Chez Scheme Version 3.0 Release Notes

July 1989

Overview

This document outlines the changes made to *Chez Scheme* for Version 3.0. The improvements to *Chez Scheme* made for Version 3.0 include: a new high-level foreign-function interface, a new facility for invoking and communicating with subprocesses, support for first-class environments and full compatibility with the text, "Structure and Interpretation of Computer Programs," improved performance, more sophisticated printing and pretty-printing, and Revised^4 Report on Scheme compatibility.

Refer to the *Chez Scheme System Manual* for detailed description of the new procedures and syntactic forms mentioned in these notes.

Version 3.0 is available for Sun-3 (SunOS 3.X, 4.X), Sun-4 (SunOS 4.X), Vax (Ultrix, 4.2 and 4.3 BSD Unix, and VMS), and Apollo (Domain/IX 10.1) computers. Support for other systems will be completed in the near future; please call or write for information.

This document contains five sections describing (1) Functionality Enhancements (2) Bugs Fixed, (3) Performance Enhancements, (4) Features Removed, and (5) Limitations.

1. Functionality Enhancements

1.1. Subprocess creation and communication

It is now possible to invoke other programs as subprocesses, either synchronously with the procedure `system` or asynchronously with the procedure `process`. The `process` procedure allows a Scheme program to communicate with the subprocess.

1.2. Foreign function interface

It is now possible to call procedures written in other, "foreign", languages (usually C) from *Chez Scheme*. On some systems, including the Sun-3, Sun-4, and VAX under Ultrix or BSD Unix, it is possible to dynamically load object files containing the foreign procedure definitions. On others, it is necessary to rebuild the *Chez Scheme* executable with the foreign code linked in.

1.3. First-Class Environments and "SICP" Compatibility

The major incompatibility between Version 2.0 of *Chez Scheme* and the Abelson and Sussman text *Structure and Interpretation of Computer Programs* ("SICP") was the lack of support for first-class environments. While first-class environments are still not a part of *Chez Scheme* by default, they can be enabled by invoking the procedure `support-first-class-environments`.

The procedure `support-sicp`, which implicitly calls `support-first-class-environments`, may be used to enter a fairly complete compatibility mode for "SICP." Previously, loading an auxiliary file, "sicp.ss," was required.

1.4. Numeric conversions

Chez Scheme now supports the numeric conversion procedures `string->number` and `number->string`, which are optional features in the "Revised^4 Report on the Algorithmic Language Scheme," which is currently in draft form. These procedures changed substantially between the Revised^3 Report and the Revised^4 Report. The versions implemented in *Chez Scheme* Version 3.0 and documented in the *Chez Scheme System Manual* follow the Revised^4 Report.

The syntax for numeric constants is also somewhat different in the Revised^4 Report; the full Revised^4 Report syntax for numbers, including exactness and radix prefixes, is supported by Version 3.0.

1.5. `char-ready?` procedure

Chez Scheme now supports the `char-ready?` procedure for input ports (created for files, strings, or subprocesses).

1.6. `peek-char` procedure

The Revised^4 Report on Scheme defines the optional procedure `peek-char`, which returns the next character to be read from a port (via `read-char`) without actually consuming it.

1.7. `nonpositive?`, `nonnegative?` procedures

The procedures `nonpositive?` and `nonnegative?` have been added to complement the existing `positive?` and `negative?` procedures.

1.8. `remprop` procedure

The new `remprop` procedure is used to remove a property from the property list of a symbol.

1.9. `set-working-directory` procedure

The new `set-working-directory` procedure is used to change the current working directory, the point from which all file accesses and system operations is performed.

1.10. New fixnum procedures

The following new fixnum procedures have been added to *Chez Scheme*:

```
fxlogand
fxlogor
fxlognot
fxlogxor
fxabs
fxeven?
fxodd?
fxmodulo
fxremainder
fxmax
fxmin
fxnegative?
fxnonnegative?
fxpositive?
fxnonpositive?
fxquotient
```

1.11. `*scheme*` parameter

The variable `*scheme*` is invoked by *Chez Scheme* upon startup with a list of file names passed on the command line. By default, it is defined as follows:

```
(define *scheme*
   (lambda filenames
      (for-each load filenames)
      (new-cafe)))
```

This variable may be set to a different procedure before the executable file or heap file is saved (see the Chez Scheme manual page, scheme.1) to alter the behavior of the new executable or heap file.

