

# BitC Language Change History<sup>†</sup>

Version 0.10

Jonathan Shapiro, Ph.D. Swaroop Sridhar Scott Doerrie  
*Systems Research Laboratory*  
Dept. of Computer Science  
Johns Hopkins University

June 17, 2006

## Abstract

This document describes the various changes that have been made to the BitC specification since its inception. Originally, the list was included in the main specification document, but it has grown to the point where separating it into a separate document became motivated.

## 1 Changes in version 0.10+

- Moved to a simplified version of Defrepr based on the scheme proposed by Iavor Diatchki, *et. al.*
- Dropped the now deprecated `(val type)` form.

## 2 Changes in version 0.10

- New default entry point `bitc.main.main`.
- Added closure conversion.
- Updated identifier syntax for interfaces. Removed `import!` and `provide!`.
- Added beginnings of formal specification section. Captured the complete grammar.
- Introduced an initial attempt to deal with locations in the specification.
- Introduced `defrepr`. Replaced old `defunion` with a new one that uses field names.
- Added specification for structure fill fields. Added field names in unions and exceptions. Added `switch` form that lets these fields be used, replacing `case`.

- Removed obsolete convenience syntax for lists.
- Improved description of union layout, and clarified elided tag rules.
- Resolved the problem of circular initialization dependency by introducing a requirement that dependencies be (transitively) observably known.
- Abandoned `case!` for good.
- Removed method types now that we have qualified types. Expanded specification of type classes, instances, and qualified types. Defined the resolution rule for ambiguous choices of instance while allowing preferred specializations.
- Clarified definition of white space and end of line processing. Added proper citation for Unicode standard.
- Introduction and definition of `definvariant`, `defaxiom`, `deftheory`, `suspend`, `disable`, and `enable`.
- Revised Unicode character literal syntax, dropping the radix syntax in favor of `\{U+digits\}`. Dropped the curly braces around the special forms for non-printing character literals. The old forms will be retired in version 0.11.
- Revised string backslash embedding rules. Backslash within a string no longer shares a common syntax with character literals. Curly braced forms for non-printing characters in strings will be retired in version 0.11.
- The following statement from the 0.9 specification has been dropped, and may lead to incompatibilities in transitioning to language version 0.10.

In general, if `#\x` is a valid character constant, then `"\x"` is a single character string consisting of the

---

<sup>†</sup> Copyright © 2006, Jonathan S. Shapiro and Swaroop Sridhar.

same character. By far the most common uses of this escaped encoding is to embed the double quote character and the backslash character within a string, but the generalization also permits encoding unicode characters by code point.

### 3 Changes in version 0.9

Cleaning up a few *more* loose ends:

- Removed `read-only`.
- Added keyword `external` to `proclaim`, along with optional external identifier.
- Renamed `vector-ref`, `array-ref`, `member-ref` to (respectively) `vector-nth`, `array-nth`, `member`.
- Restored the primitive types `bool`, `char`, `int8`, `int16`, `int32`, `int64`, `uint8`, `uint16`, `uint32`, `uint64`, `word`, `float`, `double`, `quad`.
- The `fixint` keyword is retired in favor of `bitfield`.
- Binding patterns in `letrec` forms must now be statically decomposable at compile time to avoid internal violation of the completeness restriction.
- Order of evaluation in `let` and `letrec` initializers is now specified. This is necessary because of side effects in impure expressions.
- Replaced `LOOP` with `DO`.
- Added a new type declaration qualifier `:opaque` that renders the type visible in its defining interface and providers but `opaque` to importers.
- Removed all references to `tuple`. This is replaced by a new keyword `pair`.
- Lambdas now take multiple arguments, and implicit pairing is now gone.
- The `one-of` type constraint had horrible problems, and has been removed. In consequence, `nth-ref` was split into `array-ref` and `vector-ref` and the indexing convenience syntax has been removed.
- Renamed `type-qualify` to `the`, following Common Lisp conventions. Removed the `e:T` con-  
venienced syntax for expressions. Types in binding patterns are still specified using the colon-qualified form.
- The `deftype` form was unmotivated and complicated. It is gone.
- Removed the reference chasing misfeature of `array-length`, `vector-length`, `nth-ref`, and `struct-ref`. Renamed `struct-ref` to `member-ref`.
- Renamed keyword `forall` to `method`.
- Replace `vector-ref` and `array-ref` with `nth-ref`
- Added `forall` as explicit syntax for writing method types.
- Added `one-of` as a type constraint, subject to the rule that the member types must not unify.
- *Issue*: Should we introduce labeling forms `check`, `assert`, `label`?
- Described (and required) a variant of the Cardelli optimization for list cell representations.
- Added type classes.
- Corrected definitions of `tuple`, `array` types to be value types. Corrected definition of `vector` to be a reference type.
- Removed `mutable` value constructor. Replaced `immutable` value constructor with `read-only`.
- Removed discussion of primary type conversion, since this is now part of the standard prelude.
- Removed discussion of canonical serialization.
- Discussed declaration of value vs. reference named types.
- Moved `case!` and `deftype` to the experimental section. We may restore these, but I want to try things without it first.
- Renamed `alias` to `use`.
- Removed the “literal type resolution” rule, per Mark Jones.
- Reverted the specification of `mutable`. Mutability is once again defined at field level. Temporarily *removed* the `mutable` and `immutable` value constructors..
- Renamed the `deref` type constructor to `val`.

