

Estudio y desarrollo de técnicas para el testing de programas concurrentes

CABS: traducción de C a ABS

Marco Antonio Garrido Rojo¹

18 de junio de 2018

¹<https://github.com/MaSteve/CABS-Slides>

Introducción: historia y origen

- En un principio, la mayoría de sistemas solo podían ejecutar una única secuencia de instrucciones a la vez.
- Con el avance de los ordenadores durante las últimas décadas del siglo pasado, se permite la ejecución simultánea de varios hilos de ejecución.
- Hoy en día es común el uso de la programación concurrente en la mayoría de aplicaciones y programas de todo ámbito.

Introducción: inconvenientes

- La ejecución paralela conlleva riesgos adicionales no presentes en los programas secuenciales.
- El principal problema que presenta es la presencia de una memoria compartida sobre la que los distintos hilos realizan modificaciones en cualquier momento.
- Asociado a esto, pueden ocurrir deadlocks, carreras de datos o comportamientos impredecibles.

Introducción: soluciones

- Podemos evitar que ocurran estableciendo restricciones mediante el uso de semáforos y cerrojos o implementaciones de algoritmos de más bajo nivel como el tie-breaker.
- El uso de estos mecanismos puede ser bastante **complejo**.

Introducción: soluciones

```
bool in1 = false, in2 = false;
int last = 1;
process CS1 {
    while (true) {
        last = 1; in1 = true;    /* entry protocol */
        while (in2 and last == 1) skip;
        critical section;
        in1 = false;            /* exit protocol */
        noncritical section;
    }
}
process CS2 {
    while (true) {
        last = 2; in2 = true;    /* entry protocol */
        while (in1 and last == 2) skip;
        critical section;
        in2 = false;            /* exit protocol */
        noncritical section;
    }
}
```

Figura: Algoritmo tie-breaker para dos procesos. (Andrews')

Introducción: interleavings

- En este terreno se sigue investigando para encontrar un modo definitivo que permita solucionar los problemas o detectarlos.
- Uno de los primeros asuntos estudiados es determinar el estado o estados finales a los que se llega ejecutando un programa concurrente.

Introducción: interleavings

```
1  int var;  
2  
3  proc f1() {  
4      var := 1;  
5  }  
6  
7  proc f2() {  
8      var := 2;  
9  }  
10
```

Introducción: interleavings

- Explosión exponencial en el número de interleavings en proporción al número de hilos y al número de instrucciones.
- No siempre hay una correlación con el número de estados finales.
- No todos los interleavings tienen la misma importancia y algunos de ellos son equivalentes entre sí.

Introducción: interleavings

```
1  int var1;  
2  int var2;  
3  
4  proc f1() {  
5      var1 := 1;    # 1  
6      var1 := 2;    # 2  
7  }  
8  
9  proc f2() {  
10     var2 := 2;    # 3  
11     var2 := 1;    # 4  
12 }  
13
```

Introducción: grano

- ¿Cuál es la atomicidad?
 - Asignaciones atómicas: grano grueso.
 - ¿Lectura de valores?: grano fino.
 - ¡Secciones de código enteras!: ...
- ¿De qué depende el grano? ¿Qué ocurre si no hay memoria compartida?

Introducción: actores

- Cada objeto ejecuta sus tareas de forma concurrente con respecto a las del resto de objetos.
- Una única tarea a la vez por objeto. El resto de tareas esperan en una cola cuyo orden no es determinable.
- El paso de mensajes indica qué método desea ejecutar un objeto (pudiendo ser uno propio o perteneciente a otro objeto).
- Dependiendo del tipo de llamada, una tarea puede dejar paso a otra si aún no cuenta con los valores necesarios para proseguir.
- Se trata de una concurrencia donde el scheduler es non-preemptive.

Introducción: ventajas

- Reducción notable del número de interleavings: importante cuando se aplican técnicas de testing sistemático.
- Presente en lenguajes como Erlang, Scala y **ABS**.
- Cuenta con una amplia cantidad de herramientas, entre ellas, **SYCO**, que implementa **DPOR** para obtener los posibles resultados finales de un programa, reduciendo los interleavings redundantes.

