4	44	Big Integer

Cuadro 7.1: Casos de estudio

7.1. Sistema de bicicletas

7.1.1. Especificación Z

Se trata de un sistema utilizado por un negocio de bicicletas que permite almacenar información de relevancia sobre las bicicletas disponibles.

$[BikeId, Phone, Address, Name]$

$BikeSize ::= s18 \mid s20 \mid s24 \mid s26 \mid s28 \mid s29$

$Color ::= yellow \mid red \mid blue$

Se cuenta con algunos tipos básicos

- `BikeId`
- `Phone`
- `Address`
- `Name`

y también se definen dos tipos de datos enumerados:

- `BikeSize`: se utiliza como medida de relación entre la bicicleta y la altura del individuo que la usará. Los valores considerados son:
  - `s18`: rodado 18
  - `s20`: rodado 20
  - `s24`: rodado 24
  - `s26`: rodado 26
  - `s28`: rodado 28
  - `s29`: rodado 29
- `Color` que cuenta con los siguientes valores:
  - `yellow`
  - `red`
  - `blue`

En base a estos tipos de datos se define el esquema de estado

Bikes

$bicycles : BikeId \pfun (BikeSize \times Color \times (Name \times Phone \times Address))$

donde la imagen de la función parcial `bicycles` representa los datos de una bicicleta y cuenta con la siguiente información:[1]

- Tamaño de tipo `BikeSize`
- Color de tipo `Color`

[1]Como se mencionó antes en la tesina, normalmente se definiría un esquema (posiblemente de nombrado `Bike`) compuesto por los datos contenidos en el producto cartesiano $BikeSize \times Color \times (Name \times Phone \times Address)$, pero debido a una limitación de Fastest debe expresarse como producto cartesiano.

- Fabricante de la bicicleta. Los datos considerados relevantes en este caso de estudio son:[2]
  - Nombre de tipo `Name`
  - Teléfono de tipo `Phone`
  - Dirección de tipo `Address`

Una vez definidos tanto los tipos de datos como el esquema de estado, se procede a presentar las operaciones a partir de las cuales se generaran los casos de prueba. Estas son:

- AddBike: agregar una bicicleta al sistema

$$
\begin{array}{|l}
\hline
\textit{AddBikeOk} \\
\Delta \textit{Bikes} \\
\textit{bikeid?} : \textit{BikeId} \\
\textit{b?} : \textit{BikeSize} \times \textit{Color} \times (\textit{Name} \times \textit{Phone} \times \textit{Address}) \\
\hline
\textit{bikeid?} \notin \operatorname{dom} \textit{bicycles} \\
\textit{bicycles}' = \textit{bicycles} \cup \{\textit{bikeid?} \mapsto \textit{b?}\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{BikeAlreadyAdded} \\
\Xi \textit{Bikes} \\
\textit{bikeid?} : \textit{BikeId} \\
\hline
\textit{bikeid?} \in \operatorname{dom} \textit{bicycles} \\
\hline
\end{array}
$$

$$\textit{AddBike} == \textit{AddBikeOk} \lor \textit{BikeAlreadyAdded}$$

- RemoveBike: eliminar una bicicleta del sistema

$$
\begin{array}{|l}
\hline
\textit{RemoveBikeOk} \\
\Delta \textit{Bikes} \\
\textit{bikeid?} : \textit{BikeId} \\
\hline
\textit{bikeid?} \in \operatorname{dom} \textit{bicycles} \\
\textit{bicycles}' = \{\textit{bikeid?}\} \lhd \textit{bicycles} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{BikeNotFound} \\
\Xi \textit{Bikes} \\
\textit{bikeid?} : \textit{BikeId} \\
\hline
\textit{bikeid?} \notin \operatorname{dom} \textit{bicycles} \\
\hline
\end{array}
$$

$$\textit{RemoveBike} == \textit{RemoveBikeOk} \lor \textit{BikeNotFound}$$

- GetBikeManufacturer: obtener los datos del fabricante

$$
\begin{array}{|l}
\hline
\textit{GetBikeManufacturerOk} \\
\Xi \textit{Bikes} \\
\textit{bikeid?} : \textit{BikeId} \\
\textit{bikeManufacturer!} : \textit{Name} \times \textit{Phone} \times \textit{Address} \\
\hline
\textit{bikeid?} \in \operatorname{dom} \textit{bicycles} \\
\textit{bikeManufacturer!} = (\textit{bicycles}\ \textit{bikeid?}).3 \\
\hline
\end{array}
$$

$$\textit{GetBikeManufacturer} == \textit{GetBikeManufacturerOk} \lor \textit{BikeNotFound}$$

[2] En este caso sucede lo mismo que lo mencionado para la imagen de `bicycles`.

7.1.2. Casos Abstractos de prueba

Para generar los casos abstractos de prueba se ejecutaron los comandos

```
loadspec bicycles.tex
selop AddBike
addtactic AddBike SP \cup bicycles \cup \{bikeid? \mapsto b?\}
selop RemoveBike
addtactic RemoveBike SP \ndres \{bikeid?\} \ndres bicycles
selop GetBikeManufacturer
genalltt
genalltca
```

y se obtuvieron los siguientes casos:

7.1.2.1. AddBike

AddBike_SP_2_TCASE

AddBike_SP_2

$bicycles = \emptyset$
$bikeid? = bikeId4$
$b? = (s18, yellow, (name1, phone2, address3))$

AddBike_SP_4_TCASE

AddBike_SP_4

$bicycles = \{(bikeId2 \mapsto (s20, red, (name1, phone1, address1)))\}$
$bikeid? = bikeId1$
$b? = (s18, yellow, (name2, phone3, address4))$

AddBike_SP_12_TCASE

AddBike_SP_12

$bicycles = \{(bikeId1 \mapsto (s20, red, (name1, phone1, address1)))\}$
$bikeid? = bikeId1$
$b? = (s18, yellow, (name2, phone3, address4))$

AddBike_SP_14_TCASE

AddBike_SP_14

$bicycles = \{(bikeId1 \mapsto (s18, yellow, (name1, phone2, address3))),$
$(bikeId2 \mapsto (s20, red, (name1, phone1, address4)))\}$
$bikeid? = bikeId1$
$b? = (s18, yellow, (name1, phone2, address3))$

AddBike_SP_15_TCASE

AddBike_SP_15

$bicycles = \{(bikeId1 \mapsto (s18, yellow, (name2, phone3, address4)))\}$
$bikeid? = bikeId1$
$b? = (s18, yellow, (name2, phone3, address4))$

7.1.2.2. RemoveBike

RemoveBike_SP_3_TCASE

RemoveBike_SP_3

$bicycles = \{(bikeId1 \mapsto (s20, red, (name1, phone1, address2)))\}$
$bikeid? = bikeId1$

RemoveBike_SP_4_TCASE

RemoveBike_SP_4

$bicycles = \{(bikeId1 \mapsto (s20, red, (name1, phone1, address1))),$
$(bikeId2 \mapsto (s20, red, (name1, phone1, address2)))\}$
$bikeid? = bikeId1$

RemoveBike_SP_8_TCASE

RemoveBike_SP_8

$bicycles = \emptyset$
$bikeid? = bikeId1$

RemoveBike_SP_12_TCASE

RemoveBike_SP_12

$bicycles = \{(bikeId2 \mapsto (s20, red, (name1, phone1, address1)))\}$
$bikeid? = bikeId1$

7.1.2.3. GetBikeManufacturer

GetBikeManufacturer_DNF_1_TCASE

GetBikeManufacturer_DNF_1

$bicycles = \{(bikeId1 \mapsto (s20, red, (name1, phone1, address2)))\}$
$bikeid? = bikeId1$

GetBikeManufacturer_DNF_2_TCASE

GetBikeManufacturer_DNF_2

$bicycles = \emptyset$
$bikeid? = bikeId1$

7.1.3. Implementación

Se presentarán dos implementaciones distintas, una en la que se implementa la variable de especificación `bicycles` como un array de contratos y otra que la implementa como un contrato[3] formado por records.

7.1.3.1. Implementación 1: array de contratos

Esta implementación se encuentra en el archivo `bicyclesSetAsArrayOfContracts.pl`

```
# Defino el enumerado para tamaños de bicicletas
use constant {
    s18I  => 's18I',
    s20I  => 's20I',
    s24I  => 's24I',
    s26I  => 's26I',
    s28I  => 's28I',
    s29I  => 's29I'
};

# Defino el enumerado para colores
use constant {
    yellowI => 'yellowI',
    redI  => 'redI',
    blueI => 'blueI'
};

package Pair;
```

[3]Notar que en este caso el contrato se utiliza como tipo de datos que almacena múltiples elementos del mismo tipo.

```
19
20 sub new {
21     my ($class,$args) = @_;
22     my $self = bless { }, $class;
23 }
24
25 sub set {
26   my ($self, $fst, $snd) = @_;
27   $self->{"fst"} = $fst;
28   $self->{"snd"} = $snd;
29 }
30
31 sub get {
32   my $self = shift;
33   my $arg = shift;
34   return $self->{$arg};
35 }
36
37 package Bike;
38
39 sub new {
40     my ($class,$args) = @_;
41     my $self = bless { }, $class;
42 }
43
44 sub set {
45   my ($self, $size, $color, $manufacturer) = @_;
46   $self->{"size"} = $size;
47   $self->{"color"} = $color;
48   $self->{"manufacturer"} = $manufacturer;
49 }
50
51 sub get {
52   my $self = shift;
53   my $arg = shift;
54   return $self->{$arg};
55 }
56
57 package Manufacturer;
58
59 sub new {
60     my ($class,$args) = @_;
61     my $self = bless { }, $class;
62 }
63
64 sub set {
65   my ($self, $name, $phone, $address) = @_;
66   $self->{"name"} = $name;
67   $self->{"phone"} = $phone;
68   $self->{"address"} = $address;
69 }
70
71 sub get {
72   my $self = shift;
73   my $arg = shift;
74   return $self->{$arg};
75 }
76
77 package main;
78
79 # Hash donde se almacenan los mensajes a mostrar como salida
80 %mensajes = (
81   'bikeAlreadyAdded' => 'La bicicleta ya fue agregada previamente',
82   'bikeNotFound' => 'La bicicleta no existe'
83 );
84
85 # Array de contratos donde se guardarán las bicicletas
86 @bikes = ();
87
88 # Entero utilizado para saber cual es el índice donde
89 # debe guardarse el siguiente elemento de bikes
90 $lastIndex = 0;
91
92 # Chequea si ya fue agregado un ID de bicilceta
93 # Argumento: ID de bicilceta
94 sub bikeExists {
```

```
  $bikesSize = @bikes;
  for $i (0..$bikesSize-1) {
    if($bikes[$i]->get("fst") eq $_[0]) {
      return 1;
    }
  }
  return 0;
}

sub getBikeArrayIndex {
  $bikesSize = @bikes;
  for $i (0..$bikesSize-1) {
    if($bikes[$i]->get("fst") == $_[0]) {
      return $i;
    }
  }
  return -1;
}

# Agrega una bicicleta
# Primer argumento: ID de bicicleta
# Segundo argumento: Objeto que contiene la información de la bicilceta
sub addBike {
  $bikeId = $_[0];
  $bike = $_[1];
  if(bikeExists($bikeId)) {
    say($mensajes{'bikeAlreadyAdded'})
  } else {
    $lastIndex = @bikes;

    $bikeAux = Bike->new();
    $size = $bike->get("size");
    $color = $bike->get("color");
    $manufacturerAux = $bike->get("manufacturer");
    $name = $manufacturerAux->get("name");
    $phone = $manufacturerAux->get("phone");
    $address = $manufacturerAux->get("address");
    $manufacturer = Manufacturer->new();
    $manufacturer->set($name,$phone,$address);
    $bikeAux->set($size,$color,$manufacturer);

    $bikes[$lastIndex] = Pair->new();
    $bikes[$lastIndex]->set($bikeId, $bikeAux);
    $lastIndex++;
  }
}

sub removeBike {
  $bikeId = $_[0];

  if(bikeExists($bikeId)) {
    $arrayIndex = getBikeArrayIndex($bikeId);
    delete $bikes[$arrayIndex];
  } else {
    say($mensajes{'bikeNotFound'})
  }
}

sub getBikeManufacturer {
  $bikeId = $_[0];

  if(bikeExists($bikeId)) {
    $arrayIndex = getBikeArrayIndex($bikeId);
    return $bikes[$arrayIndex]->get("manufacturer");
  } else {
    say($mensajes{'bikeNotFound'})
  }
}
```

7.1.3.2. Implementación 2: contrato de records

Esta implementación se encuentra en el archivo `bicyclesSetAsContractOfRecords.pl`

```
# Defino el enumerado para tamaños de bicicletas
use constant {
```

```
    s18I  => 's18I',
    s20I  => 's20I',
    s24I  => 's24I',
    s26I  => 's26I',
    s28I  => 's28I',
    s29I  => 's29I'
};

# Defino el enumerado para colores
use constant {
    yellowI => 'yellowI',
    redI  => 'redI',
    blueI => 'blueI'
};

package Set;

sub new {
    my ($class,$args) = @_;
    @emptyArray = ();
    my $self = bless {"elem" => \@emptyArray}, $class;
}

sub push {
  my ($self, $pair) = @_;
  @elem = @{$self->{"elem"}};
  $elemArraySize = @elem;
  $self->{"elem"}[$elemArraySize] = $pair;
}

sub pop {
  my $self = shift;
  my $arg = shift;
  @elem = @{$self->{"elem"}};
  $elemArraySize = @elem;
  if($elemArraySize == 0) {
    die;
  }
  $index = $elemArraySize-1;
  $result = $self->{"elem"}[$index];
  delete $self->{"elem"}[$index];
  return $result;
}

package Bike;

sub new {
    my ($class,$args) = @_;
    my $self = bless { }, $class;
}

sub set {
  my ($self, $size, $color, $manufacturer) = @_;
  $self->{"size"} = $size;
  $self->{"color"} = $color;
  $self->{"manufacturer"} = $manufacturer;
}

sub get {
  my $self = shift;
  my $arg = shift;
  return $self->{$arg};
}

package Manufacturer;

sub new {
    my ($class,$args) = @_;
    my $self = bless { }, $class;
}

sub set {
  my ($self, $name, $phone, $address) = @_;
  $self->{"name"} = $name;
  $self->{"phone"} = $phone;
  $self->{"address"} = $address;
```

```
}

sub get {
  my $self = shift;
  my $arg = shift;
  return $self->{$arg};
}

package main;

# Hash donde se almacenan los mensajes a mostrar como salida
%mensajes = (
  'bikeAlreadyAdded' => 'La bicicleta ya fue agregada previamente',
  'bikeNotFound' => 'La bicicleta no existe'
);

# Objeto donde se guardarán las bicicletas
$bikes = Set->new();

# Entero utilizado para saber cual es el índice donde
# debe guardarse el siguiente elemento de bikes
$lastIndex = 0;

# Chequea si ya fue agregado un ID de bicilceta
# Argumento: ID de bicilceta
sub bikeExists {
  $bikesSize = $bikes->{"elem"};
  for $i (0..$bikesSize-1) {
    if($bikes->{"elem"}[$i]->{"fst"} eq $_[0]) {
      return 1;
    }
  }
  return 0;
}

sub getBikeArrayIndex {
  $bikesSize = $bikes->{"elem"};
  for $i (0..$bikesSize-1) {
    if($bikes->{"elem"}[$i]->{"fst"} == $_[0]) {
      return $i;
    }
  }
  return -1;
}

# Agrega una bicicleta
# Primer argumento: ID de bicicleta
# Segundo argumento: Objeto que contiene la información de la bicilceta
sub addBike {
  $bikeId = $_[0];
  $bike = $_[1];
  if(bikeExists($bikeId)) {
    say($mensajes{'bikeAlreadyAdded'})
  } else {
    $lastIndex = $bikes->{"elem"};
    $bikes[$lastIndex]->{"fst"} = $bikeId;

    $bikeAux = Bike->new();
    $size = $bike->get("size");
    $color = $bike->get("color");
    $manufacturerAux = $bike->get("manufacturer");
    $name = $manufacturerAux->get("name");
    $phone = $manufacturerAux->get("phone");
    $address = $manufacturerAux->get("address");
    $manufacturer = Manufacturer->new();
    $manufacturer->set($name,$phone,$address);
    $bikeAux->set($size,$color,$manufacturer);

    $bikes->{"elem"}[$lastIndex]->{"snd"} = $bikeAux;
    $lastIndex++;
  }
}

sub removeBike {
  $bikeId = $_[0];

```

```
155   if(bikeExists($bikeId)) {
156     $arrayIndex = getBikeArrayIndex($bikeId);
157     delete $bikes->{"elem"}[$arrayIndex];
158   } else {
159     say($mensajes{'bikeNotFound'})
160   }
161 }
162
163 sub getBikeManufacturer {
164   $bikeId = $_[0];
165
166   if(bikeExists($bikeId)) {
167     $arrayIndex = getBikeArrayIndex($bikeId);
168     return $bikes->{"elem"}[$arrayIndex]->get("manufacturer");
169   } else {
170     say($mensajes{'bikeNotFound'})
171   }
172 }
```

7.1.4. Descripción de leyes de refinamiento

7.1.4.1. Array de contratos

7.1.4.1.1 Identificador de regla

```
1 @RRULE bicyclesSetAsArrayOfContractsAdd
```

Define que el nombre de la regla de refinamiento es `bicyclesSetAsArrayOfContractAdd`

7.1.4.1.2 Definición de tipos de datos

```
1 @DATATYPES
2   DATATYPE bikeSize = ENUM bikeSize ( s18I, s20I, s24I, s26I, s28I, s29I );
3   DATATYPE Color = ENUM Color ( yellowI, redI, blueI );
4   DATATYPE Manufacturer = MODULE "Manufacturer" CONSTRUCTOR new () SETTER set
        (name : STRING, phone: STRING, address: STRING) GETTER get (member :
        STRING);
5   DATATYPE Bike = MODULE "Bike" CONSTRUCTOR new () SETTER set (size : bikeSize
        , color: Color, manufacturer: Manufacturer) GETTER get (member : STRING)
        ;
6   DATATYPE Pair = MODULE "Pair" CONSTRUCTOR new () SETTER set (fst : STRING,
        snd: Bike) GETTER get (member : STRING);
```

Se definen los tipos de datos:

- bikeSize: enumerado cuyos posibles valores son:
  - s18I
  - s20I
  - s24I
  - s26I
  - s28I
  - s29I
- Color: enumerado cuyos posibles valores son:
  - yellowI
  - redI
  - blueI
- Manufacturer: contrato de constructor `new`, setter `set`, getter `get` y se compone de los siguientes miembros
  - name : STRING
  - phone : STRING
  - address : STRING

- Bike: contrato de constructor `new`, setter `set`, getter `get` y se compone de los siguientes miembros
  - size : bikeSize
  - manufacturer : Manufacturer
- Pair: contrato de constructor `new`, setter `set`, getter `get` y se compone de los siguientes miembros
  - fst : STRING
  - snd : Bike

7.1.4.1.3 Especificación de leyes de refinamiento

```
@LAWS
  bicycles ==> bikes AS ARRAY Pair (3) WITH [
    bicycles.@DOM ==> fst AS STRING,
    bicycles.@RAN ==> snd AS Bike WITH [
      bicycles.@RAN.#1 ==> size AS bikeSize MAP [
        s18 -> s18I,
        s20 -> s20I,
        s24 -> s24I,
        s26 -> s26I,
        s28 -> s28I,
        s29 -> s29I],
      bicycles.@RAN.#2 ==> color AS Color MAP [
        yellow -> yellowI,
        red -> redI,
        blue -> blueI],
      bicycles.@RAN.#3 ==> manufacturer AS Manufacturer WITH [
        bicycles.@RAN.#3.#1 ==> name AS STRING,
        bicycles.@RAN.#3.#2 ==> phone AS STRING,
        bicycles.@RAN.#3.#3 ==> address AS STRING
      ]
    ],
    bicycles.@DOM.@CARD ==> cant AS INT
  ];
  bikeid? ==> bkid AS STRING;
  b? ==> bike AS Bike WITH [
    b?.#1 ==> size AS bikeSize MAP [
      s18 -> s18I,
      s20 -> s20I,
      s24 -> s24I,
      s26 -> s26I,
      s28 -> s28I,
      s29 -> s29I
    ],
    b?.#2 ==> color AS Color MAP [
      yellow -> yellowI,
      red -> redI,
      blue -> blueI
    ],
    b?.#3 ==> manufacturer AS Manufacturer WITH [
      b?.#3.#1 ==> name AS STRING,
      b?.#3.#2 ==> phone AS STRING,
      b?.#3.#3 ==> address AS STRING
    ]
  ];
```

Se especifica que:

- La variable de especificación `bicycles` se refina como `bikes` que es un array de longitud tres cuyos elementos son de tipo `Pair`. A continuación se indica que:
  - `bicycles.@DOM ==>`...: cada elemento del dominio de `bicycles` se refina como el miembro `fst` (primer miembro de `Pair`).
  - `bicycles.@RAN ==>`...: cada elemento del rango de `bicycles` se refina como el miembro `snd` (segundo miembro de `Pair`) de tipo `Bike`. A su vez, se especifica como refinar `snd`:
    - `bicycles.@RAN.#1 ==>`...: indica que el primer componente se refina como `size`, estableciendo el mapeo correspondiente.

- `bicycles.@RAN.#2 ==>...`: indica que el segundo componente se refina como `color`, estableciendo el mapeo correspondiente.
- `bicycles.@RAN.#3 ==>...`: indica que el tercer componente se refina como `manufacturer`. Por último, se especifica en mayor detalle, como refinar cada uno de los miembros del tercer componente:
  - `bicycles.@RAN.#3.#1 ==>...`: indica que el primer componente se refina como `name`.
  - `bicycles.@RAN.#3.#2 ==>...`: indica que el segundo componente se refina como `phone`.
  - `bicycles.@RAN.#3.#3 ==>...`: indica que el tercer componente se refina como `address`.

- `bicycles.@DOM.@CARD ==>...`: indica que el cardinal del dominio de `bicycles` se refina a `cant`.

- `bikeid?` se refina a la variable de implementación `bkid`.
- `b?` se refina a la variable de implementación `bike` del mismo modo que cada uno de los elementos del rango de `bicycles`.

7.1.4.1.4 Especificación de método a testear

```
@UUT
addBike(bkid, bike);
```

Indica que el método a testear es `addBike` y sus parámetros son `bkid` y `bike`

Las reglas de refinamiento `bicyclesSetAsArrayOfContractsRemove` y `bicyclesSetAsArrayOfContractsGetManufacturer` tienen una definición similar salvo que no se especifica cómo refinar `b?` y en la sección `@UUT` se hace la llamada al método correspondiente en cada caso.

7.1.4.2. Contrato de records

7.1.4.2.1 Identificador de regla

```
@RRULE bicyclesSetAsContractOfRecordsAdd
```

Define que el nombre de la regla de refinamiento es `bicyclesSetAsContractOfRecordsAdd`.

7.1.4.2.2 Definición de tipos de datos

```
@DATATYPES
  DATATYPE bikeSize = ENUM bikeSize ( s18I, s20I, s24I, s26I, s28I, s29I );
  DATATYPE Color = ENUM Color ( yellowI, redI, blueI );
  DATATYPE Manufacturer = MODULE "Manufacturer" CONSTRUCTOR new () SETTER set
      (name : STRING, phone: STRING, address: STRING) GETTER get (member :
      STRING);
  DATATYPE Bike = MODULE "Bike" CONSTRUCTOR new () SETTER set (size : bikeSize
      , color: Color, manufacturer: Manufacturer) GETTER get (member : STRING)
      ;
  DATATYPE Pair = RECORD Pair (fst : STRING, snd: Bike);
  DATATYPE Set = MODULE "Set" CONSTRUCTOR new () SETTER push (pair : Pair)
      GETTER pop ();
```

Los tipos de datos no detallados a continuación pueden encontrarse en 7.1.4.1.2

- Pair: record que contiene dos miembros:
  - fst : STRING
  - snd : Bike
- Set: contrato de constructor `new`, setter `push`, getter `pop` y se compone del miembro `pair` de tipo `Pair`.

7.1.4.2.3 Especificación de leyes de refinamiento

```
@LAWS
  bicycles ==> bikes AS Set WITH [
    @ELEM ==> pair AS Pair WITH [
      @ELEM.#1 ==> .fst AS STRING,
      @ELEM.#2 ==> .snd AS Bike WITH [
        @ELEM.#2.#1 ==> size AS bikeSize MAP [
          s18 -> s18I,
          s20 -> s20I,
          s24 -> s24I,
          s26 -> s26I,
          s28 -> s28I,
          s29 -> s29I
        ],
        @ELEM.#2.#2 ==> color AS Color MAP [
          yellow -> yellowI,
          red -> redI,
          blue -> blueI
        ],
        @ELEM.#2.#3 ==> manufacturer AS Manufacturer WITH [
          @ELEM.#2.#3.#1 ==> name AS STRING,
          @ELEM.#2.#3.#2 ==> phone AS STRING,
          @ELEM.#2.#3.#3 ==> address AS STRING
        ]
      ]
    ]
  ];
  bikeid? ==> bkid AS STRING;
  b? ==> bike AS Bike WITH [
    b?.#1 ==> size AS bikeSize MAP [
      s18 -> s18I,
      s20 -> s20I,
      s24 -> s24I,
      s26 -> s26I,
      s28 -> s28I,
      s29 -> s29I],
    b?.#2 ==> color AS Color MAP [
      yellow -> yellowI,
      red -> redI,
      blue -> blueI],
    b?.#3 ==> manufacturer AS Manufacturer WITH [
      b?.#3.#1 ==> name AS STRING,
      b?.#3.#2 ==> phone AS STRING,
      b?.#3.#3 ==> address AS STRING]
    ];
```

Se especifica que:

- La variable de especificación `bicycles` se refina como `bikes` que es un contrato que representa un conjunto. Se indica que:
  - Cada elemento de `bicycles` se refina como `pair` de tipo `Pair` donde:
    - `@ELEM.#1 ==>...`: el primer miembro de cada elemento de `bicycles` se refina como el miembro `fst` (primer miembro de `Pair`).
    - `@ELEM.#2 ==>...`: el segundo miembro de cada elemento de `bicycles` se refina como el miembro `snd` (segundo miembro de `Pair`) de tipo `Bike`. Todas las leyes de refinamiento que se definen dentro de la cláusula `WITH` que sigue, expresan lo mismo que lo definido en la sección 7.1.4.1.3 solo que se reemplaza `bicycles.@RAN` por `@ELEM.#2`, con lo cual no se continuará con el detalle.
- `bikeid?` se refina a la variable de implementación `bkid`.
- `b?` se refina a la variable de implementación `bike` del mismo modo que el segundo miembro de cada elemento de `bicycles`.

7.1.4.2.4 Especificación de método a testear

```
@UUT
addBike(bkid, bike);
```

Indica que el método a testear es `addBike` y sus parámetros son `bkid` y `bike`.

Al igual que en la sección 7.1.4.1, las reglas de refinamiento `bicyclesSetAsContractOfRecordsRemove` y `bicyclesSetAsContractOfRecordsGetManufacturer` tienen una definición similar salvo que no se especifica cómo refinar `b?` y en la sección `@UUT` se hace la llamada al método correspondiente en cada caso.

