From *B* to ABC: the Salient Changes

Lambert Meertens
CWI

The past year, we have been working hard on the revision from *B* to ABC, making an inventory of proposed changes (using the experience collected over several years of use) and debating their pros and cons. There are two problems in that kind of undertaking. One is that new proposals may have unexpected drawbacks, and so you must be careful in adopting changes, more so than for a first 'trial' version of a language. This danger can be minimised by only considering minor changes, and in fact we felt no urge to do otherwise, since we were already unreasonably happy with unrevised *B* and only wanted to give a final high polish. The second problem is that two or more nice suggestions, fine enough by themselves to be voted in by acclamation, may be incompatible if taken together. Here, it is a good thing to 'revise by repeated approximation': take preliminary decisions on the proposals, and let the result simmer for some time to see where conflicts arise, revise the revision again etc. until stability.

We think we have now reached a point of stability, more or less. Not fully, since (a) we are still open to further input, especially if it does not entail a complete upheaval of everything that *B* stands for, and (b) we decided to categorise the changes in three classes: 'Hurry', 'Nice', and 'Wait'. This has to do with the fact that we want to bring out a next release of our system, as soon as possible, that understands (revised) ABC.

Now, some – in fact, not a few – of the proposed changes are so minor that it is hard to explain the difference with the current situation. These may have to do with behaviour in marginal situations for which user-oriented semantic descriptions, like in The *B* Programmer's Handbook, are naturally ambiguously phrased. To give one example, in order to allay the reader's now piqued curiosity: The Handbook does not specify where the extra space character should go if a text of even length

is centred in a field of odd length, as in `"at"><7`. This is defined in the much more precise Draft Proposal, the Holy Writ for implementers, and there is now a proposal to make a change there. We do not feel compelled to implement such minor changes in the first ABC'd release of the system. Therefore, this change has status: 'Wait' – that is, no 'Hurry'. Some other proposed changes are not that minor, but are strictly 'upwards compatible': they do not invalidate any existing *B* code, but impart a meaning to hitherto undefined things. This refers both to certain new syntactic forms, as to semantic extensions of existing constructs. Here, again, we felt free to confer the status 'Wait' to the proposals.

Other proposed changes are very visible and so got the label 'Hurry'. Here, existing *B* code is invalidated, either because the syntax has changed, or because the syntax is still the same but the semantics is different. Of course, we have tried to minimise such changes, especially in the group of 'same form, different content'. If we adopted one of those, we felt the merits to be substantial enough to outweigh the inconvenience to the existing user community (and don't forget that we are big-time users of *B* ourselves). To minimise the pain, the next release will in some cases still accept the old syntax style next to the new one, similar to the behaviour of the (then only) C compiler when the syntax of, among others, initialisations was changed. Possibly, this release will also come with some 'off-line' conversion tool. (The first *B* implementation pulled another stunt in this area. It was finished earlier than the Draft Proposal, and was updated along with the final polishing of that document. If it recognised a piece of syntax that got changed recently, it not only accepted the obsolete form, but recast it into current syntax and stored that back. Very convenient, that; we shall probably not go as far out of our way this time.)

The proposals that we categorised as 'Wait' are generally of minor import to the user, true trifles. Some changes, however, among those of the strictly-upwards-compatible kind, are such improvements that we would really like to bring them out in the very first next release. These constitute the 'Nice' category. Whether we succeed in incorporating all of them, we doubt, but most will be in there. (There are degrees of 'niceness' that will determine our priorities.)

This long, but perhaps interesting by itself, digression was needed to explain what I meant with 'stability'. There is another categorisation: proposals can be in the (currently) 'Accepted' or 'Rejected' sets, but also in the (as of yet) 'Undecided' set, which is where they all were, initially. The point where we are now, is that all proposals still in the Undecided category have the label 'Wait': whatever the

final decision, it will not influence the next release of the system, nor the wording in user-oriented documentation. In most of these cases, the proposal is still undecided because it appears to have possible ramifications that we could not quite fathom – safer, then, to let it stay in limbo. If one of these Waiters turns out, on further examination, to bite a Nice or even Hurried change, it will be summarily rejected.

