Revisiting `CONCATENATE-SEQUENCE`

Christophe Rhodes*

December 20, 2006

Abstract

While doing work to support user-extensible sequences (Rhodes, 2007), it was discovered that the ANSI CL standard (X3.226-1994) forbids integration of certain functions with not only user-extensible sequences but also implementation extensions of `sequence`. Irrespective of the future of user-extensible sequences, we argue that the restriction on implementations imposed by the wording adopted is too stringent, and propose an alternative.

1 Introduction

In the X3J13 Issue `CONCATENATE-SEQUENCE` (Pitman, 1991), the ANSI CL committee worried about various cases of sequence type specifiers passed to the five functions `make-sequence`, `map`, `merge`, `concatenate` and `coerce`. The essential problem which the `CONCATENATE-SEQUENCE` Issue addresses is that a type specifier can specify a recognizable subtype of `sequence` without unambiguously specifying a concrete sequence type, needed because, except for a special case in `coerce`, these functions must create an object of the specified type.

For instance, the type `sequence` itself is a recognizable subtype of `sequence`; however, the desire was that the call `(make-sequence 'sequence 8)` should be in error; other such ambiguous types can be constructed, such as `(simple-array (*) *)`, `(or bit-vector string)`, and `(and sequence (not (eql "foo")))`; although types involving conjunction, disjunction and negation are not required to be recognizable subtypes of `sequence`, most current implementations recognize these examples as such.

However, the ANSI CL standard also specifies that an implementation may offer subtypes of `sequence` that are not `list` and `vector`:

> The types [*sic*] vector and the type list are disjoint subtypes of type sequence, but are not necessarily an exhaustive partition of sequence.
>
> Pitman and Chapman (1994, System Class `sequence`)

Historically, this does not appear to have been a popular field for implementation extension; at the time of writing, the author knows of no implementation purporting to conform to Common Lisp which documents non-standard sequence types, though there exist undocumented hooks in at least GNU Clisp (Haible, 2006) which were used in a pre-CLOS implementation of generalized sequences (Haible, 1988).

---

*Goldsmiths College, New Cross Road, London SE14 6NW, `c.rhodes@gold.ac.uk`

In light of this standard definition of the `sequence` class, and of the development of user-extensible sequences, however, the wording for the Exceptional Situations of `make-sequence` overreaches the intent of the clarification of the `CONCATENATE-SEQUENCE` issue:

> An error of type type-error must be signaled if the result-type is neither a recognizable subtype of list, nor a recognizable subtype of vector.
>
> Pitman and Chapman (1994, Function `make-sequence`)

Similar requirements are placed on `map`, `merge`, `concatenate` and `coerce`.

This requirement does not permit an implementation to extend `make-sequence` to type designators for non-standard sequences, which does not seem to have been the intent behind the `CONCATENATE-SEQUENCE` issue. We therefore propose the clarification, presented in the style of an issue in the next section.

2 Issue `CONCATENATE-SEQUENCE-AGAIN`

Issue: `CONCATENATE-SEQUENCE-AGAIN`.

References: `coerce`, `concatenate`, `make-sequence`, `map`, `merge`, Pitman (1991).

Category: Clarification / Change.

Problem Description: The specification says that an error must be signalled in cases when a type specifier passed to `make-sequence` is not a recognizable subtype of either `list` or `vector`. This prevents integration of non-standard sequence types, expressly permitted by the description of `sequence`, with the standardized sequence functions.

This also affects `coerce`, `concatenate`, `map` and `merge`.

Proposal `(CONCATENATE-SEQUENCE-AGAIN:GENERALIZE)`:

- Remove from `make-sequence`, `merge`, `map` and `concatenate` the requirement that "An error [of type `type-error`] must be[1] signaled if the result-type is neither a recognizable subtype of `list`, nor a recognizable subtype of `vector`."
- Specify that if a type specifier is not a recognizable subtype of `sequence`, or is a recognizable subtype of `(or list vector)` but not of either `list` or `vector`, then an error [of type `type-error`] must be signaled.
- Specify that if a type specifier is a recognizable subtype of `sequence`, but not a recognizable subtype of any of the types `(or list vector)`, `list` or `vector`, then the consequences are unspecified.

The square brackets above reflect the slightly different requirements on the functions in their error signalling, as defined in Pitman and Chapman (1994, Section 1.4.2); the intent of this proposal is to preserve as much as possible the original requirements on these five functions while allowing for expansion of the set of acceptable sequence type specifiers.

---

[1] `concatenate` and `merge` have "is" for "must be", but this makes no difference in standardized behaviour.

Rationale: This allows implementors to make extensions of `sequence`, as seems to have been the original intent.

Test Case: No portable test case. In implementations extending the `sequence` class,

```
(make-sequence '<sequence-class> 8)
```

need not signal an error under this proposal.

Current Practice: Effectively compatible with both the standard as specified and this proposal, as no implementation extends `sequence` as of the time of writing.

Cost to Implementors: None.

Cost to Users: Minimal. Users can no longer have the guarantee that code of the form

```
(assert (typep (ignore-errors (make-sequence *x* 8))
               '(or list vector)))
```

never causes the assertion to fail.

Cost of Non-Adoption: The specification remains inconsistent.

Benefits: A natural way of providing extensions for the `sequence` type.

Aesthetics: Minimal.

References

Haible, B. (1988). The Abstract Datatype *Sequence*. Technical report, University of Karlsruhe. `http://tinyurl.com/yy3eys`.

Haible, B. (2006). personal communication.

Pitman, K. and Chapman, K., editors (1994). *Information Technology – Programming Language – Common Lisp*. Number 226–1994 in INCITS. ANSI.

Pitman, K. M. (1991). Issue `CONCATENATE-SEQUENCE`. Issue 73, X3J13, ANSI. `http://www.lisp.org/HyperSpec/Issues/iss073-writeup.html`.

Rhodes, C. (2007). User-extensible sequences. in preparation.