7.1.5. Casos concretos de prueba en Perl

7.1.5.1. Implementación 1: array de contratos

7.1.5.1.1 Operación AddBike

Ejecutando los comandos

```
loadrefrule bicyclesSetAsArrayOfContractsAdd.atcal
refine AddBike_VIS to test bicyclesSetAsArrayOfContracts.pl implemented in
    perl with bicyclesSetAsArrayOfContractsAdd
```

se generan

- AddBike_SP_14_CTCASE

```
#!/usr/bin/perl
use feature qw(say);
use YAML::XS;
use Data::Dumper;

sub __fastest_dump {
  open F, '>', 'state.yml';
  print F Dumper ( @_ );
  close F;
}

# Defino el enumerado para tamaños de bicicletas
use constant {
    s18I  => 's18I',
    s20I  => 's20I',
    s24I  => 's24I',
    s26I  => 's26I',
    s28I  => 's28I',
    s29I  => 's29I'
};

# Defino el enumerado para colores
use constant {
    yellowI => 'yellowI',
    redI  => 'redI',
    blueI => 'blueI'
};

package Pair;

sub new {
    my ($class,$args) = @_;
    my $self = bless { }, $class;
}

sub set {
  my ($self, $fst, $snd) = @_;
  $self->{"fst"} = $fst;
  $self->{"snd"} = $snd;
}

sub get {
  my $self = shift;
  my $arg = shift;
  return $self->{$arg};
}

package Bike;

sub new {
    my ($class,$args) = @_;
    my $self = bless { }, $class;
}
```

```
54
55 sub set {
56   my ($self, $size, $color, $manufacturer) = @_;
57   $self->{"size"} = $size;
58   $self->{"color"} = $color;
59   $self->{"manufacturer"} = $manufacturer;
60 }
61
62 sub get {
63   my $self = shift;
64   my $arg = shift;
65   return $self->{$arg};
66 }
67
68 package Manufacturer;
69
70 sub new {
71     my ($class,$args) = @_;
72     my $self = bless { }, $class;
73 }
74
75 sub set {
76   my ($self, $name, $phone, $address) = @_;
77   $self->{"name"} = $name;
78   $self->{"phone"} = $phone;
79   $self->{"address"} = $address;
80 }
81
82 sub get {
83   my $self = shift;
84   my $arg = shift;
85   return $self->{$arg};
86 }
87
88 package main;
89
90 # Array de contratos donde se guardarán las bicicletas
91 @bikes = ();
92
93 sub addBike {
94   # Código que implementa addBike
95 }
96
97 sub removeBike {
98   # Código que implementa removeBike
99 }
100
101 sub getBikeManufacturer {
102   # Código que implementa getBikeManufacturer
103 }
104
105 @bikes = ();
106 $bikes_snd_tmp0 = Bike->new();
107 $snd_size0 = s18I;
108 $snd_color0 = yellowI;
109 $snd_manufacturer_tmp0 = Manufacturer->new();
110 $manufacturer_name0 = "name1";
111 $manufacturer_phone0 = "phone2";
112 $manufacturer_address0 = "address3";
113 $snd_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
      $manufacturer_address0);
114 $snd_manufacturer0 = $snd_manufacturer_tmp0;
115 $bikes_snd_tmp0->set($snd_size0,$snd_color0,$snd_manufacturer0);
116 $bikes_snd0 = $bikes_snd_tmp0;
117 $bikes_snd_tmp1 = Bike->new();
118 $snd_size1 = s20I;
119 $snd_color1 = redI;
120 $snd_manufacturer_tmp1 = Manufacturer->new();
121 $manufacturer_name1 = "name1";
122 $manufacturer_phone1 = "phone1";
123 $manufacturer_address1 = "address4";
124 $snd_manufacturer_tmp1->set($manufacturer_name1,$manufacturer_phone1,
      $manufacturer_address1);
125 $snd_manufacturer1 = $snd_manufacturer_tmp1;
126 $bikes_snd_tmp1->set($snd_size1,$snd_color1,$snd_manufacturer1);
127 $bikes_snd1 = $bikes_snd_tmp1;
```

```
$cant = 2;
$bikes_tmp0 = Pair->new();
$bikes_fst0 = "bikeId1";
$bikes_tmp0->set($bikes_fst0,$bikes_snd0);
$bikes[0] = $bikes_tmp0;
$bikes_tmp1 = Pair->new();
$bikes_fst1 = "bikeId2";
$bikes_tmp1->set($bikes_fst1,$bikes_snd1);
$bikes[1] = $bikes_tmp1;
$bkid = "bikeId1";
$bike_tmp = Bike->new();
$bike_size0 = s18I;
$bike_color0 = yellowI;
$bike_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name1";
$manufacturer_phone0 = "phone2";
$manufacturer_address0 = "address3";
$bike_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$bike_manufacturer0 = $bike_manufacturer_tmp0;
$bike_tmp->set($bike_size0,$bike_color0,$bike_manufacturer0);
$bike = $bike_tmp;
addBike($bkid,$bike);
$state->{'cant'} = $cant;
$state->{'bkid'} = $bkid;
$state->{'bikes'} =  \@bikes;
$state->{'bike'} =  \$bike;
__fastest_dump($state);
```

- AddBike_SP_2_CTCASE

```
...
@bikes = ();
$cant = 0;
$bkid = "bikeId4";
$bike_tmp = Bike->new();
$bike_size0 = s18I;
$bike_color0 = yellowI;
$bike_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name1";
$manufacturer_phone0 = "phone2";
$manufacturer_address0 = "address3";
$bike_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$bike_manufacturer0 = $bike_manufacturer_tmp0;
$bike_tmp->set($bike_size0,$bike_color0,$bike_manufacturer0);
$bike = $bike_tmp;
...
```

- AddBike_SP_4_CTCASE

```
...
@bikes = ();
$bikes_snd_tmp0 = Bike->new();
$snd_size0 = s20I;
$snd_color0 = redI;
$snd_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name1";
$manufacturer_phone0 = "phone1";
$manufacturer_address0 = "address1";
$snd_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$snd_manufacturer0 = $snd_manufacturer_tmp0;
$bikes_snd_tmp0->set($snd_size0,$snd_color0,$snd_manufacturer0);
$bikes_snd0 = $bikes_snd_tmp0;
$cant = 1;
$bikes_tmp0 = Pair->new();
$bikes_fst0 = "bikeId2";
$bikes_tmp0->set($bikes_fst0,$bikes_snd0);
$bikes[0] = $bikes_tmp0;
$bkid = "bikeId1";
$bike_tmp = Bike->new();
$bike_size0 = s18I;
$bike_color0 = yellowI;
$bike_manufacturer_tmp0 = Manufacturer->new();
```

```
$manufacturer_name0 = "name2";
$manufacturer_phone0 = "phone3";
$manufacturer_address0 = "address4";
$bike_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$bike_manufacturer0 = $bike_manufacturer_tmp0;
$bike_tmp->set($bike_size0,$bike_color0,$bike_manufacturer0);
$bike = $bike_tmp;
...
```

- AddBike_SP_12_CTCASE

```
...
@bikes = ();
$bikes_snd_tmp0 = Bike->new();
$snd_size0 = s20I;
$snd_color0 = redI;
$snd_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name1";
$manufacturer_phone0 = "phone1";
$manufacturer_address0 = "address1";
$snd_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$snd_manufacturer0 = $snd_manufacturer_tmp0;
$bikes_snd_tmp0->set($snd_size0,$snd_color0,$snd_manufacturer0);
$bikes_snd0 = $bikes_snd_tmp0;
$cant = 1;
$bikes_tmp0 = Pair->new();
$bikes_fst0 = "bikeId1";
$bikes_tmp0->set($bikes_fst0,$bikes_snd0);
$bikes[0] = $bikes_tmp0;
$bkid = "bikeId1";
$bike_tmp = Bike->new();
$bike_size0 = s18I;
$bike_color0 = yellowI;
$bike_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name2";
$manufacturer_phone0 = "phone3";
$manufacturer_address0 = "address4";
$bike_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$bike_manufacturer0 = $bike_manufacturer_tmp0;
$bike_tmp->set($bike_size0,$bike_color0,$bike_manufacturer0);
$bike = $bike_tmp;
...
```

- AddBike_SP_15_CTCASE

```
...
@bikes = ();
$bikes_snd_tmp0 = Bike->new();
$snd_size0 = s18I;
$snd_color0 = yellowI;
$snd_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name2";
$manufacturer_phone0 = "phone3";
$manufacturer_address0 = "address4";
$snd_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$snd_manufacturer0 = $snd_manufacturer_tmp0;
$bikes_snd_tmp0->set($snd_size0,$snd_color0,$snd_manufacturer0);
$bikes_snd0 = $bikes_snd_tmp0;
$cant = 1;
$bikes_tmp0 = Pair->new();
$bikes_fst0 = "bikeId1";
$bikes_tmp0->set($bikes_fst0,$bikes_snd0);
$bikes[0] = $bikes_tmp0;
$bkid = "bikeId1";
$bike_tmp = Bike->new();
$bike_size0 = s18I;
$bike_color0 = yellowI;
$bike_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name2";
$manufacturer_phone0 = "phone3";
$manufacturer_address0 = "address4";
```

```
$bike_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$bike_manufacturer0 = $bike_manufacturer_tmp0;
$bike_tmp->set($bike_size0,$bike_color0,$bike_manufacturer0);
$bike = $bike_tmp;
...
```

7.1.5.1.2 Operación RemoveBike

Ejecutando los comandos

```
loadrefrule bicyclesSetAsArrayOfContractsRemove.atcal
refine RemoveBike_VIS to test bicyclesSetAsArrayOfContracts.pl implemented in
    perl with bicyclesSetAsArrayOfContractsRemove
```

se generan

- RemoveBike_SP_3_CTCASE

```
...
@bikes = ();
$bikes_snd_tmp0 = Bike->new();
$snd_size0 = s20I;
$snd_color0 = redI;
$snd_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name1";
$manufacturer_phone0 = "phone1";
$manufacturer_address0 = "address2";
$snd_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$snd_manufacturer0 = $snd_manufacturer_tmp0;
$bikes_snd_tmp0->set($snd_size0,$snd_color0,$snd_manufacturer0);
$bikes_snd0 = $bikes_snd_tmp0;
$cant = 1;
$bikes_tmp0 = Pair->new();
$bikes_fst0 = "bikeId1";
$bikes_tmp0->set($bikes_fst0,$bikes_snd0);
$bikes[0] = $bikes_tmp0;
$bkid = "bikeId1";
removeBike($bkid);
$state->{'cant'} = $cant;
$state->{'bkid'} = $bkid;
$state->{'bikes'} =  \@bikes;
__fastest_dump($state);
```

- RemoveBike_SP_4_CTCASE

```
...
@bikes = ();
$bikes_snd_tmp0 = Bike->new();
$snd_size0 = s20I;
$snd_color0 = redI;
$snd_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name1";
$manufacturer_phone0 = "phone1";
$manufacturer_address0 = "address1";
$snd_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$snd_manufacturer0 = $snd_manufacturer_tmp0;
$bikes_snd_tmp0->set($snd_size0,$snd_color0,$snd_manufacturer0);
$bikes_snd0 = $bikes_snd_tmp0;
$bikes_snd_tmp1 = Bike->new();
$snd_size1 = s20I;
$snd_color1 = redI;
$snd_manufacturer_tmp1 = Manufacturer->new();
$manufacturer_name1 = "name1";
$manufacturer_phone1 = "phone1";
$manufacturer_address1 = "address2";
$snd_manufacturer_tmp1->set($manufacturer_name1,$manufacturer_phone1,
    $manufacturer_address1);
$snd_manufacturer1 = $snd_manufacturer_tmp1;
$bikes_snd_tmp1->set($snd_size1,$snd_color1,$snd_manufacturer1);
$bikes_snd1 = $bikes_snd_tmp1;
$cant = 2;
$bikes_tmp0 = Pair->new();
```

```
$bikes_fst0 = "bikeId1";
$bikes_tmp0->set($bikes_fst0,$bikes_snd0);
$bikes[0] = $bikes_tmp0;
$bikes_tmp1 = Pair->new();
$bikes_fst1 = "bikeId2";
$bikes_tmp1->set($bikes_fst1,$bikes_snd1);
$bikes[1] = $bikes_tmp1;
$bkid = "bikeId1";
...
```

- RemoveBike_SP_8_CTCASE

```
...
@bikes = ();
$cant = 0;
$bkid = "bikeId1";
...
```

- RemoveBike_SP_12_CTCASE

```
...
@bikes = ();
$bikes_snd_tmp0 = Bike->new();
$snd_size0 = s20I;
$snd_color0 = redI;
$snd_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name1";
$manufacturer_phone0 = "phone1";
$manufacturer_address0 = "address1";
$snd_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$snd_manufacturer0 = $snd_manufacturer_tmp0;
$bikes_snd_tmp0->set($snd_size0,$snd_color0,$snd_manufacturer0);
$bikes_snd0 = $bikes_snd_tmp0;
$cant = 1;
$bikes_tmp0 = Pair->new();
$bikes_fst0 = "bikeId2";
$bikes_tmp0->set($bikes_fst0,$bikes_snd0);
$bikes[0] = $bikes_tmp0;
$bkid = "bikeId1";
...
```

7.1.5.1.3 Operación GetBikeManufacturer

Ejecutando los comandos

```
loadrefrule bicyclesSetAsArrayOfContractsGetManufacturer.atcal
refine GetBikeManufacturer_VIS to test bicyclesSetAsArrayOfContracts.pl
    implemented in perl with bicyclesSetAsArrayOfContractsGetManufacturer
```

se generan

- GetBikeManufacturer_DNF_1_CTCASE

```
...
@bikes = ();
$bikes_snd_tmp0 = Bike->new();
$snd_size0 = s20I;
$snd_color0 = redI;
$snd_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name1";
$manufacturer_phone0 = "phone1";
$manufacturer_address0 = "address2";
$snd_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$snd_manufacturer0 = $snd_manufacturer_tmp0;
$bikes_snd_tmp0->set($snd_size0,$snd_color0,$snd_manufacturer0);
$bikes_snd0 = $bikes_snd_tmp0;
$cant = 1;
$bikes_tmp0 = Pair->new();
$bikes_fst0 = "bikeId1";
$bikes_tmp0->set($bikes_fst0,$bikes_snd0);
$bikes[0] = $bikes_tmp0;
$bkid = "bikeId1";
```

```
getBikeManufacturer($bkid);
$state->{'cant'} = $cant;
$state->{'bkid'} = $bkid;
$state->{'bikes'} =  \@bikes;
__fastest_dump($state);
```

- GetBikeManufacturer_DNF_2_CTCASE

```
...
@bikes = ();
$cant = 0;
$bkid = "bikeId1";
...
```

7.1.5.2. Implementación 2: contrato de records

7.1.5.2.1 Operación AddBike

Ejecutando los comandos

```
loadrefrule bicyclesSetAsContractOfRecordsAdd.atcal
refine AddBike_VIS to test bicyclesSetAsContractOfRecords.pl implemented in
    perl with bicyclesSetAsContractOfRecordsAdd
```

se generan

- AddBike_SP_14_CTCASE

```
#!/usr/bin/perl
use feature qw(say);
use YAML::XS;
use Data::Dumper;

sub __fastest_dump {
  open F, '>', 'state.yml';
  print F Dumper ( @_ );
  close F;
}

# Defino el enumerado para tamaños de bicicletas
use constant {
    s18I  => 's18I',
    s20I  => 's20I',
    s24I  => 's24I',
    s26I  => 's26I',
    s28I  => 's28I',
    s29I  => 's29I'
};

# Defino el enumerado para colores
use constant {
    yellowI => 'yellowI',
    redI  => 'redI',
    blueI => 'blueI'
};

package Set;

sub new {
    my ($class,$args) = @_;
    @emptyArray = ();
    my $self = bless {"elem" => \@emptyArray}, $class;
}

sub push {
  my ($self, $pair) = @_;
  @elem = @{$self->{"elem"}};
  $elemArraySize = @elem;
  $self->{"elem"}[$elemArraySize] = $pair;
}

sub pop {
  my $self = shift;
  my $arg = shift;
  @elem = @{$self->{"elem"}};
```

```
  $elemArraySize = @elem;
  if($elemArraySize == 0) {
    die;
  }
  $index = $elemArraySize-1;
  $result = $self->{"elem"}[$index];
  delete $self->{"elem"}[$index];
  return $result;
}

package Bike;

sub new {
    my ($class,$args) = @_;
    my $self = bless { }, $class;
}

sub set {
  my ($self, $size, $color, $manufacturer) = @_;
  $self->{"size"} = $size;
  $self->{"color"} = $color;
  $self->{"manufacturer"} = $manufacturer;
}

sub get {
  my $self = shift;
  my $arg = shift;
  return $self->{$arg};
}

package Manufacturer;

sub new {
    my ($class,$args) = @_;
    my $self = bless { }, $class;
}

sub set {
  my ($self, $name, $phone, $address) = @_;
  $self->{"name"} = $name;
  $self->{"phone"} = $phone;
  $self->{"address"} = $address;
}

sub get {
  my $self = shift;
  my $arg = shift;
  return $self->{$arg};
}

package main;

# Objeto donde se guardarán las bicicletas
$bikes = Set->new();

sub addBike {
    # Código que implementa addBike
}

sub removeBike {
  # Código que implementa removeBike
}

sub getBikeManufacturer {
  # Código que implementa getBikeManufacturer
}

$bikes_tmp = Set->new();
$pair_fst0 = "bikeId1";
$pair_snd_tmp0 = Bike->new();
$snd_size0 = s18I;
$snd_color0 = yellowI;
$snd_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name1";
$manufacturer_phone0 = "phone2";
$manufacturer_address0 = "address3";
```

```
$snd_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$snd_manufacturer0 = $snd_manufacturer_tmp0;
$pair_snd_tmp0->set($snd_size0,$snd_color0,$snd_manufacturer0);
$pair_snd0 = $pair_snd_tmp0;
$bikes_pair0 = {"fst" => $pair_fst0, "snd" => $pair_snd0};
$pair_fst1 = "bikeId2";
$pair_snd_tmp1 = Bike->new();
$snd_size1 = s20I;
$snd_color1 = redI;
$snd_manufacturer_tmp1 = Manufacturer->new();
$manufacturer_name1 = "name1";
$manufacturer_phone1 = "phone1";
$manufacturer_address1 = "address4";
$snd_manufacturer_tmp1->set($manufacturer_name1,$manufacturer_phone1,
    $manufacturer_address1);
$snd_manufacturer1 = $snd_manufacturer_tmp1;
$pair_snd_tmp1->set($snd_size1,$snd_color1,$snd_manufacturer1);
$pair_snd1 = $pair_snd_tmp1;
$bikes_pair1 = {"fst" => $pair_fst1, "snd" => $pair_snd1};
$bikes_tmp->push($bikes_pair0);
$bikes_tmp->push($bikes_pair1);
$bikes = $bikes_tmp;
$bkid = "bikeId1";
$bike_tmp = Bike->new();
$bike_size0 = s18I;
$bike_color0 = yellowI;
$bike_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name1";
$manufacturer_phone0 = "phone2";
$manufacturer_address0 = "address3";
$bike_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$bike_manufacturer0 = $bike_manufacturer_tmp0;
$bike_tmp->set($bike_size0,$bike_color0,$bike_manufacturer0);
$bike = $bike_tmp;
addBike($bkid,$bike);
$state->{'bkid'} = $bkid;
$state->{'bikes'} =  \$bikes;
$state->{'bike'} =  \$bike;
__fastest_dump($state);
```

- AddBike_SP_2_CTCASE

```
...
$bikes_tmp = Set->new();
$bikes = $bikes_tmp;
$bkid = "bikeId4";
$bike_tmp = Bike->new();
$bike_size0 = s18I;
$bike_color0 = yellowI;
$bike_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name1";
$manufacturer_phone0 = "phone2";
$manufacturer_address0 = "address3";
$bike_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$bike_manufacturer0 = $bike_manufacturer_tmp0;
$bike_tmp->set($bike_size0,$bike_color0,$bike_manufacturer0);
$bike = $bike_tmp;
...
```

- AddBike_SP_4_CTCASE

```
...
$bikes_tmp = Set->new();
$pair_fst0 = "bikeId2";
$pair_snd_tmp0 = Bike->new();
$snd_size0 = s20I;
$snd_color0 = redI;
$snd_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name1";
$manufacturer_phone0 = "phone1";
$manufacturer_address0 = "address1";
$snd_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
```

```
$snd_manufacturer0 = $snd_manufacturer_tmp0;
$pair_snd_tmp0->set($snd_size0,$snd_color0,$snd_manufacturer0);
$pair_snd0 = $pair_snd_tmp0;
$bikes_pair0 = {"fst" => $pair_fst0, "snd" => $pair_snd0};
$bikes_tmp->push($bikes_pair0);
$bikes = $bikes_tmp;
$bkid = "bikeId1";
$bike_tmp = Bike->new();
$bike_size0 = s18I;
$bike_color0 = yellowI;
$bike_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name2";
$manufacturer_phone0 = "phone3";
$manufacturer_address0 = "address4";
$bike_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$bike_manufacturer0 = $bike_manufacturer_tmp0;
$bike_tmp->set($bike_size0,$bike_color0,$bike_manufacturer0);
$bike = $bike_tmp;
...
```

- AddBike_SP_12_CTCASE

```
...
$bikes_tmp = Set->new();
$pair_fst0 = "bikeId1";
$pair_snd_tmp0 = Bike->new();
$snd_size0 = s20I;
$snd_color0 = redI;
$snd_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name1";
$manufacturer_phone0 = "phone1";
$manufacturer_address0 = "address1";
$snd_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$snd_manufacturer0 = $snd_manufacturer_tmp0;
$pair_snd_tmp0->set($snd_size0,$snd_color0,$snd_manufacturer0);
$pair_snd0 = $pair_snd_tmp0;
$bikes_pair0 = {"fst" => $pair_fst0, "snd" => $pair_snd0};
$bikes_tmp->push($bikes_pair0);
$bikes = $bikes_tmp;
$bkid = "bikeId1";
$bike_tmp = Bike->new();
$bike_size0 = s18I;
$bike_color0 = yellowI;
$bike_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name2";
$manufacturer_phone0 = "phone3";
$manufacturer_address0 = "address4";
$bike_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$bike_manufacturer0 = $bike_manufacturer_tmp0;
$bike_tmp->set($bike_size0,$bike_color0,$bike_manufacturer0);
$bike = $bike_tmp;
...
```

- AddBike_SP_15_CTCASE

```
...
$bikes_tmp = Set->new();
$pair_fst0 = "bikeId1";
$pair_snd_tmp0 = Bike->new();
$snd_size0 = s18I;
$snd_color0 = yellowI;
$snd_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name2";
$manufacturer_phone0 = "phone3";
$manufacturer_address0 = "address4";
$snd_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$snd_manufacturer0 = $snd_manufacturer_tmp0;
$pair_snd_tmp0->set($snd_size0,$snd_color0,$snd_manufacturer0);
$pair_snd0 = $pair_snd_tmp0;
$bikes_pair0 = {"fst" => $pair_fst0, "snd" => $pair_snd0};
$bikes_tmp->push($bikes_pair0);
```

```
$bikes = $bikes_tmp;
$bkid = "bikeId1";
$bike_tmp = Bike->new();
$bike_size0 = s18I;
$bike_color0 = yellowI;
$bike_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name2";
$manufacturer_phone0 = "phone3";
$manufacturer_address0 = "address4";
$bike_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$bike_manufacturer0 = $bike_manufacturer_tmp0;
$bike_tmp->set($bike_size0,$bike_color0,$bike_manufacturer0);
$bike = $bike_tmp;
...
```

7.1.5.2.2 Operación RemoveBike

Ejecutando los comandos

```
loadrefrule bicyclesSetAsSetAsContractOfRecordsRemove.atcal
refine RemoveBike_VIS to test bicyclesSetAsContractOfRecords.pl implemented in
    perl with bicyclesSetAsSetAsContractOfRecordsRemove
```

se generan

- RemoveBike_SP_3_CTCASE

```
...
$bikes_tmp = Set->new();
$pair_fst0 = "bikeId1";
$pair_snd_tmp0 = Bike->new();
$snd_size0 = s20I;
$snd_color0 = redI;
$snd_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name1";
$manufacturer_phone0 = "phone1";
$manufacturer_address0 = "address2";
$snd_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$snd_manufacturer0 = $snd_manufacturer_tmp0;
$pair_snd_tmp0->set($snd_size0,$snd_color0,$snd_manufacturer0);
$pair_snd0 = $pair_snd_tmp0;
$bikes_pair0 = {"fst" => $pair_fst0, "snd" => $pair_snd0};
$bikes_tmp->push($bikes_pair0);
$bikes = $bikes_tmp;
$bkid = "bikeId1";
removeBike($bkid);
$state->{'bkid'} = $bkid;
$state->{'bikes'} =  \$bikes;
__fastest_dump($state);
```

- RemoveBike_SP_4_CTCASE

```
...
$bikes_tmp = Set->new();
$pair_fst0 = "bikeId1";
$pair_snd_tmp0 = Bike->new();
$snd_size0 = s20I;
$snd_color0 = redI;
$snd_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name1";
$manufacturer_phone0 = "phone1";
$manufacturer_address0 = "address1";
$snd_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$snd_manufacturer0 = $snd_manufacturer_tmp0;
$pair_snd_tmp0->set($snd_size0,$snd_color0,$snd_manufacturer0);
$pair_snd0 = $pair_snd_tmp0;
$bikes_pair0 = {"fst" => $pair_fst0, "snd" => $pair_snd0};
$pair_fst1 = "bikeId2";
$pair_snd_tmp1 = Bike->new();
$snd_size1 = s20I;
$snd_color1 = redI;
$snd_manufacturer_tmp1 = Manufacturer->new();
```

```
$manufacturer_name1 = "name1";
$manufacturer_phone1 = "phone1";
$manufacturer_address1 = "address2";
$snd_manufacturer_tmp1->set($manufacturer_name1,$manufacturer_phone1,
    $manufacturer_address1);
$snd_manufacturer1 = $snd_manufacturer_tmp1;
$pair_snd_tmp1->set($snd_size1,$snd_color1,$snd_manufacturer1);
$pair_snd1 = $pair_snd_tmp1;
$bikes_pair1 = {"fst" => $pair_fst1, "snd" => $pair_snd1};
$bikes_tmp->push($bikes_pair0);
$bikes_tmp->push($bikes_pair1);
$bikes = $bikes_tmp;
$bkid = "bikeId1";
...
```

- RemoveBike_SP_8_CTCASE

```
...
$bikes_tmp = Set->new();
$bikes = $bikes_tmp;
$bkid = "bikeId1";
...
```

- RemoveBike_SP_12_CTCASE

```
...
$bikes_tmp = Set->new();
$pair_fst0 = "bikeId2";
$pair_snd_tmp0 = Bike->new();
$snd_size0 = s20I;
$snd_color0 = redI;
$snd_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name1";
$manufacturer_phone0 = "phone1";
$manufacturer_address0 = "address1";
$snd_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$snd_manufacturer0 = $snd_manufacturer_tmp0;
$pair_snd_tmp0->set($snd_size0,$snd_color0,$snd_manufacturer0);
$pair_snd0 = $pair_snd_tmp0;
$bikes_pair0 = {"fst" => $pair_fst0, "snd" => $pair_snd0};
$bikes_tmp->push($bikes_pair0);
$bikes = $bikes_tmp;
$bkid = "bikeId1";
...
```

7.1.5.2.3 Operación GetBikeManufacturer

Ejecutando los comandos

```
loadrefrule bicyclesSetAsContractOfRecordsGetManufacturer.atcal
refine GetBikeManufacturer_VIS to test bicyclesSetAsContractOfRecords.pl
    implemented in perl with bicyclesSetAsContractOfRecordsGetManufacturer
```

se generan

- GetBikeManufacturer_DNF_1_CTCASE

```
...
$bikes_tmp = Set->new();
$pair_fst0 = "bikeId1";
$pair_snd_tmp0 = Bike->new();
$snd_size0 = s20I;
$snd_color0 = redI;
$snd_manufacturer_tmp0 = Manufacturer->new();
$manufacturer_name0 = "name1";
$manufacturer_phone0 = "phone1";
$manufacturer_address0 = "address2";
$snd_manufacturer_tmp0->set($manufacturer_name0,$manufacturer_phone0,
    $manufacturer_address0);
$snd_manufacturer0 = $snd_manufacturer_tmp0;
$pair_snd_tmp0->set($snd_size0,$snd_color0,$snd_manufacturer0);
$pair_snd0 = $pair_snd_tmp0;
$bikes_pair0 = {"fst" => $pair_fst0, "snd" => $pair_snd0};
```

```
$bikes_tmp->push($bikes_pair0);
$bikes = $bikes_tmp;
$bkid = "bikeId1";
getBikeManufacturer($bkid);
$state->{'bkid'} = $bkid;
$state->{'bikes'} =  \$bikes;
__fastest_dump($state);
```

- GetBikeManufacturer_DNF_2_CTCASE

```
...
$bikes_tmp = Set->new();
$bikes = $bikes_tmp;
$bkid = "bikeId1";
...
```

Capítulo 8

Conclusiones y trabajo futuro

8.1. Conclusiones

En el transcurso de esta tesina se desarrolló el prototipo de un sistema que permite el refinamiento de casos de pruebas abstractos generados a partir de una especificación Z. Los casos concretos generados son expresados en el lenguaje de implementación Perl. Este sistema se construyó como un módulo más de la herramienta FASTEST, la cual permite semi-automatizar el proceso de testing de software basado en especificaciones funcionales escritas en Z.

