Testowanie za pomocą wykonywalnych projektów

Maciej Zielenkiewicz

25 lutego 2010

Ogólny plan

Naszym **celem** jest weryfikacja **współbieżnego** programu obiektowego przy wykorzystaniu **modelu** (napisanego w języku wysokiego poziomu o dobrze i prosto zdefiniowanej semantyce). Jeżeli weryfikator stwierdzi, że program nie jest poprawny, chcemy uzyskać **kontrprzykład** w postaci danych i przeplotu które powodują działanie nie przewidywane przez model.

Sposób działania weryfikatora

Jednocześnie wykonywane są program i model. Dla modelu wskazuje się parametry pochodzęce *z zewnątrz*, które będą wybierane przez weryfikator. Trzeba zadbać o wskazanie sposobu przeliczania tych parametrów na parametry testowanej implementacji oraz o to, aby parametry obejmowały też możliwe rezultaty interackji z otoczeniem.

Sposób działania weryfikatora

Jednocześnie wykonywane są program i model. Dla modelu wskazuje się parametry pochodzęce *z zewnątrz*, które będą wybierane przez weryfikator. Trzeba zadbać o wskazanie sposobu przeliczania tych parametrów na parametry testowanej implementacji oraz o to, aby parametry obejmowały też możliwe rezultaty interackji z otoczeniem.
Przy wykonywaniu modelu zapisujemy czy warunki **if**ów i podobnych instrukcji były spełnione. Przed kolejnym wykonaniem jeden z warunków jest zaprzeczany, aby wygenerować **inną ścieżkę** przepływu sterowania. Do wybrania odpowiednich danych wejściowych przy znajomości warunków które mają być spełnione można wykorzystać np. **constraint solvera** lub **SMT solvera**.
Weryfikator dąży do wypróbowania wszystkich możliwych ścieżek.

Sposób działania weryfikatora

Potencjalne problemy:

- wątki, które się nie kończą
- pętle które wykonują podobne zadania (np. tworzenie wątków) określoną liczbę razy

Sposób działania weryfikatora

Potencjalne problemy:

- wątki, które się nie kończą
- pętle które wykonują podobne zadania (np. tworzenie wątków) określoną liczbę razy

Rozwiązanie:

- ograniczenie maksymalnej liczby instrukcji do wykonania
- ręczne dodefiniowanie klas abstrakcji parametrów

Creol

- *wykonywalny* język wysokiego poziomu
- obiektowy
- dobrze określona semantyka
- abstrakcja współbieżności

Dobrze określona nie oznacza, że wykorzystanie będzie trywialne:

$\langle C' : Cl \,|\, Par : \epsilon, Att : \epsilon, Mtds = \text{M}, Tok : 1\rangle$
$\langle (C;1) : Ob \,|\, Cl : C, Att : \epsilon, Pr : \langle (o := \textbf{new } C'; t!o.m(\epsilon \to \text{Nat}, AB, \epsilon); \textbf{await } t?; t?(x)),$
$(x \mapsto d_{\text{Nat}}, o \mapsto null, t \mapsto d_{\text{Label}})\rangle, PrQ : \epsilon, EvQ : \epsilon, Lab : 2\rangle$
$\longrightarrow$ R2
$\langle C' : Cl \,|\, Par : \epsilon, Att : \epsilon, Mtds = \text{M}, Tok : 2\rangle$
$\langle (C;1) : Ob \,|\, Cl : C, Att : \epsilon, Pr : \langle (o := (C';1); t!o.m(\epsilon \to \text{Nat}, AB, \epsilon); \textbf{await } t?; t?(x)),$
$(x \mapsto d_{\text{Nat}}, o \mapsto null, t \mapsto d_{\text{Label}})\rangle, PrQ : \epsilon, EvQ : \epsilon, Lab : 2\rangle$
$\langle (C';1) : Ob \,|\, Cl : C, Att : \epsilon, Pr : \langle run(\epsilon \to \epsilon, \varsigma, \epsilon); \epsilon\rangle, PrQ : \epsilon, EvQ : \epsilon, Lab : 1\rangle$
$\longrightarrow$ R1, $\longrightarrow$ R8, we omit the default invocation of the empty *run* method (e.g., **skip**) in $(C';1)$.
$\langle C' : Cl \,|\, Par : \epsilon, Att : \epsilon, Mtds = \text{M}, Tok : 2\rangle$
$\langle (C;1) : Ob \,|\, Cl : C, Att : \epsilon, Pr : \langle (t := 2; !o.m(\epsilon \to \text{Nat}, AB, \epsilon); \textbf{await } t?; t?(x)),$
$(x \mapsto d_{\text{Nat}}, o \mapsto (C';1), t \mapsto d_{\text{Label}})\rangle, PrQ : \epsilon, EvQ : \epsilon, Lab : 2\rangle$
$\langle (C';1) : Ob \,|\, Cl : C, Att : \epsilon, Pr : \langle \epsilon, \epsilon\rangle, PrQ : \epsilon, EvQ : \epsilon, Lab : 2\rangle$
$\longrightarrow$ R1, $\longrightarrow$ R9
$\langle C' : Cl \,|\, Par : \epsilon, Att : \epsilon, Mtds = \text{M}, Tok : 2\rangle$
$\langle (C;1) : Ob \,|\, Cl : C, Att : \epsilon, Pr : \langle \textbf{await } t?; t?(x), (x \mapsto d_{\text{Nat}}, o \mapsto (C';1), t \mapsto 2)\rangle, PrQ : \epsilon, EvQ : \epsilon, Lab : 3\rangle$
$\langle (C';1) : Ob \,|\, Cl : C, Att : \epsilon, Pr : \langle 1?(\epsilon), \epsilon\rangle, PrQ : \epsilon, EvQ : \epsilon, Lab : 2\rangle$
$invoc(m, \epsilon \to \text{Nat}, AB, (C;1)\ 2)$ **to** $(C';1)$
$\longrightarrow$ R11, $\longrightarrow$ R12
$\langle C' : Cl \,|\, Par : \epsilon, Att : \epsilon, Mtds = \text{M}, Tok : 2\rangle$
$\langle (C;1) : Ob \,|\, Cl : C, Att : \epsilon, Pr : \langle \textbf{await } t?; t?(x), (x \mapsto d_{\text{Nat}}, o \mapsto (C';1), t \mapsto 2)\rangle, PrQ : \epsilon, EvQ : \epsilon, Lab : 3\rangle$
$\langle (C';1) : Ob \,|\, Cl : C, Att : \epsilon, Pr : \langle \epsilon, \epsilon\rangle, PrQ : \epsilon, EvQ :, Lab : 2\rangle$
$bind(m, \epsilon \to \text{Nat}, AB, (C;1)\ 2)$ **to** C'
$\longrightarrow$ R16
$\langle C' : Cl \,|\, Par : \epsilon, Att : \epsilon, Mtds = \text{M}, Tok : 2\rangle$
$\langle (C;1) : Ob \,|\, Cl : C, Att : \epsilon, Pr : \langle \textbf{await } t?; t?(x), (x \mapsto d_{\text{Nat}}, o \mapsto (C';1), t \mapsto 2)\rangle, PrQ : \epsilon, EvQ : \epsilon, Lab : 3\rangle$
$\langle (C';1) : Ob \,|\, Cl : C, Att : \epsilon, Pr : \langle \epsilon, \epsilon\rangle, PrQ : \epsilon, EvQ : \epsilon, Lab : 2\rangle$
$bound(\langle (x := 0; return(x)), (caller \mapsto (C;1), label \mapsto 2, x \mapsto d_{\text{Nat}})\rangle)$ **to** $(C';1)$
$\longrightarrow$ R17, $\longrightarrow$ R6
$\langle C' : Cl \,|\, Par : \epsilon, Att : \epsilon, Mtds = \text{M}, Tok : 2\rangle$
$\langle (C;1) : Ob \,|\, Cl : C, Att : \epsilon, Pr : \langle \textbf{await } t?; t?(x), (x \mapsto d_{\text{Nat}}, o \mapsto (C';1), t \mapsto 2)\rangle, PrQ : \epsilon, EvQ : \epsilon, Lab : 3\rangle$

$\langle (C';1) : Ob \,|\, Cl : C, Att : \epsilon, Pr : \langle (x := 0; return(x)), (caller \mapsto (C;1), label \mapsto 2, x \mapsto d_{\mathrm{Nat}}) \rangle,$
$PrQ : \epsilon, EvQ : \epsilon, Lab : 2 \rangle$
$\longrightarrow$ R1, $\longrightarrow$ R18
$\langle C' : Cl \,|\, Par : \epsilon, Att : \epsilon, Mtds = \mathrm{M}, Tok : 2 \rangle$
$\langle (C;1) : Ob \,|\, Cl : C, Att : \epsilon, Pr : \langle \mathbf{await}\ t?; t?(x), (x \mapsto d_{\mathrm{Nat}}, o \mapsto (C';1), t \mapsto 2) \rangle, PrQ : \epsilon, EvQ : \epsilon, Lab : 3 \rangle$
$\langle (C';1) : Ob \,|\, Cl : C, Att : \epsilon, Pr : \langle \epsilon, (caller \mapsto (C;1), label \mapsto 2, x \mapsto 0) \rangle, PrQ : \epsilon, EvQ : \epsilon, Lab : 2 \rangle$
$comp(2,0)$ **to** $(C;1)$
$\longrightarrow$ R10, $\longrightarrow$ R4
$\langle C' : Cl \,|\, Par : \epsilon, Att : \epsilon, Mtds = \mathrm{M}, Tok : 2 \rangle$
$\langle (C;1) : Ob \,|\, Cl : C, Att : \epsilon, Pr : \langle t?(x), (x \mapsto d_{\mathrm{Nat}}, o \mapsto (C';1), t \mapsto 2) \rangle, PrQ : \epsilon, EvQ : comp(2,0), Lab : 3 \rangle$
$\langle (C';1) : Ob \,|\, Cl : C, Att : \epsilon, Pr : \langle \epsilon, (caller \mapsto (C;1), label \mapsto 2, x \mapsto 0) \rangle, PrQ : \epsilon, EvQ : \epsilon, Lab : 2 \rangle$
$\longrightarrow$ R13, since $t \mapsto 2$
$\langle C' : Cl \,|\, Par : \epsilon, Att : \epsilon, Mtds = \mathrm{M}, Tok : 2 \rangle$
$\langle (C;1) : Ob \,|\, Cl : C, Att : \epsilon, Pr : \langle x := 0, (x \mapsto d_{\mathrm{Nat}}, o \mapsto (C';1), t \mapsto 2) \rangle, PrQ : \epsilon, EvQ : \epsilon, Lab : 3 \rangle$
$\langle (C';1) : Ob \,|\, Cl : C, Att : \epsilon, Pr : \langle \epsilon, (caller \mapsto (C;1), label \mapsto 2, x \mapsto 0) \rangle, PrQ : \epsilon, EvQ : \epsilon, Lab : 2 \rangle$
$\longrightarrow$ R1
$\langle C' : Cl \,|\, Par : \epsilon, Att : \epsilon, Mtds = \mathrm{M}, Tok : 2 \rangle$
$\langle (C;1) : Ob \,|\, Cl : C, Att : \epsilon, Pr : \langle \epsilon, (x \mapsto 0, o \mapsto (C';1), t \mapsto 2) \rangle, PrQ : \epsilon, EvQ : \epsilon, Lab : 3 \rangle$
$\langle (C';1) : Ob \,|\, Cl : C, Att : \epsilon, Pr : \langle \epsilon, (caller \mapsto (C;1), label \mapsto 2, x \mapsto 0) \rangle, PrQ : \epsilon, EvQ : \epsilon, Lab : 2 \rangle$

Fig. 9. An example of an execution sequence. The representation of C as well as some intermediary states are omitted, $\longrightarrow$ RX denotes the application of rule RX. For convenience, we denote the method multiset of C' by M, $next(n)$ by $n + 1$, and ignore equational reduction.

