

Re: Continuations in Common Lisp (with apologies)

Source: <http://coding.derkeiler.com/Archive/Lisp/comp.lang.lisp/2005-02/0791.html>

From: William D Clinger (*cesuraSPAM_at_verizon.net*)

Date: 02/11/05

Date: 10 Feb 2005 17:15:03 -0800

Kent Pitman quoting me:

> > *Control effects, like side effects, should be documented.*

>

> *But sometimes they are not.*

We all know it is possible to write buggy software. The documentation is part of the software, and can be buggy by commission or omission.

The question is not whether it is possible to write buggy software, but whether it is possible and practical to write software that reliably performs UNWIND-PROTECT-like operations in a language that also has full continuations.

Many people, over many years, have demonstrated that the answer is "yes". My CL-style definition of UNWIND-PROTECT reliably performs all UNWIND-PROTECT-like operations that can be expressed in Common Lisp, while signalling a runtime error when it detects a class of control effects that cannot occur in Common Lisp.

If you wish to allow the control effects that my CL-style definition of UNWIND-PROTECT protects against, then you can use a Pitman-style definition of UNWIND-PROTECT. This style is also implementable in portable Scheme, without having to redefine any of Scheme's standard procedures.

If you wish to allow some but not all control effects that my CL-style definition protects against, then you can design a Pitman-style definition for the specific needs of your specific program. This is seldom necessary.

> *There are tools that I've seen written in Scheme that do*
> *backtracking, for example, by checkpointing the state at*
> *various points in ways that are not easily explained to users.*
> *In fact, the value of the tool is that it hides the control*

> *aspects and just tells you to rely on the magic of the*
> *abstraction.*

Yes. For this to work, the tool may have to redefine one or two of Scheme's standard procedures. The Scheme standards explicitly allow this. In my view, any tool that purports to hide the control aspects in some magic way should also assume the burden of implementing that magic. The specific magic you mentioned can be and has been implemented in portable Scheme.

> *As I'm reading it, you SEEM to expect programs containing*
> *UNWIND-PROTECT to be case-marked in a way that says "maybe you*
> *don't want to go here".*

No. I expect one specific control effect, namely escaping continuations, to be documented. As I explain below, the ordinary standards of documentation that are expected in languages like Java already imply that escaping continuations should be documented.

In my view, the purpose of UNWIND-PROTECT is to prevent certain bad things from happening. In Common Lisp, UNWIND-PROTECT prevents the body from performing a throw that bypasses the unwind forms. My CL-style definition of UNWIND-PROTECT does that, and also prevents the body from being continued after the unwind forms have been executed.

Programmers who want to prevent other things as well, or to allow some subset of the things that my CL-style definition prevents, can implement whatever semantics they desire. If that kind of power is too frightening, then the programmer should use someone else's abstraction.

Scheme can and should be criticized for providing too few predefined abstractions, but in this case it does provide predefined abstractions that are powerful enough to build most of the control abstractions that programmers would want, and far more than could be implemented in most other languages.

> *But I do find myself nervous about this matter.*

Relax. We now have twenty years of experience with DYNAMIC-WIND and its interactions with full continuations. We have learned, sometimes the hard way, what works and what doesn't. Most of the problems that you and others here have dreamed up are not real. A few, mainly continuation-based multithreading, coroutines, or backtracking, are real but can be, should be, and are solved by the implementors of the abstraction, so ordinary programmers don't have to worry about the control aspects.

They still have to worry about the side effects, but that's true in any such system, independent of full continuations. For example, any backtracking system is going to require programmers to guard against closing a file and then backtracking into a computation that expects the file to be open.

> *I think you're right that control abstractions need to be documented, but CALL/CC is a control abstraction. So maybe you're saying that all uses of CALL/CC should be documented. ;)*

I see the smiley wink. CALL-WITH-CURRENT-CONTINUATION and DYNAMIC-WIND are control abstractions; they are documented in the Scheme standards. Not all uses of control abstractions are control effects that require documentation, just as not all assignments are externally visible side effects that require documentation.

Most uses of CALL-WITH-CURRENT-CONTINUATION are for escapes and other things that cause no more trouble in Scheme than in Common Lisp. The only uses that could possibly cause a problem that does not arise in Common Lisp are uses in which a continuation escapes its dynamic extent. Such escapes can happen only when a continuation is (1) returned, (2) passed to a procedure, (3) stored in a variable or structure whose lifetime exceeds the continuation's dynamic extent, or (4) becomes a component of some structure (e.g. list or closure) that escapes.

Every externally visible procedure's arguments and result should be documented, as should all externally visible side effects. That already implies that all escaping continuations should be documented.

> *I wonder if other Schemers agree.*

Some of the freshmen don't, but it's our job to teach them.

Will