En este trabajo, se presenta al lenguaje ATCAL, que se ubica como nexo entre la especificación funcional y la implementación del sistema a testear. ATCAL permite expresar con suficiente nivel de detalle la relación que existe entre las variables de estado, de entrada y salida, incluidos en los casos abstractos de prueba, y las variables correspondientes en la implementación. Desde una mirada general del proceso, este lenguaje conforma la capa semántica necesaria para vincular correctamente lo abstracto (especificación funcional) con lo concreto (implementación).

De este modo, se logra el propósito planteado inicialmente de extender la herramienta de software FASTEST con un módulo que automatice el proceso de refinamiento que concretiza casos abstractos de prueba, generados a partir de una especificación Z, a casos concretos en Perl. Dicho proceso fue sometido a pruebas donde se puede observar tanto la utilidad y poder de expresividad de las leyes de refinamiento provistas por le lenguaje ATCAL, como también los scripts Perl resultantes en cada caso de estudio. El detalle de estos casos pueden encontrarse en 7.1 y en los distintos apéndices.

El código fuente de la implementación del proceso de refinamiento se puede encontrar en link, repositorio que, además, contiene los casos de estudio presentados en esta tesina.

8.2. Trabajo futuro

Si bien en esta tesina se buscó incorporar las distintas simplificaciones y funcionalidades que surgieron a lo largo de su desarrollo, el autor decidió que algunos puntos se pueden postergar y quedar como posibles mejoras a la implementación actual. A continuación se presentan dichos puntos:

- Permitir que las especificaciones de refinamiento acepten cláusulas `WITH` en las que solo se detalle el nombre de los miembros internos de records y contratos, es decir, omitir el tipo. La información no incluida en dicha cláusula puede deducirse por contexto, y en caso de llegar a una conclusión de incompatibilidad entre el tipo de la variable de especificación y la variable de implementación, se deberá mostrar el error apropiado.
- Actualmente, al definir un tipo de datos enumerado no existe la posibilidad de establecer el mapeo entre los elementos de la especificación y la implementación. Este se establece al hacer uso del tipo de datos en una ley que lo requiera. Una mejora que hará más sencillo escribir leyes de refinamiento es la que permite establecer dicho mapeo al momento de la definición del tipo de datos, lo cual permite omitir el mapeo cada vez que se utilice el tipo de datos.

- En esta tesina no se ha dado mucha relevancia a las reglas del tipo.

$$\langle \text{biRefLaw} \rangle \quad ::= \quad \langle \text{id} \rangle \ <==> \ \langle \text{refinement} \rangle (, \langle \text{refinement} \rangle)^*$$

  ya que son útiles únicamente en la etapa de abstracción, que no fue considerada como una prioridad de este trabajo. Esta etapa tiene lugar una vez finalizado el refinamiento (de los casos abstractos) y la ejecución de los casos concretos obtenidos. A partir de aquí, se procede a abstraer los resultados de dichas ejecuciones al lenguaje de especificación. Este es un punto a profundizar ya que, de completarse, agregará una funcionalidad que permite cerrar el ciclo descripto en el diagrama 3.1.

Apéndice A

ATCAL - Gramática en BNF

En esta sección se presenta la gramática completa de ATCAL usando la notación BNF, extendida con el símbolo `?`. La semántica de éste símbolo expresa que el elemento gramatical a su izquierda puede estar presente como no. Los elementos gramaticales que conforman la gramática son:

⟨refinementRule⟩ ::= @RRULE ⟨id⟩
⟨preamble⟩?
⟨datatypes⟩?
⟨laws⟩
⟨plCode⟩?
⟨uut⟩
⟨epilogue⟩?

⟨preamble⟩ ::= @PREAMBLE (⟨plCode⟩ | ⟨id⟩.@PREAMBLE;)$^{+}$

⟨datatypes⟩ ::= @DATATYPES ⟨typeDecs⟩$^{+}$

⟨typeDecs⟩ ::= DATATYPE ⟨id⟩ = ⟨type⟩;

⟨type⟩ ::= ⟨id⟩
| INT
| FLOAT
| STRING
| REFERENCE
| ARRAY ⟨type⟩(⟨number⟩)
| ENUM ⟨id⟩⟨args⟩
| RECORD ⟨id⟩ (⟨id⟩ : ⟨type⟩ (,⟨id⟩ : ⟨type⟩)*)
| (MODULE ⟨string⟩)? CONSTRUCTOR ⟨id⟩ ⟨contractMembers⟩
SETTER ⟨id⟩ ⟨contractMembers⟩
GETTER ⟨id⟩ ⟨contractMembers⟩

⟨args⟩ ::= ((⟨id⟩ (, ⟨id⟩))*?)

⟨contractMembers⟩ ::= ((⟨id⟩ : ⟨type⟩ (, ⟨id⟩ : ⟨type⟩)*)?)

⟨laws⟩ ::= @LAWS (⟨law⟩;)*

⟨law⟩ ::= (⟨id⟩ :)? (⟨biRefLaw⟩ | ⟨lawRefinement⟩ | ⟨lawReference⟩)

⟨biRefLaw⟩ ::= ⟨id⟩ <==> ⟨refinement⟩(, ⟨refinement⟩)*

⟨lawRefinement⟩ ::= ⟨zExprs⟩ ==> ⟨refinement⟩(, ⟨refinement⟩)*

⟨lawReference⟩ ::= ⟨id⟩.(@LAWS | ⟨id⟩)

⟨refinement⟩ ::= ⟨lvalue⟩ AS ⟨type⟩ ⟨constMapping⟩?⟨withRef⟩?

⟨lvalue⟩ ::= ⟨id⟩
| [⟨number⟩?]
| .⟨id⟩

⟨withRef⟩ ::= WITH [⟨lawRefinement⟩ (, ⟨lawRefinement⟩)*]

⟨constMapping⟩ ::= MAP [⟨constMap⟩ (, ⟨constMap⟩)*]

⟨constMap⟩ ::= ⟨id⟩ -> (⟨id⟩ | ⟨string⟩ | ⟨number⟩)

⟨zExprs⟩ ::= ⟨zExpr⟩ (==> ⟨zExprs⟩)?

⟨zExpr⟩ ::= ⟨id⟩
| ⟨number⟩
| ⟨string⟩
| ⟨auto⟩
| ⟨elem⟩
| ⟨zExpr⟩.⟨tupProj⟩
| < ⟨zExpr⟩(, ⟨zExpr⟩)* >
| ⟨zExpr⟩.⟨dom⟩
| ⟨zExpr⟩.⟨ran⟩
| ⟨zExpr⟩.⟨proj⟩
| ⟨zExpr⟩ ⟨inter⟩ ⟨zExpr⟩
| ⟨zExpr⟩ ⟨union⟩ ⟨zExpr⟩
| ⟨zExpr⟩ ⟨diff⟩ ⟨zExpr⟩
| {⟨zExpr⟩(, ⟨zExpr⟩)*}
| ⟨zExpr⟩.⟨card⟩
| ⟨zExpr⟩ ⟨mul⟩ ⟨zExpr⟩
| ⟨zExpr⟩ ⟨div⟩ ⟨zExpr⟩
| ⟨zExpr⟩ ⟨mod⟩ ⟨zExpr⟩
| ⟨zExpr⟩ ⟨plus⟩ ⟨zExpr⟩
| ⟨zExpr⟩ ⟨minus⟩ ⟨zExpr⟩
| ⟨zExpr⟩ ++ ⟨zExpr⟩
| (⟨zExpr⟩)

⟨epilogue⟩ ::= @EPILOGUE (⟨plCode⟩ | ⟨id⟩.@EPILOGUE;)$^+$

⟨uut⟩ ::= @UUT ⟨id⟩⟨args⟩;
| ⟨id⟩ <== @UUT ⟨id⟩⟨args⟩AS ⟨type⟩;

⟨plCode⟩ ::= @CODESTART (.)* @CODEEND;

⟨id⟩ ::= ⟨id_letter⟩ (⟨id_letter⟩ | ⟨digit⟩)*

⟨id_letter⟩ ::= a . . . z | A . . . Z | _ | ? | !

⟨digit⟩ ::= 0 . . . 9

⟨number⟩ ::= ⟨digit⟩$^+$

⟨plus⟩ ::= +

⟨minus⟩ ::= -

⟨mul⟩ ::= *

⟨div⟩ ::= /

⟨mod⟩ ::= %

⟨dom⟩ ::= @DOM

⟨ran⟩ ::= @RAN

⟨proj⟩ ::= @⟨number⟩

⟨tupProj⟩ ::= #⟨number⟩

⟨inter⟩ ::= /\

⟨union⟩ ::= \/

⟨diff⟩ ::= ~

⟨elem⟩ ::= @ELEM

⟨auto⟩ ::= @AUTOFILL

⟨card⟩ ::= @CARD

⟨string⟩ ::= “(⟨esc⟩ | .)*”

⟨esc⟩ ::= \[btnr”\]

La regla gramatical $\langle esc \rangle$ es utilizada para permitir escapar caracteres especiales y, de este modo, poder parsearlos como cualquier otro caracter.

Apéndice B

Relación de variables de especificación e implementación

En este anexo se continúa la idea de la sección 6.1.2 presentando dos ejemplos más donde se puede establecer la relación entre las variables de los casos abstractos respecto a las variables de los casos concretos. La regla de refinamiento utilizada es la siguiente:

```
@RRULE friends

@DATATYPES
  DATATYPE Person = RECORD Person (name : STRING, age: INT);
  DATATYPE Pair = RECORD Pair (dni : STRING, person: Person);

@LAWS
  people ==> peopleArray AS ARRAY Pair (5) WITH [
    people.@DOM ==> .dni AS STRING,
    people.@RAN ==> .person AS Person WITH [
      people.@RAN.#1 ==> .name AS STRING,
      people.@RAN.#2 ==> .age AS INT
    ]
  ];

  dni? ==> dni AS STRING;

  person? ==> person AS Person WITH [
    person?.#1 ==> .name AS STRING,
    person?.#2 ==> .age AS INT
  ];

@UUT
addFriend(dni, person);
```

A continuación se presentan dos casos de prueba abstractos y sus correspondientes casos concretos donde se puede observar claramente, siguiendo la descripción de la sección 6.1.2, cómo cada variable de un caso abstracto se ve reflejada en una variable de implementación asociada a través de las leyes definidas en la regla de refinamiento `friends`.

B.1. AddFriend_SP_2

- Caso abstracto AddFriend_SP_2_TCASE

AddFriend_SP_2_TCASE

AddFriend_SP_2

$person? = (nAME1 \mapsto 0)$
$people = \emptyset$
$dni? = dNI2$

- Caso concreto AddFriend_SP_2_CTCASE

```
#!/usr/bin/perl
use feature qw(say);
use YAML::XS;
```

```
use Data::Dumper;

sub __fastest_dump {
  open F, '>', 'state.yml';
  print F Dumper ( @_ );
  close F;
}

sub addFriend {
    # Definición de método que se quiere testear
}

@peopleArray = ();
$dni = "dNI2";
$person_name0 = "nAME1";
$person_age0 = 0;
$person = {"name" => $person_name0, "age" => $person_age0};
addFriend($dni,$person);
$state->{'peopleArray'} =  \@peopleArray;
$state->{'dni'} = $dni;
$state->{'person'} =  \$person;
__fastest_dump($state);
```

B.2. AddFriend_SP_14

- Caso abstracto AddFriend_SP_14_TCASE

AddFriend_SP_14_TCASE

AddFriend_SP_14

$person? = (nAME1 \mapsto 0)$
$people = \{(dNI1 \mapsto (nAME1 \mapsto 0)), (dNI2 \mapsto (nAME1 \mapsto 2))\}$
$dni? = dNI1$

- Caso concreto AddFriend_SP_14_CTCASE

```
#!/usr/bin/perl
use feature qw(say);
use YAML::XS;
use Data::Dumper;

sub __fastest_dump {
  open F, '>', 'state.yml';
  print F Dumper ( @_ );
  close F;
}

sub addFriend {
    # Definición de método que se quiere testear
}

@peopleArray = ();
$person_name0 = "nAME1";
$person_age0 = 0;
$peopleArray_person0 = {"name" => $person_name0, "age" => $person_age0};
$person_name1 = "nAME1";
$person_age1 = 2;
$peopleArray_person1 = {"name" => $person_name1, "age" => $person_age1};
$peopleArray_dni0 = "dNI1";
$peopleArray[0] = {"dni" => $peopleArray_dni0, "person" =>
    $peopleArray_person0};
$peopleArray_dni1 = "dNI2";
$peopleArray[1] = {"dni" => $peopleArray_dni1, "person" =>
    $peopleArray_person1};
$dni = "dNI1";
$person_name0 = "nAME1";
$person_age0 = 0;
$person = {"name" => $person_name0, "age" => $person_age0};
addFriend($dni,$person);
$state->{'peopleArray'} =  \@peopleArray;
$state->{'dni'} = $dni;
$state->{'person'} =  \$person;
```

```
35 __fastest_dump($state);
```

Apéndice C

Registro civil

C.1. Especificación Z

Sistema utilizado en las oficinas de un registro civil para gestionar información sobre las personas registradas. Los archivos de este caso de estudio pueden encontrarse en este link.

$[DNI, Name, StreetName]$

$PlaceType ::= house \mid building$

$Gender ::= male \mid female$

Se definen los siguientes tipos básicos:

- `DNI` representa los números de documento de las personas que se registrarán.
- `Name` representa los nombres de las personas que se registrarán.
- `StreetName` representa los nombres de calles donde se ubican las viviendas de los ciudadanos.

y los tipos de datos enumerados:

- `PlaceType` se utiliza para diferenciar el tipo de vivienda. Sus posibles valores son:
  - `house`
  - `building`
- `Gender` representa el género de las personas que se registrarán. Sus posibles valores son:
  - `male`
  - `female`

Con estos tipos de datos se puede proceder a la definición del esquema de estado

CivilRegistry

$$population : DNI \pfun (Name \times \mathbb{N} \times Gender \times (DNI \times DNI))$$
$$dwellings : DNI \pfun (StreetName \times \mathbb{N} \times PlaceType)$$
$$children : DNI \pfun \mathbb{P}\, DNI$$

donde se encuentran tres funciones parciales utilizadas para almacenar información:

- `population`: información relevante por cada ciudadano registrado. Los datos considerados son:
  - Nombre
  - Edad
  - Género
  - Progenitores

- `dwellings`: datos sobre la vivienda de los ciudadanos registrados. Estos son:
  - Nombre de la calle
  - Numero
  - Tipo de vivienda
- `children`: asocia un ciudadano con sus hijos

Habiendo definido lo anterior, es hora de presentar las operaciones de este sistema.

- CitizenRegistration: registra un ciudadano incluyendo los datos pertinentes

$$
\begin{array}{|l}
\hline
\textit{CitizenRegistrationOk} \\
\Delta \textit{CivilRegistry} \\
\textit{dni}? : \textit{DNI} \\
\textit{citizen}? : \textit{Name} \times \mathbb{N} \times \textit{Gender} \times (\textit{DNI} \times \textit{DNI}) \\
\textit{address}? : \textit{StreetName} \times \mathbb{N} \times \textit{PlaceType} \\
\textit{children}? : \mathbb{P}\, \textit{DNI} \\
\hline
\textit{dni}? \notin \operatorname{dom} \textit{population} \\
\textit{population}' = \textit{population} \cup \{\textit{dni}? \mapsto \textit{citizen}?\} \\
\textit{dwellings}' = \textit{dwellings} \cup \{\textit{dni}? \mapsto \textit{address}?\} \\
\textit{children}' = \textit{children} \cup \{\textit{dni}? \mapsto \textit{children}?\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{PersonAlreadyRegistered} \\
\Xi \textit{CivilRegistry} \\
\textit{dni}? : \textit{DNI} \\
\hline
\textit{dni}? \in \operatorname{dom} \textit{population} \\
\hline
\end{array}
$$

$$\textit{CitizenRegistration} == \textit{CitizenRegistrationOk} \lor \textit{PersonAlreadyRegistered}$$

- ChangePersonDwelling: modifica los datos de vivienda asociados a un ciudadano

$$
\begin{array}{|l}
\hline
\textit{ChangePersonDwellingOk} \\
\Delta \textit{CivilRegistry} \\
\textit{dni}? : \textit{DNI} \\
\textit{address}? : \textit{StreetName} \times \mathbb{N} \times \textit{PlaceType} \\
\hline
\textit{dni}? \in \operatorname{dom} \textit{dwellings} \\
\textit{population}' = \textit{population} \\
\textit{dwellings}' = \textit{dwellings} \oplus \{\textit{dni}? \mapsto \textit{address}?\} \\
\textit{children}' = \textit{children} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{InexistingPerson} \\
\Xi \textit{CivilRegistry} \\
\textit{dni}? : \textit{DNI} \\
\hline
\textit{dni}? \notin \operatorname{dom} \textit{population} \\
\hline
\end{array}
$$

$$\textit{ChangePersonDwelling} == \textit{ChangePersonDwellingOk} \lor \textit{InexistingPerson}$$

- AddChild: vincula un/a hijo/a a con un ciudadano

AddChildOk

$\Delta CivilRegistry$
$dni? : DNI$
$child? : DNI$

$dni? \neq child?$
$dni? \notin children\ dni?$
$child? \notin children\ child?$
$dni? \in \mathrm{dom}\ population$
$child? \in \mathrm{dom}\ population$
$child? \notin children\ dni?$
$population' = population$
$dwellings' = dwellings$
$children' = children \oplus \{dni? \mapsto (children\ dni? \cup \{child?\})\}$

ChildAlreadyRegistered

$\Xi CivilRegistry$
$dni? : DNI$
$child? : DNI$

$dni? \notin children\ dni?$
$child? \notin children\ child?$
$dni? \in \mathrm{dom}\ population$
$child? \in children\ dni?$

InexistingDad

$\Xi CivilRegistry$
$dni? : DNI$
$child? : DNI$

$dni? \notin children\ dni?$
$child? \notin children\ child?$
$dni? \notin \mathrm{dom}\ population$

NotRegisteredChild

$\Xi CivilRegistry$
$dni? : DNI$
$child? : DNI$

$dni? \notin children\ dni?$
$child? \notin children\ child?$
$child? \notin \mathrm{dom}\ population$

ChildIsSameAsDad

$\Xi CivilRegistry$
$dni? : DNI$
$child? : DNI$

$dni? \notin children\ dni?$
$child? \notin children\ child?$
$dni? = child?$

$AddChild == AddChildOk \lor ChildAlreadyRegistered \lor InexistingDad \lor NotRegisteredChild \lor ChildIsSameAsDad$

- CitizenDeath: registra el deceso de un ciudadano

*CitizenDeathOk*1

$\Delta CivilRegistry$
$dni? : DNI$

$dni? \in \mathrm{dom}\, population$
$(population\ dni?).4.1 \notin \mathrm{dom}\, children$
$(population\ dni?).4.2 \notin \mathrm{dom}\, children$
$population' = \{dni?\} \lhd population$
$dwellings' = \{dni?\} \lhd dwellings$
$children' = \{dni?\} \lhd children$

*CitizenDeathOk*2

$\Delta CivilRegistry$
$dni? : DNI$

$dni? \in \mathrm{dom}\, population$
$(population\ dni?).4.1 \in \mathrm{dom}\, children$
$(population\ dni?).4.2 \notin \mathrm{dom}\, children$
$population' = \{dni?\} \lhd population$
$dwellings' = \{dni?\} \lhd dwellings$
$children' = \{dni?\} \lhd children \oplus$
$\{(population\ dni?).4.1 \mapsto children\ ((population\ dni?).4.1) \setminus \{dni?\}\}$

*CitizenDeathOk*3

$\Delta CivilRegistry$
$dni? : DNI$

$dni? \in \mathrm{dom}\, population$
$(population\ dni?).4.1 \notin \mathrm{dom}\, children$
$(population\ dni?).4.2 \in \mathrm{dom}\, children$
$population' = \{dni?\} \lhd population$
$dwellings' = \{dni?\} \lhd dwellings$
$children' = \{dni?\} \lhd children \oplus$
$\{(population\ dni?).4.2 \mapsto children\ ((population\ dni?).4.2) \setminus \{dni?\}\}$

*CitizenDeathOk*4

$\Delta CivilRegistry$
$dni? : DNI$

$dni? \in \mathrm{dom}\, population$
$(population\ dni?).4.1 \in \mathrm{dom}\, children$
$(population\ dni?).4.2 \in \mathrm{dom}\, children$
$population' = \{dni?\} \lhd population$
$dwellings' = \{dni?\} \lhd dwellings$
$children' = \{dni?\} \lhd children \oplus$
$\{(population\ dni?).4.1 \mapsto children\ ((population\ dni?).4.1) \setminus \{dni?\}\} \oplus$
$\{(population\ dni?).4.2 \mapsto children\ ((population\ dni?).4.2) \setminus \{dni?\}\}$

$CitizenDeath == CitizenDeathOk1 \lor CitizenDeathOk2$
$\lor\ CitizenDeathOk3 \lor CitizenDeathOk4 \lor InexistingPerson$

C.2. Casos Abstractos de prueba

C.2.1. CitizenRegistration

CitizenRegistration_SP_2_TCASE

CitizenRegistration_SP_2

$children? = \emptyset$
$children = \emptyset$
$dwellings = \emptyset$
$citizen? = (name2, 0, male, (dNI1 \mapsto dNI1))$
$address? = (streetName3, 0, house)$
$dni? = dNI1$
$population = \emptyset$

CitizenRegistration_SP_4_TCASE

CitizenRegistration_SP_4

$children? = \emptyset$
$children = \emptyset$
$dwellings = \emptyset$
$citizen? = (name2, 0, male, (dNI1 \mapsto dNI1))$
$address? = (streetName3, 0, house)$
$dni? = dNI1$
$population = \{(dNI2 \mapsto (name1, 1, female, (dNI1 \mapsto dNI1)))\}$

CitizenRegistration_SP_12_TCASE

CitizenRegistration_SP_12

$children? = \emptyset$
$children = \emptyset$
$dwellings = \emptyset$
$citizen? = (name2, 0, male, (dNI1 \mapsto dNI1))$
$address? = (streetName3, 0, house)$
$dni? = dNI1$
$population = \{(dNI1 \mapsto (name1, 1, female, (dNI1 \mapsto dNI1)))\}$

CitizenRegistration_SP_14_TCASE

CitizenRegistration_SP_14

$children? = \emptyset$
$children = \emptyset$
$dwellings = \emptyset$
$citizen? = (name1, 0, male, (dNI1 \mapsto dNI1))$
$address? = (streetName3, 0, house)$
$dni? = dNI1$
$population = \{(dNI1 \mapsto (name1, 0, male, (dNI1 \mapsto dNI1))),$
$(dNI2 \mapsto (name1, 1, female, (dNI1 \mapsto dNI2)))\}$

CitizenRegistration_SP_15_TCASE

CitizenRegistration_SP_15

$children? = \emptyset$
$children = \emptyset$
$dwellings = \emptyset$
$citizen? = (name2, 0, male, (dNI1 \mapsto dNI1))$
$address? = (streetName3, 0, house)$
$dni? = dNI1$
$population = \{(dNI1 \mapsto (name2, 0, male, (dNI1 \mapsto dNI1)))\}$

C.2.2. ChangePersonDwelling

ChangePersonDwelling_SP_4_TCASE

ChangePersonDwelling_SP_4

$children = \emptyset$
$dwellings = \{(dNI1 \mapsto (streetName1, 1, building))\}$
$address? = (streetName2, 0, house)$
$dni? = dNI1$
$population = \emptyset$

ChangePersonDwelling_SP_6_TCASE

ChangePersonDwelling_SP_6

$children = \emptyset$
$dwellings = \{(dNI1 \mapsto (streetName1, 0, house)), (dNI2 \mapsto (streetName1, 1, house))\}$
$address? = (streetName1, 0, house)$
$dni? = dNI1$
$population = \emptyset$

ChangePersonDwelling_SP_7_TCASE

ChangePersonDwelling_SP_7

$children = \emptyset$
$dwellings = \{(dNI1 \mapsto (streetName2, 0, house))\}$
$address? = (streetName2, 0, house)$
$dni? = dNI1$
$population = \emptyset$

ChangePersonDwelling_SP_10_TCASE

ChangePersonDwelling_SP_10

$children = \emptyset$
$dwellings = \emptyset$
$address? = (streetName1, 0, house)$
$dni? = dNI2$
$population = \emptyset$

ChangePersonDwelling_SP_12_TCASE

ChangePersonDwelling_SP_12

$children = \emptyset$
$dwellings = \{(dNI1 \mapsto (streetName1, 1, building))\}$
$address? = (streetName1, 0, house)$
$dni? = dNI2$
$population = \emptyset$

ChangePersonDwelling_SP_14_TCASE

ChangePersonDwelling_SP_14

$children = \emptyset$
$dwellings = \{(dNI1 \mapsto (streetName1, 0, house)), (dNI2 \mapsto (streetName1, 1, house))\}$
$address? = (streetName1, 0, house)$
$dni? = dNI1$
$population = \emptyset$

ChangePersonDwelling_SP_15_TCASE

ChangePersonDwelling_SP_15

$children = \emptyset$
$dwellings = \{(dNI1 \mapsto (streetName2, 0, house))\}$
$address? = (streetName2, 0, house)$
$dni? = dNI1$
$population = \emptyset$

C.2.3. AddChild

AddChild_SP_5_TCASE

AddChild_SP_5

$children = \{(dNI2 \mapsto \emptyset), (dNI1 \mapsto \emptyset)\}$
$dwellings = \emptyset$
$child? = dNI1$
$dni? = dNI2$
$population = \{(dNI2 \mapsto (name1, 1, female, (dNI1 \mapsto dNI1))),$
$(dNI1 \mapsto (name1, 1, female, (dNI1 \mapsto dNI2)))\}$

AddChild_SP_13_TCASE

AddChild_SP_13

$children = \{(dNI2 \mapsto \{dNI1\}), (dNI1 \mapsto \emptyset)\}$
$dwellings = \emptyset$
$child? = dNI1$
$dni? = dNI2$
$population = \{(dNI2 \mapsto (name1, 1, female, (dNI1 \mapsto dNI1)))\}$

AddChild_SP_20_TCASE

AddChild_SP_20

$children = \{(dNI1 \mapsto \emptyset)\}$
$dwellings = \emptyset$
$child? = dNI1$
$dni? = dNI1$
$population = \emptyset$

AddChild_SP_21_TCASE

AddChild_SP_21

$children = \{(dNI1 \mapsto \emptyset), (dNI2 \mapsto \{dNI1\})\}$
$dwellings = \emptyset$
$child? = dNI1$
$dni? = dNI1$
$population = \emptyset$

AddChild_SP_28_TCASE

AddChild_SP_28

$children = \{(dNI1 \mapsto \emptyset)\}$
$dwellings = \emptyset$
$child? = dNI1$
$dni? = dNI1$
$population = \emptyset$

AddChild_SP_29_TCASE

AddChild_SP_29

$children = \{(dNI1 \mapsto \emptyset), (dNI2 \mapsto \{dNI1\})\}$
$dwellings = \emptyset$
$child? = dNI1$
$dni? = dNI1$
$population = \emptyset$

AddChild_SP_36_TCASE

AddChild_SP_36

$children = \{(dNI1 \mapsto \emptyset)\}$
$dwellings = \emptyset$
$child? = dNI1$
$dni? = dNI1$
$population = \emptyset$

AddChild_SP_37_TCASE

AddChild_SP_37

$children = \{(dNI1 \mapsto \emptyset), (dNI2 \mapsto \{dNI1\})\}$
$dwellings = \emptyset$
$child? = dNI1$
$dni? = dNI1$
$population = \emptyset$