Konstrukcja języka I

- klasy *podobne* do Javy

```
interface Client
begin
 with Any
  op availFiles
  op reqFile
end
```

```
interface Server
begin
 with Server
  op enquire
  op getLength
  op getPacket
end
```

```
interface Peer
 inherits Client, Server
begin
end
```

- po stworzeniu klasy automatycznie jest uruchamiana jej metoda **run**
- dostęp do klasy jedynie przez jej metody
- instrukcje pozwalające na (deterministyczne lub nie) oddanie procesora (**await** *warunek*)

Konstrukcja języka II

- **asynchroniczne** wywoływanie metod, synchronizacja w momencie odczytywania wyniku:
  label!metoda(Klasa)
  label?(zmienna)
- możliwość czekania aktywnego i nieaktywnego:
  await *label?(zmienna)*≡**await** *label?; label?(zmienna)*
- na jednym obiekcie operuje na raz co najwyżej jeden proces
- wyrażenia warunkowe nie mają efektów ubocznych

Case study: ASK

- system łączący *„inteligentnie”* użytkowników i dostawców usług/informacji
  - przykład: Koninklijke Nederlandse Redding Maatschappij - zarządzanie planami ratowników–ochotników
- napisany w C, model w Creolu
- obserwacja interesujących szczegółów wykonania w AspectC

Klasa zarządzająca wątkami

```
1  class ThreadPool(size : Int, maxNofThreads : Int)
2    contracts ThreadPool
3  begin
4    vars taskCtr, threadCtr, busyCtr :Counter;
5    var taskQueue : TaskQueue;
6    var threads : List[Thread];
7    var balancer : Task;
8
9    op init ==
10    // removed variable initialization
11     balancer:= new BalancerTask(1, taskCtr, threadCtr, busyCtr,
12       maxNofThreads, mrate, taskQueue, this);
13     this.dispatchTask(balancer;)
14
15   with Any op dispatchTask(in task : Task) ==
16     taskQueue.enqueueTask(task;)
17
18   with Any op createThreads(amount : Int) ==
19     var i : Int;
20     var thread : Thread;
21     i:= 0;
22     while (i < amount) do
23       thread:= new Worker(taskQueue, busyCtr, threadCtr);
24       threads:= threads ⊢ thread; // append thread
25       threadCtr.inc(;);
26       i:= i +1
27     end
28
29   with Any op start ==
30     this.createThreads(size;)
31 end
```

Fig. 3. ThreadPool of the ASK system (instantiation of Counter and TaskQueue omitted).

Tworzenie nowych wątków

```
1   op init ==
2     maxthreads:=maxthreads+1;
3
4   op createThreads ==
5     var amountToCreate : Int;
6     var idlethreads : Int:=threads-busythreads;
7     await ((threads<maxthreads)
8         ∧((idlethreads-tasks)<(threads/2)));
9     amountToCreate:=tasks-idlethreads+(threads/2);
10    if (amountToCreate > (maxthreads-threads)) then
11      amountToCreate:=maxthreads-threads;
12    end;
13    if (amountToCreate > 0) then
14      await threadpool.createThreads(amountToCreate);
15    end;
16    createThreads(); // infinite loop by tail-recursion
```

Fig. 5. Parts of the balancing thread to initialize and create new threads. The fields `threads`, `idlethreads` and `tasks` are updated by outside method calls, so the conditions in the **await** statements can become true.

Main

```
1  class Main(nthreadsS : Int, maxWorkThreadsS : Int)
2  begin
3    var threadpool : ThreadPool;
4    var executionCounter : Counter;
5
6    op init ==
7      threadpool:=new ThreadPool(nthreadsS, maxWorkThreadsS);
8      executionCounter:=new Counter;
9
10   op run ==
11     var task : Task;
12     var i : Int;
13     i:=0;
14     while (i<10) do
15       task:=new CounterTask(i, executionCounter);
16       threadpool.dispatchTask(task;);
17       i:=i+1;
18     end
19     threadpool.start(;)
20    //After running, the executionCounter should be 10
21 end
```

Fig. 6. Setting up a model for DSE. Here, $nthreads_S$ is the number of initial threads to be created and $maxthreads_S$ is the maximal size of the thread pool.

Przykład analizy ścieżek I

Zaczynamy od losowych parametrów $maxWorkThreads_S = 0$ i $nthreads_S = 1$. Uzyskujemy warunki:

- z pętli w linii 22, Fig. 3:
  $\{\text{"ifthenelse"} : (0 < nthreads_S)\}$
  $\{\text{"ifthenelse"} : \text{not}\,(1 < nthreads_S)\}$
- z linii 7, Fig. 5:
  $\{\text{"disabled\_wait"} : \text{not}\ (1 < (maxWorkThreads_S + 1) \wedge true)\}$

Dla drugiego testu wybieramy $maxWorkThreads_S = 15$ i otrzymujemy nowe warunki:

- $\{\text{enabled\_wait} : (1 < (maxWorkThreads_S + 1) \wedge true)\}$
- $\{\text{"ifthenelse"} : \text{not}\,(10 > maxWorkThreads_S)\}$

Liczba 10 wynika z początkowej liczby tworzonych zadań (Fig. 6).

Przykład analizy ścieżek II

Do następnego testu wybieramy $maxWorkThreads = 5$
i uzsykujemy:

- $\{\text{"disabled\_wait"} : (1 < (maxWorkThreads_S + 1) \wedge true)\}$
- $\{\text{"ifthenelse"} : \text{not}\,(10 > maxWorkThreads_S)\}$
- $\{\text{"ifthenelse"} : \text{not}\,(maxWorkThreads_S > 0)\}$

Pozostaje jeszcze zmienna $nthreads_S$. Można rozróżnić dwa istotnie różne przypadki: zwykły przebieg, który jest „trochę inny" dla każdej wartości > 0. i $nthreads_S > 0$, kiedy zadanie wyrównywania liczby wątków nie zostanie uruchomione; wtedy jedynym warunkiem będzie

- $\{\text{"ifthenelse"} : \text{not}\,(0 < nthreads_S)\}$

Dane z wykonania

```
<trace>
  <createThreads thread="3079972528" time="501911878"
                 number="10"/>
  <starting thread="3075214224" time="501911929" info=""/>
  <waiting thread="3075214224" time="501911951" info=""/>
  <starting thread="3066821520" time="501911980" info=""/>
  <waiting thread="3066821520" time="501911999" info=""/>
  ...
  <enqueue thread="3079972528" time="501912403"
           info="Sabbey - balancer (Sabbey.c 353)"/>
  ...
</trace>
```

Fig. 7. Parts of a recorded event trace from the ASK system. At the beginning, 10 threads are created; each thread emits a `starting` and a `waiting` event when created. Later, a task is added to the system.

Uzyskanie wyniku

Scenariusz testowy kończy się sukcesem, jeżeli w modelu uda się odtworzyć ślad wykonania z implementacji (i niezmienniki będą zachowane). Jeżeli niezmienniki zostaną złamane, to model jest wadliwy. Jeżli model się nie zapętli, ale program tak, to nic nie wiemy, bo dla tych samych wejść model mógł się zapętlić przy innym przeplocie.

```
1   op run ==
2     ok:= false;
3     this.allow("createThreads";);
4     await get(sem, "createThreads") = 0;
5     this.allow("starting";);
6     await get(sem, "starting") = 0;
7     this.allow("waiting";);
8     await get(sem, "waiting") = 0;
9     this.allow("starting";);
10    await get(sem, "starting") = 0;
11    this.allow("waiting";);
12    await get(sem, "waiting") = 0;
13  ...
14    this.enqueue(3079972528, 501912403,
15                 "Sabbey␣-␣balancer␣(Sabbey.c␣353)";);
16  ...
17    ok:= true
```

Fig. 10. Replaying the trace of Figure 7: tester event and action behavior in the model.

Droga do sukcesu

Solvery SMT

- Problemy sformułowane jak dla SAT-solverów, ale dodatkowe predykaty z teorii tego, co chcemy analizować (np. liczb rzeczywistych)
- przykładowa formuła:

$$3x + 2y - z > 1$$

- przykładowe metody rozwiązywania:
  - tłumaczenie do SAT
  - DPLL(T)

Solvery SMT

- Problemy sformułowane jak dla SAT-solverów, ale dodatkowe predykaty z teorii tego, co chcemy analizować (np. liczb rzeczywistych)
- przykładowa formuła:

$$3x + 2y - z > 1$$

- przykładowe metody rozwiązywania:
  - tłumaczenie do SAT
  - DPLL(T)
- często teoria tego, co chcemy badać, jest nierozstrzygalna (np. FO dla $\mathbb{N}$)
  - ale często można to co chcemy zrobić w teorii rozstrzygalnej (np. dla liczb naturalnych bez mnożenia)
  - iSAT: dołączenie do DPLL(T) pakietu do nieliniowej optymalizacji

Gotowe SMT-solvery

- ABsolver
- Barcelogic
- Beaver
- Boolector
- CVC3
- The Decision Procedure Toolkit (DPT)
- Ergo
- HySAT
- MathSAT
- OpenSMT
- SatEEn
- Spear
- STP
- SWORD
- veriT
- Yices
- Z3

z Wikipedii

Crosscutting in C Programs

AspectC.net

```
void *NutHeapAlloc(size_t size) {
  #ifdef NUTDEBUG
    //code removed
  #endif
    NODE **fpp = 0;
    //code removed
  #if defined(__arm__) ||
    defined(__m68k__) ||
    defined(__H8300H__) || ...
    while ((size & 0x03) != 0)
        size++;
  #endif
    if (size >= available) {
      #ifdef NUTDEBUG
        //code removed
      #endif
        return 0;
    }
    //code removed
...(next column)
```

```
while (node) {
    //code removed
    if (fit) {
      //split the node if too big
      if (fit->hn_size > ...) {
          //code removed
          *fpp = node;
      } else
          *fpp = fit->hn_next;
    ...}
    //code removed
}
#ifdef NUTDEBUG
    //code removed
#endif
return fit; }
```

Nut/OS, heap.c

debug concern

system-specific concern

optimization concern

main functionality

Crosscutting in C Programs

```
int is_orphaned_pgrp(int pgrp) {
    int retval;

    read_lock(&tasklist_lock);

    retval =
        will_become_orphaned
    _pgrp(pgrp, NULL);

    read_unlock(&tasklist_lock);

    return retval;

}
```

…(next column)

```
int session_of_pgrp(int pgrp) {
    struct task_struct *p;
    int sid = -1;
    read_lock(&tasklist_lock);
    do_each_task_pid(pgrp,
    PIDTYPE_PGID, p) {
    // code removed
    }
    // code removed
    read_unlock(&tasklist_lock);
    return sid;
}
```