1.12. `machine-type` procedure

The procedure `machine-type` returns a symbol defining the machine type *Chez Scheme* is running on. The value of `machine-type` may be used as input to `compile-file` for the purpose of cross-compiling. As of Version 3.0, the following machine types are recognized:

`apollo`: Apollo computers running Domain/IX 10.1 or higher
`sun3`: Sun-3 computers running SunOS 3.2 or higher
`sun4`: Sun-4 computers running SunOS 4.0.3 or higher
`vaxunx`: VAX computers running Ultrix 2.0 or higher, 4.2 BSD Unix or 4.3 BSD Unix
`vaxvms`: VAX computers running VMS 4.2 or higher

1.13. `with-output-to-file` and `with-input-from-file`

Chez Scheme now supports the procedures `with-output-to-file` and `with-input-from-file` described in the Revised^3 and Revised^4 reports on Scheme.

1.14. Trace package

The trace display now reflects the difference between tail calls and nontail calls, as the following example demonstrates:

```
> (define f
    (lambda (x)
      (cond
        [(zero? x) 0]
        [(odd? x) (f (- x 1))]
        [(even? x) (+ (f (- x 1)) 1)])))
f
> (f 5)
2
> (trace f)
(f)
> (f 5)
(f 5)
(f 4)
|  (f 3)
|  (f 2)
|  |  (f 1)
|  |  (f 0)
|  |  0
|  1
2
2
```

Tail calls do not result in an added level of indentation, while nontail calls do.

1.15. Printer enhancements

The *Chez Scheme* `write` and `pretty-print` procedures are affected by the values of five printer control parameters: `*print-radix*`, `*print-length*`, `*print-level*`, `*print-gensym*`, and `*print-graph*`. The first three were supported but not documented in Version 2.0. The latter two, `*print-gensym*` and `*print-graph*`, are new in Version 3.0.

The `*print-radix*` parameter determines the radix in which non floating-point numbers are printed.

The `*print-level*` parameter determines the number of levels of nested structure printed.

The `*print-length*` parameter determines the number of list elements printed at any level.

The `*print-gensym*` parameter determines how uninterned symbols are printed: if false, uninterned symbols are printed in the same manner as interned symbols and if true, uninterned symbols are marked with a "#:" prefix.

The `*print-graph*` parameter determines whether shared graph structures (including circular structures) are printed in a manner that allows them to be read back in in the same form.

The procedures `write` and `pretty-print` also now observe the value of `*case-sensitive?*`. When `*case-sensitive?*` is true, they no longer print escape characters (vertical bars) around a symbol containing upper case letters.

1.16. Pretty-printer enhancements

The pretty printer now handles named-`let` properly and understands many other syntactic forms that were previously left to its "generic" algorithm. It also understands where to insert brackets for readability, such as in the bindings portion of a `let` or `letrec` expression. The pretty-printer has also been extended to handle vectors, and it correctly handles the printer control parameters `*print-level*` and `*print-length*` as well as the new parameters `*print-gensym*` and `*print-graph*`

1.17. New output file open options

The `open-output-file` procedure now takes an optional second argument that determines the action of `open-output-file` when the file named by the file name argument exists. Using this argument, it is possible to open an existing file (such as /dev/null or /dev/tty on Unix-based systems), whereas in earlier versions it was always an error to open an existing file (except under VMS, under which a new version was created). The optional argument can be one of the following:

`error` (default on Unix): signal an error if the file already exists.

`replace` (default on VMS): replace the existing file with the new file (on VMS, create a new version of the existing file).

`truncate`: open the existing file and truncate to zero length (on VMS, truncate behaves like `replace`).

`append`: open the existing file and append data written to the port to the end of the file.

The procedures `call-with-output-file` and `with-output-to-file` have been extended in the same manner as `open-output-file`.

1.18. Optional `getprop` argument

The procedure `getprop` has been extended to accept an optional third argument, the value to return if the property does not exist.

```
> (getprop 'a 'b)
#f
> (getprop 'a 'b 'c)
c
```

1.19. New `cond` syntax

The `cond` syntactic form has been extended to recognize clauses of the form `[exp1 => exp2]`. If `exp1` evaluates to a true value, `exp2` is evaluated (it must evaluate to a procedure) and passed the value of `exp1`.

```
> (cond [(assq 'a '((a . b))) => cdr])
b
```

1.20. Quasiquote for vectors, boxes

The `quasiquote`, `unquote`, and `unquote-splicing` syntactic forms have been extended to handle vectors and boxes.

```
> `#(a ,(+ 3 4) b)
#(a 7 b)
> `#(a ,@(list 'b 'c))
#(a b c)
> `#&,(+ 3 4)
#&7
```

1.21. Saved executable for Apollo

The "-o" option documented in the *Chez Scheme* manual page (scheme.1) is now supported for the apollo version of *Chez Scheme*. Since this is the mechanism whereby we produce the executable to distribute, it is no longer necessary to load up a Scheme object boot file when *Chez Scheme* starts up. As a result, the lengthy (minimum 30 second) start-up time has been eliminated.