C.2.4. CitizenDeath

CitizenDeath_SP_3_TCASE

CitizenDeath_SP_3

$children = \emptyset$
$dwellings = \emptyset$
$dni? = dNI1$
$population = \{(dNI1 \mapsto (name1, 1, female, (dNI1 \mapsto dNI2)))\}$

CitizenDeath_SP_4_TCASE

CitizenDeath_SP_4

$children = \emptyset$
$dwellings = \emptyset$
$dni? = dNI1$
$population = \{(dNI1 \mapsto (name1, 1, female, (dNI1 \mapsto dNI1))),$
$(dNI2 \mapsto (name1, 1, female, (dNI1 \mapsto dNI2)))\}$

CitizenDeath_SP_10_TCASE

CitizenDeath_SP_10

$children = \{(dNI2 \mapsto \{dNI1, dNI2\})\}$
$dwellings = \emptyset$
$dni? = dNI1$
$population = \{(dNI1 \mapsto (name1, 1, female, (dNI1 \mapsto dNI2)))\}$

CitizenDeath_SP_11_TCASE

CitizenDeath_SP_11

$children = \{(dNI1 \mapsto \{dNI1, dNI2\})\}$
$dwellings = \emptyset$
$dni? = dNI1$
$population = \{(dNI1 \mapsto (name1, 1, female, (dNI1 \mapsto dNI1))),$
$(dNI2 \mapsto (name1, 1, female, (dNI1 \mapsto dNI2)))\}$

CitizenDeath_SP_17_TCASE

CitizenDeath_SP_17

$children = \{(dNI1 \mapsto \{dNI1, dNI2\})\}$
$dwellings = \emptyset$
$dni? = dNI1$
$population = \{(dNI1 \mapsto (name1, 1, female, (dNI1 \mapsto dNI2)))\}$

CitizenDeath_SP_18_TCASE

CitizenDeath_SP_18

$children = \{(dNI2 \mapsto \{dNI1, dNI2\})\}$
$dwellings = \emptyset$
$dni? = dNI1$
$population = \{(dNI1 \mapsto (name1, 1, female, (dNI1 \mapsto dNI1))),$
$(dNI2 \mapsto (name1, 1, female, (dNI1 \mapsto dNI2)))\}$

CitizenDeath_SP_24_TCASE

CitizenDeath_SP_24

$children = \{(dNI3 \mapsto \{dNI3\})\}$
$dwellings = \emptyset$
$dni? = dNI1$
$population = \{(dNI1 \mapsto (name1, 1, female, (dNI1 \mapsto dNI2)))\}$

CitizenDeath_SP_25_TCASE

CitizenDeath_SP_25

$children = \{(dNI3 \mapsto \{dNI3\})\}$
$dwellings = \emptyset$
$dni? = dNI1$
$population = \{(dNI1 \mapsto (name1, 1, female, (dNI1 \mapsto dNI1))),$
$(dNI2 \mapsto (name1, 1, female, (dNI1 \mapsto dNI2)))\}$

CitizenDeath_SP_29_TCASE

CitizenDeath_SP_29

$children = \emptyset$
$dwellings = \emptyset$
$dni? = dNI1$
$population = \emptyset$

CitizenDeath_SP_33_TCASE

CitizenDeath_SP_33

$children = \emptyset$
$dwellings = \emptyset$
$dni? = dNI1$
$population = \{(dNI2 \mapsto (name1, 1, female, (dNI1 \mapsto dNI1)))\}$

C.3. Implementación

```
#!/usr/bin/perl
use feature qw(say);
use YAML::XS;
use Data::Dumper;

sub __fastest_dump {
  open F, '>', 'state.yml';
  print F Dumper ( @_ );
  close F;
}

# Defino el enumerado para tipos de domicilio
use constant {
    houseI    => 'houseI',
    buildingI => 'buildingI'
};

# Defino el enumerado para los géneros
use constant {
    maleI => 'maleI',
    femaleI => 'femaleI'
};

package Citizen;

sub new {
    my ($class,$args) = @_;
    my $self = bless { }, $class;
}

sub set {
  my ($self, $name, $age, $gender, $progenitors) = @_;
  $self->{"name"} = $name;
  $self->{"age"} = $age;
  $self->{"gender"} = $gender;
  $self->{"progenitors"} = $progenitors;
}

sub get {
  my $self = shift;
  my $arg = shift;
  return $self->{$arg};
}

package Address;

sub new {
    my ($class,$args) = @_;
    my $self = bless { }, $class;
}

sub set {
  my ($self, $streetName, $number, $type) = @_;
  $self->{"streetName"} = $streetName;
  $self->{"number"} = $number;
  $self->{"type"} = $type;
}

sub get {
  my $self = shift;
  my $arg = shift;
  return $self->{$arg};
}

package DniAndAddress;

sub new {
    my ($class,$args) = @_;
    my $self = bless { }, $class;
}

sub set {
  my ($self, $dni, $address) = @_;
  $self->{"dni"} = $dni;
```

```
  $self->{"address"} = $address;
}

sub get {
  my $self = shift;
  my $arg = shift;
  return $self->{$arg};
}

package main;

# Hash donde se almacenan los mensajes a mostrar como salida
%mensajes = (
  'personAlreadyRegistered' => 'La persona ya se encuentra registrada',
  'inexistingPerson' => 'La persona no se encuentra registrada',
  'personAlreadyHaveDwelling' => 'La persona ya tiene domicilio',
  'childAlreadyRegistered' => 'El hija/o ya se encuentra registrada/o',
  'notRegisteredChild' => 'Hijo no se encuentra registrado'
);

# Arrays donde se guardará la información del registro civil
@populationImp = ();
@dwellingsImp = ();
@childrenImp = ();

# Chequea si ya fue agregado una persona
# a un determinado contenedor
# Argumento 1: contenedor donde buscar
# Argumento 2: DNI de persona buscada
sub personExists {
  @container = @{$_[0]};
  $dni = $_[1];

  $length = @container;
  for $i (0..$length-1) {
    if(ref($container[$i]) eq 'HASH') {
      if($container[$i]->{"dni"} eq $dni) {
        return 1;
      }
    } else {
      if($container[$i]->get("dni") eq $dni) {
        return 1;
      }
    }
  }
  return 0;
}

sub childInArray {
  @children = @{$_[0]};
  $child = $_[1];

  $childrenSize = @children;
  for $i (0..$childrenSize-1) {
    if($children[$i] eq $child) {
      return 1;
    }
  }
  return 0;
}

sub getPersonArrayIndex {
  @container = @{$_[0]};
  $dni = $_[1];

  $length = @container;
  for $i (0..$length-1) {
    if(ref($container[$i]) eq 'HASH') {
      if($container[$i]->{"dni"} == $dni) {
        return $i;
      }
    } else {
      if($container[$i]->get("dni") == $dni) {
        return $i;
      }
    }
```

```
151   }
152   return -1;
153 }
154
155
156 sub citizenRegistration {
157   $dni = $_[0];
158   $citizen = $_[1];
159   $address = $_[2];
160   @children = @{$_[3]};
161
162   if(personExists(\@populationImp, $dni)) {
163     say($mensajes{'personAlreadyRegistered'});
164   } else {
165     $lastIndex = @populationImp;
166
167     $citizenAux = Citizen->new();
168     $name = $citizen->get("name");
169     $age = $citizen->get("age");
170     $gender = $citizen->get("gender");
171     $progenitors = $citizen->get("progenitors");
172     $citizenAux->set($name,$age,$gender,$progenitors);
173
174     $populationImp[$lastIndex] = {"dni" => $dni, "citizen" => $citizenAux};
175
176
177     $lastIndex = @dwellingsImp;
178
179     $addressAux = Address->new();
180     $streetName = $address->get("streetName");
181     $number = $address->get("number");
182     $type = $address->get("type");
183     $addressAux->set($streetName,$number,$type);
184
185     $dwellingsImp[$lastIndex] = DniAndAddress->new();
186     $dwellingsImp[$lastIndex]->set($dni, $addressAux);
187
188     $lastIndex = @childrenImp;
189
190     $childrenImp[lastIndex] = {"dni" => $dni, "dnis" => \@children};
191   }
192 }
193
194 sub changePersonDwelling {
195   $dni = $_[0];
196   $address = $_[1];
197
198   if(personExists(\@dwellingsImp, $dni)) {
199     $dweIndex = getPersonArrayIndex(\@dwellingsImp, $dni);
200     $addressAux = $dwellingsImp[$dweIndex]->get("address");
201
202     $streetName = $address->get("streetName");
203     $number = $address->get("number");
204     $type = $address->get("type");
205     $addressAux->set($streetName,$number,$type);
206
207     $dwellingsImp[$dweIndex]->set($dni, $addressAux);
208   } else {
209     say($mensajes{'inexistingPerson'});
210   }
211 }
212
213 sub addChild {
214   $dni = $_[0];
215   $child = $_[1];
216
217   if(personExists(\@populationImp, $dni)) {
218     if(personExists(\@populationImp, $child)) {
219       $chiIndex = getPersonArrayIndex(\@childrenImp, $dni);
220       @children = @{$childrenImp[$chiIndex]};
221       if(childInArray(\@children, $child)) {
222         say($mensajes{'childAlreadyRegistered'});
223       } else {
224         $childrenSize = @children;
225         $children[childrenSize] = $child;
226       }
```

```
227     } else {
228       say($mensajes{'notRegisteredChild'});
229     }
230   } else {
231     say($mensajes{'inexistingPerson'});
232   }
233 }
234
235 sub citizenDeath {
236   $dni = $_[0];
237
238   if(personExists(\@populationImp, $dni)) {
239     $popIndex = getPersonArrayIndex(\@populationImp, $dni);
240     $dweIndex = getPersonArrayIndex(\@dwellingsImp, $dni);
241     $chiIndex = getPersonArrayIndex(\@childrenImp, $dni);
242     $progenitors = $populationImp[$popIndex]->{"progenitors"};
243     $dni1 = $progenitors->{"dni1"};
244     $dni2 = $progenitors->{"dni2"};
245     delete $populationImp[popIndex];
246     delete $dwellingsImp[dweIndex];
247     delete $childrenImp[chiIndex];
248
249     if (personExists(\@childrenImp, $dni1)) {
250       $dni1Index = getPersonArrayIndex(\@childrenImp, $dni1);
251       @dni1Children = @{$childrenImp[$dni1Index]};
252       $dni2Index = getPersonArrayIndex(\@childrenImp, $dni2);
253       @dni2Children = @{$childrenImp[$dni2Index]};
254       if(childInArray(\@dni1Children, $dni)) {
255         $dniIndex = getPersonArrayIndex(\@dni1Children, $dni);
256         delete $dni1Children[$dniIndex];
257       }
258       if(childInArray(\@dni2Children, $dni)) {
259         $dniIndex = getPersonArrayIndex(\@dni2Children, $dni);
260         delete $dni2Children[$dniIndex];
261       }
262     }
263     if (personExists(\@childrenImp, $dni2)) {
264
265     }
266   } else {
267     say($mensajes{'inexistingPerson'});
268   }
269 }
```

C.4. Leyes de refinamiento

```
1  @RRULE civilRegistryCR
2
3  @DATATYPES
4    DATATYPE PlaceType = ENUM PlaceType ( houseI, buildingI );
5    DATATYPE Gender = ENUM PlaceType ( maleI, femaleI );
6    DATATYPE Progenitors = RECORD Progenitors (dni1 : STRING, dni2: STRING);
7    DATATYPE Citizen = MODULE "Citizen" CONSTRUCTOR new () SETTER set (name :
         STRING, age: INT, gender: STRING, progenitors: Progenitors) GETTER get (
         member : STRING);
8    DATATYPE Address = MODULE "Address" CONSTRUCTOR new () SETTER set (
         streetName : STRING, number: INT, type: PlaceType) GETTER get (member :
         STRING);
9    DATATYPE DniAndCitizen = RECORD DniAndCitizen (dni : STRING, citizen:
         Citizen);
10   DATATYPE DniAndAddress = MODULE "DniAndAddress" CONSTRUCTOR new () SETTER
         set (dni : STRING, address: Address) GETTER get (member : STRING);
11   DATATYPE Dnis = ARRAY STRING (10);
12   DATATYPE DniAndChildren = RECORD DniAndChildren (dni : STRING, dnis: Dnis);
13
14 @LAWS
15   population ==> populationImp AS ARRAY DniAndCitizen (15) WITH [
16     population.@DOM ==> .dni AS STRING,
17     population.@RAN ==> .citizen AS Citizen WITH [
18       population.@RAN.#1 ==> name AS STRING,
19       population.@RAN.#2 ==> age AS INT,
20       population.@RAN.#3 ==> gender AS Gender MAP [
21         male -> maleI,
```

```
        female -> femaleI
      ],
      population.@RAN.#4 ==> progenitors AS Progenitors WITH [
        population.@RAN.#4.#1 ==> .dni1 AS STRING,
        population.@RAN.#4.#2 ==> .dni2 AS STRING
      ]
    ]
  ];

  dwellings ==> dwellingsImp AS ARRAY DniAndAddress (15) WITH [
    dwellings.@DOM ==> dni AS STRING,
    dwellings.@RAN ==> address AS Address WITH [
      dwellings.@RAN.#1 ==> streetName AS STRING,
      dwellings.@RAN.#2 ==> number AS INT,
      dwellings.@RAN.#3 ==> type AS PlaceType MAP [
        house -> houseI,
        building -> buildingI
      ]
    ]
  ];

  children ==> childrenImp AS ARRAY DniAndChildren (15) WITH [
    children.@DOM ==> .dni AS STRING,
    children.@RAN ==> .dnis AS Dnis
  ];

  dni? ==> dni AS STRING;

  citizen? ==> citizen AS Citizen WITH [
    citizen?.#1 ==> name AS STRING,
    citizen?.#2 ==> age AS INT,
    citizen?.#3 ==> gender AS Gender MAP [
      male -> maleI,
      female -> femaleI
    ],
    citizen?.#4 ==> progenitors AS Progenitors WITH [
      citizen?.#4.#1 ==> .dni1 AS STRING,
      citizen?.#4.#2 ==> .dni2 AS STRING
    ]
  ];

  address? ==> address AS Address WITH [
    address?.#1 ==> streetName AS STRING,
    address?.#2 ==> number AS INT,
    address?.#3 ==> type AS PlaceType MAP [
      house -> houseI,
      building -> buildingI
    ]
  ];

  children? ==> personsChildren AS Dnis;

@UUT
citizenRegistration(dni, citizen, address, personsChildren);
```

Las reglas de refinamiento `civilRegistryCPD`, `civilRegistryAC` y `civilRegistryCD` contienen especificaciones de refinamiento similares a la regla presentada en esta sección.

C.5. Casos concretos de prueba en Perl

C.5.1. Operación CitizenRegistration

Ejecutando los comandos

```
loadrefrule civilRegistryCR.atcal
refine CitizenRegistration_VIS to test civilRegistry.pl implemented in perl
    with civilRegistryCR
```

se generan

- CitizenRegistration_SP_2_CTCASE

```
#!/usr/bin/perl
use feature qw(say);
```

```
use YAML::XS;
use Data::Dumper;

sub __fastest_dump {
  open F, '>', 'state.yml';
  print F Dumper ( @_ );
  close F;
}

# Defino el enumerado para tipos de domicilio
use constant {
    houseI    => 'houseI',
    buildingI => 'buildingI'
};

# Defino el enumerado para los géneros
use constant {
    maleI => 'maleI',
    femaleI => 'femaleI'
};

package Citizen;

sub new {
    my ($class,$args) = @_;
    my $self = bless { }, $class;
}

sub set {
  my ($self, $name, $age, $gender, $progenitors) = @_;
  $self->{"name"} = $name;
  $self->{"age"} = $age;
  $self->{"gender"} = $gender;
  $self->{"progenitors"} = $progenitors;
}

sub get {
  my $self = shift;
  my $arg = shift;
  return $self->{$arg};
}

package Address;

sub new {
    my ($class,$args) = @_;
    my $self = bless { }, $class;
}

sub set {
  my ($self, $streetName, $number, $type) = @_;
  $self->{"streetName"} = $streetName;
  $self->{"number"} = $number;
  $self->{"type"} = $type;
}

sub get {
  my $self = shift;
  my $arg = shift;
  return $self->{$arg};
}

package DniAndAddress;

sub new {
    my ($class,$args) = @_;
    my $self = bless { }, $class;
}

sub set {
  my ($self, $dni, $address) = @_;
  $self->{"dni"} = $dni;
  $self->{"address"} = $address;
}

sub get {
```

```
  my $self = shift;
  my $arg = shift;
  return $self->{$arg};
}

package main;

# Arrays donde se guardará la información del registro civil
@populationImp = ();
@dwellingsImp = ();
@childrenImp = ();

sub citizenRegistration {
  # Código que implementa citizenRegistration
}

sub changePersonDwelling {
  # Código que implementa changePersonDwelling
}

sub addChild {
  # Código que implementa addChild
}

sub citizenDeath {
  # Código que implementa citizenDeath
}

@populationImp = ();
@dwellingsImp = ();
@childrenImp = ();
$dni = "dNI1";
$citizen_tmp = Citizen->new();
$citizen_name0 = "name2";
$citizen_age0 = 0;
$citizen_gender0 = maleI;
$progenitors_dni10 = "dNI1";
$progenitors_dni20 = "dNI1";
$citizen_progenitors0 = {"dni1" => $progenitors_dni10, "dni2" =>
    $progenitors_dni20};
$citizen_tmp->set($citizen_name0,$citizen_age0,$citizen_gender0,
    $citizen_progenitors0);
$citizen = $citizen_tmp;
$address_tmp = Address->new();
$address_streetName0 = "streetName3";
$address_number0 = 0;
$address_type0 = houseI;
$address_tmp->set($address_streetName0,$address_number0,$address_type0);
$address = $address_tmp;
@personsChildren = ();
citizenRegistration($dni,$citizen,$address,@personsChildren);
$state->{'populationImp'} =  \@populationImp;
$state->{'personsChildren'} =  \@personsChildren;
$state->{'citizen'} =  \$citizen;
$state->{'address'} =  \$address;
$state->{'dni'} = $dni;
$state->{'dwellingsImp'} =  \@dwellingsImp;
$state->{'childrenImp'} =  \@childrenImp;
__fastest_dump($state);
```

- CitizenRegistration_SP_4_CTCASE

```
...
@populationImp = ();
$populationImp_citizen_tmp0 = Citizen->new();
$citizen_name0 = "name1";
$citizen_age0 = 1;
$citizen_gender0 = femaleI;
$progenitors_dni10 = "dNI1";
$progenitors_dni20 = "dNI1";
$citizen_progenitors0 = {"dni1" => $progenitors_dni10, "dni2" =>
    $progenitors_dni20};
$populationImp_citizen_tmp0->set($citizen_name0,$citizen_age0,
    $citizen_gender0,$citizen_progenitors0);
$populationImp_citizen0 = $populationImp_citizen_tmp0;
$populationImp_dni0 = "dNI2";
```

```
$populationImp[0] = {"dni" => $populationImp_dni0, "citizen" =>
    $populationImp_citizen0};
@dwellingsImp = ();
@childrenImp = ();
$dni = "dNI1";
$citizen_tmp = Citizen->new();
$citizen_name0 = "name2";
$citizen_age0 = 0;
$citizen_gender0 = maleI;
$progenitors_dni10 = "dNI1";
$progenitors_dni20 = "dNI1";
$citizen_progenitors0 = {"dni1" => $progenitors_dni10, "dni2" =>
    $progenitors_dni20};
$citizen_tmp->set($citizen_name0,$citizen_age0,$citizen_gender0,
    $citizen_progenitors0);
$citizen = $citizen_tmp;
$address_tmp = Address->new();
$address_streetName0 = "streetName3";
$address_number0 = 0;
$address_type0 = houseI;
$address_tmp->set($address_streetName0,$address_number0,$address_type0);
$address = $address_tmp;
@personsChildren = ();
...
```

- CitizenRegistration_SP_12_CTCASE

```
...
@populationImp = ();
$populationImp_citizen_tmp0 = Citizen->new();
$citizen_name0 = "name1";
$citizen_age0 = 1;
$citizen_gender0 = femaleI;
$progenitors_dni10 = "dNI1";
$progenitors_dni20 = "dNI1";
$citizen_progenitors0 = {"dni1" => $progenitors_dni10, "dni2" =>
    $progenitors_dni20};
$populationImp_citizen_tmp0->set($citizen_name0,$citizen_age0,
    $citizen_gender0,$citizen_progenitors0);
$populationImp_citizen0 = $populationImp_citizen_tmp0;
$populationImp_dni0 = "dNI1";
$populationImp[0] = {"dni" => $populationImp_dni0, "citizen" =>
    $populationImp_citizen0};
@dwellingsImp = ();
@childrenImp = ();
$dni = "dNI1";
$citizen_tmp = Citizen->new();
$citizen_name0 = "name2";
$citizen_age0 = 0;
$citizen_gender0 = maleI;
$progenitors_dni10 = "dNI1";
$progenitors_dni20 = "dNI1";
$citizen_progenitors0 = {"dni1" => $progenitors_dni10, "dni2" =>
    $progenitors_dni20};
$citizen_tmp->set($citizen_name0,$citizen_age0,$citizen_gender0,
    $citizen_progenitors0);
$citizen = $citizen_tmp;
$address_tmp = Address->new();
$address_streetName0 = "streetName3";
$address_number0 = 0;
$address_type0 = houseI;
$address_tmp->set($address_streetName0,$address_number0,$address_type0);
$address = $address_tmp;
@personsChildren = ();
...
```

- CitizenRegistration_SP_14_CTCASE

```
...
@populationImp = ();
$populationImp_citizen_tmp0 = Citizen->new();
$citizen_name0 = "name1";
$citizen_age0 = 0;
$citizen_gender0 = maleI;
$progenitors_dni10 = "dNI1";
```

```
$progenitors_dni20 = "dNI1";
$citizen_progenitors0 = {"dni1" => $progenitors_dni10, "dni2" =>
    $progenitors_dni20};
$populationImp_citizen_tmp0->set($citizen_name0,$citizen_age0,
    $citizen_gender0,$citizen_progenitors0);
$populationImp_citizen0 = $populationImp_citizen_tmp0;
$populationImp_citizen_tmp1 = Citizen->new();
$citizen_name1 = "name1";
$citizen_age1 = 1;
$citizen_gender1 = femaleI;
$progenitors_dni11 = "dNI1";
$progenitors_dni21 = "dNI2";
$citizen_progenitors1 = {"dni1" => $progenitors_dni11, "dni2" =>
    $progenitors_dni21};
$populationImp_citizen_tmp1->set($citizen_name1,$citizen_age1,
    $citizen_gender1,$citizen_progenitors1);
$populationImp_citizen1 = $populationImp_citizen_tmp1;
$populationImp_dni0 = "dNI1";
$populationImp[0] = {"dni" => $populationImp_dni0, "citizen" =>
    $populationImp_citizen0};
$populationImp_dni1 = "dNI2";
$populationImp[1] = {"dni" => $populationImp_dni1, "citizen" =>
    $populationImp_citizen1};
@dwellingsImp = ();
@childrenImp = ();
$dni = "dNI1";
$citizen_tmp = Citizen->new();
$citizen_name0 = "name1";
$citizen_age0 = 0;
$citizen_gender0 = maleI;
$progenitors_dni10 = "dNI1";
$progenitors_dni20 = "dNI1";
$citizen_progenitors0 = {"dni1" => $progenitors_dni10, "dni2" =>
    $progenitors_dni20};
$citizen_tmp->set($citizen_name0,$citizen_age0,$citizen_gender0,
    $citizen_progenitors0);
$citizen = $citizen_tmp;
$address_tmp = Address->new();
$address_streetName0 = "streetName3";
$address_number0 = 0;
$address_type0 = houseI;
$address_tmp->set($address_streetName0,$address_number0,$address_type0);
$address = $address_tmp;
@personsChildren = ();
...
```

- CitizenRegistration_SP_15_CTCASE

```
...
@populationImp = ();
$populationImp_citizen_tmp0 = Citizen->new();
$citizen_name0 = "name2";
$citizen_age0 = 0;
$citizen_gender0 = maleI;
$progenitors_dni10 = "dNI1";
$progenitors_dni20 = "dNI1";
$citizen_progenitors0 = {"dni1" => $progenitors_dni10, "dni2" =>
    $progenitors_dni20};
$populationImp_citizen_tmp0->set($citizen_name0,$citizen_age0,
    $citizen_gender0,$citizen_progenitors0);
$populationImp_citizen0 = $populationImp_citizen_tmp0;
$populationImp_dni0 = "dNI1";
$populationImp[0] = {"dni" => $populationImp_dni0, "citizen" =>
    $populationImp_citizen0};
@dwellingsImp = ();
@childrenImp = ();
$dni = "dNI1";
$citizen_tmp = Citizen->new();
$citizen_name0 = "name2";
$citizen_age0 = 0;
$citizen_gender0 = maleI;
$progenitors_dni10 = "dNI1";
$progenitors_dni20 = "dNI1";
$citizen_progenitors0 = {"dni1" => $progenitors_dni10, "dni2" =>
    $progenitors_dni20};
```

```
$citizen_tmp->set($citizen_name0,$citizen_age0,$citizen_gender0,
    $citizen_progenitors0);
$citizen = $citizen_tmp;
$address_tmp = Address->new();
$address_streetName0 = "streetName3";
$address_number0 = 0;
$address_type0 = houseI;
$address_tmp->set($address_streetName0,$address_number0,$address_type0);
$address = $address_tmp;
@personsChildren = ();
...
```

C.5.2. Operación ChangePersonDwelling

Ejecutando los comandos

```
loadrefrule civilRegistryCPD.atcal
refine ChangePersonDwelling_VIS to test civilRegistry.pl implemented in perl
    with civilRegistryCPD
```

se generan

- ChangePersonDwelling_SP_4_CTCASE

```
...
@populationImp = ();
@dwellingsImp = ();
$dwellingsImp_address_tmp0 = Address->new();
$address_streetName0 = "streetName1";
$address_number0 = 1;
$address_type0 = buildingI;
$dwellingsImp_address_tmp0->set($address_streetName0,$address_number0,
    $address_type0);
$dwellingsImp_address0 = $dwellingsImp_address_tmp0;
$dwellingsImp_tmp0 = DniAndAddress->new();
$dwellingsImp_dni0 = "dNI1";
$dwellingsImp_tmp0->set($dwellingsImp_dni0,$dwellingsImp_address0);
$dwellingsImp[0] = $dwellingsImp_tmp0;
@childrenImp = ();
$dni = "dNI1";
$address_tmp = Address->new();
$address_streetName0 = "streetName2";
$address_number0 = 0;
$address_type0 = houseI;
$address_tmp->set($address_streetName0,$address_number0,$address_type0);
$address = $address_tmp;
changePersonDwelling($dni,$address);
$state->{'populationImp'} =  \@populationImp;
$state->{'address'} =  \$address;
$state->{'dni'} = $dni;
$state->{'dwellingsImp'} =  \@dwellingsImp;
$state->{'childrenImp'} =  \@childrenImp;
__fastest_dump($state);
```

- ChangePersonDwelling_SP_6_CTCASE

```
...
@populationImp = ();
@dwellingsImp = ();
$dwellingsImp_address_tmp0 = Address->new();
$address_streetName0 = "streetName1";
$address_number0 = 0;
$address_type0 = houseI;
$dwellingsImp_address_tmp0->set($address_streetName0,$address_number0,
    $address_type0);
$dwellingsImp_address0 = $dwellingsImp_address_tmp0;
$dwellingsImp_address_tmp1 = Address->new();
$address_streetName1 = "streetName1";
$address_number1 = 1;
$address_type1 = houseI;
$dwellingsImp_address_tmp1->set($address_streetName1,$address_number1,
    $address_type1);
$dwellingsImp_address1 = $dwellingsImp_address_tmp1;
$dwellingsImp_tmp0 = DniAndAddress->new();
```

```
$dwellingsImp_dni0 = "dNI1";
$dwellingsImp_tmp0->set($dwellingsImp_dni0,$dwellingsImp_address0);
$dwellingsImp[0] = $dwellingsImp_tmp0;
$dwellingsImp_tmp1 = DniAndAddress->new();
$dwellingsImp_dni1 = "dNI2";
$dwellingsImp_tmp1->set($dwellingsImp_dni1,$dwellingsImp_address1);
$dwellingsImp[1] = $dwellingsImp_tmp1;
@childrenImp = ();
$dni = "dNI1";
$address_tmp = Address->new();
$address_streetName0 = "streetName1";
$address_number0 = 0;
$address_type0 = houseI;
$address_tmp->set($address_streetName0,$address_number0,$address_type0);
$address = $address_tmp;
...
```

