Análisis de especificaciones Z (con Z/EVES)

Análisis

- La idea es verificar propiedades a partir de la especificación o descubrir que propiedades que se creían válidas en realidad no lo son.
- Existen ciertas verificaciones estándar y otras que dependen de cada especificación o del lenguaje utilizado o del interés del desarrollador.
- También, algunos teoremas pueden usarse como documentación de ciertas propiedades del modelo que de otra forma quedarían implícitas.

El asistente de pruebas Z/EVES

- Mientras Z es un lenguaje tipado, Z/EVES está basado en el asistente de pruebas EVES que no trabaja con un lenguaje tipado.
- Por lo tanto el sistema traduce los términos Z al lenguaje de EVES y viceversa.
- Esto ocasiona al menos dos problemas:
  - ciertas traducciones no son posibles lo que hace que ciertas pruebas no se puedan realizar
  - ciertas deducciones que el sistema de tipos de Z garantiza no son posibles en EVES.

El asistente de pruebas... (2)

- Como muchos asistentes Z/EVES puede utilizar automáticamente teoremas ya demostrados, definiciones y reglas de reescritura.
- Por defecto los únicos resultados que Z/EVES aplica de forma automática son un subconjunto de los resultados incluidos en el *Z/EVES 2.0 Mathematical Toolkit*.
- El toolkit incluye muchos otros resultados que no son usados automáticamente pero que pueden ser invocados por el ingeniero.
- Es muy conveniente revisar el *toolkit*.

Ejemplo: un editor muy simple

$[CHAR] \qquad TEXT == \text{seq}\ CHAR$

$$
\begin{array}{|l}
maxsize : \mathbb{N} \\
\hline
maxsize \leqslant 32000
\end{array}
$$

Editor

$$
\begin{array}{|l}
left, right : TEXT \\
\hline
\#(left \frown right) \leqslant maxsize \\
\hline
\end{array}
$$

Init

$$
\begin{array}{|l}
Editor \\
\hline
left = right = \langle\rangle \\
\hline
\end{array}
$$

El teorema de inicialización

- Si la especificación es consistente el estado inicial debe satisfacer el invariante de estado:

$$\exists\ \text{SystemState} \cdot \text{InitSystem}$$

$\exists Editor \cdot Init$

$\exists left, right : TEXT \mid \#(left \frown right) \leqslant maxsize \cdot left = right = \langle\rangle$

$\exists left, right : TEXT \mid \#(left \frown right) \leqslant maxsize \wedge left = right = \langle\rangle$

$\#(\langle\rangle \frown \langle\rangle) \leqslant maxsize$

$0 \leqslant maxsize$

En Z/EVES

theorem *EditorInit*

$\exists Editor \bullet Init$

proof of *EditorInit*

prove by reduce $\qquad 0 \leqslant maxsize \wedge true$

Z/EVES no es capaz de deducir que *maxsize* es mayor o igaul a cero. Eso está codificado en un axioma (incluido automáticamentepor Z/EVES) que "tipa" a *maxsize* y en la definición de $\mathbb{N}$. El problema es que Z/EVES no usa automáticamente un axioma ni expande la definición de $\mathbb{N}$.

Este comando aplica una combinación usual de otros comandos de prueba (*prenex*, *rearrange*, *equality*, *substitution* y *reduce*), hasta que la prueba finaliza (*true*) o no hay más progreso.