Linux Kernel 2.6
kernel/exit.c

synchronization concern

main functionality

Crosscutting in C Programs

synch.c

```
struct lock *
lock_create(const char *name) {
   struct lock *lock;
   lock = kmalloc(sizeof(struct lock));
   if (lock == NULL) { return NULL; }
   //code removed
}
```

thread.c

```
static struct thread *
thread_create(const char *name) {
   struct thread *thread =
       kmalloc(sizeof(struct thread));
   if (thread==NULL) {  return NULL; }
   //code removed
}
```

OS161

misc.c

```
char * kstrdup(const char *s) {
   char *z = kmalloc(strlen(s)+1);
   if (z==NULL) { return NULL; }
   //code removed
}
```

device.c

```
struct vnode *
dev_create_vnode(…) {
       int result;
       struct vnode *v;
       v = kmalloc(sizeof(struct vnode));
       if (v==NULL) {  return NULL; }
       //code removed
}
```

error checking concern

main functionality

Crosscutting in C Programs

```
void * rb_find (const struct rb_table* tree, const void *item) {
        //local variable declaration
        assert (tree != NULL && item != NULL);
        //code removed
}
void ** rb_probe (struct rb_table *tree, void *item) {
        //local variable declaration
        assert (tree != NULL && item != NULL );
        //code removed
}
void * rb_delete (struct rb_table *tree, const void *item) {
        //local variable declaration
        assert (tree != NULL && item != NULL);
        //code removed
}
```

