Pruebas

Ma. Laura Cobo

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Argentina

Hacia donde avanzamos …

La idea ahora es … conectar JML con la lógica dinámica

Esta conexión lleva a **generar**, **entender** y **probar**.

A partir de las especificaciones JML se tiene pruebas de obligación en DL (lógica dinámica)

pure vs. assignable \nothing

assignable \nothing prohíbe los efectos colaterales.

Se diferencia del modificador **pure** en el hecho que:

- El modificador pure es global al método y además prohíbe la no-terminación y las excepciones.
- La cláusula assignable es local a un caso de especificación.
- El modificador no es utilizable es contextos particulares

Generación de "proof obligations"

Se generan fórmulas para el comportamiento normal del servicio

- Se inspeccionan los Assumed Invariants
- Hay que asegurarse de seleccionar el contrato con **modifies** `balance` (tengan en cuenta que **modifies** es sinónimo de **assignable** en JML)
- En el panel se muestra la "poof obligation" como un consecuente de lógica dinámica (DL)

Generación de “proof obligations”

Re-abriendo el browser que corresponde a las “proof obligations” se pueden generar los EnsuresPost PO para los comportamientos normales y excepcionales de los diferentes servicios

Traducción de JML a DL

A fin de realizar las "proof obligations" es necesario realizar una traducción. La misma sigue los siguientes pasos:

- Tratamiento para la traducción de expresiones aritméticas.
- Traducción de `this`.
- Identificación de la implementación de métodos
- Traducción de expresiones JML booleanas a fórmulas FOL
- Traducción de precondiciones
- Traducción de invariantes de clase
- Traducción de post-condiciones
- Almacenamientos de los atributos/campos `/old`
- Almacenamiento de parámetros actuales (antes de la invocación)
- Expresión de "excepciones no señalizadas"
- Unión de las partes …

Traducción de JML a DL

La idea es poder entender y leer mejor las pruebas

Los operadores aritméticos son reemplazados por operadores generalizados, así por ejemplo el operador + se reemplaza por "`javaAddInt`" si los operandos son enteros

En la traducción se agregan cast cuando es necesario para respetar la jerarquía de tipos. Así por ejemplo "0" se traduce como "`(jint)(0)`"

Traducción `this`

Las referencias a "`this`" se traducen como "`self`"

Si se tiene una clase como la siguiente:

```
public class MiClase {
  . . .
 private int atrib;
. . . }
```

La traducción es la siguiente:

`atrib`	`self.atrib`
`this.atrib`	`self.atrib`

métodos

Una llamada a un método en JML se ve como `metodo(argumentos)` la clave está en que el método pertenece a una clase que Key conoce. Esto se indica a través de `package.clase`

La traducción por lo tanto es:

`metodo(argumentos)@package.clase`

ejemplo

`Cargar(x)@tarjeta.TajetaDebito`

Ejecuta la implementación que provee la clase `tarjeta.TajetaDebito` para el método `Cargar(x)`

Expresiones booleanas

JML no separa las fórmulas lógicas en una categoría diferente. De esta manera la expresiones booleanas de Java están permitidas en las fórmulas JML

Al hacer la traducción a la lógica de KeY:

- Las fórmulas y las expresiones se vuelven conceptos separados.
- Las constantes de verdad true y false son fórmulas mientras que las constantes `TRUE, FALSE` son expresiones.
- Las fórmulas atómicas toman expresiones como argumentos
  - X – y < 5
  - b = `TRUE`

Función de traducción

$$
\begin{array}{lcl}
\mathcal{F}(\texttt{v}) & = & \texttt{v = TRUE} \\
\mathcal{F}(\texttt{f}) & = & \mathcal{T}(\texttt{f}) \texttt{ = TRUE} \\
\mathcal{F}(\texttt{m()}) & = & \mathcal{T}(\texttt{m})\texttt{() = TRUE} \\
\mathcal{F}(\texttt{!b\_0}) & = & \texttt{!}\mathcal{F}(\texttt{b\_0}) \\
\mathcal{F}(\texttt{b\_0 \&\& b\_1}) & = & \mathcal{F}(\texttt{b\_0}) \texttt{ \& } \mathcal{F}(\texttt{b\_1}) \\
\mathcal{F}(\texttt{b\_0 || b\_1}) & = & \mathcal{F}(\texttt{b\_0}) \texttt{ | } \mathcal{F}(\texttt{b\_1}) \\
\mathcal{F}(\texttt{b\_0 ==> b\_1}) & = & \mathcal{F}(\texttt{b\_0}) \texttt{ -> } \mathcal{F}(\texttt{b\_1}) \\
\mathcal{F}(\texttt{b\_0 <==> b\_1}) & = & \mathcal{F}(\texttt{b\_0}) \texttt{ <-> } \mathcal{F}(\texttt{b\_1}) \\
\mathcal{F}(\texttt{e\_0 == e\_1}) & = & \mathcal{E}(\texttt{e\_0}) \texttt{ = } \mathcal{E}(\texttt{e\_1}) \\
\mathcal{F}(\texttt{e\_0 != e\_1}) & = & \texttt{!}\mathcal{E}(\texttt{e\_0}) \texttt{ = } \mathcal{E}(\texttt{e\_1}) \\
\mathcal{F}(\texttt{e\_0 >= e\_1}) & = & \mathcal{E}(\texttt{e\_0}) \texttt{ >= } \mathcal{E}(\texttt{e\_1})
\end{array}
$$

v/f/m()	Variables boolean – atributos – métodos puros
b_0, b_1	Expresiones JML boolean
e_0, e_1	Expresiones Java