Una prueba más detallada

proof of *EditorInit*

invoke	$\exists Editor \bullet left \in \text{seq}\ CHAR$ $\wedge\ right \in \text{seq}\ CHAR$ $\wedge\ \#(left \frown right) \leqslant maxsize$ $\wedge\ left = right$ $\wedge\ right = \langle\rangle$
rewrite	$Editor[left := \langle\rangle, right := \langle\rangle] \wedge 0 \quad maxsize$
invoke	$\langle\rangle \in \text{seq}\ CHAR \wedge \langle\rangle \in \text{seq}\ CHAR$ $\wedge\ \#(\langle\rangle \frown \langle\rangle) \leqslant maxsize \wedge 0 \leqslant maxsize$
reduce	$0 \leqslant maxsize \wedge true$
use *maxsize$declaration*	$maxsize \in \mathbb{N} \Rightarrow 0 \leqslant maxsize$
invoke	$maxsize \in \{ n : \mathbb{Z} \mid n \geqslant 0 \} \Rightarrow 0 \leqslant maxsize$
reduce	$true$

Otra prueba para *EditorInit*

theorem grule *maxsizeBound* //regla de suposición

$0 \leqslant maxsize$

proof of *maxsizeBound*

use *maxsize$declaration*	**if** $maxsize \in \mathbb{N}$ **then** $0 \leqslant maxsize$ **else** *true*
invoke	**if** $maxsize \in \{ n: \mathbb{Z} \mid n \geqslant 0 \}$
	then $0 \leqslant maxsize$
	else *true*
reduce	*true*

Antes de poder usar la regla deben grabar

theorem *EditorInit*

$\exists Editor \bullet Init$

proof of *EditorInit*

prove by reduce	*true*

Otra prueba más para *EditorInit*

Sin definir la grule *maxSizeBound* se puede probar de esta forma:

proof of *EditorInit*

use *maxsize$declaration*	$maxsize \in \mathbb{N} \Rightarrow \exists Editor \bullet Init$
invoke $\mathbb{N}$	$maxsize \in \{ n: \mathbb{Z} \mid n \geqslant 0 \}$
	$\Rightarrow \exists Editor \bullet Init$
prove by reduce	*true*

`invoke` *name*

- Si *name* es el nombre de un esquema o el nombre de un término introducido en una definición, todas las apariciones del nombre en el gol actual son reemplazadas por su definición.
- Si no se especifica ningún nombre todos los nombres de definiciones y esquemas son invocados.
- `invoke predicate` *name*

`rewrite`

Enteros, igualdad, lógica proposicional, tautologías

- Al re−escribir, Z/EVES simplifica y aplica reglas de re−escritura siempre que sea posible.
- Una regla de re−escritura es un teorema de la forma *Condición* ⇒ *Patrón* = *Reemplazo*.
- El toolkit de Z está lleno de reglas de re−escritura:

theorem `disabled rule` *capSubsetLeft[X]*

$S \subseteq[X]\ T \Rightarrow S \cap T = S$

theorem `rule` *eqTuple2*

$(x,y) = (x',y') \Leftrightarrow x = x' \wedge y = y'$

`apply` *theorem−name*

- Las reglas de re−escritura deshabiltadas pueden aplicarse mediante el comando `apply`
  - También seleccionando una expresión, pulsando el botón derecho del mouse y examinando la opción "**Apply to expresion**".
- Las reglas habilitadas son aplicadas automáticamente por el asistente de pruebas.
- `with enabled (`*lista−de−reglas*`) rewrite`
- `apply` *theorem−name* `to predicate` *predicate*

`reduce` y `prove`

- Al reducir, Z/EVES simplifica y re−escribe, y si una subfórmula es un esquema o el nombre de una abreviatura, la subfórmula será reemplazada por la definición y el resultado será reducido nuevamente.
- El comando `prove` es muy similar pero no efectúa expansiones de esquemas o definiciones.

use theorem−name ···

- La sintaxis general es algo compleja pues se deben instanciar los parámetros y variables del teorema a ser usado.
- Por ejemplo si tenemos:

theorem `rule` *inDom [X, Y]*

$\forall R: X \leftrightarrow Y \bullet x \in \mathrm{dom}\ R \Leftrightarrow (\exists y: Y \bullet (x, y) \in R)$

y estamos probando un teorema sobre $f{:}\mathbb{N}\leftrightarrow\mathbb{N}$:

`use` *inDom* [ℕ,ℕ] [*R*:=*f*]

use theorem−name ··· (2)

- El teorema usado es agregado como hipótesis del gol actual de manera que los comandos `simplify`, `reduce` o `rewrite` lo usarán para hacer avanzar la prueba.
- Si el gol, Q, no es una implicación entonces "`use` A" lo transforma en $A \Rightarrow Q$.
- Si el gol es la implicación $P \Rightarrow Q$, entonces "`use` A" lo transforma en $A \wedge P \Rightarrow Q$.