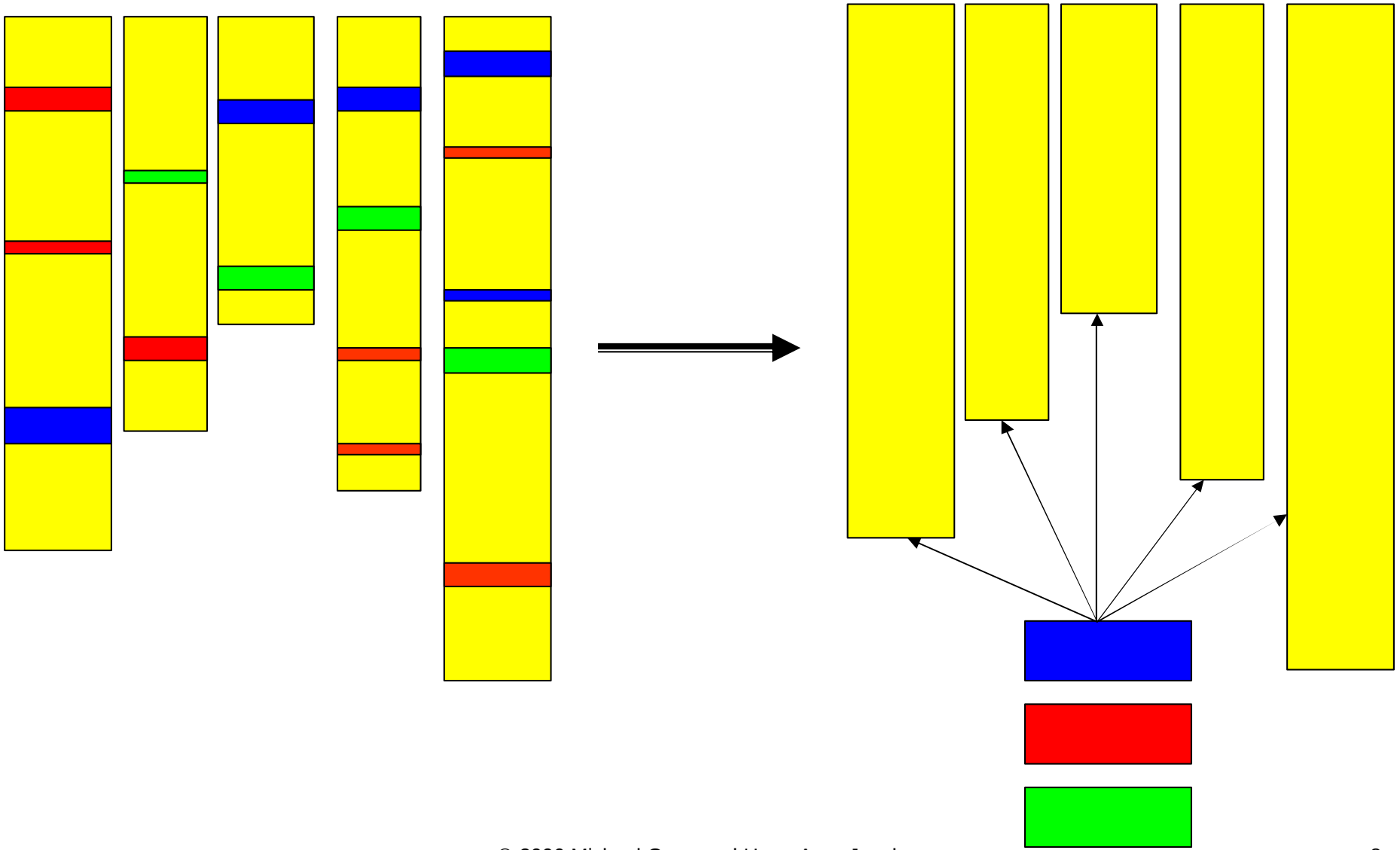
GNU libavl 2.0.2

rb.c

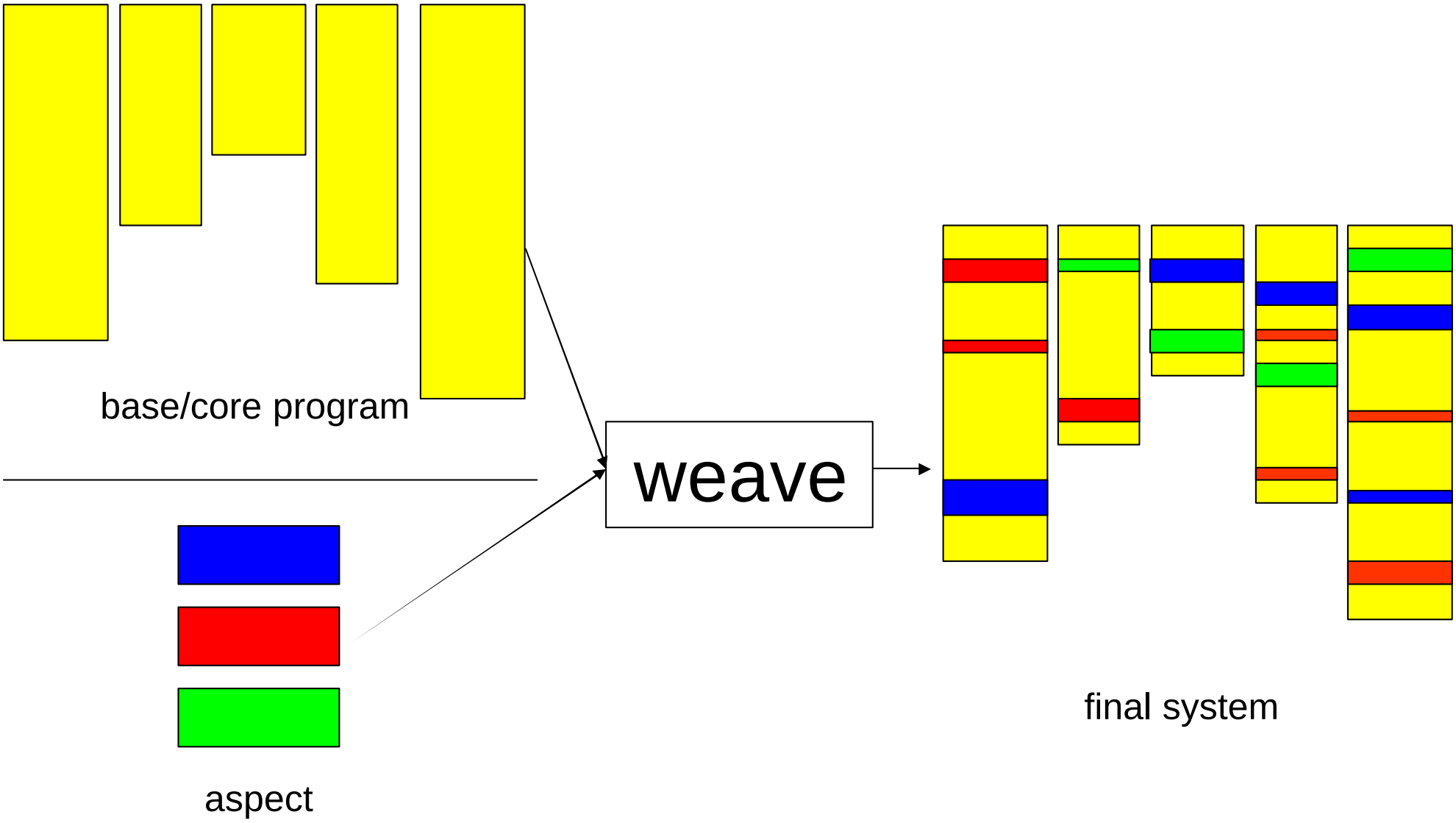
precondition check concern

main functionality

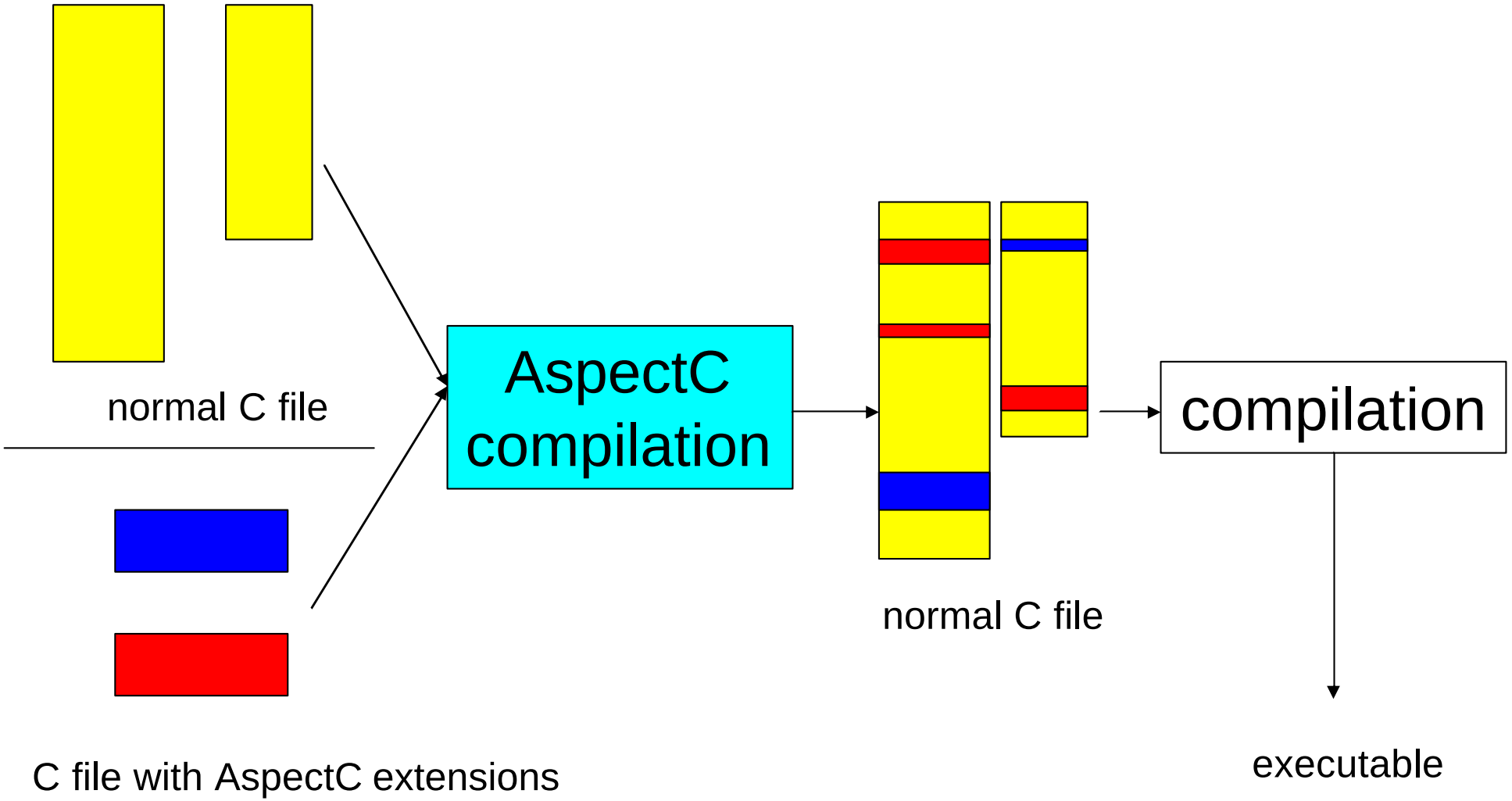
Is There a Solution?



AOP Key Idea

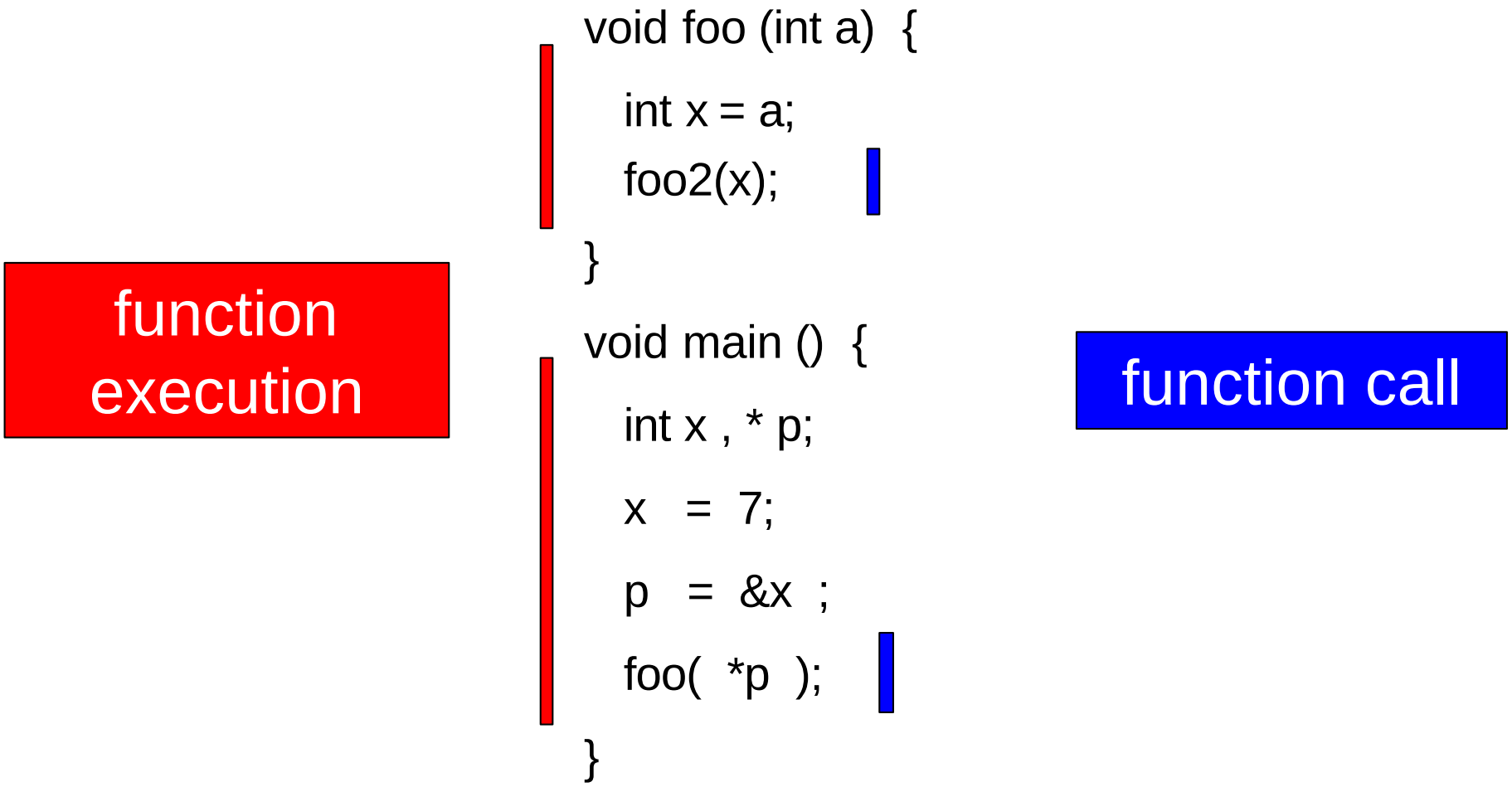


AspectC Key Idea



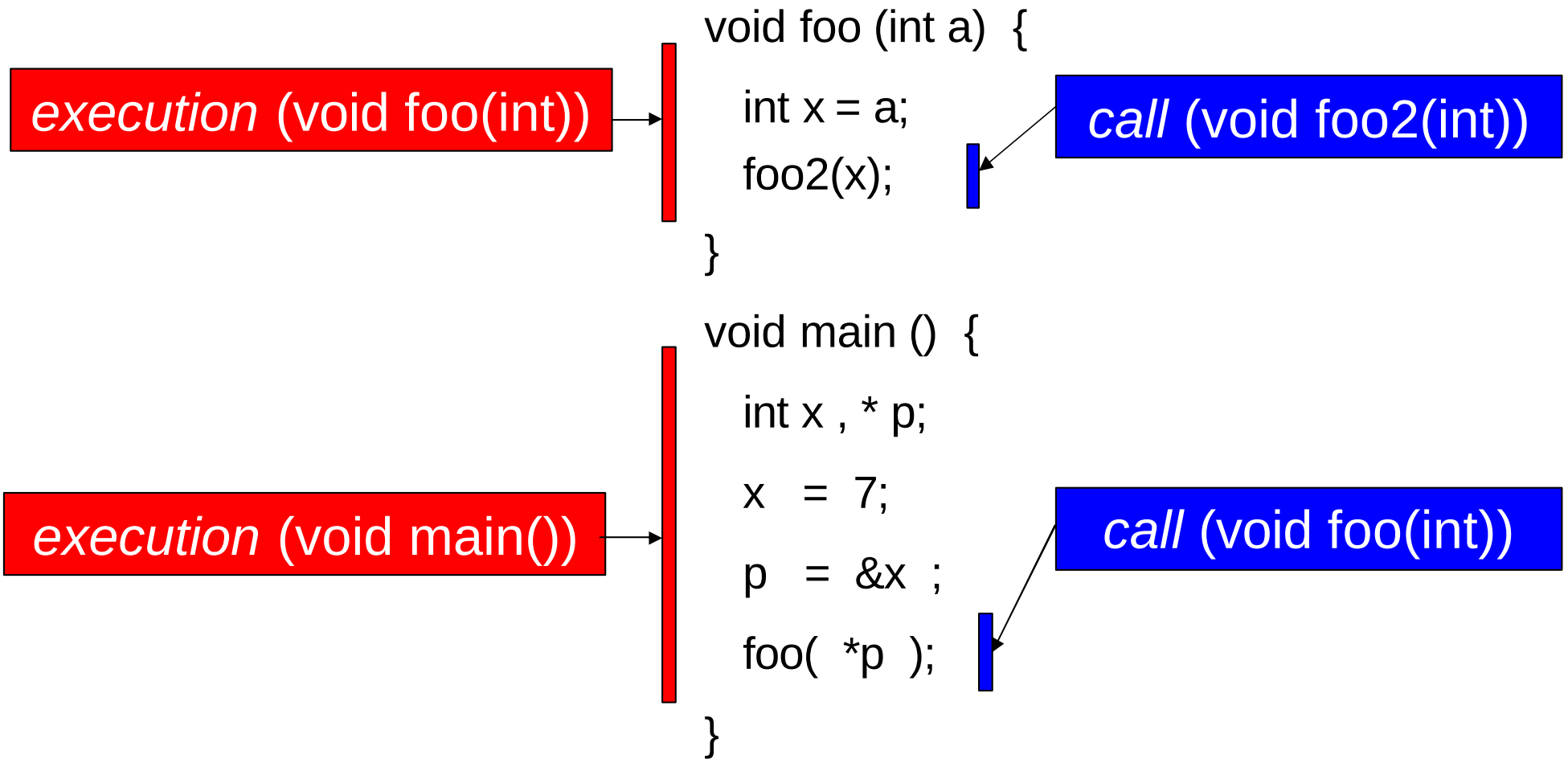
AspectC Join Point Model

- join point
  - the **location** in the **base** program where **aspects** take effect