Introducción: objetivo

- Acercar las herramientas de ABS a un lenguaje que no sea de modelado, como C.
- Un lenguaje que permita recrear una concurrencia entre procesos a nivel de grano fino.
- Un lenguaje sencillo pero completo: **CABS**.

CABS: sintaxis

- Subconjunto de C con sintaxis para hebras.
- Tipos estáticos y arrays.

```
1 type global_var1;  
2 type global_var2;  
3  
4 type nombre_de_funcion(args...) {  
5     ...  
6     codigo  
7     ...  
8     return exp  
9 }
```

CABS: sintaxis

```
1 int array[10];
2 int global_var;
3
4 int f(int res, int
    arr[10]) {
5     int var;
6     var = 2 * res +
        arr[1];
7     return var;
8 }
```

```
9
10 int main() {
11     array[0] = 10;
12     array[1] = 5;
13     int res;
14     res = array[0] + 1;
15     global_var = f(res,
        array);
16     return 0;
17 }
```

CABS: sintaxis

```
1 int var;  
2  
3 int main() {  
4     thread f(1);  
5     thread f(2);  
6     return 0;  
7 }  
8  
9 void f(int value) {  
10     var = value;  
11 }
```


- Estado: $(\mathbf{G}, \mathbf{F}, \mathbf{RP})$.
- Ámbito global de variables (\mathbf{G}): asocia nombres de variables a valores de \mathbb{V} .
- Definición de funciones: $\mathbf{F}(\mathbf{func}) = (t, S_F, args_F, a_{ret})$.
- Lista de marcos de ejecución: $\mathbf{RP} \rightsquigarrow (local : s, S)$

$$\mathbf{init}(\varepsilon) = (\mathbf{nil}, \mathbf{nil}, [])$$
$$\mathbf{init}(\mathit{int} \text{ var}; \mathbf{P}) = (\mathbf{G} [\text{var} \mapsto 0], \mathbf{F}, \mathbf{RP})$$

donde $\mathbf{init}(\mathbf{P}) = (\mathbf{G}, \mathbf{F}, \mathbf{RP})$

$$\mathbf{init}(\mathit{bool} \text{ var}; \mathbf{P}) = (\mathbf{G} [\text{var} \mapsto \text{FALSE}], \mathbf{F}, \mathbf{RP})$$

donde $\mathbf{init}(\mathbf{P}) = (\mathbf{G}, \mathbf{F}, \mathbf{RP})$

$$\mathbf{init}(\mathit{int} \text{ func}(\arg)\{S; \text{return } a\} \mathbf{P}) = (\mathbf{G}, \mathbf{F} [\text{func} \mapsto (\mathit{int}, S, \arg, a)], \mathbf{RP})$$

donde $\mathbf{init}(\mathbf{P}) = (\mathbf{G}, \mathbf{F}, \mathbf{RP})$

$$\mathbf{init}(\mathit{bool} \text{ func}(\arg)\{S; \text{return } b\} \mathbf{P}) = (\mathbf{G}, \mathbf{F} [\text{func} \mapsto (\mathit{bool}, S, \arg, b)], \mathbf{RP})$$

donde $\mathbf{init}(\mathbf{P}) = (\mathbf{G}, \mathbf{F}, \mathbf{RP})$

$$\mathbf{init}(\mathit{void} \text{ func}(\arg)\{S\} \mathbf{P}) = (\mathbf{G}, \mathbf{F} [\text{func} \mapsto (\mathit{void}, S, \arg, \varepsilon)], \mathbf{RP})$$

donde $\mathbf{init}(\mathbf{P}) = (\mathbf{G}, \mathbf{F}, \mathbf{RP})$

$$\mathbf{start}((\mathbf{G}, \mathbf{F}, \mathbf{RP})) = (\mathbf{G}, \mathbf{F}, [(\mathbf{nil} : [], S)]) \text{ donde } \mathbf{F}(\mathit{main}) = (\mathit{int}, S, \arg, 0)$$

