

Re: left-to-right (was In-Out Parameters for functions)

Source: <http://coding.derkeiler.com/Archive/Ada/comp.lang.ada/2004-02/0734.html>

From: Dmitry A. Kazakov (mailbox_at_dmitry-kazakov.de)

Date: 02/27/04

Date: Fri, 27 Feb 2004 10:43:25 +0100

On 26 Feb 2004 23:58:22 -0500, Stephen Leake <Stephe.Leake@nasa.gov> wrote:

>Dmitry A.Kazakov <mailbox@dmitry-kazakov.de> writes:

>

>> On 24 Feb 2004 19:44:12 -0500, Stephen Leake <stephen_leake@acm.org>

>> wrote:

>>

>> >I think Hyman has two valid points:

>> >

>> >1) if the language specified left-to-right order, there would never be

>> > a need to "kill the original smart-ass programmer"

>>

>> >But also, if the language disallowed side-effects when there could be

>> > more than one order. I prefer the second, provided that the term

>> > "side-effect" should be formally defined by the language and visible

>> > from the specifications.

>>

>> >That's never going to happen!

Alas.

>> >2) Does anyone have a real example of a compiler taking advantage

>> > of the evaluation order freedom to speed up a program?

>>

>> >The question is what gets lost if an evaluation order is fixed. I gave

>> > an example earlier:

>>

>> >for I in A'Range loop

>> > Sum := Sum + A (I) + B (B'First + I - A'First);

>> > ...

>>

>> >Here with left-to-right order the compiler cannot optimize neither the

>> > array index of B,

>>

>> >What do you mean by "optimize" in this sentence? the array index of B

comp.lang.ada: Re: left-to-right (was In-Out Parameters for functions)

>is $B'First + I - A'First$. It must be computed before $B()$. In what way
>would this be done differently by a "real Ada" compiler vs a
>"left-to-right" Ada compiler?

A left-to-right compiler shall first evaluate $B'First + I$, then decrement it by $A'First$.

What I would expect from a good compiler is factoring out the cycle constant $B'First - A'First$ and omitting all index checks for B when $A'Length = B'Length$. I am not sure if that will be possible for a strict left-to-right compiler without dealing with program semantics.

>> nor can it increment Sum by adding $A()+B()$.
>
>True; it must compute $temp := Sum + A()$, then $Sum := temp + B()$. What
>difference does that make, in actual practice? How many compilers do
>anything else?
>
>Hmm. For scalars, if A and B happen to be in registers, and Sum in
>ram, I can see compilers messing with the order. Can anyone confirm
>that actually happens in real compilers? It's been a while since I
>messed with gcc code generation.

It is a pity that Robert Dewar isn't here.

>> Well, these optimizations could be made possible if the compiler
>> might prove that the result does not depend on the order. That could
>> be quite difficult to formally define, especially regarding numeric
>> exceptions and floating-point accuracy stuff. Even more difficult it
>> would be check.
>
>That is precisely the point. Since it is difficult to formally
>define and check the meaning of "does not depend on order", why do we
>expect people to be able to do it?

Exactly, they should not! People have to write programs so that they would be valid for any allowed order. If they don't the code is broken. In my view it is broken *even* if the language defined implicit order is the one assumed by the programmer.

My point is: either 1) the order is specified explicitly by the programmer, or 2) the code is valid for any order.

>On the other hand, that's also why it's not likely such restrictions
>will be added to the definition of "pure" functions in Ada.

There is no need to check program semantics very precisely. The language may simply outlaw "pure" functions like:

```
X : Integer;  
function Pure return Integer is
```

Re: left-to-right (was In-Out Parameters for functions)

```
-- pragma Pure (Pure);  
begin  
  return X - X; -- Does not depend on X  
end Pure;
```

>> *But finally, if all that could be achieved, then the compiler will be
>> capable to simply detect that different orders give different
>> results, and treat it as an error. Much better than fixing an
>> arbitrary order!*
>
>*Why is that better? The goal is to have code that works (meaning
>produces correct results :), on all compilers/platforms.*

But the code has some semantics it implements, at least I hope it is so (-:). This semantics is either order-dependent or not. That does not depend on platform. The goal is to implement that semantics. The rest does Ada.

> *If I only
> have to prove that one evaluation order is correct, that is easier
> than if I have to think about many evaluation orders.*

True. Weaker the precondition, harder to write the code. It is quite easy to write sqrt provided that the only valid value of the argument is 0. Alas, that would be not what people understand under sqrt. From software design point of view it is always worth to try to write order-independent code, like one with a weakest precondition. The opposite should be treated rather as an exception.

>> *So the bottom line, IMO, I makes no sense.*
>
>*But you haven't answered my question. Is there, in actual fact, any
>compiler that actually does these optimizations?*

Randy Brukardt gave one example. However I would like to stress that optimization and implementation freedom is only one, though important argument. Software design practice is no less important. Fixing order not imposed by the semantics, and thus an arbitrary order, is that sort of premature optimization Knuth wrote about. My concern is that fixing an arbitrary order would bless bad software design in favor of getting results now.

>> *>Ada is supposed to be about clear, unsurprising code. Subtle order
>> >issues are just that - "subtle". If the language _could_ make them a
>> >non-issue, at very little cost, I think it _should_.
>>
>> Right, the present situation is not good. But IMO the solution lies in
>> introducing formally pure functions and imposing limitations on use of
>> impure functions (and procedures with results) in expressions.
>
>*That's way more work than simply requiring left-to-right, and**

comp.lang.ada: Re: left-to-right (was In-Out Parameters for functions)

>therefore far less likely to happen.

AFAIK Ravenscar profile would be a part of the standard. A time may come when some SPARK ideas will be evaluated too.

--

Regards,

Dmitry A. Kazakov

www.dmitry-kazakov.de