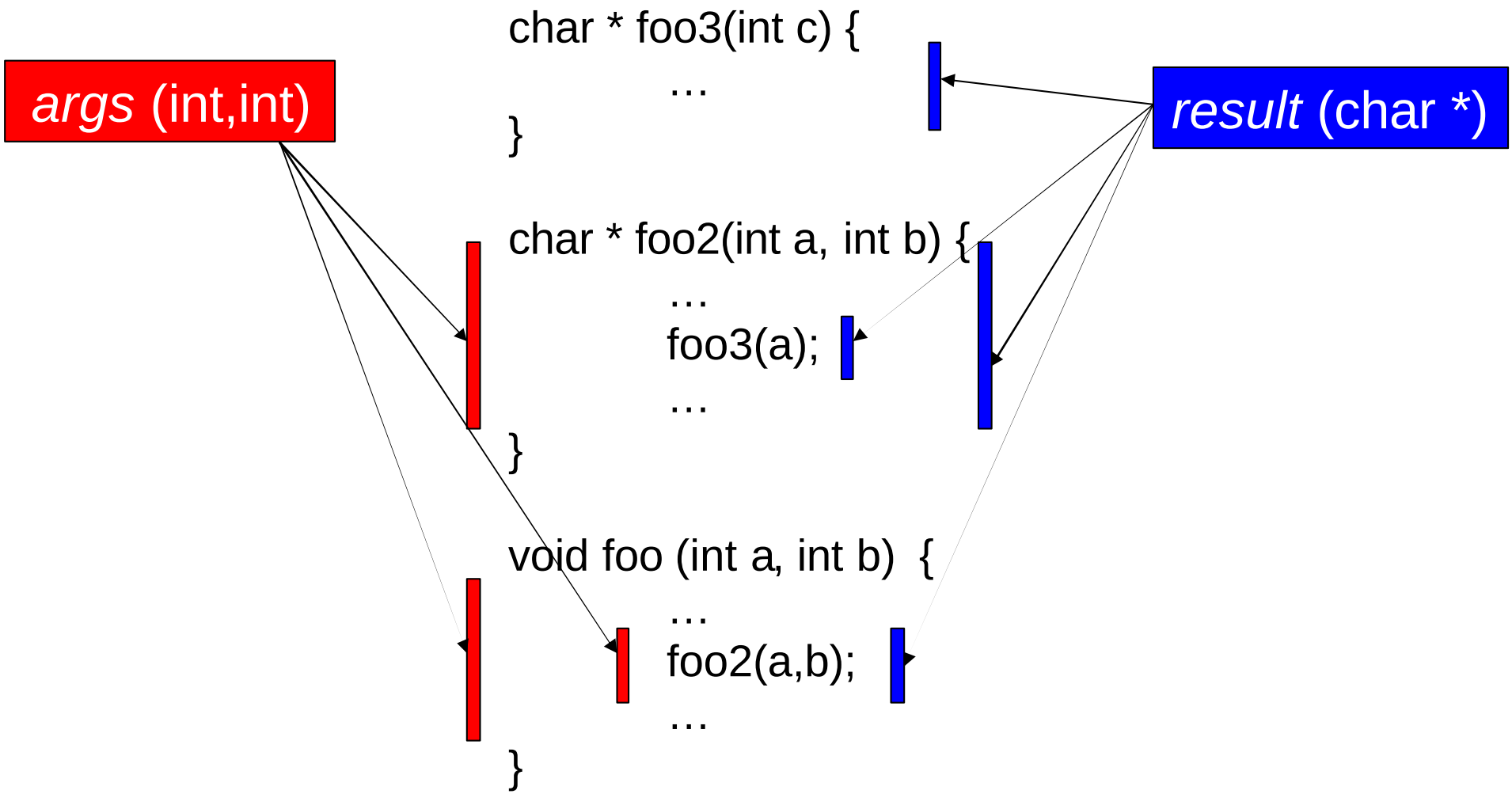
AspectC Pointcut

- pointcut
  - a language construct to denote join points



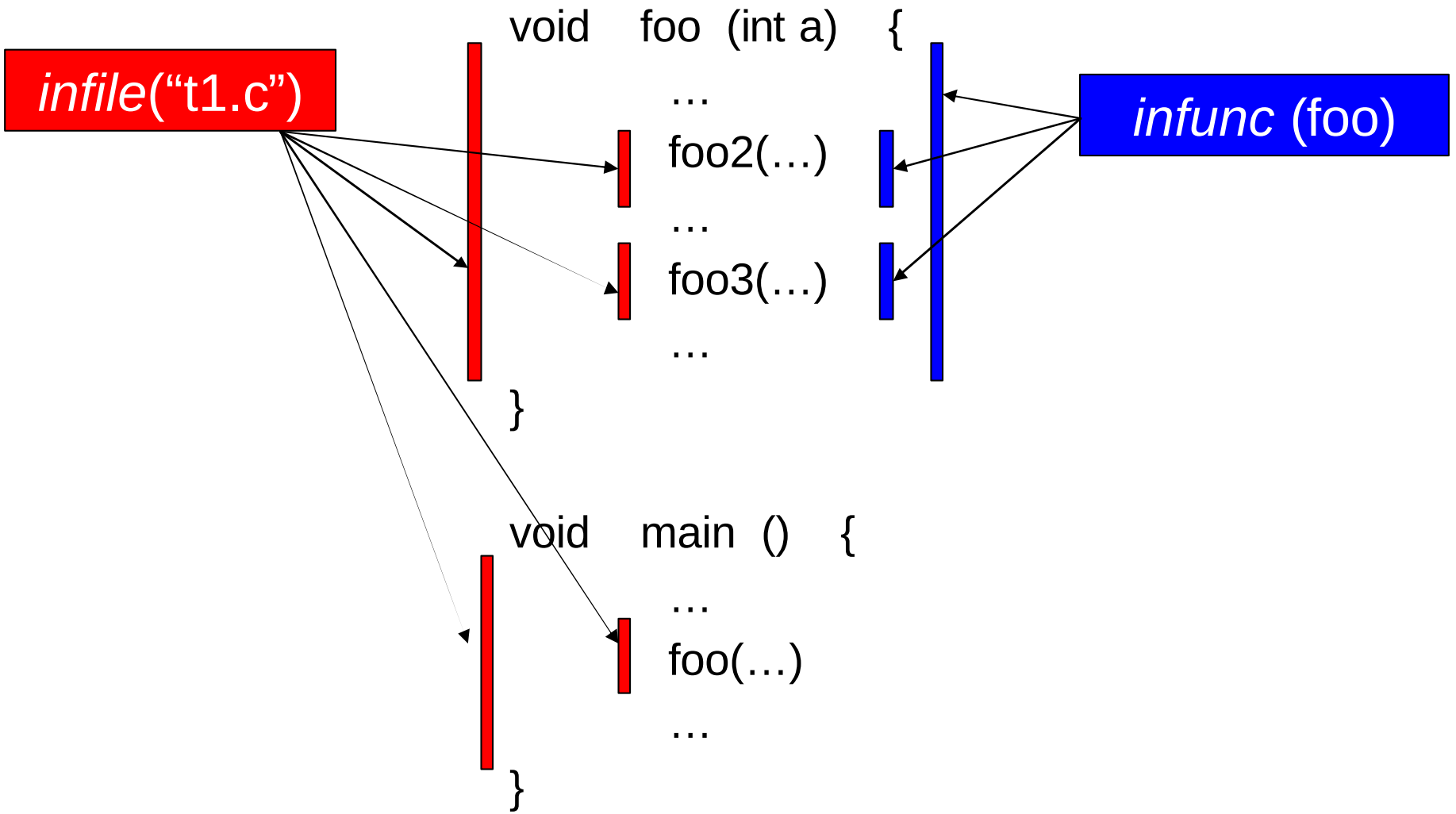
AspectC Pointcut

- *args* (paramter-type-list)
- *result* (return-type)



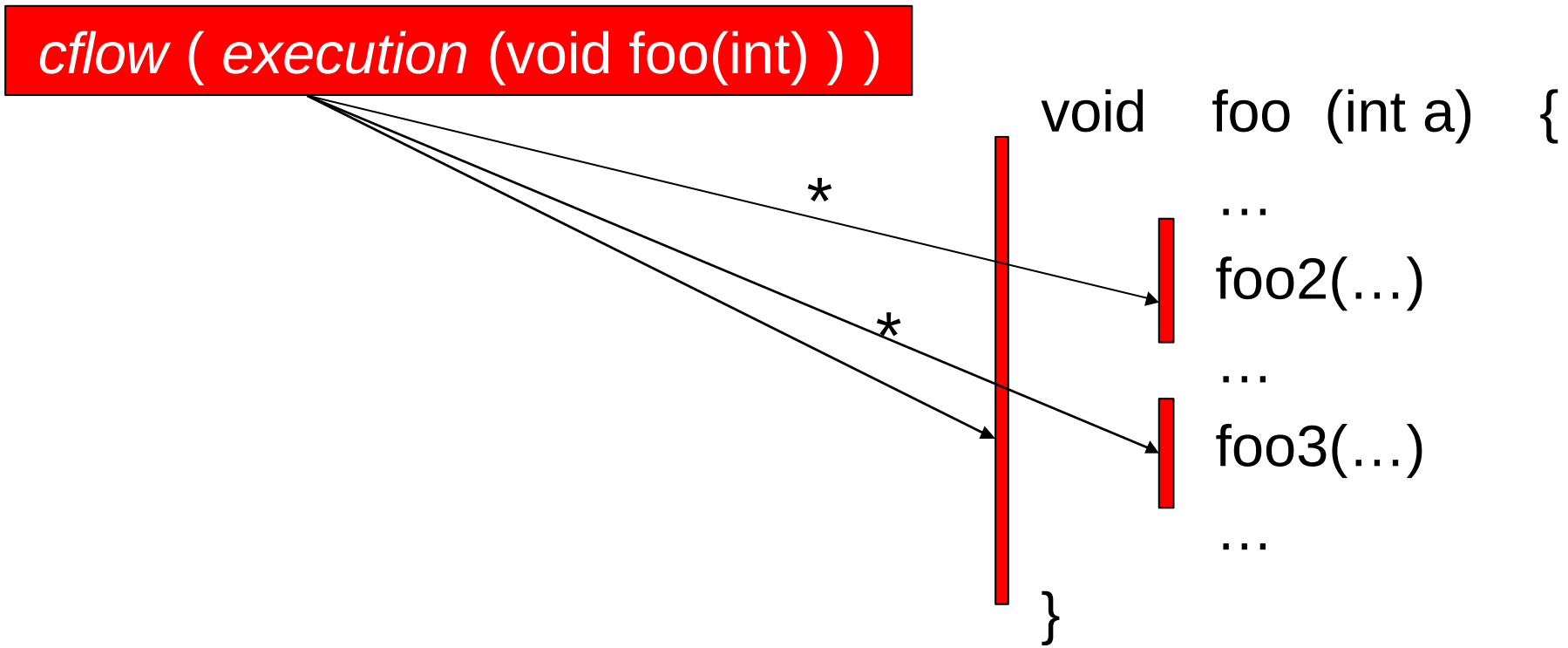
AspectC Pointcut

- *infile* (“file-name”)
- *infunc* (func-name)



AspectC Pointcut

- *cflow* (pointcut-declaration)
  - all join points happening under the control flow of other join points



* all join points happening inside foo2() or foo3() function calls

AspectC Pointcut Composition

- compose pointcuts with &&, ||, ! (i.e., *and*, *or*, and *not*)

infile (“t1.c”) && *infunc* (foo)

execution (void foo(int)) || *args* (int,int)

(*result* (char *) || *infunc* (foo)) && ! *call* (void foo2(int))

cflow (*execution* (void foo(int))) && *call* (void foo2(int))

AspectC Named Pointcut

- *pointcut* name (parameter-list) : pointcut-declaration

pointcut CallFoo2(): *call* (void foo2(int))

CallFoo2() || *args* (int,int)

cflow (*execution* (void foo(int))) && CallFoo2()

(*result* (char *) || *infunc* (foo)) && ! CallFoo2()

AspectC Pointcut Using Wildcard Character

$: match any single item

... : match any list of items

call (lon$ foo$())

call (long long foo())
call (long foo2())
call (long int foo3())
...

args (int , ... , char *)

args (int , char *)
args (int, char, char *)
args (int, int *, char *)
args (int, char, char , char *)
...

