Estudio y desarrollo de técnicas para el testing de programas concurrentes

CABS: traducción de C a ABS

Marco Antonio Garrido Rojo[1]

17 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 }
```

CABS: semántica

- Estado: $(\mathbf{G}, \mathbf{F}, \mathbf{RP})$.
- Ámbito global de varibles ($\mathbf{G}$): 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: $\mathbf{RP} \rightsquigarrow (local : s, S)$

CABS: semántica

$$\begin{aligned}
\mathbf{init}(\varepsilon) &= (\mathbf{nil}, \mathbf{nil}, []) \\
\mathbf{init}(\mathit{int}\ \text{var}; \mathbf{P}) &= (\mathbf{G}\,[\ \text{var} \mapsto 0]\,, \mathbf{F}, \mathbf{RP}) \\
&\quad \text{donde } \mathbf{init}(\mathbf{P}) = (\mathbf{G}, \mathbf{F}, \mathbf{RP}) \\
\mathbf{init}(\mathit{bool}\ \text{var}; \mathbf{P}) &= (\mathbf{G}\,[\ \text{var} \mapsto \text{FALSE}]\,, \mathbf{F}, \mathbf{RP}) \\
&\quad \text{donde } \mathbf{init}(\mathbf{P}) = (\mathbf{G}, \mathbf{F}, \mathbf{RP}) \\
\mathbf{init}(\mathit{int}\ \mathbf{func}(\text{arg})\{S; \mathbf{return}\ a\}\mathbf{P}) &= (\mathbf{G}, \mathbf{F}\,[\mathbf{func} \mapsto (\mathit{int}, S, \text{arg}, a)]\,, \mathbf{RP}) \\
&\quad \text{donde } \mathbf{init}(\mathbf{P}) = (\mathbf{G}, \mathbf{F}, \mathbf{RP}) \\
\mathbf{init}(\mathit{bool}\ \mathbf{func}(\text{arg})\{S; \mathbf{return}\ b\}\mathbf{P}) &= (\mathbf{G}, \mathbf{F}\,[\mathbf{func} \mapsto (\mathit{bool}, S, \text{arg}, b)]\,, \mathbf{RP}) \\
&\quad \text{donde } \mathbf{init}(\mathbf{P}) = (\mathbf{G}, \mathbf{F}, \mathbf{RP}) \\
\mathbf{init}(\mathit{void}\ \mathbf{func}(\text{arg})\{S\}\mathbf{P}) &= (\mathbf{G}, \mathbf{F}\,[\mathbf{func} \mapsto (\mathit{void}, S, \text{arg}, \varepsilon)]\,, \mathbf{RP}) \\
&\quad \text{donde } \mathbf{init}(\mathbf{P}) = (\mathbf{G}, \mathbf{F}, \mathbf{RP})
\end{aligned}$$

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

CABS: semántica de las **Aexp**

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

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

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

CABS: semántica de las **Aexp**

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

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

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

CABS: semántica de las **Aexp**

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

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

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

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

CABS: semántica de las asignaciones

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

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

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

CABS: semántica del if

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

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

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

CABS: semántica del if

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

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

$$\left[\text{thread}_C^2\right] \quad \frac{\mathbf{F}(\mathbf{func}) = (t, S_F, args_F, e) \qquad check\_args(args_F, args) \qquad eval(args) = v_1 : \cdots : v_n}{(\mathbf{G}, \mathbf{F}, \mathbf{RP} \rightsquigarrow (s, \mathbf{thread\ func}(args); S)) \rightarrow (\mathbf{G}, \mathbf{F}, (\mathbf{RP} \cup (\mathbf{nil}\ [args \mapsto v_1 : \cdots : v_n] : [], S_F)) \rightsquigarrow (s, S))}$$