1.22. New `define` syntax

The `define` syntactic form has been extended to accept the syntax `(define variable)`, which defines `variable` and gives it an unspecified value.

1.23. Extended file name syntax

File and directory names on Unix-based systems can now be specified with the "~*user*" and "~/" prefixes supported by certain shells, including "csh." File names prefixed by "~" are interpreted relative to the current user's "home directory," while file names prefixed by "~*user*" are interpreted relative to *user's* home directory.

1.24. Improper list check in `apply`

The `apply` procedure now signals an error when its last argument is an improper list.

1.25. Octal character syntax

The reader now supports the syntax `#\\`*ddd* for characters, where *ddd* is the ascii code for the character in octal. The `write` and `pretty-print` procedures prints certain control characters using this syntax.

1.26. random number generator

The random number generator has been extended to accept positive inexact real numbers. The value returned, whether the argument is a positive integer or a positive inexact real number, is a nonnegative pseudo-random number less than the argument.

The new procedure `random-seed` may be used to obtain the current state of the random number generator as an integer. The procedure `set-random-seed!` may be used to restore the random number generator to a previous state by passing it a value returned by a previous call to `random-seed`.

1.27. `extend-syntax`

The `extend-syntax` facility has been extended to allow the use of `quasiquote`, `unquote`, and `unquote-splicing` expressions within expansions, fenders, and `with` values.

The `extend-syntax` form also checks for misplaced `with` and `...` symbols.

1.28. Circular lists passed to `length`.

The `length` procedure now signals an error if passed a circular list.

1.29. Optional `compile-file` arguments

The `compile-file` procedure has been extended to allow explicit specification of the output file name via an optional second argument.

It has also been extended to allow cross-compilation via an optional third argument that specifies the target machine type.

1.30. New example program

The examples directory contains a new example program, “m4.ss”, which is an implementation of part of the functionality of the “m4” macro preprocessor found on most Unix-based systems.

1.31. Improved diagnostic messages from the I/O system

The input/output subsystem now provides more descriptive error messages when an operation cannot be performed. Rather than saying simply that the operation cannot be performed, the error messages attempt to describe why the operation cannot be performed.

1.32. Memory use statistics

The “-o” option described in the *Chez Scheme* manual page (scheme.1) now prints memory usage statistics.

1.33. Keyboard interrupt “s” option

The “s” option in the keyboard interrupt menu now reports elapsed statistics from last command typed to the current waiter, rather than overall statistics for the current session. The current session statistics may be displayed by entering a new cafe (the “n” option in the keyboard interrupt menu) and typing `(display-statistics)`.

1.34. Eight-bit input

The reader now treats all characters with bit seven set as alphabetic, rather than signaling an error.

1.35. “Nonsensical” applications

The evaluator now allows actual procedures or quoted procedures to be placed in the first position of an application, *e.g.*, `(eval ‘(,cons 3 4))`.

1.36. Parameter list checking

Various syntactic forms, including `lambda`, `define`, and `let` now check for duplicate bound variable names and for syntactic keywords used as variables.

1.37. `sort/merge` side effects

The procedures `sort` and `merge` no longer use any side effects. Previously, `sort` was implemented by copying its input list and calling `sort!`, and merge was implemented similarly. This change only matters if you create a continuation while sorting and invoke the continuation more than once (implicitly or explicitly)

2. Bugs Fixed

2.1. Rational number division bug

Division of ratios can no longer produce ratios with negative denominators.

2.2. `let/set!` bug

The following code no longer causes a memory fault:

```
(let ((f 0)) (set! f (lambda (x) x)))
```

2.3. Boolean `vector-ref` bug

Using the result of a `vector-ref` in the test part of an `if` expression now works.

2.4. VAX `expt` underflow

Computing the exponent of an inexact quantity with a very large negative exponent on the VAX now returns zero rather than causing an floating point overflow.