CABS: semántica de las **Aexp**

$$[\text{num}_{\mathcal{A}}] \frac{}{\langle n, (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s, S)) \rangle \rightarrow_{Aexp} \langle \mathcal{N} \llbracket n \rrbracket, (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s, S)) \rangle}$$

$$[\text{var}_{\mathcal{A}}^L] \frac{\text{local}(x) = v}{\langle x, (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (\text{local} : s, S)) \rangle \rightarrow_{Aexp} \langle v, (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (\text{local} : s, S)) \rangle}$$

$$[\text{var}_{\mathcal{A}}^G] \frac{G(x) = v \quad \text{local}(x) = \text{undef}}{\langle x, (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (\text{local} : s, S)) \rangle \rightarrow_{Aexp} \langle v, (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (\text{local} : s, S)) \rangle}$$

CABS: semántica de las **Aexp**

$$\left[\odot_{\mathcal{A}}^1 \right] \frac{\langle a_1, (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s, S)) \rangle \rightarrow_{Aexp} \langle a'_1, (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s', S')) \rangle}{\langle a_1 \odot a_2, (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s, S)) \rangle \rightarrow_{Aexp} \langle a'_1 \odot a_2, (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s', S')) \rangle}$$

$$\left[\odot_{\mathcal{A}}^2 \right] \frac{\langle a_2, (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s, S)) \rangle \rightarrow_{Aexp} \langle a'_2, (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s', S')) \rangle}{\langle v \odot a_2, (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s, S)) \rangle \rightarrow_{Aexp} \langle v \odot a'_2, (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s', S')) \rangle}$$

$$\left[\odot_{\mathcal{A}}^3 \right] \frac{}{\langle v_1 \odot v_2, (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s, S)) \rangle \rightarrow_{Aexp} \langle v_1 \odot_{\mathcal{N}} v_2, (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s, S)) \rangle}$$

CABS: semántica de las **Aexp**

$$[\text{unstack}_{\mathcal{A}}^1] \frac{\langle a, (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s, S)) \rangle \rightarrow_{Aexp} \langle a', (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s', S')) \rangle}{\langle \text{UNSTACK}(a), (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s, S)) \rangle \rightarrow_{Aexp} \langle \text{UNSTACK}(a'), (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s', S')) \rangle}$$

$$[\text{unstack}_{\mathcal{A}}^2] \frac{}{\langle \text{UNSTACK}(v), (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (local : s, S)) \rangle \rightarrow_{Aexp} \langle v, (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s, S)) \rangle}$$

$$[\text{call}_{\mathcal{A}}^1] \frac{\mathbf{F}(\text{func}) = (int, S_F, args_F, a) \quad \text{check_args}(args_F, args) \quad \text{eval}(args) = args'}{\langle \text{func}(args), (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (local : s, S)) \rangle \rightarrow_{Aexp} \langle \text{func}(args'), (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (local : s, S)) \rangle}$$

$$[\text{call}_{\mathcal{A}}^2] \frac{\mathbf{F}(\text{func}) = (int, S_F, args_F, a) \quad \text{check_args}(args_F, args) \quad \text{eval}(args) = v_1 : \dots : v_n}{\langle \text{func}(args), (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (local : s, S)) \rangle \rightarrow_{Aexp} \langle \text{UNSTACK}(a), (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (\text{nil} [args \mapsto v_1 : \dots : v_n] : local : s, S_F; S)) \rangle}$$

CABS: semántica de las asignaciones

$$[\text{ass}_C^1] \frac{\langle a, (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (\text{local} : s, \varepsilon)) \rangle \rightarrow_{Aexp} \langle a', (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s', S_A)) \rangle}{(\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (\text{local} : s, \text{var} = a; S)) \rightarrow (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s', S_A; \text{var} = a'; S))}$$

$$[\text{ass}_C^2] \frac{\text{is_local}(\text{var})}{(\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (\text{local} : s, \text{var} = v; S)) \rightarrow (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (\text{local} [\text{var} \mapsto v] : s, S))}$$

$$[\text{ass}_C^3] \frac{\text{is_global}(\text{var})}{(\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (\text{local} : s, \text{var} = v; S)) \rightarrow (\mathbf{G} [\text{var} \mapsto v], \mathbf{F}, \mathbf{RP} \rightsquigarrow (\text{local} : s, S))}$$