AspectC Advice

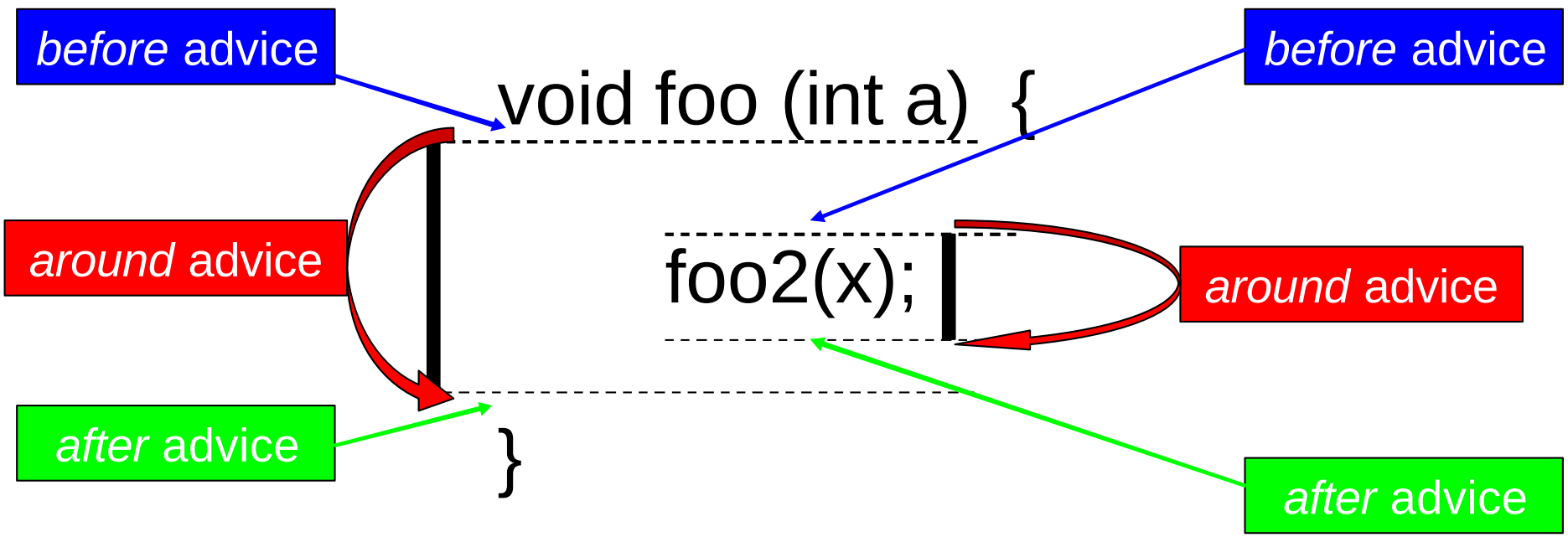
- the code to run for a pointcut

  before/after (parameter-list) : pointcut-declaration

  { //advice body }

  return-type *around* (parameter-list) : pointcut-declaration

  { //advice body }



AspectC Advice Example

```
before (): execution (void foo(int)) {
        printf(" before exec\n");
}
```

```
void foo (int a) {
        foo2(x);
}
```

```
void around ():
        execution (void foo(int)) {
                printf("around exec\n");
}
```

```
after (): execution (void foo(int)) {
        printf(" after exec\n");
}
```

AspectC Advice Example

```
before (): call (void foo2(int)) {
        printf(" before call\n");
}
```

```
void foo (int a)  {
        foo2(x);
}
```

```
void around ():
        call (void foo2(int)) {
        printf("around call\n");
}
```

```
after (): call (void foo2(int)) {
        printf(" after call\n");
}
```

AspectC Advice Example

- access argument value by using *args* ()
- access return value by using *result* ()

```
before (int i): call (void foo2(int))  && args (i) {
        printf(" before call foo2, argument = %d\n", i);
}
```

```
after (int res): call (int foo2(int)) && result (res) {
        printf(" after call foo2, return %d\n", res);
}
```

AspectC Advice Example

- around advice
  - invoke original function via *proceed* ()

```
void around ( ): call (void foo2(int)) {
        printf("around begin\n");
        proceed();
        printf("around end\n");
}
```

```
void foo2(int a) {
    printf("in foo2\n");
}
int main() {
    foo2(3);
}
```

if no *proceed()* used:
  around begin
  around end

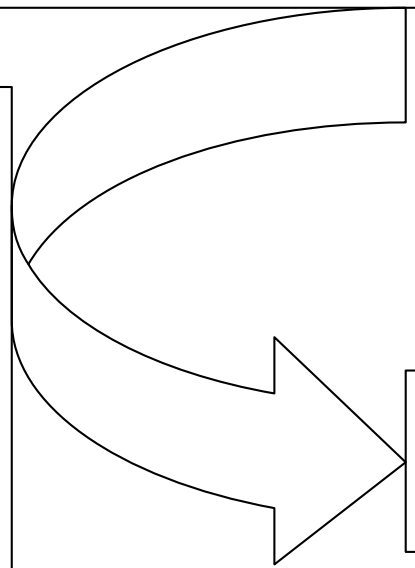
```
around begin
in foo2
around end
```

AspectC Advice Example

- reflective information about join point
  - *this->funcName*, *this->kind*

```
before ( ): call (void foo2(int)) {
        printf("before %s %s \n", this->kind, this->funcName);
}
```

```
void foo2(int a) {
   printf("in foo2\n");
}
int main() {
    foo2(3);
}
```



```
before call foo2
in foo2
```

AspectC Advice Example

- wildcard matching increases the usability

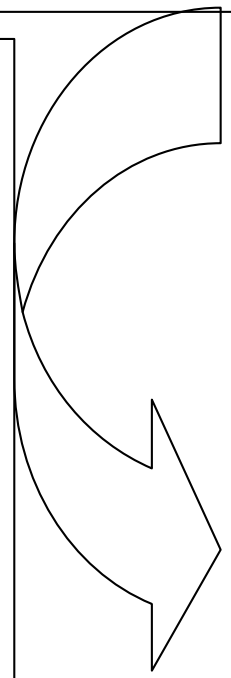
```
before ( ): call (void $(…)) {
        printf("before %s %s \n", this->kind, this->funcName);
}
```

```
void fun1() {
   printf("in fun1\n\n");
}

void foo2(int a) {
   printf("in foo2\n\n");
}

void foo3(int a, char * s) {
   printf("in foo3\n\n");
}
```

```
int main() {
    fun1();
    foo2(3);
    foo3(3, "s");
}
```



```
before call fun1
in fun1

before call foo2
in foo2

before call foo3
in foo3
```

AspectC Static Crosscutting

- add new data members to struct/union types

introduce ( ) : *intype* (type-name) {

// new member declaration

}

```
introduce ( ): intype ( struct X ) {
        double x;
        char * parent;
}
```

```
struct X {
    int a;
    char b;
};
int main() {
    printf("size of X = %d\n",
        sizeof(struct X);
}
```

```
struct X {
    int a;
    char b;
    double x;
    char * parent;
}
```

size of X = 20

Aspects in AspectC

- Aspect = a file having **AspectC extension** & **C code**

```
#include <stdio.h>
int functionCounter;
void printResult() {
  printf("total functions called = %d\n", functionCounter;
}
pointcut ExecMain(): execution (int main()) ;
before(): ExecMain() {
        functionCounter = 0;
}
before() : call($ $(…)) {
        functionCounter ++;
}
after(): ExecMain() {
        printResult();
}
```

counter

helper function

named pointcut

before any function call, increase the counter

before executing main, initialize counter

after executing main, print result

an aspect counting # of function calls

Use AspectC in Real C Code

- GNU libavl
  - a collection of binary search trees and balanced tree library routines
  - http://www.stanford.edu/~blp/avl
  - version 2.0.2a
  - mostly complete and well-documented
  - for simplification, we focus on
    - *red-black tree (RBT) routines and its correctness testing*

Trace Concern

```
int test_correctness (…, int verbosity) { …
    if (verbosity >= 2) printf ("  Inserting %d...\n", insert[i]);
   {  void **p = rb_probe (tree, &insert[i]);
      …
      if (verbosity >= 2)
          printf ("   Checking traversal from item %d...\n",
insert[i]);
       if (rb_t_find (&x, tree, &insert[i]) == NULL) {  … }
      …
       if (verbosity >= 3)
          printf ("    Deleting item %d.\n", delete[i]);
      deleted = rb_delete (tree, &delete[i]);
       ...
       if (verbosity >= 3)
           printf ("    Re-inserting item %d.\n", delete[i]);
       rb_t_insert (&z, tree, &delete[i])…
       …
        if (verbosity >= 2)
           printf ("  Deleting %d...\n", delete[i]);
        deleted = rb_delete (tree, &delete[i]);
        ...
        if (verbosity >= 2)
            printf ("  Copying tree and comparing...\n");
        { …rb_copy (tree, NULL, NULL, NULL); … } …}
```

rb-test.c

a trace concern controlled by the variable “verbosity”

* it crosscuts the core logic of this function.

* the core logic code is *polluted.*

Trace Aspect

```
#include "rb.h"

pointcut INTEST (): infunc(test_correctness);

before(void * node ): INTEST() && call($ rb_probe(...)) && args($, node) {
        printf ("  Inserting %d...\n", *(int*)(node));
}
before(void * node) : INTEST() && call($ rb_t_find(…)) && args($, $, node) {
        printf ("   Checking traversal from item %d...\n", *(int*)(node));
}
before(const void * node) : INTEST() && call($ rb_delete(…)) && args($, node)
{
        printf ("    Deleting item %d.\n", *(int*)(node));
}
before(void * node): INTEST() && call($ rb_t_insert(…)) && args($,$,node){
            printf(" Re-inserting item %d.\n“, *(int*)(node));
}
before(): INTEST() && call($ rb_copy(...)) {
            printf ("  Copying tree and comparing...\n");
}
```