Departamento de Ciencias e Ingeniería de la Computación – Universidad Nacional del Sur, Argentina

Función de traducción

$$\mathcal{F}((\texttt{\textbackslash forall T x; e\_0})) = \texttt{\textbackslash forall T x; !x = null -> } \mathcal{F}(\texttt{e\_0})$$

$$\mathcal{F}((\texttt{\textbackslash exists T x; e\_0})) = \texttt{\textbackslash exists T x; !x = null \& } \mathcal{F}(\texttt{e\_0})$$

$$\mathcal{F}((\texttt{\textbackslash forall T x; e\_0; e\_1})) = \texttt{\textbackslash forall T x; !x = null \& } \mathcal{F}(\texttt{e\_0}) \texttt{ -> } \mathcal{F}(\texttt{e\_1})$$

$$\mathcal{F}((\texttt{\textbackslash exists T x; e\_0; e\_1})) = \texttt{\textbackslash exists T x; !x = null \& } \mathcal{F}(\texttt{e\_0}) \texttt{ \& } \mathcal{F}(\texttt{e\_1})$$

Traducción de precondiciones

Un contrato seleccionado con las siguientes precondiciones:

```
@ requires b_1;
@ . . .
@ requires b_n;
```

Se traduce como

$$\mathcal{PRE}(Contr) = \mathcal{F}(\text{b\_1}) \ \& \ \ldots \ \& \ \mathcal{F}(\text{b\_n})$$

Traducción de invariantes

Un invariante como el siguiente:

```
Class C {
. . .
//@ invariant inv_i;
```

Se traduce como

$$\mathcal{INV}(\texttt{inv\_i}) = \texttt{\textbackslash forall C o; ((o.<created> = TRUE \& !o = null) -> } \{\texttt{self:=o}\}\mathcal{F}(\texttt{inv\_i}))$$

Traducción de postcondiciones

Un contrato seleccionado con la siguientes postcondiciones:

```
@ ensures b_1;
@ . . .
@ ensures b_n;
```

Se traduce como

$$\mathcal{POST}(Contr) = \mathcal{F}(\text{b\_1}) \;\&\; \ldots \;\&\; \mathcal{F}(\text{b\_n})$$

Traducción de postcondiciones

Es necesario tener en cuenta la clausula

```
@ assignable lista_de-atributos_asignables;
```

Las expresiones en la post-condición deberán tener en cuenta:

$$\mathcal{E}(\texttt{\textbackslash result}) = \texttt{result}$$

$$\mathcal{E}(\texttt{\textbackslash old(e)}) = \mathcal{E}_{old}(\texttt{e})$$

$\mathcal{E}_{old}$ defined like $\mathcal{E}$, with the exception of:

$$\mathcal{E}_{old}(\texttt{e.f}) = \texttt{fAtPre}(\mathcal{E}_{old}(\texttt{e}))$$
$$\mathcal{E}_{old}(\texttt{f}) = \texttt{fAtPre(self)}$$

for f $\in$ *<assignable_fields>*

Almacenado Pre-Estado de un atributo

Dado un campo assignable de una clase

```
class C {
 . . .
 private tipo f;
```

Se traduce como (es un update cuantificado)

$$\mathcal{STORE}(\mathtt{f}) = \mathtt{\backslash for\ C\ o;\ fAtPre(o)\ :=\ o.f}$$

atributos asignables

Se traducen como una secuencia de updates paralelos

Expresando terminación normal

En DL se expresa la terminación normal de un método `m()`, es decir se indica que no lanza ninguna excepción

```
\<{ exc = null;
    try {
      m()@p.C;
    } catch (Throwable e) {
      exc = e;
    }
  }\> exc = null
```

Se asume que el cuerpo del método que se ejecuta es el que está en la clase `C` del paquete `p`.
Lo que esta indicado con azul son asignaciones Java, lo rojo son fórmulas

Expresando terminación excepcional

En DL se expresa la terminación anormal de un método `m()`, por el lanzamiento de una excepción

```
\<{ exc = null;
    try {
      m()@p.C;
    } catch (Throwable e) {
      exc = e;
    }
  }\> !exc = null & <exc has right type>
```

Se asume que el cuerpo del método que se ejecuta es el que está en la clase `C` del paquete `p`.

Proof obligation para un contrato normal

PO for a normal behavior contract *Contr* for void method m(), with chosen assumed invariants inv_1, ..., inv_n

```
==>
      INV(inv_1)
    & ...
    & INV(inv_n)
    & PRE(Contr)
 -> STORE(Contr)
      \<{ exc = null;
          try {
            m()@p.C;
          } catch (Throwable e) {
            exc = e;
          }
        }\> exc = null & POST(Contr)
```

Proof obligation: contrato normal con/sin terminación

PO for a normal behavior contract *Contr* for method m(), where *Contr* has clause **diverges true;**

==>

$\mathcal{INV}$(inv_1)
& ...
& $\mathcal{INV}$(inv_n)
& $\mathcal{PRE}$(*Contr*)
-> $\mathcal{STORE}$(*Contr*)

```
\[{ exc = null;
    try {
      m()@p.C;
    } catch (Throwable e) {
      exc = e;
    }
  }\] exc = null  & POST(Contr)
```

Comportamiento normal con resultado

PO for a normal behavior contract *Contr* for non-**void** method m(),

```
==>
      INV(inv_1)
    & ...
    & INV(inv_n)
    & PRE(Contr)
 -> STORE(Contr)
      \<{ exc = null;
          try {
            result = m()@p.C;
          } catch (Throwable e) {
            exc = e;
          }
        }\> exc = null & POST(Contr)
```