CABS: semántica del if

$$[\text{if}_C] \frac{\langle b, (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (\text{local} : s, \varepsilon)) \rangle \rightarrow_{Bexp} \langle b', (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s', S_{\mathbb{B}})) \rangle}{(\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (\text{local} : s, \text{if}(b)\{S_1\}\text{else}\{S_2\}S)) \rightarrow (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s', S_{\mathbb{B}}; \text{if}(b')\{S_1\}\text{else}\{S_2\}S))}$$

$$[\text{if}_C^{\text{TRUE}}] \frac{}{(\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (\text{local} : s, \text{if}(\text{TRUE})\{S_1\}\text{else}\{S_2\}S)) \rightarrow (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (\text{local} : s, S_1; S))}$$

$$[\text{if}_C^{\text{FALSE}}] \frac{}{(\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (\text{local} : s, \text{if}(\text{FALSE})\{S_1\}\text{else}\{S_2\}S)) \rightarrow (\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (\text{local} : s, S_2; S))}$$

CABS: semántica del thread

$$[\text{end}_C] \frac{}{(\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s, \varepsilon))) \rightarrow (\mathbf{G}, \mathbf{F}, \mathbf{RP}))}$$

$$[\text{thread}_C^1] \frac{\mathbf{F}(\text{func}) = (t, S_F, \text{args}_F, e) \quad \text{check_args}(\text{args}_F, \text{args}) \quad \text{eval}(\text{args}) = \text{args}'}{(\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s, \text{thread func}(\text{args}); S)) \rightarrow (\mathbf{G}, \mathbf{F}, (\mathbf{RP} \rightsquigarrow (s, \text{thread func}(\text{args}'); S)))}$$

$$[\text{thread}_C^2] \frac{\mathbf{F}(\text{func}) = (t, S_F, \text{args}_F, e) \quad \text{check_args}(\text{args}_F, \text{args}) \quad \text{eval}(\text{args}) = v_1 : \dots : v_n}{(\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s, \text{thread func}(\text{args}); S)) \rightarrow (\mathbf{G}, \mathbf{F}, (\mathbf{RP} \cup (\text{nil} [\text{args} \mapsto v_1 : \dots : v_n] : [], S_F)) \rightsquigarrow (s, S))}$$

CABS: ejemplo

```
1 int var;  
2  
3 int main() {  
4     thread f(1);  
5     thread f(2);  
6     return 0;  
7 }  
8  
9 void f(int value) {  
10     var = value;  
11 }
```

CABS: ejemplo

Excluyendo algunos pasos:

$$\begin{aligned} &(\mathbf{G}, \mathbf{F}, (\mathbf{nil}, \mathbf{thread } f(1); \mathbf{thread } f(2);)) \rightarrow \\ &\quad (\mathbf{G}, \mathbf{F}, (\mathbf{nil} [value \mapsto 1], var = value;) : (\mathbf{nil}, \mathbf{thread } f(2);)) \rightarrow \\ &(\mathbf{G}, \mathbf{F}, (\mathbf{nil} [value \mapsto 1], var = value;) : (\mathbf{nil} [value \mapsto 2], var = value;) : (\mathbf{nil}, \varepsilon)) \rightarrow \\ &(\mathbf{G} [var \mapsto 1], \mathbf{F}, (\mathbf{nil} [value \mapsto 1], \varepsilon) : (\mathbf{nil} [value \mapsto 2], var = value;) : (\mathbf{nil}, \varepsilon)) \rightarrow \\ &\quad (\mathbf{G} [var \mapsto 2], \mathbf{F}, (\mathbf{nil} [value \mapsto 1], \varepsilon) : (\mathbf{nil} [value \mapsto 2], \varepsilon) : (\mathbf{nil}, \varepsilon)) \rightarrow \\ &\quad\quad\quad (\mathbf{G} [var \mapsto 2], \mathbf{F}, []) \end{aligned}$$