`split` predicado

- Si el gol es *Q*, el comando `split` *P* lo transforma en *if P then Q else Q*.
- Obviamente, se usa para pruebas por casos.
- Con los comandos `cases` y `next` Z/EVES considera cada uno de los casos necesarios.
- Ejemplo: ver prueba de *SCInterfacePI*.

Errores de dominio

- El sistema de tipos de Z no es tan poderoso como para garantizar que todas las expresiones sean *significativas*.
  - 1 div 0, *max* $\mathbb{Z}$, # $\mathbb{N}$, etc.
- Por este motivo, Z/EVES verifica cada párrafo y determina si es necesaria una *comprobación de dominio*, en cuyo caso plantea una *obligación de prueba* que debe ser *descargada*.

`instantiate` [var==expr,...,var==expr]

- Este comando se utiliza para instanciar una o más variables cuantificadas.
  - Las variables deben estar ligadas al mismo cuantificador.
- Al instanciar la variable *x* con *e* convierte el predicado $\exists\, x{:}S \bullet P(x)$ en $(e \in S \land P(e)) \lor \exists\, x{:}S \bullet P(x)$ para preservar la equivalencia entre los dos goles.
- Ejemplo: *setComprEqualPfun*.

Ejemplo

La mayoría de las obligaciones de prueba provienen de expresiones donde intervienen aplicaciones de funciones parciales

Ejemplo

$f: \mathbb{Z} \pfun \mathbb{Z}$

$\forall z: \mathbb{Z} \bullet f\, z < 5$

proof of *Ejemplo$domainCheck*

$f \in \mathbb{Z} \pfun \mathbb{Z} \land z \in \mathbb{Z} \Rightarrow z \in \mathrm{dom}\, f$

Es imposible de probar.

EjemploCorr

$f: \mathbb{Z} \pfun \mathbb{Z}$

$\forall z: \mathbb{Z} \mid z \in \mathrm{dom}\, f \bullet f\, z < 5$

proof of *Ejemplo$domainCheck*

$f \in \mathbb{Z} \pfun \mathbb{Z} \land z \in \mathbb{Z} \land z \in \mathrm{dom}\, f$

$\Rightarrow z \in \mathrm{dom}\, f$

Se pueba facilmente con `simplify`

Satisfacción de esquemas

- Un error posible es definir un esquema cuyo predicado sea (siempre) falso, es decir un esquema *insatisfacible*.
- Para evitar ese error, se debe probar:

$$\exists\ Schema;\ Inputs?\ \bullet\ true$$

- Si el esquema erróneo corresponde:
  - al estado, entonces el sistema es imposible;
  - a una operación, entonces esta nunca puede ser invocada exitosamente.

Ascensor

sentido: SENTIDOS
puerta: ESTPUERTA

$sentido \neq Parado \Rightarrow puerta = Cerrada$

AbrirPuerta

$\Delta Ascensor$

$sentido = Arriba$
$puerta = Cerrada$
$puerta' = Abierta$
$sentido' = sentido$

theorem *AbrirPuertaInsat*
$\exists AbrirPuerta\ \bullet\ true$

proof of *AbrirPuertaInsat*
instantiate *sentido* == *Arriba*,
puerta == *Cerrada*,
sentido' == *Arriba*,
puerta' == *Abierta*

invoke

Instancia variables cuantificadas con valores constantes ya definidos.