- ChangePersonDwelling_SP_7_CTCASE

```
...
@populationImp = ();
@dwellingsImp = ();
$dwellingsImp_address_tmp0 = Address->new();
$address_streetName0 = "streetName2";
$address_number0 = 0;
$address_type0 = houseI;
$dwellingsImp_address_tmp0->set($address_streetName0,$address_number0,
    $address_type0);
$dwellingsImp_address0 = $dwellingsImp_address_tmp0;
$dwellingsImp_tmp0 = DniAndAddress->new();
$dwellingsImp_dni0 = "dNI1";
$dwellingsImp_tmp0->set($dwellingsImp_dni0,$dwellingsImp_address0);
$dwellingsImp[0] = $dwellingsImp_tmp0;
@childrenImp = ();
$dni = "dNI1";
$address_tmp = Address->new();
$address_streetName0 = "streetName2";
$address_number0 = 0;
$address_type0 = houseI;
$address_tmp->set($address_streetName0,$address_number0,$address_type0);
$address = $address_tmp;
...
```

- ChangePersonDwelling_SP_10_CTCASE

```
...
@populationImp = ();
@dwellingsImp = ();
@childrenImp = ();
$dni = "dNI2";
$address_tmp = Address->new();
$address_streetName0 = "streetName1";
$address_number0 = 0;
$address_type0 = houseI;
$address_tmp->set($address_streetName0,$address_number0,$address_type0);
$address = $address_tmp;
...
```

- ChangePersonDwelling_SP_12_CTCASE

```
...
@populationImp = ();
@dwellingsImp = ();
$dwellingsImp_address_tmp0 = Address->new();
$address_streetName0 = "streetName1";
$address_number0 = 1;
$address_type0 = buildingI;
$dwellingsImp_address_tmp0->set($address_streetName0,$address_number0,
    $address_type0);
$dwellingsImp_address0 = $dwellingsImp_address_tmp0;
$dwellingsImp_tmp0 = DniAndAddress->new();
$dwellingsImp_dni0 = "dNI1";
$dwellingsImp_tmp0->set($dwellingsImp_dni0,$dwellingsImp_address0);
$dwellingsImp[0] = $dwellingsImp_tmp0;
```

```
@childrenImp = ();
$dni = "dNI2";
$address_tmp = Address->new();
$address_streetName0 = "streetName1";
$address_number0 = 0;
$address_type0 = houseI;
$address_tmp->set($address_streetName0,$address_number0,$address_type0);
$address = $address_tmp;
...
```

- ChangePersonDwelling_SP_14_CTCASE

```
...
@populationImp = ();
@dwellingsImp = ();
$dwellingsImp_address_tmp0 = Address->new();
$address_streetName0 = "streetName1";
$address_number0 = 0;
$address_type0 = houseI;
$dwellingsImp_address_tmp0->set($address_streetName0,$address_number0,
    $address_type0);
$dwellingsImp_address0 = $dwellingsImp_address_tmp0;
$dwellingsImp_address_tmp1 = Address->new();
$address_streetName1 = "streetName1";
$address_number1 = 1;
$address_type1 = houseI;
$dwellingsImp_address_tmp1->set($address_streetName1,$address_number1,
    $address_type1);
$dwellingsImp_address1 = $dwellingsImp_address_tmp1;
$dwellingsImp_tmp0 = DniAndAddress->new();
$dwellingsImp_dni0 = "dNI1";
$dwellingsImp_tmp0->set($dwellingsImp_dni0,$dwellingsImp_address0);
$dwellingsImp[0] = $dwellingsImp_tmp0;
$dwellingsImp_tmp1 = DniAndAddress->new();
$dwellingsImp_dni1 = "dNI2";
$dwellingsImp_tmp1->set($dwellingsImp_dni1,$dwellingsImp_address1);
$dwellingsImp[1] = $dwellingsImp_tmp1;
@childrenImp = ();
$dni = "dNI1";
$address_tmp = Address->new();
$address_streetName0 = "streetName1";
$address_number0 = 0;
$address_type0 = houseI;
$address_tmp->set($address_streetName0,$address_number0,$address_type0);
$address = $address_tmp;
...
```

- ChangePersonDwelling_SP_15_CTCASE

```
...
@populationImp = ();
@dwellingsImp = ();
$dwellingsImp_address_tmp0 = Address->new();
$address_streetName0 = "streetName2";
$address_number0 = 0;
$address_type0 = houseI;
$dwellingsImp_address_tmp0->set($address_streetName0,$address_number0,
    $address_type0);
$dwellingsImp_address0 = $dwellingsImp_address_tmp0;
$dwellingsImp_tmp0 = DniAndAddress->new();
$dwellingsImp_dni0 = "dNI1";
$dwellingsImp_tmp0->set($dwellingsImp_dni0,$dwellingsImp_address0);
$dwellingsImp[0] = $dwellingsImp_tmp0;
@childrenImp = ();
$dni = "dNI1";
$address_tmp = Address->new();
$address_streetName0 = "streetName2";
$address_number0 = 0;
$address_type0 = houseI;
$address_tmp->set($address_streetName0,$address_number0,$address_type0);
$address = $address_tmp;
...
```

C.5.3. Operación AddChild

Ejecutando los comandos

```
loadrefrule civilRegistryAC.atcal
refine AddChild_VIS to test civilRegistry.pl implemented in perl with
    civilRegistryAC
```

se generan

- AddChild_SP_5_CTCASE

```
...
@populationImp = ();
$populationImp_citizen_tmp0 = Citizen->new();
$citizen_name0 = "name1";
$citizen_age0 = 1;
$citizen_gender0 = femaleI;
$progenitors_dni10 = "dNI1";
$progenitors_dni20 = "dNI1";
$citizen_progenitors0 = {"dni1" => $progenitors_dni10, "dni2" =>
    $progenitors_dni20};
$populationImp_citizen_tmp0->set($citizen_name0,$citizen_age0,
    $citizen_gender0,$citizen_progenitors0);
$populationImp_citizen0 = $populationImp_citizen_tmp0;
$populationImp_citizen_tmp1 = Citizen->new();
$citizen_name1 = "name1";
$citizen_age1 = 1;
$citizen_gender1 = femaleI;
$progenitors_dni11 = "dNI1";
$progenitors_dni21 = "dNI2";
$citizen_progenitors1 = {"dni1" => $progenitors_dni11, "dni2" =>
    $progenitors_dni21};
$populationImp_citizen_tmp1->set($citizen_name1,$citizen_age1,
    $citizen_gender1,$citizen_progenitors1);
$populationImp_citizen1 = $populationImp_citizen_tmp1;
$populationImp_dni0 = "dNI2";
$populationImp[0] = {"dni" => $populationImp_dni0, "citizen" =>
    $populationImp_citizen0};
$populationImp_dni1 = "dNI1";
$populationImp[1] = {"dni" => $populationImp_dni1, "citizen" =>
    $populationImp_citizen1};
@dwellingsImp = ();
@childrenImp = ();
@childrenImp_dnis0 = ();
@childrenImp_dnis1 = ();
$childrenImp_dni0 = "dNI1";
$childrenImp[0] = {"dni" => $childrenImp_dni0, "dnis" => \
    @childrenImp_dnis0};
$childrenImp_dni1 = "dNI2";
$childrenImp[1] = {"dni" => $childrenImp_dni1, "dnis" => \
    @childrenImp_dnis1};
$dni = "dNI2";
$child = "dNI1";
addChild($dni,$child);
$state->{'populationImp'} =  \@populationImp;
$state->{'dni'} = $dni;
$state->{'dwellingsImp'} =  \@dwellingsImp;
$state->{'child'} = $child;
$state->{'childrenImp'} =  \@childrenImp;
__fastest_dump($state);
```

- AddChild_SP_13_CTCASE

```
...
@populationImp = ();
$populationImp_citizen_tmp0 = Citizen->new();
$citizen_name0 = "name1";
$citizen_age0 = 1;
$citizen_gender0 = femaleI;
$progenitors_dni10 = "dNI1";
$progenitors_dni20 = "dNI1";
$citizen_progenitors0 = {"dni1" => $progenitors_dni10, "dni2" =>
    $progenitors_dni20};
$populationImp_citizen_tmp0->set($citizen_name0,$citizen_age0,
    $citizen_gender0,$citizen_progenitors0);
```

```
$populationImp_citizen0 = $populationImp_citizen_tmp0;
$populationImp_dni0 = "dNI2";
$populationImp[0] = {"dni" => $populationImp_dni0, "citizen" =>
    $populationImp_citizen0};
@dwellingsImp = ();
@childrenImp = ();
@childrenImp_dnis0 = ();
@childrenImp_dnis1 = ();
$childrenImp_dnis1[0] = "dNI1";
$childrenImp_dni0 = "dNI1";
$childrenImp[0] = {"dni" => $childrenImp_dni0, "dnis" => \
    @childrenImp_dnis0};
$childrenImp_dni1 = "dNI2";
$childrenImp[1] = {"dni" => $childrenImp_dni1, "dnis" => \
    @childrenImp_dnis1};
$dni = "dNI2";
$child = "dNI1";
...
```

- AddChild_SP_20_CTCASE

```
...
@populationImp = ();
@dwellingsImp = ();
@childrenImp = ();
@childrenImp_dnis0 = ();
$childrenImp_dni0 = "dNI1";
$childrenImp[0] = {"dni" => $childrenImp_dni0, "dnis" => \
    @childrenImp_dnis0};
$dni = "dNI1";
$child = "dNI1";
...
```

- AddChild_SP_21_CTCASE

```
...
@populationImp = ();
@dwellingsImp = ();
@childrenImp = ();
@childrenImp_dnis0 = ();
@childrenImp_dnis1 = ();
$childrenImp_dnis1[0] = "dNI1";
$childrenImp_dni0 = "dNI1";
$childrenImp[0] = {"dni" => $childrenImp_dni0, "dnis" => \
    @childrenImp_dnis0};
$childrenImp_dni1 = "dNI2";
$childrenImp[1] = {"dni" => $childrenImp_dni1, "dnis" => \
    @childrenImp_dnis1};
$dni = "dNI1";
$child = "dNI1";
...
```

- AddChild_SP_28_CTCASE

```
...
@populationImp = ();
@dwellingsImp = ();
@childrenImp = ();
@childrenImp_dnis0 = ();
$childrenImp_dni0 = "dNI1";
$childrenImp[0] = {"dni" => $childrenImp_dni0, "dnis" => \
    @childrenImp_dnis0};
$dni = "dNI1";
$child = "dNI1";
...
```

- AddChild_SP_29_CTCASE

```
...
@populationImp = ();
@dwellingsImp = ();
@childrenImp = ();
@childrenImp_dnis0 = ();
@childrenImp_dnis1 = ();
```

```
$childrenImp_dnis1[0] = "dNI1";
$childrenImp_dni0 = "dNI1";
$childrenImp[0] = {"dni" => $childrenImp_dni0, "dnis" => \
    @childrenImp_dnis0};
$childrenImp_dni1 = "dNI2";
$childrenImp[1] = {"dni" => $childrenImp_dni1, "dnis" => \
    @childrenImp_dnis1};
$dni = "dNI1";
$child = "dNI1";
...
```

- AddChild_SP_36_CTCASE

```
...
@populationImp = ();
@dwellingsImp = ();
@childrenImp = ();
@childrenImp_dnis0 = ();
$childrenImp_dni0 = "dNI1";
$childrenImp[0] = {"dni" => $childrenImp_dni0, "dnis" => \
    @childrenImp_dnis0};
$dni = "dNI1";
$child = "dNI1";
...
```

- AddChild_SP_37_CTCASE

```
...
@populationImp = ();
@dwellingsImp = ();
@childrenImp = ();
@childrenImp_dnis0 = ();
@childrenImp_dnis1 = ();
$childrenImp_dnis1[0] = "dNI1";
$childrenImp_dni0 = "dNI1";
$childrenImp[0] = {"dni" => $childrenImp_dni0, "dnis" => \
    @childrenImp_dnis0};
$childrenImp_dni1 = "dNI2";
$childrenImp[1] = {"dni" => $childrenImp_dni1, "dnis" => \
    @childrenImp_dnis1};
$dni = "dNI1";
$child = "dNI1";
...
```

C.5.4. Operación CitizenDeath

Ejecutando los comandos

```
loadrefrule civilRegistryCD.atcal
refine CitizenDeath_VIS to test civilRegistry.pl implemented in perl with
    civilRegistryCD
```

se generan

- CitizenDeath_SP_3_CTCASE

```
...
@populationImp = ();
$populationImp_citizen_tmp0 = Citizen->new();
$citizen_name0 = "name1";
$citizen_age0 = 1;
$citizen_gender0 = femaleI;
$progenitors_dni10 = "dNI1";
$progenitors_dni20 = "dNI2";
$citizen_progenitors0 = {"dni1" => $progenitors_dni10, "dni2" =>
    $progenitors_dni20};
$populationImp_citizen_tmp0->set($citizen_name0,$citizen_age0,
    $citizen_gender0,$citizen_progenitors0);
$populationImp_citizen0 = $populationImp_citizen_tmp0;
$populationImp_dni0 = "dNI1";
$populationImp[0] = {"dni" => $populationImp_dni0, "citizen" =>
    $populationImp_citizen0};
@dwellingsImp = ();
```

```
@childrenImp = ();
$dni = "dNI1";
citizenDeath($dni);
$state->{'populationImp'} =  \@populationImp;
$state->{'dni'} = $dni;
$state->{'dwellingsImp'} =  \@dwellingsImp;
$state->{'childrenImp'} =  \@childrenImp;
__fastest_dump($state);
```

- CitizenDeath_SP_4_CTCASE

```
...
@populationImp = ();
$populationImp_citizen_tmp0 = Citizen->new();
$citizen_name0 = "name1";
$citizen_age0 = 1;
$citizen_gender0 = femaleI;
$progenitors_dni10 = "dNI1";
$progenitors_dni20 = "dNI1";
$citizen_progenitors0 = {"dni1" => $progenitors_dni10, "dni2" =>
    $progenitors_dni20};
$populationImp_citizen_tmp0->set($citizen_name0,$citizen_age0,
    $citizen_gender0,$citizen_progenitors0);
$populationImp_citizen0 = $populationImp_citizen_tmp0;
$populationImp_citizen_tmp1 = Citizen->new();
$citizen_name1 = "name1";
$citizen_age1 = 1;
$citizen_gender1 = femaleI;
$progenitors_dni11 = "dNI1";
$progenitors_dni21 = "dNI2";
$citizen_progenitors1 = {"dni1" => $progenitors_dni11, "dni2" =>
    $progenitors_dni21};
$populationImp_citizen_tmp1->set($citizen_name1,$citizen_age1,
    $citizen_gender1,$citizen_progenitors1);
$populationImp_citizen1 = $populationImp_citizen_tmp1;
$populationImp_dni0 = "dNI1";
$populationImp[0] = {"dni" => $populationImp_dni0, "citizen" =>
    $populationImp_citizen0};
$populationImp_dni1 = "dNI2";
$populationImp[1] = {"dni" => $populationImp_dni1, "citizen" =>
    $populationImp_citizen1};
@dwellingsImp = ();
@childrenImp = ();
$dni = "dNI1";
...
```

- CitizenDeath_SP_10_CTCASE

```
...
@populationImp = ();
$populationImp_citizen_tmp0 = Citizen->new();
$citizen_name0 = "name1";
$citizen_age0 = 1;
$citizen_gender0 = femaleI;
$progenitors_dni10 = "dNI1";
$progenitors_dni20 = "dNI2";
$citizen_progenitors0 = {"dni1" => $progenitors_dni10, "dni2" =>
    $progenitors_dni20};
$populationImp_citizen_tmp0->set($citizen_name0,$citizen_age0,
    $citizen_gender0,$citizen_progenitors0);
$populationImp_citizen0 = $populationImp_citizen_tmp0;
$populationImp_dni0 = "dNI1";
$populationImp[0] = {"dni" => $populationImp_dni0, "citizen" =>
    $populationImp_citizen0};
@dwellingsImp = ();
@childrenImp = ();
@childrenImp_dnis0 = ();
$childrenImp_dnis0[0] = "dNI1";
$childrenImp_dnis0[1] = "dNI2";
$childrenImp_dni0 = "dNI2";
$childrenImp[0] = {"dni" => $childrenImp_dni0, "dnis" => \
    @childrenImp_dnis0};
$dni = "dNI1";
...
```

- CitizenDeath_SP_11_CTCASE

```
...
@populationImp = ();
$populationImp_citizen_tmp0 = Citizen->new();
$citizen_name0 = "name1";
$citizen_age0 = 1;
$citizen_gender0 = femaleI;
$progenitors_dni10 = "dNI1";
$progenitors_dni20 = "dNI1";
$citizen_progenitors0 = {"dni1" => $progenitors_dni10, "dni2" =>
    $progenitors_dni20};
$populationImp_citizen_tmp0->set($citizen_name0,$citizen_age0,
    $citizen_gender0,$citizen_progenitors0);
$populationImp_citizen0 = $populationImp_citizen_tmp0;
$populationImp_citizen_tmp1 = Citizen->new();
$citizen_name1 = "name1";
$citizen_age1 = 1;
$citizen_gender1 = femaleI;
$progenitors_dni11 = "dNI1";
$progenitors_dni21 = "dNI2";
$citizen_progenitors1 = {"dni1" => $progenitors_dni11, "dni2" =>
    $progenitors_dni21};
$populationImp_citizen_tmp1->set($citizen_name1,$citizen_age1,
    $citizen_gender1,$citizen_progenitors1);
$populationImp_citizen1 = $populationImp_citizen_tmp1;
$populationImp_dni0 = "dNI1";
$populationImp[0] = {"dni" => $populationImp_dni0, "citizen" =>
    $populationImp_citizen0};
$populationImp_dni1 = "dNI2";
$populationImp[1] = {"dni" => $populationImp_dni1, "citizen" =>
    $populationImp_citizen1};
@dwellingsImp = ();
@childrenImp = ();
@childrenImp_dnis0 = ();
$childrenImp_dnis0[0] = "dNI1";
$childrenImp_dnis0[1] = "dNI2";
$childrenImp_dni0 = "dNI1";
$childrenImp[0] = {"dni" => $childrenImp_dni0, "dnis" => \
    @childrenImp_dnis0};
$dni = "dNI1";
...
```

- CitizenDeath_SP_17_CTCASE

```
...
@populationImp = ();
$populationImp_citizen_tmp0 = Citizen->new();
$citizen_name0 = "name1";
$citizen_age0 = 1;
$citizen_gender0 = femaleI;
$progenitors_dni10 = "dNI1";
$progenitors_dni20 = "dNI2";
$citizen_progenitors0 = {"dni1" => $progenitors_dni10, "dni2" =>
    $progenitors_dni20};
$populationImp_citizen_tmp0->set($citizen_name0,$citizen_age0,
    $citizen_gender0,$citizen_progenitors0);
$populationImp_citizen0 = $populationImp_citizen_tmp0;
$populationImp_dni0 = "dNI1";
$populationImp[0] = {"dni" => $populationImp_dni0, "citizen" =>
    $populationImp_citizen0};
@dwellingsImp = ();
@childrenImp = ();
@childrenImp_dnis0 = ();
$childrenImp_dnis0[0] = "dNI1";
$childrenImp_dnis0[1] = "dNI2";
$childrenImp_dni0 = "dNI1";
$childrenImp[0] = {"dni" => $childrenImp_dni0, "dnis" => \
    @childrenImp_dnis0};
$dni = "dNI1";
...
```

- CitizenDeath_SP_18_CTCASE

```
...
```

```
@populationImp = ();
$populationImp_citizen_tmp0 = Citizen->new();
$citizen_name0 = "name1";
$citizen_age0 = 1;
$citizen_gender0 = femaleI;
$progenitors_dni10 = "dNI1";
$progenitors_dni20 = "dNI1";
$citizen_progenitors0 = {"dni1" => $progenitors_dni10, "dni2" =>
    $progenitors_dni20};
$populationImp_citizen_tmp0->set($citizen_name0,$citizen_age0,
    $citizen_gender0,$citizen_progenitors0);
$populationImp_citizen0 = $populationImp_citizen_tmp0;
$populationImp_citizen_tmp1 = Citizen->new();
$citizen_name1 = "name1";
$citizen_age1 = 1;
$citizen_gender1 = femaleI;
$progenitors_dni11 = "dNI1";
$progenitors_dni21 = "dNI2";
$citizen_progenitors1 = {"dni1" => $progenitors_dni11, "dni2" =>
    $progenitors_dni21};
$populationImp_citizen_tmp1->set($citizen_name1,$citizen_age1,
    $citizen_gender1,$citizen_progenitors1);
$populationImp_citizen1 = $populationImp_citizen_tmp1;
$populationImp_dni0 = "dNI1";
$populationImp[0] = {"dni" => $populationImp_dni0, "citizen" =>
    $populationImp_citizen0};
$populationImp_dni1 = "dNI2";
$populationImp[1] = {"dni" => $populationImp_dni1, "citizen" =>
    $populationImp_citizen1};
@dwellingsImp = ();
@childrenImp = ();
@childrenImp_dnis0 = ();
$childrenImp_dnis0[0] = "dNI1";
$childrenImp_dnis0[1] = "dNI2";
$childrenImp_dni0 = "dNI2";
$childrenImp[0] = {"dni" => $childrenImp_dni0, "dnis" => \
    @childrenImp_dnis0};
$dni = "dNI1";
...
```

- CitizenDeath_SP_24_CTCASE

```
...
@populationImp = ();
$populationImp_citizen_tmp0 = Citizen->new();
$citizen_name0 = "name1";
$citizen_age0 = 1;
$citizen_gender0 = femaleI;
$progenitors_dni10 = "dNI1";
$progenitors_dni20 = "dNI2";
$citizen_progenitors0 = {"dni1" => $progenitors_dni10, "dni2" =>
    $progenitors_dni20};
$populationImp_citizen_tmp0->set($citizen_name0,$citizen_age0,
    $citizen_gender0,$citizen_progenitors0);
$populationImp_citizen0 = $populationImp_citizen_tmp0;
$populationImp_dni0 = "dNI1";
$populationImp[0] = {"dni" => $populationImp_dni0, "citizen" =>
    $populationImp_citizen0};
@dwellingsImp = ();
@childrenImp = ();
@childrenImp_dnis0 = ();
$childrenImp_dnis0[0] = "dNI3";
$childrenImp_dni0 = "dNI3";
$childrenImp[0] = {"dni" => $childrenImp_dni0, "dnis" => \
    @childrenImp_dnis0};
$dni = "dNI1";
...
```

- CitizenDeath_SP_25_CTCASE

```
...
@populationImp = ();
$populationImp_citizen_tmp0 = Citizen->new();
$citizen_name0 = "name1";
$citizen_age0 = 1;
```

```
$citizen_gender0 = femaleI;
$progenitors_dni10 = "dNI1";
$progenitors_dni20 = "dNI1";
$citizen_progenitors0 = {"dni1" => $progenitors_dni10, "dni2" =>
    $progenitors_dni20};
$populationImp_citizen_tmp0->set($citizen_name0,$citizen_age0,
    $citizen_gender0,$citizen_progenitors0);
$populationImp_citizen0 = $populationImp_citizen_tmp0;
$populationImp_citizen_tmp1 = Citizen->new();
$citizen_name1 = "name1";
$citizen_age1 = 1;
$citizen_gender1 = femaleI;
$progenitors_dni11 = "dNI1";
$progenitors_dni21 = "dNI2";
$citizen_progenitors1 = {"dni1" => $progenitors_dni11, "dni2" =>
    $progenitors_dni21};
$populationImp_citizen_tmp1->set($citizen_name1,$citizen_age1,
    $citizen_gender1,$citizen_progenitors1);
$populationImp_citizen1 = $populationImp_citizen_tmp1;
$populationImp_dni0 = "dNI1";
$populationImp[0] = {"dni" => $populationImp_dni0, "citizen" =>
    $populationImp_citizen0};
$populationImp_dni1 = "dNI2";
$populationImp[1] = {"dni" => $populationImp_dni1, "citizen" =>
    $populationImp_citizen1};
@dwellingsImp = ();
@childrenImp = ();
@childrenImp_dnis0 = ();
$childrenImp_dnis0[0] = "dNI3";
$childrenImp_dni0 = "dNI3";
$childrenImp[0] = {"dni" => $childrenImp_dni0, "dnis" => \
    @childrenImp_dnis0};
$dni = "dNI1";
...
```

- CitizenDeath_SP_29_CTCASE

```
...
@populationImp = ();
@dwellingsImp = ();
@childrenImp = ();
$dni = "dNI1";
...
```

- CitizenDeath_SP_33_CTCASE

```
...
@populationImp = ();
$populationImp_citizen_tmp0 = Citizen->new();
$citizen_name0 = "name1";
$citizen_age0 = 1;
$citizen_gender0 = femaleI;
$progenitors_dni10 = "dNI1";
$progenitors_dni20 = "dNI1";
$citizen_progenitors0 = {"dni1" => $progenitors_dni10, "dni2" =>
    $progenitors_dni20};
$populationImp_citizen_tmp0->set($citizen_name0,$citizen_age0,
    $citizen_gender0,$citizen_progenitors0);
$populationImp_citizen0 = $populationImp_citizen_tmp0;
$populationImp_dni0 = "dNI2";
$populationImp[0] = {"dni" => $populationImp_dni0, "citizen" =>
    $populationImp_citizen0};
@dwellingsImp = ();
@childrenImp = ();
$dni = "dNI1";
...
```

Apéndice D

Almacen de elementos

Los archivos de este caso de estudio pueden encontrarse en este link.