ABS: sintaxis

- Sintaxis parecida a la de Java (aproximadamente).
- Clases que implementan interfaces para exponer los métodos públicos.
- Tipos futuro para esperar resultados del paso de mensajes (await).

```
1 interface nombre_de_interfaz {  
2     ...  
3     signaturas  
4     ...  
5 }
```

```
1 Tipo nombre_metodo(args);
```

```
1 class nombre_de_clase(args) implements nombre_de_interfaz {  
2     ...  
3     declaraciones de atributos privados  
4     ...  
5     ...  
6     implementaciones  
7     ...  
8 }
```

- Estado: (\mathbf{O}, \mathbf{C}) .
- Definición de las clases: $\mathbf{C} \llbracket c \rrbracket = (\text{Inter}, \text{attr}_c, \text{met}, \text{args}_c)$.
- Lista de objetos instanciados (actores):
 $\mathbf{O} \rightsquigarrow (id, c, \mathbf{RT} \rightsquigarrow (loc, S, t), t, \mathbf{attr})$

ABS: algunas reglas semánticas

$$\left[\text{ass}_{\text{ABS}}^1 \right] \frac{\text{is_local}(x) \quad \text{is_Int}(x)}{(\mathbf{O} \rightsquigarrow (id, c, \mathbf{RT} \rightsquigarrow (loc, x = a; S, t), t, \mathbf{attr}), \mathbf{C}) \rightarrow (\mathbf{O} \rightsquigarrow (id, c, \mathbf{RT} \rightsquigarrow (loc \left[x \mapsto \mathcal{A} \llbracket a \rrbracket_{loc, \mathbf{attr}} \right], S, t), t, \mathbf{attr}), \mathbf{C})}$$

$$\left[\text{obj}_{\text{ABS}} \right] \frac{\mathbf{C} \llbracket c' \rrbracket = (\text{Inter}, \text{attr}_c, \text{met}, \text{args}_c) \quad \text{check_args}(\text{args}_c, \text{args})}{(\mathbf{O} \rightsquigarrow (id, c, \mathbf{RT} \rightsquigarrow (loc, \text{Inter } \text{inter} = \text{new } c'(\text{args}); S, t), t, \mathbf{attr}), \mathbf{C}) \rightarrow (\mathbf{O} : o \rightsquigarrow (id, c, \mathbf{RT} \rightsquigarrow (loc \left[\text{inter} \mapsto id' \right], S, t), t, \mathbf{attr}), \mathbf{C})}$$

donde $o = (id', c', [], \perp, \mathbf{attr}')$ con id' un nuevo identificador de objeto no utilizado y

$\mathbf{attr}' = \text{attr}_c \left[\text{args}_c \mapsto \mathcal{E} \llbracket \text{args} \rrbracket_{loc, \mathbf{attr}} \right]$ los atributos del nuevo objeto creado.

ABS: algunas reglas semánticas

$$\left[\text{tsk}_{\text{SYNC}}^{\text{ABS1}} \right] \frac{\text{loc} \cup \text{attr} \llbracket \text{int} \rrbracket = \text{id}' \quad \mathbf{C} \llbracket c' \rrbracket = (\text{Inter}, \text{attr}_C, \text{met}, \text{args}_C) \quad \text{contains}(\text{met}, m) \quad \text{check_args}(\text{args}_m, \text{args})}{(\mathbf{O} \rightsquigarrow (\text{id}', c', \mathbf{RT}', t', \text{attr}')(\text{id}, c, \mathbf{RT} \rightsquigarrow (\text{loc}, \text{Int } x = \text{await } \text{int!}m(\text{args}); S, t), t, \text{attr}), \mathbf{C}) \rightarrow st}$$

