

Estudio y desarrollo de técnicas para el testing de programas concurrentes

CABS: traducción de C a ABS

Marco Antonio Garrido Rojo¹

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,
16                    array);
17     return 0;
18 }
```

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

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