D.1. Especificación Z

Se presenta un sistema sencillo que permite almacenar elementos y establecer relación entre ellos. El esquema de estado contiene el conjunto `elements` para almacenar los elementos y la relación `relations` donde se vinculan los elementos entre sí.

$$
\begin{array}{|l}
\hline
\textit{Storage} \\
\textit{elements} : \mathbb{P}\,\mathbb{N} \\
\textit{relations} : \mathbb{N} \leftrightarrow \mathbb{N} \\
\hline
\end{array}
$$

- AddInitialElement: permite almacenar el primer elemento

$$
\begin{array}{|l}
\hline
\textit{AddInitialElementOk} \\
\Delta \textit{Storage} \\
\textit{element}? : \mathbb{N} \\
\hline
\#\textit{elements} = 0 \\
\textit{elements}' = \textit{elements} \cup \{\textit{element}?\} \\
\textit{relations}' = \textit{relations} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{AlreadyExistElements} \\
\Xi \textit{Storage} \\
\hline
\#\textit{elements} \neq 0 \\
\hline
\end{array}
$$

$$\textit{AddInitialElement} == \textit{AddInitialElementOk} \lor \textit{AlreadyExistElements}$$

- AddRelationToElement: vincula un elemento con otro

$$
\begin{array}{|l}
\hline
\textit{AddRelationToElementOk} \\
\Delta \textit{Storage} \\
\textit{dadElement}? : \mathbb{N} \\
\textit{sonElement}? : \mathbb{N} \\
\hline
\textit{dadElement}? \in \textit{elements} \\
\#(\{\textit{dadElement}?\} \lhd \textit{relations}) < 2 \\
\textit{sonElement}? \notin \operatorname{dom} \textit{relations} \\
\textit{sonElement}? \notin \operatorname{ran} \textit{relations} \\
\textit{relations}' = \textit{relations} \cup \{\textit{dadElement}? \mapsto \textit{sonElement}?\} \\
\textit{elements}' = \textit{elements} \cup \{\textit{sonElement}?\} \\
\hline
\end{array}
$$

SonElementAlreadyRelatedAsDad

$\Xi Storage$
$sonElement? : \mathbb{N}$

$sonElement? \in \text{dom } relations$

SonElementAlreadyRelatedAsChild

$\Xi Storage$
$sonElement? : \mathbb{N}$

$sonElement? \in \text{ran } relations$

DadElementDoesntExist

$\Xi Storage$
$dadElement? : \mathbb{N}$

$dadElement? \notin elements$

DadElementAlreadyHaveTwoRelatedElements

$\Xi Storage$
$dadElement? : \mathbb{N}$

$dadElement? \in elements$
$\#(\{dadElement?\} \lhd relations) = 2$

$AddRelationToElement == AddRelationToElementOk \lor SonElementAlreadyRelatedAsDad \lor SonElementAlreadyRelatedAsChild \lor DadElementDoesntExist \lor DadElementAlreadyHaveTwoRelatedElements$

- RemoveElement: elimina un elemento del almacen

RemoveElementOk

$\Delta Storage$
$element? : \mathbb{N}$

$element? \in \text{dom } relations \lor element? \in \text{ran } relations$
$relations' = (\{element?\} \lhd relations) \rhd \{element?\}$
$elements' = elements \setminus \{element?\}$

ElementNotIncludedInRelations

$\Xi Storage$
$element? : \mathbb{N}$

$element? \notin \text{dom } relations$
$element? \notin \text{ran } relations$

$RemoveElement == RemoveElementOk \lor ElementNotIncludedInRelations$

D.2. Casos Abstractos de prueba

Para generar los casos abstractos de prueba se ejecutaron los comandos

```
loadspec elements.tex
selop AddInitialElement
addtactic AddInitialElement SP \cup elements \cup \{element?\}
selop AddRelationToElement
addtactic AddRelationToElement SP \cup relations \cup \{dadElement? \mapsto
    sonElement?\}
selop RemoveElement
addtactic RemoveElement SP \ndres \{element?\} \ndres relations
genalltt
genalltca
```

y se obtuvieron los siguientes casos:

D.2.1. AddInitialElement

AddInitialElement_SP_2_TCASE

AddInitialElement_SP_2

$relations = \emptyset$
$elements = \emptyset$
$element? = 0$

AddInitialElement_SP_12_TCASE

AddInitialElement_SP_12

$relations = \emptyset$
$elements = \{1\}$
$element? = 0$

AddInitialElement_SP_14_TCASE

AddInitialElement_SP_14

$relations = \emptyset$
$elements = \{0, 1\}$
$element? = 0$

AddInitialElement_SP_15_TCASE

AddInitialElement_SP_15

$relations = \emptyset$
$elements = \{0\}$
$element? = 0$

D.2.2. AddRelationToElement

AddRelationToElement_SP_2_TCASE

AddRelationToElement_SP_2

$sonElement? = 0$
$dadElement? = 0$
$relations = \emptyset$
$elements = \{0\}$

AddRelationToElement_SP_4_TCASE

AddRelationToElement_SP_4

$sonElement? = 0$
$dadElement? = 0$
$relations = \{(1, 1)\}$
$elements = \{0\}$

AddRelationToElement_SP_12_TCASE

AddRelationToElement_SP_12

$sonElement? = 0$
$dadElement? = 0$
$relations = \{(1, 0), (0, 1)\}$
$elements = \emptyset$

AddRelationToElement_SP_14_TCASE

AddRelationToElement_SP_14

$sonElement? = 0$
$dadElement? = 0$
$relations = \{(0,0),(1,0)\}$
$elements = \emptyset$

AddRelationToElement_SP_15_TCASE

AddRelationToElement_SP_15

$sonElement? = 0$
$dadElement? = 0$
$relations = \{(0,0)\}$
$elements = \emptyset$

AddRelationToElement_SP_20_TCASE

AddRelationToElement_SP_20

$sonElement? = 0$
$dadElement? = 0$
$relations = \{(1,0)\}$
$elements = \emptyset$

AddRelationToElement_SP_22_TCASE

AddRelationToElement_SP_22

$sonElement? = 0$
$dadElement? = 0$
$relations = \{(0,0),(1,0)\}$
$elements = \emptyset$

AddRelationToElement_SP_23_TCASE

AddRelationToElement_SP_23

$sonElement? = 0$
$dadElement? = 0$
$relations = \{(0,0)\}$
$elements = \emptyset$

AddRelationToElement_SP_26_TCASE

AddRelationToElement_SP_26

$sonElement? = 0$
$dadElement? = 0$
$relations = \emptyset$
$elements = \emptyset$

AddRelationToElement_SP_28_TCASE

AddRelationToElement_SP_28

$sonElement? = 0$
$dadElement? = 0$
$relations = \{(1,0)\}$
$elements = \emptyset$

AddRelationToElement_SP_30_TCASE

AddRelationToElement_SP_30

$sonElement? = 0$
$dadElement? = 0$
$relations = \{(0,0),(1,0)\}$
$elements = \emptyset$

AddRelationToElement_SP_31_TCASE

AddRelationToElement_SP_31

$sonElement? = 0$
$dadElement? = 0$
$relations = \{(0,0)\}$
$elements = \emptyset$

AddRelationToElement_SP_36_TCASE

AddRelationToElement_SP_36

$sonElement? = 0$
$dadElement? = 0$
$relations = \{(0,1),(0,2)\}$
$elements = \{0\}$

AddRelationToElement_SP_38_TCASE

AddRelationToElement_SP_38

$sonElement? = 0$
$dadElement? = 0$
$relations = \{(0,0),(1,0),(0,1)\}$
$elements = \{0\}$

D.2.3. RemoveElement

RemoveElement_SP_3_TCASE

RemoveElement_SP_3

$relations = \{(0,0)\}$
$elements = \emptyset$
$element? = 0$

RemoveElement_SP_4_TCASE

RemoveElement_SP_4

$relations = \{(0,0),(1,0)\}$
$elements = \emptyset$
$element? = 0$

RemoveElement_SP_10_TCASE

RemoveElement_SP_10

$relations = \{(0,0)\}$
$elements = \emptyset$
$element? = 0$

RemoveElement_SP_11_TCASE

RemoveElement_SP_11

$relations = \{(0,0),(1,1)\}$
$elements = \emptyset$
$element? = 0$

RemoveElement_SP_12_TCASE

RemoveElement_SP_12

$relations = \{(1, 0)\}$
$elements = \emptyset$
$element? = 0$

RemoveElement_SP_15_TCASE

RemoveElement_SP_15

$relations = \emptyset$
$elements = \emptyset$
$element? = 0$

RemoveElement_SP_19_TCASE

RemoveElement_SP_19

$relations = \{(1, 1)\}$
$elements = \emptyset$
$element? = 0$

D.3. Implementación

```
#!/usr/bin/perl
use feature qw(say);
use YAML::XS;
use Data::Dumper;

sub __fastest_dump {
  open F, '>', 'state.yml';
  print F Dumper ( @_ );
  close F;
}

package main;

# Hash donde se almacenan los mensajes a mostrar como salida
%mensajes = (
  'alreadyExistElements' => 'Ya existen elementos actualmente',
  'sonElementAlreadyRelatedAsDad' => 'El elemento hijo existe como padre de un
       elemento',
  'sonElementAlreadyRelatedAsChild' => 'El elemento hijo existe como hijo de
      un elemento',
  'dadElementDoesntExist' => 'El elemento padre no existe',
  'dadElementAlreadyHaveTwoRelatedElements' => 'El elemento padre ya tiene dos
       elementos asociados',
  'elementNotIncludedInRelations' => 'El elemento no se encuentra entre las
      relaciones'
);

@elementsImp = ();
@relationsImp = ();

sub isAnElement {
  $element = $_[0];
  $elementsCount = getElementsCount();
  for $i (0..$elementsCount-1) {
    if($elementsImp[$i] eq $element) {
      return 1;
    }
  }
  return 0;
}

sub getElementIndexFromElements {
  $element = $_[0];
  $elementsCount = getElementsCount();
  for $i (0..$elementsCount-1) {
```

```
    if($elementsImp[$i] eq $element) {
      return $i;
    }
  }
  return -1;
}

sub getRightElementIndexFromRelation {
  $element = $_[0];
  $relationsSize = @relationsImp;
  for $i (0..$relationsSize-1) {
    if($relationsImp[$i]->{"right"} eq $element) {
      return $i;
    }
  }
  return -1;
}

sub getRightRelations {
  $element = $_[0];
  @results = ();
  $relationsSize = @relationsImp;

  for $i (0..$relationsSize-1) {
    if($relationsImp[$i]->{"left"} eq $element) {
      push(@results, $relationsImp[$i]->{"right"});
    }
  }
  return $results;
}

sub isALeftRelationElement {
  $element = $_[0];
  $relationsSize = @relationsImp;
  for $i (0..$relationsSize-1) {
    if($relationsImp[$i]->{"left"} eq $element) {
      return 1;
    }
  }
  return 0;
}

sub isARightRelationElement {
  $element = $_[0];
  $relationsSize = @relationsImp;
  for $i (0..$relationsSize-1) {
    if($relationsImp[$i]->{"right"} eq $element) {
      return 1;
    }
  }
  return 0;
}

sub removeRightElementFromRelations {
  $element = $_[0];
  $relationsSize = @relationsImp;
  @indexes = ();
  for $i (0..$relationsSize-1) {
    if($relationsImp[$i]->{"right"} eq $element) {
      push(@indexes, $i);
    }
  }
  $size = @indexes;
  for $j (0..$size-1) {
    delete $relationsImp[$j];
  }
}

sub getElementsCount {
  $size = @elementsImp;
  return $size;
}


sub addInitialElement {
  $element = $_[0];
```

```
  $elementsCount = getElementsCount();
  if($elementsCount > 0) {
    say($mensajes{'alreadyExistElements'})
  } else {
    push(@elementsImp, $element);
  }
}

sub addRelationToElement {
  $dadElement = $_[0];
  $sonElement = $_[1];
  if(isAnElement($dadElement)) {
    @rightRelations = @{getRightRelations($dadElement)};
    $rightRelationsSize = @rightRelations;
    if ($rightRelationsSize > 2) {
      say($mensajes{'dadElementAlreadyHaveTwoRelatedElements'});
    } else {
      if (isALeftRelationElement($sonElement)) {
        say($mensajes{'sonElementAlreadyRelatedAsDad'});
      } else {
        if (isARightRelationElement($sonElement)) {
          say($mensajes{'sonElementAlreadyRelatedAsChild'});
        } else {
          if(isAnElement($sonElement) != 1) {
            push(@elementsImp, $sonElement);
          }
          $pair = {"left" => $dadElement, "right" => $sonElement};
          push(@relationsImp, $pair);
        }
      }
    }
  } else {
    say($mensajes{'dadElementDoesntExist'})
  }
}

sub removeElement {
  $element = $_[0];
  if (isALeftRelationElement($element) || isARightRelationElement($element)) {
    if(isALeftRelationElement($element)) {
      removeRightElementFromRelations($element);
    }
    if(isARightRelationElement($element)) {
      $index = getRightElementIndexFromRelation($element);
      delete $relationsImp[$index];
    }
    if(isAnElement($element)) {
      $index = getElementIndexFromElements($element);
      delete $elementsImp[$index];
    }
  } else {
    say($mensajes{'elementNotIncludedInRelations'});
  }
}
```

D.4. Leyes de refinamiento

```
@RRULE elementsAR

@DATATYPES
  DATATYPE Pair = RECORD Pair (left : INT, right: INT);

@LAWS
  elements ==> elementsImp AS ARRAY INT (4);
  relations ==> relationsImp AS ARRAY Pair (15) WITH [
    relations.@DOM ==> .left AS INT,
    relations.@RAN ==> .right AS INT
  ];
  dadElement? ==> dadElement AS INT;
  sonElement? ==> sonElement AS INT;

@UUT
addRelationToElement(dadElement, sonElement);
```

Las reglas de refinamiento `elementsAIE` y `elementsRE` son muy similares a `elementsAR` salvo por ciertos detalles tales como la sección `@UUT` donde se especifica la llamada al método correspondiente en cada caso particular.

D.5. Casos concretos de prueba en Perl

D.5.1. Operación AddInitialElement

Ejecutando los comandos

```
loadrefrule elementsAIE.atcal
ne AddInitialElement_VIS to test elements.pl implemented in perl with
    elementsAIE
```

se generan

- AddInitialElement_SP_2_CTCASE

```
#!/usr/bin/perl
use feature qw(say);
use YAML::XS;
use Data::Dumper;

sub __fastest_dump {
  open F, '>', 'state.yml';
  print F Dumper ( @_ );
  close F;
}

package Pair;

sub new {
    my ($class,$args) = @_;
    my $self = bless { }, $class;
}

sub set {
  my ($self, $left, $right) = @_;
  $self->{"left"} = $left;
  $self->{"right"} = $ageRange;
}

sub get {
  my $self = shift;
  my $arg = shift;
  return $self->{$arg};
}

package main;

@elements = ();
@relations = ();


sub addInitialElement {
  # Código que implementa addInitialElement
}

sub addRelationToElement {
  # Código que implementa addRelationToElement
}

sub removeElement {
  # Código que implementa removeElement
}

@elementsImp = ();
@relationsImp = ();
$element = 0;
addInitialElement($element);
$state->{'element'} = $element;
$state->{'relationsImp'} =  \@relationsImp;
$state->{'elementsImp'} =  \@elementsImp;
__fastest_dump($state);
```

- AddInitialElement_SP_12_CTCASE

```
...
@elementsImp = ();
$elementsImp[0] = 1;
@relationsImp = ();
$element = 0;
...
```

- AddInitialElement_SP_14_CTCASE

```
...
@elementsImp = ();
$elementsImp[0] = 0;
$elementsImp[1] = 1;
@relationsImp = ();
$element = 0;
...
```

- AddInitialElement_SP_15_CTCASE

```
...
@elementsImp = ();
$elementsImp[0] = 0;
@relationsImp = ();
$element = 0;
...
```

D.5.2. Operación AddRelationToElement

Ejecutando los comandos

```
loadrefrule elementsAR.atcal
ne AddRelationToElement_VIS to test elements.pl implemented in perl with
    elementsAR
```

se generan

- AddRelationToElement_SP_2_CTCASE

```
...
@elementsImp = ();
$elementsImp[0] = 0;
@relationsImp = ();
$dadElement = 0;
$sonElement = 0;
addRelationToElement($dadElement,$sonElement);
$state->{'relationsImp'} =  \@relationsImp;
$state->{'elementsImp'} =  \@elementsImp;
$state->{'sonElement'} = $sonElement;
$state->{'dadElement'} = $dadElement;
__fastest_dump($state);
```

- AddRelationToElement_SP_4_CTCASE

```
...
@elementsImp = ();
$elementsImp[0] = 0;
@relationsImp = ();
$relationsImp_left0 = 1;
$relationsImp_right0 = 1;
$relationsImp[0] = {"left" => $relationsImp_left0, "right" =>
    $relationsImp_right0};
$dadElement = 0;
$sonElement = 0;
...
```

- AddRelationToElement_SP_12_CTCASE

```
...
@elementsImp = ();
@relationsImp = ();
$relationsImp_left0 = 1;
$relationsImp_right0 = 0;
$relationsImp[0] = {"left" => $relationsImp_left0, "right" =>
    $relationsImp_right0};
$relationsImp_left1 = 0;
$relationsImp_right1 = 1;
$relationsImp[1] = {"left" => $relationsImp_left1, "right" =>
    $relationsImp_right1};
$dadElement = 0;
$sonElement = 0;
...
```

- AddRelationToElement_SP_14_CTCASE

```
...
@elementsImp = ();
@relationsImp = ();
$relationsImp_left0 = 1;
$relationsImp_right0 = 0;
$relationsImp[0] = {"left" => $relationsImp_left0, "right" =>
    $relationsImp_right0};
$relationsImp_left1 = 0;
$relationsImp_right1 = 0;
$relationsImp[1] = {"left" => $relationsImp_left1, "right" =>
    $relationsImp_right1};
$dadElement = 0;
$sonElement = 0;
...
```

- AddRelationToElement_SP_15_CTCASE

```
...
@elementsImp = ();
@relationsImp = ();
$relationsImp_left0 = 0;
$relationsImp_right0 = 0;
$relationsImp[0] = {"left" => $relationsImp_left0, "right" =>
    $relationsImp_right0};
$dadElement = 0;
$sonElement = 0;
...
```

- AddRelationToElement_SP_20_CTCASE

```
...
@elementsImp = ();
@relationsImp = ();
$relationsImp_left0 = 1;
$relationsImp_right0 = 0;
$relationsImp[0] = {"left" => $relationsImp_left0, "right" =>
    $relationsImp_right0};
$dadElement = 0;
$sonElement = 0;
...
```

- AddRelationToElement_SP_22_CTCASE

```
...
@elementsImp = ();
@relationsImp = ();
$relationsImp_left0 = 1;
$relationsImp_right0 = 0;
$relationsImp[0] = {"left" => $relationsImp_left0, "right" =>
    $relationsImp_right0};
$relationsImp_left1 = 0;
$relationsImp_right1 = 0;
$relationsImp[1] = {"left" => $relationsImp_left1, "right" =>
    $relationsImp_right1};
$dadElement = 0;
$sonElement = 0;
...
```

- AddRelationToElement_SP_23_CTCASE

```
...
@elementsImp = ();
@relationsImp = ();
$relationsImp_left0 = 0;
$relationsImp_right0 = 0;
$relationsImp[0] = {"left" => $relationsImp_left0, "right" =>
    $relationsImp_right0};
$dadElement = 0;
$sonElement = 0;
...
```

- AddRelationToElement_SP_26_CTCASE

```
...
@elementsImp = ();
@relationsImp = ();
$dadElement = 0;
$sonElement = 0;
...
```

- AddRelationToElement_SP_28_CTCASE

```
...
@elementsImp = ();
@relationsImp = ();
$relationsImp_left0 = 1;
$relationsImp_right0 = 0;
$relationsImp[0] = {"left" => $relationsImp_left0, "right" =>
    $relationsImp_right0};
$dadElement = 0;
$sonElement = 0;
...
```

- AddRelationToElement_SP_30_CTCASE

```
...
@elementsImp = ();
@relationsImp = ();
$relationsImp_left0 = 1;
$relationsImp_right0 = 0;
$relationsImp[0] = {"left" => $relationsImp_left0, "right" =>
    $relationsImp_right0};
$relationsImp_left1 = 0;
$relationsImp_right1 = 0;
$relationsImp[1] = {"left" => $relationsImp_left1, "right" =>
    $relationsImp_right1};
$dadElement = 0;
$sonElement = 0;
...
```

- AddRelationToElement_SP_31_CTCASE

```
...
@elementsImp = ();
@relationsImp = ();
$relationsImp_left0 = 0;
$relationsImp_right0 = 0;
$relationsImp[0] = {"left" => $relationsImp_left0, "right" =>
    $relationsImp_right0};
$dadElement = 0;
$sonElement = 0;
...
```

- AddRelationToElement_SP_36_CTCASE

```
...
@elementsImp = ();
$elementsImp[0] = 0;
@relationsImp = ();
$relationsImp_left0 = 0;
$relationsImp_right0 = 1;
```

```
$relationsImp[0] = {"left" => $relationsImp_left0, "right" =>
    $relationsImp_right0};
$relationsImp_left1 = 0;
$relationsImp_right1 = 2;
$relationsImp[1] = {"left" => $relationsImp_left1, "right" =>
    $relationsImp_right1};
$dadElement = 0;
$sonElement = 0;
...
```

- AddRelationToElement_SP_38_CTCASE

```
...
@elementsImp = ();
$elementsImp[0] = 0;
@relationsImp = ();
$relationsImp_left0 = 1;
$relationsImp_right0 = 0;
$relationsImp[0] = {"left" => $relationsImp_left0, "right" =>
    $relationsImp_right0};
$relationsImp_left1 = 0;
$relationsImp_right1 = 0;
$relationsImp[1] = {"left" => $relationsImp_left1, "right" =>
    $relationsImp_right1};
$relationsImp_left2 = 0;
$relationsImp_right2 = 1;
$relationsImp[2] = {"left" => $relationsImp_left2, "right" =>
    $relationsImp_right2};
$dadElement = 0;
$sonElement = 0;
...
```

D.5.3. Operación AddRelationToElement

Ejecutando los comandos

```
loadrefrule elementsRE.atcal
refine RemoveElement_VIS to test elements.pl implemented in perl with
    elementsRE
```

se generan

- RemoveElement_SP_3_CTCASE

```
...
@elementsImp = ();
@relationsImp = ();
$relationsImp_left0 = 0;
$relationsImp_right0 = 0;
$relationsImp[0] = {"left" => $relationsImp_left0, "right" =>
    $relationsImp_right0};
$element = 0;
removeElement($element);
$state->{'element'} = $element;
$state->{'relationsImp'} =  \@relationsImp;
$state->{'elementsImp'} =  \@elementsImp;
__fastest_dump($state);
```

- RemoveElement_SP_4_CTCASE

```
...
@elementsImp = ();
@relationsImp = ();
$relationsImp_left0 = 1;
$relationsImp_right0 = 0;
$relationsImp[0] = {"left" => $relationsImp_left0, "right" =>
    $relationsImp_right0};
$relationsImp_left1 = 0;
$relationsImp_right1 = 0;
$relationsImp[1] = {"left" => $relationsImp_left1, "right" =>
    $relationsImp_right1};
$element = 0;
...
```

- RemoveElement_SP_10_CTCASE

```
...
@elementsImp = ();
@relationsImp = ();
$relationsImp_left0 = 0;
$relationsImp_right0 = 0;
$relationsImp[0] = {"left" => $relationsImp_left0, "right" =>
    $relationsImp_right0};
$element = 0;
...
```

- RemoveElement_SP_11_CTCASE

```
...
@elementsImp = ();
@relationsImp = ();
$relationsImp_left0 = 0;
$relationsImp_right0 = 0;
$relationsImp[0] = {"left" => $relationsImp_left0, "right" =>
    $relationsImp_right0};
$relationsImp_left1 = 1;
$relationsImp_right1 = 1;
$relationsImp[1] = {"left" => $relationsImp_left1, "right" =>
    $relationsImp_right1};
$element = 0;
...
```

- RemoveElement_SP_12_CTCASE

```
...
@elementsImp = ();
@relationsImp = ();
$relationsImp_left0 = 1;
$relationsImp_right0 = 0;
$relationsImp[0] = {"left" => $relationsImp_left0, "right" =>
    $relationsImp_right0};
$element = 0;
...
```

- RemoveElement_SP_15_CTCASE

```
...
@elementsImp = ();
@relationsImp = ();
$element = 0;
...
```

- RemoveElement_SP_19_CTCASE

```
...
@elementsImp = ();
@relationsImp = ();
$relationsImp_left0 = 1;
$relationsImp_right0 = 1;
$relationsImp[0] = {"left" => $relationsImp_left0, "right" =>
    $relationsImp_right0};
$element = 0;
...
```

Apéndice E

Rango etario

Tanto en este caso de estudio como en el presentado en Blog se especifican sistemas funcionalmente similares, pero que representan escenarios diferentes. En particular, el caso de este apéndice describe un sistema que almacena personas asociándolas con su rango etario. Los archivos de este caso de estudio pueden encontrarse en este link.

E.1. Especificación Z

$[DNI]$

$AgeRange ::= young \mid adult \mid old$

Se cuenta con el tipo básico `DNI` utilizado como identificación para las personas del sistema y el tipo enumerado `AgeRange` con los siguientes posibles valores:

- `young`
- `adult`
- `old`

Luego, a partir de ellos se presenta el esquema de estados:

System

$$personRange : DNI \pfun AgeRange$$

donde se relaciona cada persona con su rango etario correspondiente.

Las operaciones disponibles son:

- Insert: agrega a una persona junto a su rango etario en el sistema

InsertOk

$$\Delta System$$
$$d? : DNI$$
$$r? : AgeRange$$

$$d? \notin \operatorname{dom} personRange$$
$$personRange' = personRange \cup \{d? \mapsto r?\}$$

PersonAlreadyRegistered

$$\Xi System$$
$$d? : DNI$$

$$d? \in \operatorname{dom} personRange$$

$$Insert == InsertOk \lor PersonAlreadyRegistered$$

- Update: actualiza el rango etario de una persona del sistema

$$
\begin{array}{|l}
\hline
\textit{UpdateOk} \\
\Delta \textit{System} \\
d? : \textit{DNI} \\
r? : \textit{AgeRange} \\
\hline
d? \in \operatorname{dom} \textit{personRange} \\
\textit{personRange}' = \textit{personRange} \oplus \{d? \mapsto r?\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{NonRegisteredPerson} \\
\Xi \textit{System} \\
d? : \textit{DNI} \\
\hline
d? \notin \operatorname{dom} \textit{personRange} \\
\hline
\end{array}
$$

$$\textit{Update} == \textit{UpdateOk} \lor \textit{NonRegisteredPerson}$$

- GetPeopleInRange: retorna las personas dentro de un mismo rango etario

$$
\begin{array}{|l}
\hline
\textit{GetPeopleInRange} \\
\Xi \textit{System} \\
r? : \textit{AgeRange} \\
\textit{result}! : \mathbb{P}\, \textit{DNI} \\
\hline
\textit{result}! = \operatorname{dom}(\textit{personRange} \rhd \{r?\}) \\
\hline
\end{array}
$$

- Delete: elimina una persona del sistema

$$
\begin{array}{|l}
\hline
\textit{DeleteOk} \\
\Delta \textit{System} \\
d? : \textit{DNI} \\
\hline
d? \in \operatorname{dom} \textit{personRange} \\
\textit{personRange}' = \{d?\} \lhd \textit{personRange} \\
\hline
\end{array}
$$

$$\textit{Delete} == \textit{DeleteOk} \lor \textit{NonRegisteredPerson}$$

E.2. Casos Abstractos de prueba

Para generar los casos abstractos de prueba se ejecutaron los comandos

```
loadspec ageRange.tex
selop Insert
addtactic Insert SP \cup personRange \cup \{d? \mapsto r?\}
selop Update
addtactic Update SP \oplus personRange \oplus \{d? \mapsto r?\}
selop Delete
addtactic Delete SP \ndres \{d?\} \ndres personRange
selop GetPeopleInRange
addtactic GetPeopleInRange SP \rres personRange \rres \{ r? \}
genalltt
genalltca
```

y se obtuvieron los siguientes casos:

E.2.1. Insert

$$
\begin{array}{|l}
\hline
\textit{Insert\_SP\_2\_TCASE} \\
\textit{Insert\_SP\_2} \\
\hline
\textit{personRange} = \emptyset \\
r? = \textit{young} \\
d? = \textit{dNI}1 \\
\hline
\end{array}
$$

Insert_SP_4_TCASE

Insert_SP_4

$personRange = \{(dNI2 \mapsto adult)\}$
$r? = young$
$d? = dNI1$

Insert_SP_12_TCASE

Insert_SP_12

$personRange = \{(dNI1 \mapsto adult)\}$
$r? = young$
$d? = dNI1$

Insert_SP_14_TCASE

Insert_SP_14

$personRange = \{(dNI1 \mapsto young), (dNI2 \mapsto adult)\}$
$r? = young$
$d? = dNI1$

Insert_SP_15_TCASE

Insert_SP_15

$personRange = \{(dNI1 \mapsto young)\}$
$r? = young$
$d? = dNI1$

E.2.2. Update

Update_SP_4_TCASE

Update_SP_4

$personRange = \{(dNI1 \mapsto old)\}$
$r? = young$
$d? = dNI1$

Update_SP_5_TCASE

Update_SP_5

$personRange = \{(dNI1 \mapsto adult), (dNI2 \mapsto old)\}$
$r? = young$
$d? = dNI1$