donde $o' = (\text{id}', c', \mathbf{RT}' : \text{tsk}, t', \text{attr}')$, $\text{tsk} = (\text{loc}', S', t'')$ con t'' un identificador de tarea nuevo y $\text{loc}' = \text{nil} [\text{args}_m \mapsto \text{args}]$, $\text{met} \llbracket m \rrbracket = (S', \text{args}_m)$ y $st = (\mathbf{O} \rightsquigarrow o'(\text{id}, c, \mathbf{RT} \rightsquigarrow (\text{loc}, \text{Int } x = \text{await } t'; S, t), t, \text{attr}), \mathbf{C})$

$$\left[\text{ret}_{\text{ABS}}^1 \right] \frac{\text{tsk} = (\text{loc}', \varepsilon(\nu), t'')}{(\mathbf{O} \rightsquigarrow (\text{id}', c', \mathbf{RT}' \rightsquigarrow \text{tsk}, t', \text{attr}')(\text{id}, c, \mathbf{RT} \rightsquigarrow (\text{loc}, \text{Int } x = \text{await } t'; S, t), t, \text{attr}), \mathbf{C}) \rightarrow st}$$

donde $o' = (\text{id}', c', \mathbf{RT}', t', \text{attr}')$ y $st = (\mathbf{O} \rightsquigarrow o'(\text{id}, c, \mathbf{RT} \rightsquigarrow (\text{loc} [x \mapsto \nu], S, t), t, \text{attr}), \mathbf{C})$

$$\left[\text{end}_{\text{ABS}} \right] \frac{}{(\mathbf{O} \rightsquigarrow (\text{id}, c, \mathbf{RT} \rightsquigarrow (\text{loc}, \text{return } a; S, t), t, \text{attr}), \mathbf{C}) \rightarrow (\mathbf{O} \rightsquigarrow (\text{id}, c, \mathbf{RT} \rightsquigarrow (\text{loc}, \varepsilon(\mathcal{A} \llbracket a \rrbracket_{\text{loc}, \text{attr}}), t), \perp, \text{attr}), \mathbf{C})}$$

ABS: ejemplo

```
1 module Cabs;
2 import * from ABS.StdLib;
3
4 interface GLOBAL {
5   Int getvar();
6   Unit setvar(Int val);
7 }
8 class GlobalVariables() implements GLOBAL {
9   Int var = 0;
10
11   Int getvar() {
12     return var;
13   }
14
15   Unit setvar(Int val) {
16     var = val;
17   }
18 }
19
20 interface Intf {
21   Unit f(Int value);
22 }
23
24 interface Intmain {
25   Int main();
26 }
```

```
27
28 class Impf(GLOBAL globalval) implements Intf {
29   Unit f(Int value) {
30     await globalval!setvar(value);
31   }
32 }
33
34 class Impmain(GLOBAL globalval) implements Intmain
35 {
36   Int main() {
37     Intf funcf1 = new Impf(globalval);
38     funcf1!f(1);
39
40     Intf funcf2 = new Impf(globalval);
41     funcf2!f(2);
42
43     return 0;
44   }
45 }
46 {
47   GLOBAL globalval = new GlobalVariables();
48   Intmain prog = new Impmain(globalval);
49   await prog!main();
50 }
```

- Necesitamos crear de forma artificial un ámbito global de variables (memoria compartida ausente en ABS).
- La traducción de las funciones y procedimientos tienen que ser visibles desde cualquier punto del programa (¿métodos públicos de interfaz o métodos privados de una única clase?).
- La concurrencia tiene que ser de grano fino (ABS implementa el modelo de actores).
- Hay que dar un soporte a los arrays en ámbito local y global (ABS tiene un tipo de lista enlazada).

Traducción: ámbito global de variables y funciones

```
1 int var1;
2
3 void f() {
4     var1 = 2;
5 }
6
7 int main() {
8     thread f();
9     var1 = 1;
10    return 0;
11 }
```

Así solo es posible un interleaving.