AbrirPuerta[*puerta* := *Cerrada*,
puerta':= *Abierta*, *sentido* :=*Arriba*,
sentido' := *Arriba*]
$\vee\ (\exists AbrirPuerta\ \bullet\ true)$

$Arriba \neq Parado \Rightarrow Abierta = Cerrada$

Cálculo de precondiciones

- La precondición de una operación es un predicado que describe todos los estados de partida en los que la operación está definida.
- Así, la precondición sólo contiene variables de estado no primadas y variables de entrada.
- La precondición de una operación es:

$$\exists\ SystemState;\ Outputs!\ \bullet\ Op$$

o en Z/EVES,

$$\forall\ SystemState;\ Inputs?\ \bullet\ \mathrm{pre}\ Op$$

- Es conveniente documentar la precondición, *P*, de cada operación:

$$\forall\ SystemState;\ Inputs?\ |\ P\ \bullet\ \mathrm{pre}\ Op$$

FDoc

$miembros: DNI \rightarrowtail\!\!\!\!\!\!+ NAME;\ prohibidos: \mathbb{P}\ DNI$

$prohibidos \subseteq \mathrm{dom}\ miembros$

AddMember

$\Delta FDoc$
$candidato?: NAME$
$doc!: DNI$

$candidato? \notin \mathrm{ran}\ miembros$
$doc! \notin \mathrm{dom}\ miembros$
miembros' = *miembros* ∪ {(*doc!* Œ *candidato?*)}
$prohibidos' = prohibidos$

En Z/EVES

theorem *AddMemberPre*

$\forall FDoc; candidato?: NAME \bullet$ pre $AddMember$

proof of *AddMemberPre*

prove by reduce $\quad tipos \Rightarrow (\exists doc!: DNI$

$\bullet\ miembros \cup \{(doc!, candidato?)\}$

$\in DNI \rightarrowtail\!\!\!\!\rightarrow NAME$

$\land \neg\, candidato? \in \mathrm{ran}\ miembros$

$\land \neg\, doc! \in \mathrm{dom}\ miembros)$

apply *cupInPinj* $\quad$ *un predicado muy largo*

prove $\quad tipos \Rightarrow (\exists doc!: DNI \bullet$

$\neg\, candidato? \in \mathrm{ran}\ miembros$

$\land \neg\, doc! \in \mathrm{dom}\ miembros)$

Propiedades de un modelo

- Estas propiedades pueden haberse establecido en los requerimientos informales o pueden ser puntos clave de la especificación.

BanMember

$FDoc; mem?: DNI$

$mem? \in \mathrm{dom}\ miembros$

$prohibidos' = prohibidos \cup \{mem?\}$

$miembros' = miembros$

En Z/EVES

theorem *YaEstaProhibido*

$\forall mem?: DNI \bullet\ BanMember \land mem? \in prohibidos \Rightarrow \Xi FDoc$

proof of *YaEstaProhibido*

prove by reduce $\quad \ldots \land mem? \in prohibidos$

$\land\ prohibidos' = prohibidos \cup \{mem?\}$

$\Rightarrow prohibidos = prohibidos \cup \{mem?\}$

apply *cupSubsetRight to expression prohibidos* $\cup \{mem?\}$

$\ldots \land mem? \in prohibidos$

$\land\ prohibidos' = \mathbf{if}\ \{mem?\} \subseteq prohibidos$

$\mathbf{then}\ prohibidos\ \mathbf{else}\ prohibidos \cup \{mem?\}$

$\Rightarrow prohibidos = \mathbf{if}\ \{mem?\} \subseteq prohibidos$

$\mathbf{then}\ prohibidos\ \mathbf{else}\ prohibidos \cup \{mem?\}$

prove $\quad$ *true*

```
The teacher stands in front of the class
But the lesson plan he cant’t recall
The student’s eyes don’t perceive the lies
Bouncing off every fucking wall
His composture is well kept
I guess he fears playing the fool
The complacent students sit and listen to the
Bullshit that he learned in school
                Rage against the machine
```

FIN