Update_SP_10_TCASE

Update_SP_10

$personRange = \emptyset$
$r? = young$
$d? = dNI1$

Update_SP_14_TCASE

Update_SP_14

$personRange = \{(dNI2 \mapsto adult)\}$
$r? = young$
$d? = dNI1$

E.2.3. GetPeopleInRange

GetPeopleInRange_SP_1_TCASE

GetPeopleInRange_SP_1

$personRange = \emptyset$
$r? = young$

GetPeopleInRange_SP_3_TCASE

GetPeopleInRange_SP_3

$personRange = \{(dNI1 \mapsto young)\}$
$r? = young$

GetPeopleInRange_SP_4_TCASE

GetPeopleInRange_SP_4

$personRange = \{(dNI1 \mapsto young), (dNI2 \mapsto adult)\}$
$r? = young$

GetPeopleInRange_SP_5_TCASE

GetPeopleInRange_SP_5

$personRange = \{(dNI1 \mapsto adult)\}$
$r? = young$

E.2.4. Delete

Delete_SP_3_TCASE

Delete_SP_3

$personRange = \{(dNI1 \mapsto old)\}$
$d? = dNI1$

Delete_SP_4_TCASE

Delete_SP_4

$personRange = \{(dNI1 \mapsto adult), (dNI2 \mapsto old)\}$
$d? = dNI1$

Delete_SP_8_TCASE

Delete_SP_8

$personRange = \emptyset$
$d? = dNI1$

E.3. Implementación

```
#!/usr/bin/perl
use feature qw(say);
use YAML::XS;
use Data::Dumper;

sub __fastest_dump {
  open F, '>', 'state.yml';
  print F Dumper ( @_ );
  close F;
}

```

```
# Defino el enumerado para rangos de edad
use constant {
    young => 'young',
    adult => 'adult',
    old => 'old'
};

package Pair;

sub new {
    my ($class,$args) = @_;
    my $self = bless { }, $class;
}

sub set {
  my ($self, $dni, $ageRange) = @_;
  $self->{"dni"} = $dni;
  $self->{"ageRange"} = $ageRange;
}

sub get {
  my $self = shift;
  my $arg = shift;
  return $self->{$arg};
}

package Ranges;

sub new {
    my ($class,$args) = @_;
    @emptyArray = ();
    my $self = bless {"elem" => \@emptyArray}, $class;
}

sub push {
  my ($self, $pair) = @_;
  @elem = @{$self->{"elem"}};
  $elemArraySize = @elem;
  $self->{"elem"}[$elemArraySize] = $pair;
}

sub pop {
  my $self = shift;
  my $arg = shift;
  @elem = @{$self->{"elem"}};
  $elemArraySize = @elem;
  if($elemArraySize == 0) {
    die;
  }
  $index = $elemArraySize-1;
  $result = $self->{"elem"}[$index];
  delete $self->{"elem"}[$index];
  return $result;
}

package main;

# Hash donde se almacenan los mensajes a mostrar como salida
%mensajes = (
  'personAlreadyRegistered' => 'La persona ya fue registrada',
  'nonRegisteredPerson' => 'La persona aún no fue registrada'
);

$ranges = Ranges->new();

sub isRegistered {
  $dni = $_[0];
  $rangesCount = getRangesSize();
  for $i (0..$rangesCount-1) {
    if($ranges->{"elem"}[$i]->{"dni"} eq $dni) {
      return 1;
    }
  }
  return 0;
}

```

```
sub getPersonIndex {
  $dni = $_[0];
  $rangesSize = getRangesSize();
  for $i (0..$rangesSize-1) {
    if($ranges->{"elem"}[$i]->{"dni"} eq $dni) {
      return $i;
    }
  }
  return -1;
}

sub getRangesSize {
  $size = @{$ranges->{"elem"}};
  return $size;
}

sub insert {
  $dni = $_[0];
  $range = $_[1];
  if(isRegistered($dni)) {
    say($mensajes{'personAlreadyRegistered'})
  } else {
    push(@{$ranges->{"elem"}}, {"dni" => $dni, "ageRange" => $range});
  }
}

sub update {
  $dni = $_[0];
  $range = $_[1];
  if(isRegistered($dni)) {
    $index = getPersonIndex($dni);
    $ranges->{"elem"}[$index]->{"ageRange"} = $range;
  } else {
    say($mensajes{'nonRegisteredPerson'})
  }
}

sub getPeopleInRange {
  $range = $_[0];
  $rangesSize = getRangesSize();
  @result = ();
  for $i (0..$rangesSize-1) {
    if($ranges->{"elem"}[$i]->{"ageRange"} eq $range) {
      push(@result, $ranges->{"elem"}[$i]->{"dni"});
    }
  }
  return $result;
}

sub deletePerson {
  $dni = $_[0];
  if(isRegistered($dni)) {
    $index = getPersonIndex($dni);
    delete $ranges->{"elem"}[$index];
  } else {
    say($mensajes{'nonRegisteredPerson'})
  }
}
```

E.4. Leyes de refinamiento

E.4.1. Identificador de regla

```
@RRULE ageRangeInsert
```

Define que el nombre de la regla de refinamiento es `ageRangeInsert`

E.4.2. Definición de tipos de datos

```
@DATATYPES
  DATATYPE ageRange = ENUM ageRange ( young, adult, old );
```

```
  DATATYPE Pair = RECORD Pair (dni : STRING, ageRange: ageRange);
  DATATYPE Ranges = MODULE "Ranges" CONSTRUCTOR new () SETTER push (pair :
      Pair) GETTER pop ();
```

Se definen los tipos de datos:

- ageRange: enumerado cuyos posibles valores son:
  - young
  - adult
  - old
- Pair: record con los siguientes miembros
  - dni : STRING
  - ageRange : ageRange
- Ranges: contrato de constructor `new`, setter `push`, getter `pop` y contiene a `pair` de tipo `Pair` como único miembro.

E.4.3. Especificación de leyes de refinamiento

```
@LAWS
  personRange ==> ranges AS Ranges WITH [
    @ELEM ==> pair AS Pair WITH [
      @ELEM.#1 ==> .dni AS STRING,
      @ELEM.#2 ==> .ageRange AS ageRange MAP [
        young -> young,
        adult -> adult,
        old -> old
      ]
    ]
  ];
  d? ==> dni AS STRING;
  r? ==> range AS ageRange MAP [
    young -> young,
    adult -> adult,
    old -> old
  ]
```

Se especifica que:

- La variable de especificación `personRange` se refina como `ranges` que es un contrato. A continuación se indica que cada elemento se refina como `pair` de tipo `Pair` donde:
  - `@ELEM.#1 ==>`...: indica que el primer componente del elemento se refina como el miembro `dni` (primer miembro de `Pair`)
  - `@ELEM.#2 ==>`...: indica que el segundo componente del elemento se refina como el miembro `ageRange` (segundo miembro de `Pair`) para el cual se especifica el mapeo entre los enumerados de la especificación funcional y la implementación
- `d?` se refina a la variable de implementación `dni`
- `r?` se refina a la variable de implementación `range`

E.4.4. Especificación de método a testear

```
@UUT
insert(dni, range);
```

Indica que el método a testear es `insert` y sus parámetros son `dni` y `range`.

Las reglas de refinamiento `ageRangeUpdate`, `ageRangeGetPeopleInRange` y `ageRangeDelete` tienen una definición similar salvo por algunos detalles inherentes a cada operación, como por ejemplo la llamada al método correspondiente en la sección `@UUT`.

E.5. Casos concretos de prueba en Perl

E.5.1. Operación Insert

Ejecutando los comandos

```
loadrefrule ageRangeInsert.atcal
refine Insert_VIS to test ageRange.pl implemented in perl with ageRangeInsert
```

se generan

- Insert_SP_2_CTCASE

```
#!/usr/bin/perl
use feature qw(say);
use YAML::XS;
use Data::Dumper;

sub __fastest_dump {
  open F, '>', 'state.yml';
  print F Dumper ( @_ );
  close F;
}


# Defino el enumerado para rangos de edad
use constant {
    young => 'young',
    adult => 'adult',
    old => 'old'
};

package Pair;

sub new {
    my ($class,$args) = @_;
    my $self = bless { }, $class;
}

sub set {
  my ($self, $dni, $ageRange) = @_;
  $self->{"dni"} = $dni;
  $self->{"ageRange"} = $ageRange;
}

sub get {
  my $self = shift;
  my $arg = shift;
  return $self->{$arg};
}

package Ranges;

sub new {
    my ($class,$args) = @_;
    @emptyArray = ();
    my $self = bless {"elem" => \@emptyArray}, $class;
}

sub push {
  my ($self, $pair) = @_;
  @elem = @{$self->{"elem"}};
  $elemArraySize = @elem;
  $self->{"elem"}[$elemArraySize] = $pair;
}

sub pop {
  my $self = shift;
  my $arg = shift;
  @elem = @{$self->{"elem"}};
  $elemArraySize = @elem;
  if($elemArraySize == 0) {
    die;
  }
  $index = $elemArraySize-1;
  $result = $self->{"elem"}[$index];
```

```
  delete $self->{"elem"}[$index];
  return $result;
}

package main;

$ranges = Ranges->new();

sub insert {
  # Código que implementa insert
}

sub update {
  # Código que implementa update
}

sub getPeopleInRange {
  # Código que implementa getPeopleInRange
}

sub deletePerson {
    # Código que implementa deletePerson
}

$ranges_tmp = Ranges->new();
$ranges = $ranges_tmp;
$dni = "dNI1";
$range = young;
insert($dni,$range);
$state->{'range'} =  \$range;
$state->{'ranges'} =  \$ranges;
$state->{'dni'} = $dni;
__fastest_dump($state);
```

- Insert_SP_4_CTCASE

```
...
$ranges_tmp = Ranges->new();
$pair_dni0 = "dNI2";
$pair_ageRange0 = adult;
$ranges_pair0 = {"dni" => $pair_dni0, "ageRange" => $pair_ageRange0};
$ranges_tmp->push($ranges_pair0);
$ranges = $ranges_tmp;
$dni = "dNI1";
$range = young;
...
```

- Insert_SP_12_CTCASE

```
...
$ranges_tmp = Ranges->new();
$pair_dni0 = "dNI1";
$pair_ageRange0 = adult;
$ranges_pair0 = {"dni" => $pair_dni0, "ageRange" => $pair_ageRange0};
$ranges_tmp->push($ranges_pair0);
$ranges = $ranges_tmp;
$dni = "dNI1";
$range = young;
...
```

- Insert_SP_14_CTCASE

```
...
$ranges_tmp = Ranges->new();
$pair_dni0 = "dNI2";
$pair_ageRange0 = adult;
$ranges_pair0 = {"dni" => $pair_dni0, "ageRange" => $pair_ageRange0};
$pair_dni1 = "dNI1";
$pair_ageRange1 = young;
$ranges_pair1 = {"dni" => $pair_dni1, "ageRange" => $pair_ageRange1};
$ranges_tmp->push($ranges_pair0);
$ranges_tmp->push($ranges_pair1);
$ranges = $ranges_tmp;
$dni = "dNI1";
```

```
$range = young;
...
```

- Insert_SP_15_CTCASE

```
...
$ranges_tmp = Ranges->new();
$pair_dni0 = "dNI1";
$pair_ageRange0 = young;
$ranges_pair0 = {"dni" => $pair_dni0, "ageRange" => $pair_ageRange0};
$ranges_tmp->push($ranges_pair0);
$ranges = $ranges_tmp;
$dni = "dNI1";
$range = young;
...
```

E.5.2. Operación Update

Ejecutando los comandos

```
loadrefrule ageRangeUpdate.atcal
refine Update_VIS to test ageRange.pl implemented in perl with ageRangeUpdate
```

se generan

- Update_SP_4_CTCASE

```
...
$ranges_tmp = Ranges->new();
$pair_dni0 = "dNI1";
$pair_ageRange0 = old;
$ranges_pair0 = {"dni" => $pair_dni0, "ageRange" => $pair_ageRange0};
$ranges_tmp->push($ranges_pair0);
$ranges = $ranges_tmp;
$dni = "dNI1";
$range = young;
update($dni,$range);
$state->{'range'} =  \$range;
$state->{'ranges'} =  \$ranges;
$state->{'dni'} = $dni;
__fastest_dump($state);
```

- Update_SP_5_CTCASE

```
...
$ranges_tmp = Ranges->new();
$pair_dni0 = "dNI1";
$pair_ageRange0 = adult;
$ranges_pair0 = {"dni" => $pair_dni0, "ageRange" => $pair_ageRange0};
$pair_dni1 = "dNI2";
$pair_ageRange1 = old;
$ranges_pair1 = {"dni" => $pair_dni1, "ageRange" => $pair_ageRange1};
$ranges_tmp->push($ranges_pair0);
$ranges_tmp->push($ranges_pair1);
$ranges = $ranges_tmp;
$dni = "dNI1";
$range = young;
...
```

- Update_SP_10_CTCASE

```
...
$ranges_tmp = Ranges->new();
$ranges = $ranges_tmp;
$dni = "dNI1";
$range = young;
...
```

- Update_SP_14_CTCASE

```
...
$ranges_tmp = Ranges->new();
$pair_dni0 = "dNI2";
$pair_ageRange0 = adult;
$ranges_pair0 = {"dni" => $pair_dni0, "ageRange" => $pair_ageRange0};
$ranges_tmp->push($ranges_pair0);
$ranges = $ranges_tmp;
$dni = "dNI1";
$range = young;
...
```

E.5.3. Operación GetPeopleInRange

Ejecutando los comandos

```
loadrefrule ageRangeGetPeopleInRange.atcal
refine GetPeopleInRange_VIS to test ageRange.pl implemented in perl with
    ageRangeGetPeopleInRange
```

se generan

- GetPeopleInRange_SP_1_CTCASE

```
...
$ranges_tmp = Ranges->new();
$ranges = $ranges_tmp;
$range = young;
$result = getPeopleInRange($range);
$state->{'range'} =  \$range;
$state->{'ranges'} =  \$ranges;
$state->{'result'} =  \@result;
__fastest_dump($state);
```

- GetPeopleInRange_SP_3_CTCASE

```
...
$ranges_tmp = Ranges->new();
$pair_dni0 = "dNI1";
$pair_ageRange0 = young;
$ranges_pair0 = {"dni" => $pair_dni0, "ageRange" => $pair_ageRange0};
$ranges_tmp->push($ranges_pair0);
$ranges = $ranges_tmp;
$range = young;
...
```

- GetPeopleInRange_SP_4_CTCASE

```
...
$ranges_tmp = Ranges->new();
$pair_dni0 = "dNI2";
$pair_ageRange0 = adult;
$ranges_pair0 = {"dni" => $pair_dni0, "ageRange" => $pair_ageRange0};
$pair_dni1 = "dNI1";
$pair_ageRange1 = young;
$ranges_pair1 = {"dni" => $pair_dni1, "ageRange" => $pair_ageRange1};
$ranges_tmp->push($ranges_pair0);
$ranges_tmp->push($ranges_pair1);
$ranges = $ranges_tmp;
$range = young;
...
```

- GetPeopleInRange_SP_5_CTCASE

```
...
$ranges_tmp = Ranges->new();
$pair_dni0 = "dNI1";
$pair_ageRange0 = adult;
$ranges_pair0 = {"dni" => $pair_dni0, "ageRange" => $pair_ageRange0};
$ranges_tmp->push($ranges_pair0);
$ranges = $ranges_tmp;
$range = young;
...
```

E.5.4. Operación Delete

Ejecutando los comandos

```
loadrefrule ageRangeDelete.atcal
refine Delete_VIS to test ageRange.pl implemented in perl with ageRangeDelete
```

se generan

- Delete_SP_3_CTCASE

```
...
$ranges_tmp = Ranges->new();
$pair_dni0 = "dNI1";
$pair_ageRange0 = old;
$ranges_pair0 = {"dni" => $pair_dni0, "ageRange" => $pair_ageRange0};
$ranges_tmp->push($ranges_pair0);
$ranges = $ranges_tmp;
$dni = "dNI1";
deletePerson($dni);
$state->{'ranges'} =  \$ranges;
$state->{'dni'} = $dni;
__fastest_dump($state);
```

- Delete_SP_4_CTCASE

```
...
$ranges_tmp = Ranges->new();
$pair_dni0 = "dNI1";
$pair_ageRange0 = adult;
$ranges_pair0 = {"dni" => $pair_dni0, "ageRange" => $pair_ageRange0};
$pair_dni1 = "dNI2";
$pair_ageRange1 = old;
$ranges_pair1 = {"dni" => $pair_dni1, "ageRange" => $pair_ageRange1};
$ranges_tmp->push($ranges_pair0);
$ranges_tmp->push($ranges_pair1);
$ranges = $ranges_tmp;
$dni = "dNI1";
...
```

- Delete_SP_8_CTCASE

```
...
$ranges_tmp = Ranges->new();
$ranges = $ranges_tmp;
$dni = "dNI1";
...
```

- Delete_SP_12_CTCASE

```
...
$ranges_tmp = Ranges->new();
$pair_dni0 = "dNI2";
$pair_ageRange0 = adult;
$ranges_pair0 = {"dni" => $pair_dni0, "ageRange" => $pair_ageRange0};
$ranges_tmp->push($ranges_pair0);
$ranges = $ranges_tmp;
$dni = "dNI1";
...
```

Apéndice F

Blog

Se presenta un sistema que permite agregar, actualizar y eliminar usuarios vinculados a un blog. Los archivos de este caso de estudio pueden encontrarse en este link.

F.1. Especificación Z

$[Nickname, Name, Address]$

Se cuenta con los siguientes tipos básicos:

- `Nickname`: nombres de usuario dentro del blog
- `Name`: se utiliza para representar los nombres reales de las personas del blog
- `Address`: representa la dirección donde habita la persona

Luego, se presenta el esquema de estados:

System

$$peoplesBlog : Nickname \pfun (Name \times \mathbb{N} \times Address)$$

donde se vincula un nombre de usuario con la información correspondiente a la persona. En este caso se contemplan únicamente:

- Nombre
- Edad
- Dirección

Una vez definido lo antedicho, se procede a hacer lo propio con las operaciones

- Insert: agrega un usuario

InsertOk

$$\Delta System$$
$$nickname? : Nickname$$
$$person? : Name \times \mathbb{N} \times Address$$

$$nickname? \notin \dom peoplesBlog$$
$$peoplesBlog' = peoplesBlog \cup \{nickname? \mapsto person?\}$$

UserAlreadyRegistered

$$\Xi System$$
$$nickname? : Nickname$$

$$nickname? \in \dom peoplesBlog$$

$$Insert == InsertOk \lor UserAlreadyRegistered$$

- Update: actualiza los datos de un usuario

$$
\begin{array}{|l}
\hline
\textit{UpdateOk} \\
\Delta \textit{System} \\
\textit{nickname?} : \textit{Nickname} \\
\textit{person?} : \textit{Name} \times \mathbb{N} \times \textit{Address} \\
\hline
\textit{nickname?} \in \operatorname{dom} \textit{peoplesBlog} \\
\textit{peoplesBlog}' = \textit{peoplesBlog} \oplus \{\textit{nickname?} \mapsto \textit{person?}\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{NonRegisteredUser} \\
\Xi \textit{System} \\
\textit{nickname?} : \textit{Nickname} \\
\hline
\textit{nickname?} \notin \operatorname{dom} \textit{peoplesBlog} \\
\hline
\end{array}
$$

$$\textit{Update} == \textit{UpdateOk} \lor \textit{NonRegisteredUser}$$

- Delete: elimina un usuario

$$
\begin{array}{|l}
\hline
\textit{DeleteOk} \\
\Delta \textit{System} \\
\textit{nickname?} : \textit{Nickname} \\
\hline
\textit{nickname?} \in \operatorname{dom} \textit{peoplesBlog} \\
\textit{peoplesBlog}' = \{\textit{nickname?}\} \lhd \textit{peoplesBlog} \\
\hline
\end{array}
$$

$$\textit{Delete} == \textit{DeleteOk} \lor \textit{NonRegisteredUser}$$

F.2. Casos Abstractos de prueba

Para generar los casos abstractos de prueba se ejecutaron los comandos

```
loadspec peoplesBlog.tex
selop Insert
addtactic Insert SP \cup peoplesBlog \cup \{nickname? \mapsto person?\}
selop Update
addtactic Update SP \oplus peoplesBlog \oplus \{nickname? \mapsto person?\}
selop Delete
addtactic Delete SP \ndres \{nickname?\} \ndres peoplesBlog
genalltt
genalltca
```

y se obtuvieron los siguientes casos:

F.2.1. Insert

$$
\begin{array}{|l}
\hline
\textit{Insert\_SP\_2\_TCASE} \\
\textit{Insert\_SP\_2} \\
\hline
\textit{person?} = (\textit{name}1, 0, \textit{address}2) \\
\textit{nickname?} = \textit{nickname}3 \\
\textit{peoplesBlog} = \emptyset \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{Insert\_SP\_4\_TCASE} \\
\textit{Insert\_SP\_4} \\
\hline
\textit{person?} = (\textit{name}2, 0, \textit{address}3) \\
\textit{nickname?} = \textit{nickname}1 \\
\textit{peoplesBlog} = \{(\textit{nickname}2 \mapsto (\textit{name}1, 1, \textit{address}1))\} \\
\hline
\end{array}
$$

Insert_SP_12_TCASE

Insert_SP_12

$person? = (name2, 0, address3)$
$nickname? = nickname1$
$peoplesBlog = \{(nickname1 \mapsto (name1, 1, address1))\}$

Insert_SP_14_TCASE

Insert_SP_14

$person? = (name1, 0, address2)$
$nickname? = nickname1$
$peoplesBlog = \{(nickname1 \mapsto (name1, 0, address2)), (nickname2 \mapsto (name1, 1, address3))\}$

Insert_SP_15_TCASE

Insert_SP_15

$person? = (name2, 0, address3)$
$nickname? = nickname1$
$peoplesBlog = \{(nickname1 \mapsto (name2, 0, address3))\}$

F.2.2. Update

Update_SP_4_TCASE

Update_SP_4

$person? = (name3, 0, address4)$
$nickname? = nickname1$
$peoplesBlog = \{(nickname1 \mapsto (name1, 1, address2))\}$

Update_SP_5_TCASE

Update_SP_5

$person? = (name3, 0, address4)$
$nickname? = nickname1$
$peoplesBlog = \{(nickname1 \mapsto (name1, 1, address1)), (nickname2 \mapsto (name1, 1, address2))\}$

Update_SP_10_TCASE

Update_SP_10

$person? = (name1, 0, address2)$
$nickname? = nickname3$
$peoplesBlog = \emptyset$

Update_SP_14_TCASE

Update_SP_14

$person? = (name2, 0, address3)$
$nickname? = nickname1$
$peoplesBlog = \{(nickname2 \mapsto (name1, 1, address1))\}$

F.2.3. Delete

Delete_SP_3_TCASE

Delete_SP_3

$nickname? = nickname1$
$peoplesBlog = \{(nickname1 \mapsto (name1, 1, address2))\}$

Delete_SP_4_TCASE

Delete_SP_4

$nickname? = nickname1$

$peoplesBlog = \{(nickname1 \mapsto (name1, 1, address1)), (nickname2 \mapsto (name1, 1, address2))\}$

Delete_SP_8_TCASE

Delete_SP_8

$nickname? = nickname1$

$peoplesBlog = \emptyset$

Delete_SP_12_TCASE

Delete_SP_12

$nickname? = nickname1$

$peoplesBlog = \{(nickname2 \mapsto (name1, 1, address1))\}$

F.3. Implementación

```
#!/usr/bin/perl
use feature qw(say);
use YAML::XS;
use Data::Dumper;

sub __fastest_dump {
  open F, '>', 'state.yml';
  print F Dumper ( @_ );
  close F;
}

package Person;

sub new {
    my ($class,$args) = @_;
    my $self = bless { }, $class;
}

sub set {
  my ($self, $name, $age, $address) = @_;
  $self->{"name"} = $name;
  $self->{"age"} = $age;
  $self->{"address"} = $address;
}

sub get {
  my $self = shift;
  my $arg = shift;
  return $self->{$arg};
}

package main;

# Hash donde se almacenan los mensajes a mostrar como salida
%mensajes = (
  'userAlreadyRegistered' => 'La persona ya fue registrada',
  'nonRegisteredUser' => 'La persona aún no fue registrada'
);

@blog = ();

sub isRegistered {
  $nickname = $_[0];
  $blogSize = getBlogSize();
  for $i (0..$blogSize-1) {
    if($blog[$i]->{"nickname"} eq $nickname) {
      return 1;
    }
```

```
  }
  return 0;
}

sub getUserIndex {
  $nickname = $_[0];
  $blogSize = getBlogSize();
  for $i (0..$blogSize-1) {
    if($blog[$i]->{"nickname"} eq $nickname) {
      return $i;
    }
  }
  return -1;
}

sub getBlogSize {
  $size = @blog;
  return $size;
}

sub insert {
  my ($nickname, $person) = @_;
  if(isRegistered($nickname)) {
    say($mensajes{'userAlreadyRegistered'})
  } else {
    push(@blog, {"nickname" => $nickname, "person" => $person});
  }
}

sub update {
  my ($nickname, $person) = @_;
  if(isRegistered($nickname)) {
    $index = getUserIndex($nickname);
    $blog[$index]->{"person"} = $person;
  } else {
    say($mensajes{'nonRegisteredUser'})
  }
}

sub deleteUser {
  $nickname = $_[0];
  if(isRegistered($nickname)) {
    $index = getUserIndex($nickname);
    delete $blog[$index];
  } else {
    say($mensajes{'nonRegisteredUser'})
  }
}
```

F.4. Leyes de refinamiento

F.4.1. Identificador de regla

```
@RRULE peoplesBlogInsert
```

Define que el nombre de la regla de refinamiento es `peoplesBlogInsert`

F.4.2. Definición de tipos de datos

```
@DATATYPES
  DATATYPE NickName = STRING;
  DATATYPE Person = MODULE "Person" CONSTRUCTOR new () SETTER set (name :
      STRING, age : INT, address : STRING) GETTER get (member : STRING);
  DATATYPE Pair = RECORD Pair (nickname : NickName, person : Person);
  DATATYPE Blog = ARRAY Pair (5);
```

Se definen los tipos de datos:

- NickName: tipo sinónimo de string
- Person: contrato de constructor `new`, setter `set`, getter `get` y compuesto de los siguientes miembros:

- `name`
- `age`
- `address`

- Pair: record con los miembros
  - `nickname` : STRING
  - `person` : ageRange
- Blog: array de longitud cinco cuyos elementos son de tipo `Pair`

F.4.3. Especificación de leyes de refinamiento

```
@LAWS
  peoplesBlog ==> blog AS Blog WITH [
    peoplesBlog.@DOM ==> .nickname AS NickName,
    peoplesBlog.@RAN ==> .person AS Person WITH [
      peoplesBlog.@RAN.#1 ==> name AS STRING,
      peoplesBlog.@RAN.#2 ==> age AS INT,
      peoplesBlog.@RAN.#3 ==> address AS STRING
    ]
  ];
  nickname? ==> nickname AS NickName;
  person? ==> person AS Person WITH [
    person?.#1 ==> name AS STRING,
    person?.#2 ==> age AS INT,
    person?.#3 ==> address AS STRING
  ];
```

Se especifica que:

- La variable de especificación `peoplesBlog` se refina como `blog` que es un array con tipo `Pair`. A continuación se indica que:
  - `@peoplesBlog.@DOM ==>...`: cada elemento del dominio de `peoplesBlog` se refina como `nickname` (primer miembro de `Pair`)
  - `@peoplesBlog.@RAN ==>...`: cada elemento del dominio de `peoplesBlog` se refina como `nickname` (primer miembro de `Pair`) de tipo `Person`. A su vez se indica que:
    - `peoplesBlog.@RAN.#1 ==>...`: indica que el primer componente se refina como `name`
    - `peoplesBlog.@RAN.#2 ==>...`: indica que el segundo componente se refina como `age`
    - `peoplesBlog.@RAN.#3 ==>...`: indica que el tercer componente se refina como `address`
- `nickname?` se refina a la variable de implementación `nickname`
- `person?` se refina a la variable de implementación `person` del mismo modo que cada uno de los elementos del rango de `peoplesBlog`

F.4.4. Especificación de método a testear

```
@UUT
insert(nickname, person);
```

Indica que el método a testear es `insert` y sus parámetros son `nickname` y `person`.

Las reglas de refinamiento `peoplesBlogUpdate` y `peoplesBlogDelete` tienen una definición similar salvo por algunos detalles inherentes a cada operación, como por ejemplo la llamada al método correspondiente en la sección `@UUT`.