Tal vez usando:

```
1 this!f();
2 suspend;
3
```

```
1 module Cabs;
2 import * from ABS.StdLib;
3
4 interface Functions {
5     Unit f();
6     Unit main();
7 }
8
9 class Prog() implements Functions {
10     Int var1 = 0;
11
12     Unit f() {
13         var1 = 2;
14     }
15
16     Unit main() {
17         this!f();
18         var1 = 1;
19     }
20 }
21
22 {
23     Functions prog = new Prog();
24     prog.main();
25 }
```

Traducción: ámbito global de variables y funciones

Tampoco funciona la idea del *suspend*

```
1 int var1;  
2 int var2;  
3  
4 void f() {  
5     var1 = 2;  
6     var2 = 4;  
7 }  
8  
9 int main() {  
10     thread f();  
11     var1 = 1;  
12     var2 = 3;  
13     return 0;  
14 }
```

Concluimos que cada función tiene que ir en un objeto distinto.

Traducción: ámbito global de variables y funciones

¿Qué hacemos ahora con las variables globales? Ya no pueden ser atributos de una clase que implemente una función. Necesita su propio objeto.

```
1 module Cabs;
2 import * from ABS.StdLib;
3
4 interface GLOBAL {
5   Int getvar1();
6   Unit setvar1(Int val);
7   Int getvar2();
8   Unit setvar2(Int val);
9 }
10
11 class GlobalVariables() implements GLOBAL {
12   Int var1 = 0;
13   Int var2 = 0;
14
15   Int getvar1() {
16     return var1;
17   }
18
19   Unit setvar1(Int val) {
20     var1 = val;
21   }
22
23   Int getvar2() {
24     return var2;
25   }
26
27   Unit setvar2(Int val) {
28     var2 = val;
29   }
30 }
```

Usaremos llamadas asíncronas para manejar este objeto:

```
1 globalval!setvar(1);
2 Int x = await globalval!getvar();
3
```

Traducción: arrays locales y globales

El mismo argumento que hemos empleado para las variables globales (preservación de los interleavings) se puede aplicar a los arrays encapsulando una lista de ABS en una clase.

```
1 interface ArrayInt {
2     Int getV(Int indx);
3     Unit setV(Int indx, Int value);
4 }
5
6 class ArrayIntC(Int size) implements
7     ArrayInt{
8     List<Int> list = copy(0, size);
9
10    Int getV(Int indx) {
11        return nth(this.list, indx);
12    }
13
14    Unit setV(Int indx, Int value) {
15        Int i = 0;
16        List<Int> prev = Nil;
17        List<Int> post = list;
18        while (i < indx) {
19            Int elem = head(post);
20            prev = appendright(prev, elem);
21            post = tail(post);
22            i = i + 1;
23        }
24        prev = appendright(prev, value);
25        post = tail(post);
26        this.list = concatenate(prev, post);
27    }
```

Para el ámbito global hay que añadir algunos métodos extra (*retrievearray*, *init* ...).

Traducción: función de traducción **Aexp**

$$\mathcal{C}_{\mathbf{Aexp}} \llbracket n \rrbracket v = (\text{Int}(\mathbf{Aux} v) = n, v + 1)$$

$$\mathcal{C}_{\mathbf{Aexp}} \llbracket x \rrbracket v = (\text{Int}(\mathbf{Aux} v) = x, v + 1) \text{ donde } x \text{ es variable entera local.}$$

$$\mathcal{C}_{\mathbf{Aexp}} \llbracket x \rrbracket v = (\text{Int}(\mathbf{Aux} v) = \text{await globalval!getx}(), v + 1)$$

donde x es variable entera global.

$$\mathcal{C}_{\mathbf{Aexp}} \llbracket a_1 \odot_{\mathcal{A}} a_2 \rrbracket v = (c_1; c_2; \text{Int}(\mathbf{Aux} v'') = (\mathbf{Aux}(v' - 1)) \odot_{\mathcal{A}} (\mathbf{Aux}(v'' - 1)), v'' + 1)$$

donde $\mathcal{C}_{\mathbf{Aexp}} \llbracket a_1 \rrbracket v = (c_1, v')$ y $\mathcal{C}_{\mathbf{Aexp}} \llbracket a_2 \rrbracket v' = (c_2, v'')$

