AFL Change History

Sam Wilmott
420 Tweedsmuir Ave, Ottawa, ON, Canada
sam@wilmott.ca

Abstract

These are the changes to the AFL implementation, its libraries and its documentation since the first public release (afla001) on 25 February 2005.

1 afla009 (4 September - 12 November 2005)

The following is a collection of changes that have built up over the last couple of months.

- Added multi-thread references (a.k.a. "threadSafeRef") to afldefs.afl, and in the process made currentTextReader, currentTextWriter and currentSource all thread safe, without any need to do extra includes, other than including "threading.afl".
- Each of the compiler options /a /e, /i, /k and /s may now be specified more than once on the command line. This is especially useful for /a and /i, where there may be more than one source of configuration or "include" files. For /e, /k and /s, all the specified options are combined as if specified together.
- Added the ".." operator to "patterns.afl" (for matching a character in a range of values, as in ~: "A" .. "Z").
- Added the "random" function to "afldefs.afl".
- Removed the "else" forms of the "for-do", "while-do", "until-do", "do-until" and "do-while" operators from "afldefs.afl" and "afldefs2.afl". I found that constructs like:

  ```
  if a.count () > 0 then
    (for each a do (aItem):
      print aItem)
  else
    print "no items!";
  ```

  more often than not intended that the "else" be part of the outer "if" rather than the inner loop, requiring an extra set of parentheses to disambiguate the association of the "else". Removing the loops' "else" parts reduces potential ambiguity, and means the parentheses aren't needed.
- Made "afldefs2.afl" the default AFL2 definition file for the AFL compiler. This provides an approximately 20% speed up in AFL run times. "afldefs.afl" is still there, for those of you who want to play with Church-like booleans.
- Removed "ccall" as a keyword, and added a definition for it (using "ccallWithEH") to "afldefs.afl" and "afldefs2.afl".
- Added the /a option to the compiler's command line, allowing specification of files containing further command-line-like options (one per line of the command file, and with /a not recognized within configuration files).
- Added the /f option to the run-time's command line. /f says to not run any frame "atEnd" parts once and exception has been encountered, and to treat all exceptions as program terminating. This is typically a bad idea, but is useful when trying to pin-point the reason for an exception – not cluttering up the situation with processing following the exception.
- Added the /s=z option on the AFL compiler's command line. /s=z causes the compiler to generate code that gives you a function/continuation stack trace of where you are in the current thread/coroutine when the program ends in an error. Helps a lot in debugging.
- Removed the ".suspend" and ".resume" methods from the the "thread" maker in "threading.afl". These methods are deprecated by .NET 2.0. One should use a monitor instead.
- Added a note to the "xmlparser.afl". This include produces a warning under .NET 2.0 but still runs OK. The note tells how to make the include .NET 2.0 compatible. (Updating it to be .NET 2.0 compatible would have made it not run under .NET 1.1 and lower or under Mono 1.9.x and lower.) "xmlparser2.afl" is an include with this update applied.

Some error corrections:

- Corrected an error in the "<pattern> rep <number> upto <number>" operator in "patterns.afl": it always returned an inappropriate function value rather than true/false.
- Corrected an error in the "pos <number>" operator when it was passed a negative value: it always failed.
- Corrected an error in the compiler: any top-level closing parend, bracket or brace would end the program without complaint.
- Changed the way the "hasField" and "field" operators work: it used to be that the second argument had to be a string whose value is a C# name rather than an AFL name, which was inappropriate. The second argument's value now has to be an AFL name, i.e. "0field" rather than "_00field".

And changes to (`www.wilmott.ca/afl/afloverview.html`) afloverview.html:

- The section titled "Variables Revisited" has been added, describing the creation and use of variables in more detail.

- The section titled “Multi-Thread Variables” has been added, describing a strange but useful alternate form of variable – that exists independently in each thread. With the help of these values, currentTextReader, currentTextWriter and currentSource are now thread-safe.
- The section titled “currentTextReader, currentTextWriter and currentSource in Threads” has been added, describing the thread-safe properties of these values.
- The section titled “Include” now describes the “+” template and “?” conditional include options.
- The new afldefs.afl/afldefs2.afl-defined operators are included in the lists of operators.
- Any mention of the “else” forms of the looping operators has been removed.
- The “grouped arguments” for operators have been added to the AFL2 syntax.

There’s a few additions to the samples:

- A very small, simple XML parser implemented in AFL, illustrating the relationship between “push”, “pull” and “DOM” parsers.
- An include file (“midiutil.afl”) and a set of programs that create MIDI audio files.
- The include file “xmlparser2.afl” and sample “xmltest2.afl” have been added, doing XML parsing the .NET 2.0 way.