## 4 Changes in version 0.8

The objective in Version 0.8 is to clean up a few loose ends before I dig in to type classes.

One cleanup that did *not* make it into this document was eliminating the references to "complete" and "incomplete" types. These aren't really issues of type at all. There is no problem from a typing perspective with a recursively defined value type. The issue is really an issue of instantiability, and if we treat it that way and discuss it separately from the discussion of types, we can deal with some of the complications that arise from mutable monomorphism in a more sensible way as well.

- Corrected the "literal type resolution" rule in section 3.2 to apply only to literals, noted Mark Jones's objection, and marked this as provisional.
- Re-introduced named `let` under the keyword `loop`. Corrected specification of `let`, which is not a derived form.
- Non-escaping types have been removed. This includes `restricted` and `sequence`. This required me to expand `sequence-ref` and `sequence-length` into `array-ref`, `array-length`, `vector-ref`, and `vector-length`.

This change introduces an open issue, which is whether the array types should be redefined as `(array T len)`, where the type constructor is restricted to require a literal at the argument `len`. I am deferring the resolution of this issue until type classes have been added.

- Revised the specification of the `mutable` type constructor to indicate that its use appearances are syntactically restricted.
- Introduced `dup`, removing the `ref` value constructor.

## 5 Changes in version 0.7

Version 0.7 is primarily a pass to reconcile the provisional compiler with the specification, and perform some cleanups. The primary substantive change in this pass is to replace modules with interfaces.

Need to dig into the library some more and define equality types.

- Migrated `typecase` into the experimental section. Added a discussion of inference resolution in the pri-

mary types section and a set of type coercion operators in the expression section.

- Removed the Swaroop restrictions, since the implementor is no longer confused.
- Added `case!` to the specification.
- As a consequence of the change to interfaces and the deferral of `typecase`, clarified that `int32` and friends are proper keywords.
- Added new keywords `alias`, `import!` (was: `stateful-import`), `provide`, `provide!`, and `interface` to provide means for interface specification and name aliasing. Updated the description of compilation units to reflect this addition. Dropped the `module` and `export` keywords in favor of interfaces.
- Added new keyword `proclaim` for forward and external declarations. This obviated the need for `rectypes`, which is now removed.
- Added new keyword `immutable`, with the effect that it returns its argument, stripped of mutability. `immutable` is syntax, because it does not copy its argument.

Clarified definition of `mutable` value constructor to indicate that it is a non-copying syntactic form.

Relocated the Pragmatics section, which should have been part of the language specification rather than the library specification.

- Removed `defmacro`, which was a really bad idea.
- Re-merged the core and convenience forms into a single section, but annotate which is which. Having two sections to describe expressions made things unreasonably obfuscated.

## 6 Changes in version 0.6

Version 0.6 introduce interfaces, modules, and support for capability-safe programming. We are vaguely hopeful that the only major changes after this point will be fleshing out the standard library.

- Added a definition of the module system. Changed the definition of compilation units to reflect it.
- Removed discussion of storage model, layout, and alignment in the types section. Temporarily removed the major section on storage model, which needs serious rework.