$$\mathcal{C}_{\mathbf{Aexp}} \llbracket f(e_1 \dots e_n) \rrbracket v^1 = (c_1 \dots c_n$$

$$\text{Int} f(\mathbf{Aux}(v^{n+1})) = \text{new Impf}(\text{globalval})$$

$$\text{Int}(\mathbf{Aux}(v^{n+1} + 1)) = (\mathbf{Aux}(v^{n+1}))!((\mathbf{Aux}(v^2 - 1)), \dots,$$

$$(\mathbf{Aux}(v^{n+1} - 1))), v^{n+1} + 2)$$

donde $\mathcal{C}_{\mathbf{Exp}} \llbracket e_i \rrbracket v^i = (c_i, v^{i+1})$ (escogiendo $\mathcal{C}_{\mathbf{Aexp}}$ o $\mathcal{C}_{\mathbf{Bexp}}$ según convenga)

Traducción: función de traducción del if y del while

$\mathcal{C} \llbracket \text{if}(b)\{S_1\}\text{else}\{S_2\} \rrbracket v = (c_1; \text{if}(\text{Aux}(v' - 1))\{c_2\}\text{else}\{c_3\}, v''')$

donde $\mathcal{C}_{\text{Bexp}} \llbracket b \rrbracket v = (c_1, v')$, $\mathcal{C} \llbracket S_1 \rrbracket v' = (c_2, v'')$ y $\mathcal{C} \llbracket S_2 \rrbracket v'' = (c_3, v''')$

$\mathcal{C} \llbracket \text{while}(b)\{S\} \rrbracket v = (c_1; \text{while}(\text{Aux}(v' - 1))\{c_2 c_3 \text{Aux}(v' - 1) = \text{Aux}(v''' - 1)\}; v''')$

donde $\mathcal{C}_{\text{Bexp}} \llbracket b \rrbracket v = (c_1, v')$, $\mathcal{C} \llbracket S \rrbracket v' = (c_2, v'')$ y $\mathcal{C}_{\text{Bexp}} \llbracket b \rrbracket v'' = (c_3, v''')$

Traducción: función de traducción de una función y del thread

$\mathcal{C} \llbracket \text{int func}(arg)\{S \text{ return } a\} \rrbracket v = (\text{interface Intfunc}\{Int \text{ func}(\mathcal{C}_{arg} arg);\}$
class Impfunc(GLOBAL globalval) implements Intfunc{
 $Int \text{ func}(\mathcal{C}_{arg} arg)\{c \text{ c' return Aux}(v'' - 1)\}, v''$
 donde $\mathcal{C} \llbracket S \rrbracket v = (c, v')$ y $\mathcal{C}_{Aexp} \llbracket a \rrbracket v' = (c', v'')$

$\mathcal{C} \llbracket \text{threadf}(e_1 \dots e_n) \rrbracket v^1 = (c_1 \dots c_n$
 $Int f(\mathbf{Aux}(v^{n+1})) = \text{new Impf}(\text{globalval})$
 $(\mathbf{Aux}(v^{n+1}))!((\mathbf{Aux}(v^2 - 1)), \dots, (\mathbf{Aux}(v^{n+1} - 1))), v^{n+1} + 1)$
 donde $\mathcal{C}_{Exp} \llbracket e_i \rrbracket v^i = (c_i, v^{i+1})$ (escogiendo \mathcal{C}_{Aexp} o \mathcal{C}_{Bexp} según convenga)

- Se trata de ver que existe una equivalencia semántica entre el programa en CABS y su traducción a ABS.
- Ambas semánticas son de paso corto luego la idea propuesta pasa por hacer una bisimulación en ambas semánticas.
- Partiendo de dos estados equivalentes, uno de CABS y otro de ABS, hay que ver que un paso en una de las semánticas se corresponde con un paso en la otra semántica (obviando transiciones que no afectan al significado).
- Dándose esto para un paso, por inducción en la longitud de la secuencia de derivación, tenemos que se cumple para ejecuciones arbitrariamente largas.