Using the AspectC Compiler in Make

- core/aspect files should be preprocessed
- suffix for core file : .mc (*)
- suffix for aspect file: .ac (*)

```
trace: test.c rb.c rb-test.c trace.ac
    gcc -E test.c > test_mc.mc
    gcc -E rb.c > rb_mc.mc
    gcc -E rb-test.c > rb-test_mc.mc
    cp trace.ac trace_temp.c
    gcc -E trace_temp.c > trace_temp.ac
    acc test_mc.mc rb_mc.mc rb-test_mc.mc trace_temp.ac
    gcc -o rbtest_aspect -g test_mc.c rb_mc.c rb-test_mc.c trace_temp.c
```

preprocess

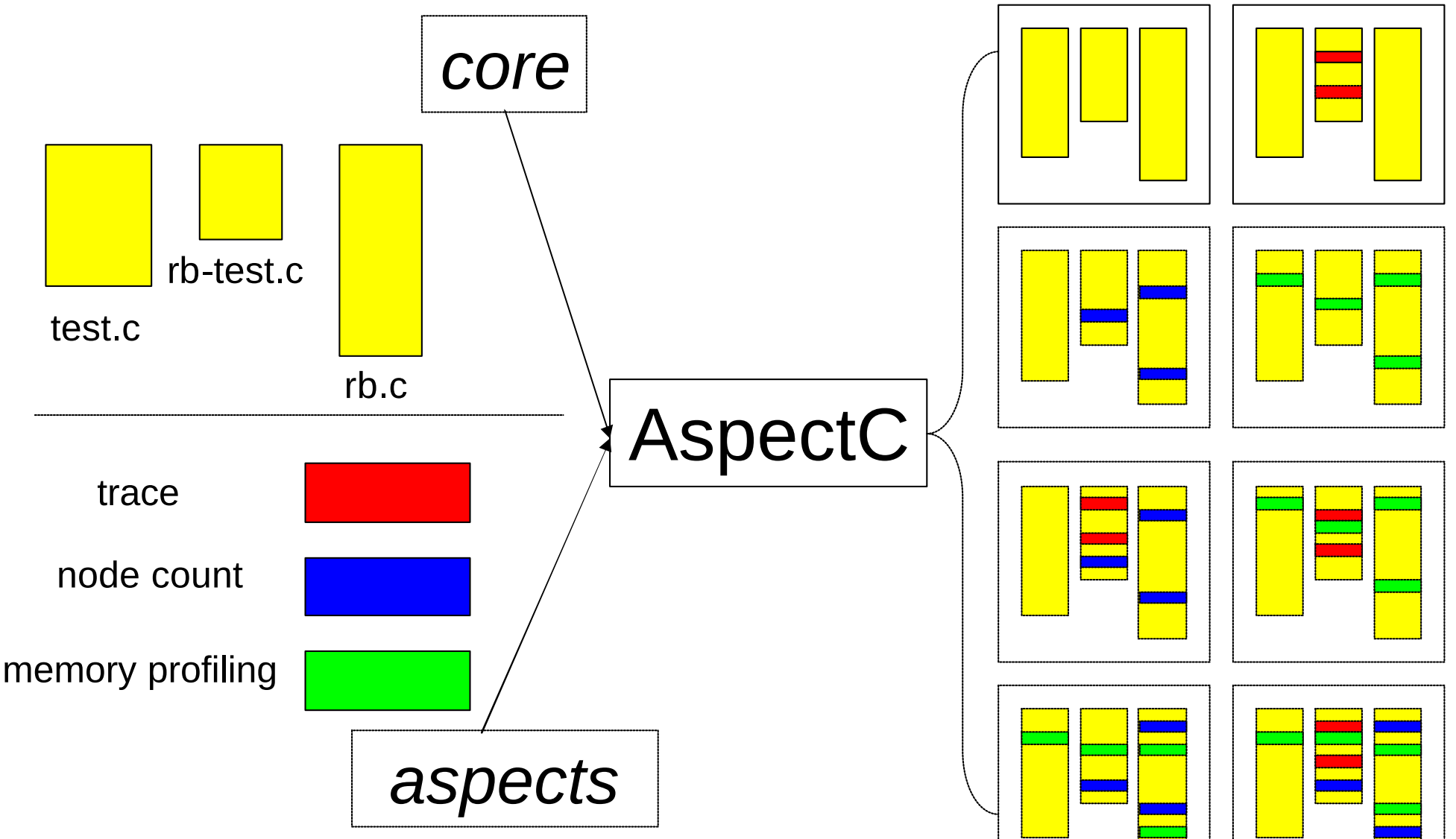
Unfortunately, gcc doesn't recognize ".ac" suffix, we have to copy it to a ".c" file ☹

AspectC compiler !

generated by AspectC compiler.

* file suffix rule might be revised in the future.

AOP-based Software Product Lines



Future Work

- AspectC Compiler
  - semantic checking
  - debugging on the original source file
  - global static crosscutting
    - introduce function/variable/header files in file scope
- Case Studies
  - thread-RBT and RBT with parent pointer in Libavl
    - use aspects because they are crosscutting concerns
  - refactor an embedded operating system: EtherNUT
  - refactor the C-based Orbit object request broker

Semantyka denotacyjna

(R1) $\langle o : Ob \mid Att : \mathrm{A}, Pr : \langle (v\ \mathrm{V} := e\ \mathrm{E}); \mathrm{S}, \mathrm{L} \rangle\ \rangle$
$\longrightarrow$ **if** v *in* L **then** $\langle o : Ob \mid Att : \mathrm{A}, Pr : \langle (\mathrm{V} := \mathrm{E}); \mathrm{S}, (\mathrm{L} + [v \mapsto eval(e, (\mathrm{A}; \mathrm{L}))]) \rangle\ \rangle$
else $\langle o : Ob \mid Att : (\mathrm{A} + [v \mapsto eval(e, (\mathrm{A}; \mathrm{L}))]), Pr : \langle (\mathrm{V} := \mathrm{E}); \mathrm{S}, \mathrm{L} \rangle\ \rangle$ **fi**

(R2) $new\ C(\mathrm{E})\ \langle C : Cl \mid Par : (\mathrm{V} : T), Att : \mathrm{A}, Tok : n \rangle$
$\longrightarrow \langle C : Cl \mid Par : (\mathrm{V} : T), Att : \mathrm{A}, Tok : next(n) \rangle$
$\langle (C; n) : Ob \mid Cl : C, Att : \epsilon, Pr : \langle (\langle self : Any = (C; n); \mathrm{V} : T := \mathrm{E}; \mathrm{A} \rangle \downarrow; run), \epsilon \rangle,$
$PrQ : \epsilon, EvQ : \epsilon, Lab : 1 \rangle$

(R3) $\langle o : Ob \mid Att : \mathrm{A}, Pr : \langle (v := new\ C(\mathrm{E}); \mathrm{S}), \mathrm{L} \rangle\ \rangle$
$\langle C : Cl \mid Par : (\mathrm{V} : T), Att : \mathrm{A}', Tok : n \rangle$
$\longrightarrow \langle o : Ob \mid Att : \mathrm{A}, Pr : \langle (v := (C; n); \mathrm{S}), \mathrm{L} \rangle\ \rangle\ \langle (C; n) : Ob \mid Cl : C, Att : \epsilon,$
$Pr : \langle (\langle self : Any = (C; n); \mathrm{V} : T = eval(\mathrm{E}, (\mathrm{A}; \mathrm{L})); \mathrm{A}' \rangle \downarrow; run), \epsilon \rangle, PrQ : \epsilon,$
$EvQ : \epsilon, Lab : 1 \rangle\ \langle C : Cl \mid Par : (\mathrm{V} : T), Att : \mathrm{A}', Tok : next(n) \rangle$

(R4) $\langle o : Ob \mid Att : \mathrm{A}, Pr : \langle await\ g; \mathrm{S}, \mathrm{L} \rangle, EvQ : \mathrm{Q} \rangle$
$\longrightarrow \langle o : Ob \mid Att : \mathrm{A}, Pr : \langle \mathrm{S}, \mathrm{L} \rangle, EvQ : \mathrm{Q} \rangle$ **if** $enabled(g, (\mathrm{A}, \mathrm{L}), \mathrm{Q})$

(R5) $\langle o : Ob \mid Att : \mathrm{A}, Pr : \langle \mathrm{S}, \mathrm{L} \rangle, PrQ : \mathrm{W}, EvQ : \mathrm{Q} \rangle$
$\longrightarrow \langle o : Ob \mid Att : \mathrm{A}, Pr : \langle \epsilon, \epsilon \rangle, PrQ : (\mathrm{W}\ \langle clear(\mathrm{S}), \mathrm{L} \rangle), EvQ : \mathrm{Q} \rangle$
if *not* $enabled(\mathrm{S}, (\mathrm{A}; \mathrm{L}), \mathrm{Q})$

(R6) $\langle o : Ob \mid Att : \mathrm{A}, Pr : \langle \epsilon, \mathrm{L}' \rangle, PrQ : \langle \mathrm{S}, \mathrm{L} \rangle\ \mathrm{W}, EvQ : \mathrm{Q} \rangle$
$\longrightarrow \langle o : Ob \mid Att : \mathrm{A}, Pr : \langle \mathrm{S}, \mathrm{L} \rangle, PrQ : \mathrm{W}, EvQ : \mathrm{Q} \rangle$ **if** $enabled(\mathrm{S}, (\mathrm{A}, \mathrm{L}), \mathrm{Q})$

(R7) $\langle o : Ob \mid Pr : \langle (r(Sig, Co, \mathrm{E}; \mathrm{V}); \mathrm{S}), \mathrm{L} \rangle, Lab : n \rangle$
$\longrightarrow \langle o : Ob \mid Pr : \langle (!r(Sig, Co, \mathrm{E}); n?(\mathrm{V}); \mathrm{S}), \mathrm{L} \rangle, Lab : n \rangle$

(R8) $\langle o : Ob \mid Att : \mathrm{A}, Pr : \langle (t!r(Sig, Co, \mathrm{E}); \mathrm{S}), \mathrm{L} \rangle, Lab : n \rangle$
$\longrightarrow \langle o : Ob \mid Att : \mathrm{A}, Pr : \langle (t := n; !r(Sig, Co, \mathrm{E}); \mathrm{S}), \mathrm{L} \rangle, Lab : n \rangle$

(R9) $\langle o : Ob \mid Att : \mathrm{A}, Pr : \langle (!x.m(Sig, Co, \mathrm{E}); \mathrm{S}), \mathrm{L} \rangle, Lab : n \rangle$
$\longrightarrow \langle o : Ob \mid Att : \mathrm{A}, Pr : \langle \mathrm{S}, \mathrm{L} \rangle, Lab : next(n) \rangle$
$invoc(m, Sig, Co, (o\ n\ eval(\mathrm{E}, (\mathrm{A}; \mathrm{L}))))$ **to** $eval(x, (\mathrm{A}; \mathrm{L}))$

(R10) $\langle o : Ob \mid Att : \mathrm{A}, Pr : \langle (!m(Sig, Co, \mathrm{E}); \mathrm{S}), \mathrm{L} \rangle, Lab : n \rangle$
$\longrightarrow \langle o : Ob \mid Att : \mathrm{A}, Pr : \langle \mathrm{S}, \mathrm{L} \rangle, Lab : next(n) \rangle$
$invoc(m, Sig, Co, (o\ n\ eval(\mathrm{E}, (\mathrm{A}; \mathrm{L}))))$ **to** o

(R11) $(msg$ **to** $o)\ \langle o : Ob \mid EvQ : \mathrm{Q} \rangle \longrightarrow \langle o : Ob \mid EvQ : \mathrm{Q}\ msg \rangle$

(R12) $\langle o : Ob \mid Cl : C, EvQ : \mathrm{Q}\ invoc(m, Sig, Co, \mathrm{E}) \rangle$
$\longrightarrow \langle o : Ob \mid Cl : C, EvQ : \mathrm{Q} \rangle\ bind(m, Sig, Co, \mathrm{E}, o)$ **to** C

(R13) $\langle o : Ob \mid Pr : \langle (t\,?\,(\mathrm{V}); \mathrm{S}), \mathrm{L} \rangle, EvQ : \mathrm{Q}\ comp(n, \mathrm{E}) \rangle$
$\longrightarrow \langle o : Ob \mid Pr : \langle (\mathrm{V} := \mathrm{E}; \mathrm{S}), \mathrm{L} \rangle, EvQ : \mathrm{Q} \rangle$ **if** $n = eval(t, \mathrm{L})$