- Moved `letrec` back into the core, since it cannot be lambda-expanded without the Y combinator, which does not type check in a statically typed lambda calculus.
- Added mention of the value restriction rule.
- Repaired the erroneous parenthesization of `cond`, `case`, and `typecase`. Changed definition of `cond` and `case` to have an explicit otherwise case.
- Clarified that the type inference engine must handle `typecase` with care to avoid type unification.
- Revised `typecase` to note that its result expressions need not have the same type. Also noted that failure to match is a static type error.
- Removed `int` and `nat`. Specified default types of unqualified integer literals. Added new type `word` which is the least unsigned integer type sufficient to hold a pointer representation.
- Split existing `vector` type into `array` and `vector`. Array is fixed length, vector is dynamic length. Vectors are unboxable, but internally reference heap-allocated storage. Both are served by the primitive accessor `sequence-ref` in order to continue to support the convenience syntax for array element reference. Added new core form `sequence-length` that returns the length of its argument value of sequence type.
- Replaced `box`, `unbox` with `ref`, `restricted-ref`. Changed `struct-ref`, `sequence-ref` to implicitly dereference their left hand argument as needed. In consequence it became possible to eliminate the convenience syntax for `deref`, which is now rare.
- Introduced `defmacro` and specified the expansion of convenience syntax into core language forms.
- Split the specification into core expression forms and convenience expressions.
- Noted that BitC is polymorphic, and that it imposes the Value restriction similar to ML.
- Added a syntax for describing procedure types.
- Removed field names from unions
- Replaced `typealias` with `deftype`
- Noticed that binding patterns in `define` already handled mutual recursion and consequently removed `mutual-recursion`. Introduced `rectypes` for recursive type declarations.
- Removed the `ref` keyword, replacing it with `mutable`, with the intended meaning that what we used to write as `(unbox ref x)` is now `(mutable x)`.
- Added an explicit type modifier keyword `box` to the language (the dual of `unbox`).
- State that primitive fixed-size types are unboxed by default, and the constructed types are boxed by default (compare Java).
- **Pending** Add explicit operators `box` and `unbox` to the language. Each creates a copy of its argument. The compiler is free to optimize out the copy when it can do so correctly.
- Removed the character `^` from the legal identifier characters. `^` is now a postfix operator indicating pointer dereference.
- Added a new form `(bitc-version "version")` where `version` is a string identifying the BitC language version in which the program was implemented. This form is syntax, and is permitted only at top-level. It *must* appear exactly once as the first form in every unit of compilation. It is a compile-time error if the compiler implementation does not implement the language-version specified in the input compilation unit.
- Resolved to specify the layout of data structure created using the `defunion` declaration, and to provide a means using for specifying the representation size of the “hidden” union tag:

```
(defunion u
  (cons1 arg1)
  (cons2 args)
  (declare (tag-type T)))
```

where  $T$  must be compatible with some `fixint` of unsigned type.

## 7 Changes in version 0.5

Version 0.5 is a major rework of the language. The reader is advised to reread the entire specification, as there have been many changes with potentially subtle interactions.

- Added a “reserved words” section, including some reserved words set aside for use in future module and object systems.

- **pending** Need to specify assignment and binding compatibility of boxed and unboxed types. Resolution is that the language does *not* provide implicit boxing or unboxing.
- **pending** Need to specify assignment and binding compatibility of mutable vs non-mutable things. Resolution is that assignment has copy semantics, and so both assignments are legal. Further, consider the procedures:

```
(define (f x : (mutable T))
  body)
(define (g x : T)
  body)
```

the mutability of  $T$  in  $\mathcal{F}$  is strictly *internal* to the procedure. Because BitC has by-value semantics of procedure arguments the signature of  $\mathcal{F}$  for purposes of external type checking and pattern matching is the same as the signature of  $\mathcal{G}$ .

- Added radix prefixes and negation for integer literals. Introduced string literal syntax and specified full syntax for floating point literals. Revised character literal syntax for code points to be lexically similar. This may still need further revision, as the current syntax is exceptionally awkward. I am considering using HTML-style character entity syntax in place of the current one. Note that left and right curly brace are no longer supported using the simple character syntax.
- Added means to describe bitfields via `fixint`.
- Specified string literal syntax. Type of such literals is `(vector char)`. String type is needed in the primitive language in order to support diagnostic output. More precisely, it is needed in order to provide for string literals.
- Revised the definition of `tuple` to remove the requirement that `(tuple x)` is treated like `x`. This should never have been true in SML either.
- Removed the comma-style tuple construction syntax in the expression language, eliminating the syntactic ambiguity of how to interpret `( a )` (application or tuplization?).
- Swaroop and Jonathan initially had a difference of opinion concerning whether `deftype` should be resolved by textual substitution. The problem is that most of our types are resolved by name equivalence, so this appears to have rather limited utility in reducing verbiage.

Actually, this is rather a big mess, as is illustrated by the definition of `tree-of` in the discussion of structures. There appears to be no *correct* definition of `tree-of` under our present rules.

## 8 Changes in version 0.4

- Description of tuples was augmented to describe syntax of tuple patterns and allow field names in tuple patterns.
- Added `typealias` declaration.
- Added pragmatics section requiring limited tail recursion.