2 afla007 (25 July - 11 August 2005)

Dealt with a few (mostly what I consider) oversights:

- Added “rep”, “repupto”, “rep ... orMore” and “rep ... upto” operators to “patterns.afl”, as in:

```
if ~: ="-" rep 80 then # match exactly 80 of "-"
if ~: ="-" repupto 80 then
                  # match no more than 80 of "-"
if ~: ="-" rep 6 upto 80 then   # 6 to 80 of "-"
if ~: ="-" rep 6 orMore then # at least 6 of "-"
```

- Added “streamFrom” and “streamInto” to “io.afl”. These two operators create pairs of coroutines that are “piped” into each other using “read” and “write” logic, and allow one coroutine to provide the pattern matching input for another.
- Added the “scanReader” no-argument operator to “patterns.afl”. It makes using the current text reader as the current matching source easier.
- Added the *> and <* operators to “io.afl”. These operators support “filters”, functions that perform transformations on data read from their default input to their default output.
- Added “sourceio.afl”, supporting the “<~” and “scanFrom” operators, combining some of the new functionality from “io.afl” and “patterns.afl”.
- Modified coroutine logic so that each coroutine has its own “currentTextWriter”, “currentTextReader” and “currentSource”. These values aren’t really usable in the context of coroutines unless they are coroutine-local.

And something extra to play with:

- When you apply call syntax to a frame value, as if it were a function or continuation value, AFL attempts to invoke the frame’s “0callable” property. This means, for example, that you can have functions with named properties and indexers.

3 afla006 (20 May - 13 July 2005)

A few minor language changes, one biggish one, some updates to the Overview document, and some internal experiments:

- Added “until do”, and the “else” forms of “while do”, “until do”, “do while”, “do until” and “for do”.
- Changed the “rethrow” operator to “resignal”, for more consistency with “signal”, and less confusion with with the “throw” keyword, which is a different thing.
- Added the “...” operator.
- Changed the “arraylist” creator from a no-argument operator to a function, and added origin index values.
- Changed the “hashtable” creator from a no-argument operator to a function, for consistency with the change to “arraylist”.

  The changes to the creators for “hashtable” and “arraylist”, and the renaming “rethrow” to “resignal” are the backward-incompatibility changes for this release.
- Introduced “def op”. That’s a biggie, but it’s strictly an addition.
- “withSource” has been added to the pattern matching functionality.
- The two-argument ~:, +: and -: operators in “patterns.afl” have been modified so that they no longer persistently reset the value of currentSource, but rather localize that resetting to the scope of the application of the second argument.
- “withTextReader”, “withTextWriter” and “readFrom” have been added to the I/O functionality.

As part of release 6, there are a number of changes that don’t affect the language but help in the implementation:

- The syntax analyzer was rewritten using a pattern-matching approach.
- The “annotater”, that gathers information about what frames contain what names, was rewritten into a separate module. It’s not too smart right now, but has potential.

4 afla005 (2 May 2005)

This is just a reissue of afla004 (the AFL compiler reports itself as a004) with a major update to (`www.wilmott.ca/afl/afloverview.html`) afloverview.html. Rereading the overview, I found that the discussion of continuations was too spread around and early – they now have their own chapter.

5 afla004 (4-8 April 2005)

Minor cleanups and revisions:

- Added the “.exceptionHandler” accessor for continuation values, and removed “ccall” from AFL0. (“ccall” is still part of AFL1. It’s just that it now gets rewritten into “callWithEH” form.)

- Added the /d run-time command-line option.
- Removed “foreach” from the language but added generator operations “each”, “next ... to”, “next”, “forever”, “once” and “whilst”.

  The precedence numbers for comparisons and the generator related operators have been updated to allow a few things to be done more naturally, like “1 to 9 :++: 11 to 19” without parenthesization.
- Added “octet” I/O to “io.afl”.
- Added the “ord” and “field” operators.
- Added the unqualified ’[...] arraylist and ’[...,...] hashtable creator operations
- Corrected the AFL0 grammar in (`www.wilmott.ca/afl/afloverview.html`) afloverview.html: removed the “throw” from “catch:” and the “contArg” from “{}:”.
- Changed the names of some of the generator operators for more uniformity and ease of recognition:
  - :+: to :++: (because it’s akin to string joining)
  - :*: to inner (to spell it out a bit)
  - :**: to outer (ditto)
  - :/: to :+: (by analogy with :-:)

6 afla003 (31 March 2005)

Another minor release. A small but important addition in exception handling, and a fix and revision in template interpolation.

6.1 Exception Handling

I’ve added a relatively primitive form of exception handling and synchronous finalization:

```
try
  {
    doStuff ();
    true            # Return true if all was well.
  }
except (e):
  if typeOf e == "DivideByZeroException"
    then false      # Return false on a divide by
    else rethrow e; # zero exception and rethrow
                    # the exception otherwise.

def out: outputFile ("myOutputFile.txt");
try
 {
   // use "out"
 }
finally
  out.close ();
```

There are also supporting low-level facilities for establishing and using exception handlers. The (`www.wilmott.ca/afl/afloverview.html`) afloverview.html document describes the exception handling facility and its implementation.