2.5. Storage manager bug

An obscure storage manager bug that occurred only when there were many large and small objects of the same type allocated at the same time was fixed. This bug sometimes resulted in a corrupted memory image.

2.6. `call/cc` argument check

`call/cc` now signals an error when passed a non-procedure as an argument.

2.7. `new-cafe` and `load` argument check

`new-cafe` and `load` now signal an error if passed a non-procedure to use as the evaluator.

2.8. `apply` and circular lists

`apply` no longer generates a memory fault when passed a circular or excessively long list.

2.9. `force` argument check

The error signaled by `force` when passed a non-procedure argument has been fixed.

2.10. Transcript/storage manager bug

The transcript port can no longer be reclaimed prematurely by the storage manager.

2.11. VMS real-time clock

The `real-time` procedure now understands both the forward-moving (VMS Version 4.5 or earlier) and backwards moving (VMS Version 4.6 or later) real-time counter.

2.12. `letrec/call/cc` bug

An obscure bug related to `call/cc` and `letrec` has been fixed. Using `call/cc`, it was possible to force `letrec` to create one or more bindings more than once: once before the continuation was created and after it was invoked.

2.13. `rationalize` on exact numbers

Rationalize now correctly simplifies exact rational numbers when passed a nonzero second argument, instead of simply returning the argument.

2.14. `write-char` argument check

`write-char` now signals an error when passed a non-character first argument.

2.15. `newline` argument check

`newline` now signals an error when passed a non-port argument.

2.16. `floating-point`/end-of-file bug

The reader now correctly handles floating point numbers occurring at end-of-file with no trailing white space.

2.17. Waiter more robust

The Scheme waiter now always interacts with the console. Previously, it was possible to confuse the waiter by rebinding the `*standard-input*` and `*standard-output*` ports.

2.18. Nonsymbol in parameter list

The error message signaled for a nonsymbol in a parameter list has been fixed.

2.19. `exact->inexact` and large ratios

`exact->inexact` now handles ratios with large components as long as the resulting floating point number is in the range of numbers representable internally.

2.20. `sqrt` and rational numbers

`sqrt` now accurately finds the rational square root of all rational numbers with rational square roots.

2.21. SunOS keyboard interrupt bug

The *Chez Scheme* output routines now work around a bug in the SunOS 3.5 `write` system call that sometimes caused *Chez Scheme* to report a noncontinuable interrupt when the user attempted to continue after an keyboard interrupt that occurred during output to the console.

3. Performance Enhancements

3.1. Optimization

Overall system performance has increased by about 15% over Version 2 due to new compiler optimizations and code generation strategies.

3.2. `extend-syntax`

`extend-syntax` now uses a much faster matching algorithm, and generates more efficient code, specifically when the output pattern is a subexpression of the input pattern.

3.3. `length`, `list-ref`, and `list-tail`

`length`, `list-ref`, and `list-tail` are now much faster.

3.4. Stack/continuation management

Stack management has been greatly improved, sometimes resulting in an order of magnitude improvement in performance for deeply recursive routines. The stack management changes have also resulted in greatly increased speed for certain continuation-intensive computations.

4. Features Removed

4.1. `*eof*`

Changing the value of the variable `*eof*` no longer has any affect on the end-of-file value returned by `read-char` and `read`.

4.2. `engine`

The `engine` syntactic extension supported by previous versions has been removed.

4.3. `*generate-timer-trap*`

Changing the value of `*generate-timer-trap*` no longer has any affect on generated code (the timer trap check is no longer separate from the interrupt trap check).

4.4. `*generate-stack-check*` and `*stack-overflow-handler*`

Changing the value of `*generate-stack-check*` no longer has any affect on generated code. Likewise, changing the value of `*stack-overflow-handler*` no longer has any affect on stack overflow handling (stack overflow can no longer occur).

5. Limitations

5.1. IEEE Infinity and NaN handling

IEEE floating point infinity and NaN values are not trapped by *Chez Scheme* floating point operations. If one of these values is produced by a floating point operation and passed as an argument to a subsequent floating point operation, the behavior of the subsequent operation may be incorrect.

5.2. Apollo save heap

The "-s" and "-h" options documented in the Unix manual page scheme.1 for saving and restoring heap files is not supported on the Apollo.

5.3. VMS and `process`

The `process` procedure is not supported for VAX VMS.

5.4. VMS and `char-ready?`

The `char-ready?` procedure is not supported for VAX VMS.