(R14) $\langle o : Ob \mid Pr : \langle (t\,?\,(\mathrm{V}); \mathrm{S}), \mathrm{L} \rangle, PrQ : \langle \mathrm{S}', \mathrm{L}' \rangle\ \mathrm{W} \rangle$
$\longrightarrow \langle o : Ob \mid Pr : \langle \mathrm{S}'; cont(eval(t, \mathrm{L})), \mathrm{L}' \rangle, PrQ : \langle await\ t\,?\,(\mathrm{V}); \mathrm{S}, \mathrm{L} \rangle\ \mathrm{W} \rangle$
if $eval(caller, \mathrm{L}') = o \wedge eval(label, \mathrm{L}') = eval(t, \mathrm{L})$

(R15) $\langle o : Ob \mid Pr : \langle cont(n), \mathrm{L} \rangle, PrQ : \langle (await\ (t'?); \mathrm{S}), \mathrm{L}' \rangle\ \mathrm{W} \rangle$
$\longrightarrow \langle o : Ob \mid Pr : \langle \mathrm{S}, \mathrm{L}' \rangle, PrQ : \mathrm{W} \rangle$ **if** $eval(t', \mathrm{L}') = n$

(R16) $(bind(m, Sig, Co, \mathrm{E}, o)$ **to** $C)\ \langle C : Cl \mid Mtds : \mathrm{M} \rangle$
$\longrightarrow$ **if** $match(m, Sig, Co, \mathrm{M})$
then $(bound(get(m, \mathrm{M}, \mathrm{E}))$ **to** $o)$
else $(bind(m, Sig, Co, \mathrm{E}, o)$ **to** $\epsilon)$ **fi** $\langle C : Cl \mid Mtds : \mathrm{M} \rangle$

(R17) $(bound(w)$ **to** $o)\ \langle o : Ob \mid PrQ : \mathrm{W} \rangle \longrightarrow \langle o : Ob \mid PrQ : \mathrm{W}\ w \rangle$

(R18) $\langle o : Ob \mid Att : \Lambda, Pr : \langle (return(\mathrm{V}); \mathrm{P}), \mathrm{L} \rangle \rangle$
$\longrightarrow \langle o : Ob \mid Att : \mathrm{A}, Pr : \langle \mathrm{P}, \mathrm{L} \rangle \rangle$
$comp(eval(label, \mathrm{L}), eval(\mathrm{V}, (\mathrm{L})))$ **to** $eval(caller, \mathrm{L})$

Fig. 8. An operational semantics in rewriting logic.

Semantyka operacyjna

$$\text{(R2')}\quad \begin{array}{l} \mathit{new}\ C(\mathrm{E})\ \langle C : \mathit{Cl} \mid \mathit{Tok} : n\rangle \\ \quad\longrightarrow\ \langle (C;n) : \mathit{Ob} \mid \mathit{Cl}: C, \mathit{Att} : \epsilon, \mathit{Pr} : \langle \epsilon, \epsilon\rangle, \mathit{PrQ}: \epsilon, \mathit{EvQ} : \epsilon, \mathit{Lab} : 1\rangle \\ \qquad \langle C : \mathit{Cl} \mid \mathit{Tok} : \mathit{next}(n)\rangle \\ \qquad \mathit{inherit}((C;n), \mathit{self} : \mathit{Any} = (C;n), \epsilon)\ \mathbf{to}\ C(\mathit{eval}\,(\mathrm{E}, (\mathrm{A};\mathrm{L}))) \end{array}$$

$$\text{(R3')}\quad \begin{array}{l} \langle o : \mathit{Ob} \mid \mathit{Att} : \mathrm{A}, \mathit{Pr} : \langle (v := \mathit{new}\ C(\mathrm{E}); \mathrm{S}), \mathrm{L}\rangle\rangle\ \langle C : \mathit{Cl} \mid \mathit{Tok} : n\rangle \\ \quad\longrightarrow\ \langle o : \mathit{Ob} \mid \mathit{Att} : \mathrm{A}, \mathit{Pr} : \langle (v := (C;n); \mathrm{S}), \mathrm{L}\rangle\rangle\ \langle C : \mathit{Cl} \mid \mathit{Tok} : \mathit{next}(n)\rangle \\ \qquad \langle (C;n) : \mathit{Ob} \mid \mathit{Cl} : C, \mathit{Att} : \epsilon, \mathit{Pr} : \langle \epsilon, \epsilon\rangle, \mathit{PrQ} : \epsilon, \mathit{EvQ} : \epsilon, \mathit{Lab} : 1\rangle \\ \qquad \mathit{inherit}((C;n), \mathit{self} : \mathit{Any} = (C;n), \epsilon)\ \mathbf{to}\ C(\mathit{eval}\,(\mathrm{E}, (\mathrm{A};\mathrm{L}))) \end{array}$$

$$\text{(R10')}\quad \begin{array}{l} \langle o : \mathit{Ob} \mid \mathit{Att} : \mathrm{A}, \mathit{Pr} : \langle\, !mq(\mathit{Sig}, \mathit{Co}, \mathrm{E}); \mathrm{S}, \mathrm{L}\rangle, \mathit{Lab} : n\rangle \\ \quad\longrightarrow\ \langle o : \mathit{Ob} \mid \mathit{Att} : \mathrm{A}, \mathit{Pr} : \langle \mathrm{S}, \mathrm{L}\rangle, \mathit{Lab} : \mathit{next}(n)\rangle \\ \qquad \mathit{invoc}(mq, \mathit{Sig}, \mathit{Co}, (n\ o\ \mathit{eval}(\mathrm{E}, (\mathrm{A};\mathrm{L}))))\ \mathbf{to}\ o \end{array}$$

$$\text{(R24)}\quad \begin{array}{l} \langle o : \mathit{Ob} \mid \mathit{EvQ} : \mathit{invoc}(m@C, \mathit{Sig}, \mathit{Co}, \mathrm{E})\ \mathrm{Q}\rangle \\ \quad\longrightarrow \langle o : \mathit{Ob} \mid \mathit{EvQ} : \mathrm{Q}\rangle\ (\mathit{bind}(m, \mathit{Sig}, \mathit{Co}, \mathrm{E}, o)\ \mathbf{to}\ C) \end{array}$$

$$\text{(R25)}\quad \begin{array}{l} \langle o : \mathit{Ob} \mid \mathit{Cl} : C, \mathit{EvQ} : \mathit{invoc}(m < C', \mathit{Sig}, \mathit{Co}, \mathrm{E})\ \mathrm{Q}\rangle \\ \quad\longrightarrow \langle o : \mathit{Ob} \mid \mathit{Cl} : C, \mathit{EvQ} : \mathrm{Q}\rangle\ (\mathit{bind}(m < C', \mathit{Sig}, \mathit{Co}, \mathrm{E}, o)\ \mathbf{to}\ C) \end{array}$$

$$\text{(R16')}\quad \begin{array}{l} (\mathit{bind}(m, \mathit{Sig}, \mathit{Co}, \mathrm{E}, o)\ \mathbf{to}\ C\ \mathrm{I})\ \langle C : \mathit{Cl} \mid \mathit{Inh} : \mathrm{I}', \mathit{Mtds} : \mathrm{M}\rangle \\ \quad\longrightarrow\ \mathbf{if}\ \mathit{match}(m, \mathit{Sig}, \mathit{Co}, \mathrm{M}) \\ \qquad \mathbf{then}\ \mathit{bound}(\mathit{get}(m, \mathrm{M}, \mathrm{E}))\ \mathbf{to}\ o \\ \qquad \mathbf{else}\ \mathit{bind}(m, \mathit{Sig}, \mathit{Co}, \mathrm{E}, o)\ \mathbf{to}\ (\mathrm{I}'\ \mathrm{I})\ \mathbf{fi}\ \langle C : \mathit{Cl} \mid \mathit{Inh} : \mathrm{I}', \mathit{Mtds} : \mathrm{M}\rangle \end{array}$$

$$\text{(R26)}\quad \begin{array}{l} (\mathit{bind}(m < C', \mathit{Sig}, \mathit{Co}, \mathrm{E}, o)\ \mathbf{to}\ C\ \mathrm{I})\ \langle C : \mathit{Cl} \mid \mathit{Inh} : \mathrm{I}', \mathit{Mtds} : \mathrm{M}, \mathit{Tok} : n\rangle \\ \quad\longrightarrow\ \langle C : \mathit{Cl} \mid \mathit{Inh} : \mathrm{I}', \mathit{Mtds} : \mathrm{M}, \mathit{Tok} : \mathit{next}(n)\rangle \\ \qquad (\mathbf{if}\ \mathit{match}(m, \mathit{Sig}, \mathit{Co}, \mathrm{M}) \\ \qquad \mathbf{then}\ (\mathit{find}(n, C', C)\ \mathbf{to}\ C)\ (\mathit{stopbind}(n, m < C', \mathit{Sig}, \mathit{Co}, \mathrm{E}, o)\ \mathbf{to}\ C\ \mathrm{I}) \\ \qquad \mathbf{else}\ \mathit{bind}(m < C', \mathit{Sig}, \mathit{Co}, \mathrm{E}, o)\ \mathbf{to}\ (\mathrm{I}'\ \mathrm{I})\ \mathbf{fi}) \end{array}$$

$$\text{(R27)}\quad \begin{array}{l} (\mathit{found}(n, b, C')\ \mathbf{to}\ C)\ (\mathit{stopbind}(n, m, \mathit{Sig}, \mathit{Co}, \mathrm{E}, o)\ \mathbf{to}\ C\ \mathrm{I}) \\ \langle C : \mathit{Cl} \mid \mathit{Inh} : \mathrm{I}', \mathit{Mtds} : \mathrm{M}\rangle \\ \quad\longrightarrow\ \mathbf{if}\ b\ \mathbf{then}\ \mathit{bound}(\mathit{get}(m, \mathrm{M}, \mathrm{E}))\ \mathbf{to}\ o\ \mathbf{else}\ \mathit{bind}(m, \mathit{Sig}, \mathit{Co}, \mathrm{E}, o)\ \mathbf{to}\ \mathrm{I}\ \mathbf{fi} \\ \qquad \langle C : \mathit{Cl} \mid \mathit{Inh} : \mathrm{I}', \mathit{Mtds} : \mathrm{M}\rangle \end{array}$$

$$\text{(R28)}\quad \mathit{find}(n, C, C'')\ \mathbf{to}\ \epsilon\ \longrightarrow\ \mathit{found}(n, \mathit{false}, C)\ \mathbf{to}\ C''$$

$$\text{(R29)}\quad \mathit{find}(n, C, C'')\ \mathbf{to}\ \mathrm{I}\ C\ \mathrm{I}'\ \longrightarrow\ \mathit{found}(n, \mathit{true}, C)\ \mathbf{to}\ C''$$

$$\text{(R30)}\quad \begin{array}{l} (\mathit{find}(n, C, C'')\ \mathbf{to}\ C'\ \mathrm{I})\ \langle C' : \mathit{Cl} \mid \mathit{Inh} : \mathrm{I}'\rangle \\ \quad\longrightarrow\ (\mathit{find}(n, C, C'')\ \mathbf{to}\ \mathrm{I}\ \mathrm{I}')\ \langle C' : \mathit{Cl} \mid \mathit{Inh} : \mathrm{I}'\rangle\ \mathbf{if}\ (C \neq C') \end{array}$$

Fig. 16. An operational semantics with multiple inheritance. Note that the rules R2′, R3′, R10′, and R16′ redefine the previous rules R2, R3, R10,