There’s a supporting addition to “threading.afl”: there’s now a “critical” construct, that combines monitor locking with the facilities of “try ... finally”.

There’s one backward-incompatibility with this release: the ’{...} form of continuation invocation and {...} form of function call use to allow and expression list that was processed like a parenthesized group – evaluate all in order and return the value of the last expression. As of this release both these forms can have zero one or two arguments:

- With zero arguments, **nil** is passed as the argument value (or list).
- With one argument, its value is passed as the argument value (or list).

  In both the zero and one argument case, the current exception handler is set as in the previous release.
- With two arguments, the first the passed as the argument value (or list) and the second as the current exception handler of the called continuation or function.

6.2 Templating

There was a serious error in how template interpolation worked. Many things that shouldn’t have, got compile-time errors. It’s been fixed. At the same time, how things are interpolated into strings by the \{...} form has changed:

- Where the value is a string, it’s used as-is. (No change)
- Where the value is a number, it’s converted to its string representation (as in “repr”), and that string value is used. (Change)
- Otherwise, nothing is added to the string. (No change)

This change means that the following prints “10**2 = 100”, rather than an extra line break followed by “**2 = d”:

```
print "\{10}**2 = \{10*10}";
```

To make a character from its numeric encoding you have to convert yourself, as in:

```
def tab : "\{char 9}"; # tab
def lf : "\{char 10}"; # line feed
def cr : "\{char 13}"; # carriage return
```

7 afla002 (16-21 March 2005)

This is a minor release, fixing some errors, adding a few minor features, and removing a few others. It’s primary purpose is to clean up a few rough edges and round out the pre-type-system shape of the language. It illustrates, if nothing else, the way in which the language changes as it develops.

I expect at least one more pre-type-system release, the major feature of which will be a primitive form of exception handling, forming the basis of a fuller implementation when the type system starts to flesh out.

There have been a number of corrections to the documentation. For non-Windows users, the shell script “`aflmc.sh`” has been renamed “`aflmc`”. As well, the following changes have been made to the language:

- The **parent** keyword, and the dot-qualified forms of **parent**, **self** and **throw** have been removed.

  These queries depended too much on a particular implementation technique for the language, excluding some implementation techniques I want to experiment with, made future static analysis overly difficult, and don't seem to add to the language's functionality. So out they go.

- The **fn** and **cont** keywords have been removed from the language.

  The only things you could previously do that needed these keywords were using **fn** to prefix multiple argument lists, and specify that operator arguments' evaluation was to be delayed. You can not specify multiply argument lists sans **fn** with the same effect as of old:

  ```
  {a} {b} : a + b  # was: fn {a} {b} : a + b
  ```

  and the **fn** keyword used in operator arguments has been replaced by "()" (a zero-length parenthesized group), as in:

  ```
  def if [90 a] then [90 b ()] else [90 c ()]:
      a (b, c) ();
  # was:
  # def if [90 a] then [90 b fn] else [90 c fn]:
  #     a (b, c) ();
  ```

  The removal of the **fn** and **cont** keywords simplifies the language. They were initially used to help make programs more readable, but the variety of resulting forms seems to work in the other direction. As well, the use of () in function arguments seems to be a better choice than **fn**, as the () following the argument name brings its syntax closer to that of declaring the argument name as a function.

- The way interpolated values in literal strings are handled has changed. Previously each \{ ... } group defined a new frame, and any defined values within the group were local to that frame. Now, each literal string value defines a frame and the interpolated groups within it are considered parts of that frame. Most importantly, a program that is compiled in "template mode" (`/k=t`) can now define "global" names within the template.

  To complement this change, the '{ ... } operation has been added. It sets and accesses a frame's string value, providing a lower-level rewrite for template strings, and to lower the boundary between template and non-template oriented programs.

- "Grouped" arguments are now supported for user-defined operators, as in:

  ```
  def if [() a] [90 b ()] else [90 c ()] :
      a (b, c) ();
  # as in: if (a < 0) print "negative"
  #        else print "non-negative";
  ```

  Grouped operators have two or more arguments with no intervening operator name, but required that all but the last argument in a sequence be parenthesized, braced or bracketed.

  Grouped arguments allow experimenting with C/Java-like syntax.

- A number of operators have been added to the language:
  - The "do ... while ..." looping form has been added. (afldefs.afl)
  - The "<<" prefix and infix operators have been added. (io.afl) It implements C++-like output, as in:

    ```
    << "Hello World" << lf;
    ```

  - The "ungenerate" function has been added. (afldefs.afl) It converts a generator into a coroutine. For example:

    ```
    def nextNumber:
        ungenerate (1 to 1000, exit);
    ```

  defines "nextNumber" as a function that returns a new numeric value on each call.

The language changes and new operators are described in more detail in (`www.wilmott.ca/afl/afloverview.html`) AFL Overview.