F.5. Casos concretos de prueba en Perl

F.5.1. Operación Insert

Ejecutando los comandos

```
loadrefrule peoplesBlogInsert.atcal
refine Insert_VIS to test peoplesBlog.pl implemented in perl with
    peoplesBlogInsert
```

se generan

- Insert_SP_2_CTCASE

```
#!/usr/bin/perl
use feature qw(say);
use YAML::XS;
use Data::Dumper;

sub __fastest_dump {
  open F, '>', 'state.yml';
  print F Dumper ( @_ );
  close F;
}


package Person;

sub new {
    my ($class,$args) = @_;
    my $self = bless { }, $class;
}

sub set {
  my ($self, $name, $age, $address) = @_;
  $self->{"name"} = $name;
  $self->{"age"} = $age;
  $self->{"address"} = $address;
}

sub get {
  my $self = shift;
  my $arg = shift;
  return $self->{$arg};
}

package main;

# Hash donde se almacenan los mensajes a mostrar como salida
%mensajes = (
  'userAlreadyRegistered' => 'La persona ya fue registrada',
  'nonRegisteredUser' => 'La persona aún no fue registrada'
);

@blog = ();

sub insert {
  # Código que implementa insert
}

sub update {
  # Código que implementa update
}

sub deleteUser {
  # Código que implementa deleteUser
}

@blog = ();
$nickname = "nickname3";
$person_tmp = Person->new();
$person_name0 = "name1";
$person_age0 = 0;
$person_address0 = "address2";
$person_tmp->set($person_name0,$person_age0,$person_address0);
$person = $person_tmp;
```

```
insert($nickname,$person);
$state->{'blog'} =  \@blog;
$state->{'nickname'} = $nickname;
$state->{'person'} =  \$person;
__fastest_dump($state);
```

- Insert_SP_4_CTCASE

```
...
@blog = ();
$blog_person_tmp0 = Person->new();
$person_name0 = "name1";
$person_age0 = 1;
$person_address0 = "address1";
$blog_person_tmp0->set($person_name0,$person_age0,$person_address0);
$blog_person0 = $blog_person_tmp0;
$blog_nickname0 = "nickname2";
$blog[0] = {"nickname" => $blog_nickname0, "person" => $blog_person0};
$nickname = "nickname1";
$person_tmp = Person->new();
$person_name0 = "name2";
$person_age0 = 0;
$person_address0 = "address3";
$person_tmp->set($person_name0,$person_age0,$person_address0);
$person = $person_tmp;
...
```

- Insert_SP_12_CTCASE

```
...
@blog = ();
$blog_person_tmp0 = Person->new();
$person_name0 = "name1";
$person_age0 = 1;
$person_address0 = "address1";
$blog_person_tmp0->set($person_name0,$person_age0,$person_address0);
$blog_person0 = $blog_person_tmp0;
$blog_nickname0 = "nickname1";
$blog[0] = {"nickname" => $blog_nickname0, "person" => $blog_person0};
$nickname = "nickname1";
$person_tmp = Person->new();
$person_name0 = "name2";
$person_age0 = 0;
$person_address0 = "address3";
$person_tmp->set($person_name0,$person_age0,$person_address0);
$person = $person_tmp;
...
```

- Insert_SP_14_CTCASE

```
...
@blog = ();
$blog_person_tmp0 = Person->new();
$person_name0 = "name1";
$person_age0 = 1;
$person_address0 = "address3";
$blog_person_tmp0->set($person_name0,$person_age0,$person_address0);
$blog_person0 = $blog_person_tmp0;
$blog_person_tmp1 = Person->new();
$person_name1 = "name1";
$person_age1 = 0;
$person_address1 = "address2";
$blog_person_tmp1->set($person_name1,$person_age1,$person_address1);
$blog_person1 = $blog_person_tmp1;
$blog_nickname0 = "nickname2";
$blog[0] = {"nickname" => $blog_nickname0, "person" => $blog_person0};
$blog_nickname1 = "nickname1";
$blog[1] = {"nickname" => $blog_nickname1, "person" => $blog_person1};
$nickname = "nickname1";
$person_tmp = Person->new();
$person_name0 = "name1";
$person_age0 = 0;
$person_address0 = "address2";
$person_tmp->set($person_name0,$person_age0,$person_address0);
```

```
$person = $person_tmp;
...
```

- Insert_SP_15_CTCASE

```
...
@blog = ();
$blog_person_tmp0 = Person->new();
$person_name0 = "name2";
$person_age0 = 0;
$person_address0 = "address3";
$blog_person_tmp0->set($person_name0,$person_age0,$person_address0);
$blog_person0 = $blog_person_tmp0;
$blog_nickname0 = "nickname1";
$blog[0] = {"nickname" => $blog_nickname0, "person" => $blog_person0};
$nickname = "nickname1";
$person_tmp = Person->new();
$person_name0 = "name2";
$person_age0 = 0;
$person_address0 = "address3";
$person_tmp->set($person_name0,$person_age0,$person_address0);
$person = $person_tmp;
...
```

F.5.2. Operación Update

Ejecutando los comandos

```
loadrefrule peoplesBlogUpdate.atcal
refine Update_VIS to test peoplesBlog.pl implemented in perl with
    peoplesBlogUpdate
```

se generan

- Update_SP_4_CTCASE

```
...
@blog = ();
$blog_person_tmp0 = Person->new();
$person_name0 = "name1";
$person_age0 = 1;
$person_address0 = "address2";
$blog_person_tmp0->set($person_name0,$person_age0,$person_address0);
$blog_person0 = $blog_person_tmp0;
$blog_nickname0 = "nickname1";
$blog[0] = {"nickname" => $blog_nickname0, "person" => $blog_person0};
$nickname = "nickname1";
$person_tmp = Person->new();
$person_name0 = "name3";
$person_age0 = 0;
$person_address0 = "address4";
$person_tmp->set($person_name0,$person_age0,$person_address0);
$person = $person_tmp;
update($nickname,$person);
$state->{'blog'} =  \@blog;
$state->{'nickname'} = $nickname;
$state->{'person'} =  \$person;
__fastest_dump($state);
```

- Update_SP_5_CTCASE

```
...
@blog = ();
$blog_person_tmp0 = Person->new();
$person_name0 = "name1";
$person_age0 = 1;
$person_address0 = "address1";
$blog_person_tmp0->set($person_name0,$person_age0,$person_address0);
$blog_person0 = $blog_person_tmp0;
$blog_person_tmp1 = Person->new();
$person_name1 = "name1";
$person_age1 = 1;
$person_address1 = "address2";
```

```
$blog_person_tmp1->set($person_name1,$person_age1,$person_address1);
$blog_person1 = $blog_person_tmp1;
$blog_nickname0 = "nickname1";
$blog[0] = {"nickname" => $blog_nickname0, "person" => $blog_person0};
$blog_nickname1 = "nickname2";
$blog[1] = {"nickname" => $blog_nickname1, "person" => $blog_person1};
$nickname = "nickname1";
$person_tmp = Person->new();
$person_name0 = "name3";
$person_age0 = 0;
$person_address0 = "address4";
$person_tmp->set($person_name0,$person_age0,$person_address0);
$person = $person_tmp;
...
```

- Update_SP_10_CTCASE

```
...
@blog = ();
$nickname = "nickname3";
$person_tmp = Person->new();
$person_name0 = "name1";
$person_age0 = 0;
$person_address0 = "address2";
$person_tmp->set($person_name0,$person_age0,$person_address0);
$person = $person_tmp;
...
```

- Update_SP_14_CTCASE

```
...
@blog = ();
$blog_person_tmp0 = Person->new();
$person_name0 = "name1";
$person_age0 = 1;
$person_address0 = "address1";
$blog_person_tmp0->set($person_name0,$person_age0,$person_address0);
$blog_person0 = $blog_person_tmp0;
$blog_nickname0 = "nickname2";
$blog[0] = {"nickname" => $blog_nickname0, "person" => $blog_person0};
$nickname = "nickname1";
$person_tmp = Person->new();
$person_name0 = "name2";
$person_age0 = 0;
$person_address0 = "address3";
$person_tmp->set($person_name0,$person_age0,$person_address0);
$person = $person_tmp;
...
```

F.5.3. Operación Delete

Ejecutando los comandos

```
loadrefrule peoplesBlogDelete.atcal
refine Delete_VIS to test peoplesBlog.pl implemented in perl with
    peoplesBlogDelete
```

se generan

- Delete_SP_3_CTCASE

```
...
@blog = ();
$blog_person_tmp0 = Person->new();
$person_name0 = "name1";
$person_age0 = 1;
$person_address0 = "address2";
$blog_person_tmp0->set($person_name0,$person_age0,$person_address0);
$blog_person0 = $blog_person_tmp0;
$blog_nickname0 = "nickname1";
$blog[0] = {"nickname" => $blog_nickname0, "person" => $blog_person0};
$nickname = "nickname1";
deleteUser($nickname);
```

```
$state->{'blog'} =  \@blog;
$state->{'nickname'} = $nickname;
__fastest_dump($state);
```

- Delete_SP_4_CTCASE

```
...
@blog = ();
$blog_person_tmp0 = Person->new();
$person_name0 = "name1";
$person_age0 = 1;
$person_address0 = "address1";
$blog_person_tmp0->set($person_name0,$person_age0,$person_address0);
$blog_person0 = $blog_person_tmp0;
$blog_person_tmp1 = Person->new();
$person_name1 = "name1";
$person_age1 = 1;
$person_address1 = "address2";
$blog_person_tmp1->set($person_name1,$person_age1,$person_address1);
$blog_person1 = $blog_person_tmp1;
$blog_nickname0 = "nickname1";
$blog[0] = {"nickname" => $blog_nickname0, "person" => $blog_person0};
$blog_nickname1 = "nickname2";
$blog[1] = {"nickname" => $blog_nickname1, "person" => $blog_person1};
$nickname = "nickname1";
...
```

- Delete_SP_8_CTCASE

```
...
@blog = ();
$nickname = "nickname1";
...
```

- Delete_SP_12_CTCASE

```
...
@blog = ();
$blog_person_tmp0 = Person->new();
$person_name0 = "name1";
$person_age0 = 1;
$person_address0 = "address1";
$blog_person_tmp0->set($person_name0,$person_age0,$person_address0);
$blog_person0 = $blog_person_tmp0;
$blog_nickname0 = "nickname2";
$blog[0] = {"nickname" => $blog_nickname0, "person" => $blog_person0};
$nickname = "nickname1";
...
```

Apéndice G

Big Integer

Los archivos de este caso de estudio pueden encontrarse en este link.

G.1. Especificación Z

Se trata de un sistema simple que considera las siguientes operaciones matemáticas:

- Suma
- Resta
- Multipliación
- División

En este caso no es necesario mantener ningún esquema de estado debido a la simplicidad del caso de estudio. Solo son necesarias variables de entrada y una de salida.

PlusOperation

$$in1? : \mathbb{Z}$$
$$in2? : \mathbb{Z}$$
$$out! : \mathbb{Z}$$

$$out! = in1? + in2?$$

MinusOperation

$$in1? : \mathbb{Z}$$
$$in2? : \mathbb{Z}$$
$$out! : \mathbb{Z}$$

$$out! = in1? - in2?$$

MultiplicationOperation

$$in1? : \mathbb{Z}$$
$$in2? : \mathbb{Z}$$
$$out! : \mathbb{Z}$$

$$out! = in1? * in2?$$

DivisionOperationOk

$$in1? : \mathbb{Z}$$
$$in2? : \mathbb{Z}$$
$$out! : \mathbb{Z}$$

$$in2? \neq 0$$
$$out! = in1? \ \mathsf{div} \ in2?$$

DivisionOperationError

$in1? : \mathbb{Z}$
$in2? : \mathbb{Z}$
$out! : \mathbb{Z}$

$in2? = 0$

$DivisionOperation == DivisionOperationOk \lor DivisionOperationError$

G.2. Casos Abstractos de prueba

Para generar los casos abstractos de prueba se ejecutaron los comandos

```
loadspec bigInteger.tex
selop PlusOperation
addtactic PlusOperation SP + in1? + in2?
selop MinusOperation
addtactic MinusOperation SP - in1? - in2?
selop MultiplicationOperation
addtactic MultiplicationOperation SP * in1? * in2?
selop DivisionOperation
addtactic DivisionOperation SP \div in1? \div in2?
genalltt
genalltca
```

y se obtuvieron los siguientes casos:

G.2.1. PlusOperation

PlusOperation_SP_1_TCASE

PlusOperation_SP_1

$in1? = -2147483648$
$in2? = -2147483647$

PlusOperation_SP_2_TCASE

PlusOperation_SP_2

$in1? = -2147483648$
$in2? = -2147483648$

PlusOperation_SP_3_TCASE

PlusOperation_SP_3

$in1? = -2147483647$
$in2? = -2147483648$

PlusOperation_SP_4_TCASE

PlusOperation_SP_4

$in1? = -2147483648$
$in2? = 0$

PlusOperation_SP_5_TCASE

PlusOperation_SP_5

$in1? = -2147483648$
$in2? = 1$

PlusOperation_SP_6_TCASE

PlusOperation_SP_6

$in1? = 0$
$in2? = -2147483648$

PlusOperation_SP_7_TCASE

PlusOperation_SP_7

$in1? = 0$
$in2? = 0$

PlusOperation_SP_8_TCASE

PlusOperation_SP_8

$in1? = 0$
$in2? = 1$

PlusOperation_SP_9_TCASE

PlusOperation_SP_9

$in1? = 1$
$in2? = 2$

PlusOperation_SP_10_TCASE

PlusOperation_SP_10

$in1? = 1$
$in2? = 1$

PlusOperation_SP_11_TCASE

PlusOperation_SP_11

$in1? = 2$
$in2? = 1$

G.2.2. MinusOperation

Debido a que los predicados del esquema de la operación `MinusOperation` son los mismos que los de la operación `PlusOperation`, los casos abstractos generados son los mismos excepto el nombre de los mismos que tienen el formato MinusOperation_ SP_ N_ TCASE, donde N es el número de caso en cuestión. Por lo tanto, el autor considera que no es necesario volver a incluirlos en esta sección.

G.2.3. MultiplicationOperation

Por la misma razón que en la sección G.2.2, los casos abstractos de prueba serán los mismos que para la operación `PlusOperation` y no se incluyen.

G.2.4. DivisionOperation

DivisionOperation_SP_1_TCASE

DivisionOperation_SP_1

$in1? = -2147483648$
$in2? = -2147483647$

DivisionOperation_SP_2_TCASE

DivisionOperation_SP_2

$in1? = -2147483648$
$in2? = -2147483648$

DivisionOperation_SP_3_TCASE

DivisionOperation_SP_3

$in1? = -2147483647$
$in2? = -2147483648$

DivisionOperation_SP_5_TCASE

DivisionOperation_SP_5

$in1? = -2147483648$
$in2? = 1$

DivisionOperation_SP_6_TCASE

DivisionOperation_SP_6

$in1? = 0$
$in2? = -2147483648$

DivisionOperation_SP_8_TCASE

DivisionOperation_SP_8

$in1? = 0$
$in2? = 1$

DivisionOperation_SP_9_TCASE

DivisionOperation_SP_9

$in1? = 1$
$in2? = 2$

DivisionOperation_SP_10_TCASE

DivisionOperation_SP_10

$in1? = 1$
$in2? = 1$

DivisionOperation_SP_11_TCASE

DivisionOperation_SP_11

$in1? = 2$
$in2? = 1$

DivisionOperation_SP_15_TCASE

DivisionOperation_SP_15

$in1? = -2147483648$
$in2? = 0$

DivisionOperation_SP_18_TCASE

DivisionOperation_SP_18

$in1? = 0$
$in2? = 0$

G.3. Implementación

```
#!/usr/bin/perl
use feature qw(say);
use YAML::XS;
use Data::Dumper;

sub __fastest_dump {
  open F, '>', 'state.yml';
  print F Dumper ( @_ );
  close F;
}

sub plus {
  $in1 = $_[0];
  $in2 = $_[1];

  return $in1 + $in2;
}

sub minus {
  $in1 = $_[0];
  $in2 = $_[1];

  return $in1 - $in2;
}

sub multiplication {
  $in1 = $_[0];
  $in2 = $_[1];

  return $in1 * $in2;
}

sub division {
  $in1 = $_[0];
  $in2 = $_[1];

  if ($in2 == 0) {
    say('Error de división por 0')
  } else {
    return $in1 / $in2;
  }
}
```

G.4. Leyes de refinamiento

En este caso de estudio se decide mostrar todas las reglas de refinamiento que se utilizarán para generar los casos concretos de prueba ya que se presenta una característica del lenguaje ATCAL, que, si bien es un concepto sencillo, es muy útil a la hora de reutilizar leyes de refinamiento definidas en reglas previamente cargadas.

- bigIntegerP

```
@RRULE bigIntegerP

@LAWS
  I1 : in1? ==> in1 AS INT;
  I2 : in2? ==> in2 AS INT;

out <== @UUT plus(in1, in2) AS INT;
```

- bigIntegerMI

```
@RRULE bigIntegerMI

@LAWS
  bigIntegerP.I1;
  bigIntegerP.I2;

out <== @UUT minus(in1, in2) AS INT;
```

- bigIntegerMU

```
@RRULE bigIntegerMU

@LAWS
  bigIntegerP.I1;
  bigIntegerP.I2;

out <== @UUT multiplication(in1, in2) AS INT;
```

- bigIntegerD

```
@RRULE bigIntegerD

@LAWS
  bigIntegerP.@LAWS;

out <== @UUT division(in1, in2) AS INT;
```

Como se puede observar, en la regla `bigIntegerP` se definen dos leyes de refinamiento donde se especifica que tanto `in1?` como `in2?` se refinan como un `INT` y para ambas se define un identificador (`I1` e `I2`) que podrá ser utilizado para referenciar la ley desde otra regla. Una vez definida esta regla es posible reutilizar las distintas leyes de refinamiento que contiene en otras reglas de refinamiento, como es el caso de las reglas `bigIntegerMI` y `bigIntegerMU` que reutilizan de manera independiente a `I1` e `I2`. Por otro lado, la regla `bigIntegerD` también reutiliza las leyes definidas en `bigIntegerP`, pero en este caso se está indicando que deben incluirse todas las leyes definidas en `bigIntegerP`.[1]

G.5. Casos concretos de prueba en Perl

G.5.1. Operación PlusOperation

Ejecutando los comandos

```
loadrefrule bigIntegerP.atcal
refine PlusOperation_VIS to test bigInteger.pl implemented in perl with
    bigIntegerP
```

se generan

- PlusOperation_SP_1_CTCASE

```
#!/usr/bin/perl
use feature qw(say);
use YAML::XS;
use Data::Dumper;

sub __fastest_dump {
  open F, '>', 'state.yml';
  print F Dumper ( @_ );
  close F;
}

sub plus {
  # Código que implementa plus
}

sub minus {
  # Código que implementa minus
}

sub multiplication {
  # Código que implementa multiplication
}

sub division {
  # Código que implementa division
```

[1] Es importante destacar que en este caso sencillo las leyes `bigIntegerP.I1` y `bigIntegerP.I2` son equivalentes a `bigIntegerP.@LAWS`, pero con esta última forma de inclusión se consideran todas las leyes la regla en cuestión, mientras que de la primer forma solo se incluye una ley en particular

```
}

$in1 = -2147483648;
$in2 = -2147483647;
$out = plus($in1,$in2);
$state->{'in2'} = $in2;
$state->{'in1'} = $in1;
$state->{'out'} = $out;
__fastest_dump($state);
```

- PlusOperation_SP_2_CTCASE

```
...
$in1 = -2147483648;
$in2 = -2147483648;
...
```

- PlusOperation_SP_3_CTCASE

```
...
$in1 = -2147483647;
$in2 = -2147483648;
...
```

- PlusOperation_SP_4_CTCASE

```
...
$in1 = -2147483648;
$in2 = 0;
...
```

- PlusOperation_SP_5_CTCASE

```
...
$in1 = -2147483648;
$in2 = 1;
...
```

- PlusOperation_SP_6_CTCASE

```
...
$in1 = 0;
$in2 = -2147483648;
...
```

- PlusOperation_SP_7_CTCASE

```
...
$in1 = 0;
$in2 = 0;
...
```

- PlusOperation_SP_8_CTCASE

```
...
$in1 = 0;
$in2 = 1;
...
```

- PlusOperation_SP_9_CTCASE

```
...
$in1 = 1;
$in2 = 2;
...
```

- PlusOperation_SP_10_CTCASE

```
...
$in1 = 1;
$in2 = 1;
...
```

- PlusOperation_SP_11_CTCASE

```
...
$in1 = 2;
$in2 = 1;
...
```

G.5.2. Operaciones MinusOperation y MultiplicationOperation

Por la misma razón que se menciona en la sección G.2.2, los casos de prueba concretos para las operaciones `MinusOperation` y `MultiplicationOperation` son idénticos a los de `PlusOperation` salvo por la llamada al método correspondiente de cada operación.

G.5.3. Operación DivisionOperation

Ejecutando los comandos

```
loadrefrule bigIntegerD.atcal
refine DivisionOperation_VIS to test bigInteger.pl implemented in perl with
    bigIntegerD
```

se generan los mismos casos que en G.5.1 exceptuando los casos `DivisionOperation_SP_4_CTCASE` y `DivisionOperation_SP_7_CTCASE` y se suman otros casos adicionales. A continuación se detallan los agregados:

- DivisionOperation_SP_15_CTCASE

```
...
$in1 = -2147483648;
$in2 = 0;
$out = division($in1,$in2);
$state->{'in2'} = $in2;
$state->{'in1'} = $in1;
$state->{'out'} = $out;
__fastest_dump($state);
```

- DivisionOperation_SP_18_CTCASE

```
...
$in1 = 0;
$in2 = 0;
...
```

Bibliografía

[1] Jean-Raymond Abrial, S Schuman, and Bertrand Meyer. A specification language. on the construction of programs, mcnaughten, r., and mckeag, r, 1980.

[2] Pablo Albertengo. Poda de árboles de testing a través de la detección de contradicciones matemáticas. B.S. thesis, Facultad de Ciencias Exactas, Ingeniería y Agrimensura. Universidad Nacional . . . , 2011.

[3] Axel Belinfante, Jan Feenstra, René G de Vries, Jan Tretmans, Nicolae Goga, Loe Feijs, Sjouke Mauw, and Lex Heerink. Formal test automation: A simple experiment. In *Testing of Communicating Systems*, pages 179–196. Springer, 1999.

[4] Sebastian Benz. Aspectt: aspect-oriented test case instantiation. In *Proceedings of the 7th international conference on Aspect-oriented software development*, pages 1–12, 2008.

[5] Fabrice Bouquet and Bruno Legeard. Reification of executable test scripts in formal specification-based test generation: The java card transaction mechanism case study. In *International Symposium of Formal Methods Europe*, pages 778–795. Springer, 2003.

[6] Khusbu Bubna. An approach for generating concrete test cases utilizing formal specifications of web applications. *International Journal of Information and Electronics Engineering*, 6(3):166, 2016.

[7] Khusbu Bubna and Sujit Chakrabarti. End to end specification based test generation of web applications. In *ENASE*, pages 296–302, 2016.

[8] Khusbu Bubna and Sujit Kumar Chakrabarti. Act (abstract to concrete tests)-a tool for generating concrete test cases from formal specification of web applications. In *ModSym+ SAAAS@ ISEC*, pages 16–22, 2016.

[9] Pablo D Coca. Refinamiento a java de casos de prueba abstractos generados por fastest, un sistema de testing automatizado. B.S. thesis, Facultad de Ciencias Exactas, Ingeniería y Agrimensura. Universidad Nacional . . . , 2010.

[10] Maximiliano Cristiá, Diego A. Hollmann, Pablo Albertengo, Claudia S. Frydman, and Pablo Rodríguez Monetti. A language for test case refinement in the test template framework. In *Formal Methods and Software Engineering - 13th International Conference on Formal Engineering Methods, ICFEM 2011, Durham, UK, October 26-28, 2011. Proceedings*, pages 601–616, 2011.

[11] Maximiliano Cristiá and Pablo Rodríguez Monetti. Implementing and applying the stocks-carrington framework for model-based testing. In *International Conference on Formal Engineering Methods*, pages 167–185. Springer, 2009.

[12] M. Cristiá. Tácticas de testing. `http://www.fceia.unr.edu.ar/ingsoft/testing-func-a.pdf`.

[13] Joaquín Cuenca. Definición de estrategias para la aplicación automática de tácticas de testing en el marco del ttf y fastest. B.S. thesis, Facultad de Ciencias Exactas, Ingeniería y Agrimensura. Universidad Nacional . . . , 2014.

[14] Leonardo De Moura and Nikolaj Bjørner. Satisfiability modulo theories: introduction and applications. *Communications of the ACM*, 54(9):69–77, 2011.

[15] X Devroey and G Perrouin. Variability intensive system behavioural testing framework (vibes). *URL https://projects. info. unamur. be/vibes*, 2014.

[16] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design patterns: Abstraction and reuse of object-oriented design. In *European Conference on Object-Oriented Programming*, pages 305–317. Springer, 1993.

[17] Ben Hamilton, Terence Parr, and Steinbacher Bradley. Antlr grammar syntax. `https://github.com/antlr/antlr4/blob/master/doc/lexer-rules.md`.

[18] Diego Ariel Hollmann. Tcrl - refinamiento de casos de prueba para sistema de testing automatizado. 2009.

[19] Hans-Martin Hörcher and Jan Peleska. Using formal specifications to support software testing. *Software Quality Journal*, 4(4):309–327, 1995.

[20] Claude Jard and Thierry Jéron. Tgv: theory, principles and algorithms. *International Journal on Software Tools for Technology Transfer*, 7(4):297–315, 2005.

[21] JUnit. Junit 4 / about. `https://junit.org/junit4/`, October 2013.

[22] James C King. Symbolic execution and program testing. *Communications of the ACM*, 19(7):385–394, 1976.

[23] Joaquín Mesuro. Extensión del ttf a testing de integración. B.S. thesis, Facultad de Ciencias Exactas, Ingeniería y Agrimensura. Universidad Nacional . . . , 2015.

[24] Oracle. Java programming language. `https://www.java.com/es/`.

[25] Apache org. Maven. `https://maven.apache.org/what-is-maven.html`.

[26] Python Org. Python programming language. `https://www.python.org/`.

[27] Terence Parr. *The definitive ANTLR 4 reference*. Pragmatic Bookshelf, 2013.

[28] Perl.org. Perl web site. `https://www.perl.org/`. 2002-2019.

[29] Pablo Rodríguez Monetti. Fastest: Automatizando el testing de software. B.S. thesis, Departamento de Ciencias de la Computación; Facultad de Ciencias Exactas . . . , 2009.

[30] QSpin SA. Qspin tailored automated system test environment. `https://github.com/qspin/qtaste`.

[31] SeleniumHQ. Selenium remoto control. `https://www.seleniumhq.org/projects/remote-control/`.

[32] Phil Stocks and David Carrington. A framework for specification-based testing. *IEEE Transactions on software Engineering*, 22(11):777–793, 1996.

[33] Philip Alan Stocks. *Applying formal methods to software testing*. PhD thesis, University of Queensland, 1993.

[34] Julián Tomasi. Generación de lenguaje natural a partir de clases de prueba del test template framework. B.S. thesis, Departamento de Ciencias de la Computación; Facultad de Ciencias Exactas . . . .

[35] Machiel Van Der Bijl, Arend Rensink, and Jan Tretmans. Atomic action refinement in model based testing. *CTIT Technical Report Series*, (LNCS4549/TR-CTIT-07-64), 2007.

[36] Jeremy Vanhecke, Xavier Devroey, and Gilles Perrouin. Abscon: a test concretizer for model-based testing. In *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 15–22. IEEE, 2019.

[37] Andrius Velykis. Community z tools. `http://czt.sourceforge.net/`.

[38] Wikipedia. C programming language. `https://en.wikipedia.org/wiki/C_(programming_language)`.

[39] Wikipedia. Satisfiability modulo theories. https://en.wikipedia.org/wiki/Satisfiability_modulo_theories.

[40] Wikipedia. Symbolic model checking. https://en.wikipedia.org/wiki/Model_checking#Symbolic_model_checking.

[41] Wikipedia.org. Backus-naur form. https://en.wikipedia.org/wiki/Backus%E2%80%93Naur_form.