Some statistics. We started out with a collection of some 69 proposals. (Sometimes it is a matter of taste if you count something as one single proposal or as two closely related proposals.) Of these, 36 are now Accepted and 15 Rejected, leaving 18 Undecided. Of the thirty-six chosen ones, 22 are Hurried, 7 are Nice, and another 7 lie in Wait.

Below, I only describe the more important changes. (Some of the Accepted–Hurry ones are too petty to mention here. A full account is available on request.)

- The most prominent change, one that will in fact influence every existing *B* program, is that `HOW'TO` will henceforth be spelled `HOW TO`.
- Not less dramatic is the following change: instead of `YIELD` we have now `HOW TO RETURN`, and instead of `TEST` similarly `HOW TO REPORT`. So all units will start with `HOW TO`. It is no longer appropriate to call command definitions 'how-to-units' then, and we shall rename them 'command-definitions'. Instead of 'yield-units' we have then 'function-definitions', and 'test-units' become 'predicate-definitions'.
- The signs `''` and `"` are allowed in names (tags) and keywords, and spaces are not. In current *B*, the `''`-sign provides a way to introduce pseudo-spaces, as in '`last'time'seen`'. In the future, a '`.`'-sign can be used for the same purpose: '`last.time.seen`'. This is an addition: the use of `''` and `"` stays allowed since it is handy for such locutions as '`x''`'. For a long time, we hesitated between the signs '`.`', '`_`' and '`~`'. The decisive advantage of the '`.`' is that it is a lower-case character on all keyboards (except the Struldbrugg-YUC models).
- Moreover, the name of a user-defined command will no longer consist of its first keyword, but of all keywords together preceding the first parameter, if any (and otherwise simply all keywords). So then users can define and use commands like `PUT OFF task`. The predefined command `SET'RANDOM` is also changed into `SET RANDOM`. A further proposal is to consider the whole 'keyword skeleton' as determining the command name, allowing

`PUT a OVER b` next to `PUT a IN b`. This last business is Undecided: we fear adverse effects on the friendliness of the editor.

- Comparison between exact and approximate numbers will be changed in accordance with the proposal in issue 4 of The *B* Newsletter. Henceforth, `root 4 = 2` shall succeed.

- The parameter mechanism for user-defined commands, currently call-by-name as in ALGOL 60, will be replaced by copy-restore. Side effects of evaluating or locating a parameter (which, because of the scratchpad-copy semantics of expressions, are already restricted to run-time errors, I/O, or changing the state of the pseudo-random-number generator) are thus incurred only once. The main change, therefore, is in the reduction of aliasing. Moreover, putting different values in actually identical output parameters will be a (dynamically) signalled error, exactly as happens already in

  ```
  PUT 1, 2 IN i, i.
  ```

  It will remain an error to put a value in a formal parameter that does not correspond to an actual *target* parameter.

- Also, multiple (collateral) names will be allowed as formal parameters in a command definition, e.g. `HOW TO SWAP a, b`, so that the parameter mechanisms of commands and of functions and predicates are almost completely unified – differences are that the parameters of functions and predicates are *always* copied in, and that there is no 'restore' (which would be invisible anyway because of the scratchpad copying), so that the formal parameter can be used as a local target, even if the actual parameter is a formula. If one of the fields of a multiple formal parameter is assigned to, but not all fields, then either the actual must be multiple as well, or it may be single but must then be already initialised at the time of the call. This prevents the creation of compound locations not all of whose fields have a value.

- A new command `PASS` will serve to specify the dummy (null) action. Empty command-suites will no longer do: each command-suite must contain at least one command. Next to improving clarity, this also does away with the

problem that an indentation error, as in

```
IF x > 1:
PUT x-1 IN x
```

gets uncaught; this will cause a syntax error in the future. (In fact, the editor will leave a hole for the missing command-suite.)

- The PARSING construct will be replaced by a somewhat less general, but in other respects more powerful and in any case more user-friendly construct, although probably not in the next release. Even then, PARSING will be gone (at least from the documentation; it may temporarily remain in the system to alleviate switch-over problems). The big problem with PARSING was that it was so excruciatingly slow that no-one in their right minds used it in the first place. This was not a matter of a more efficient implementation; its inefficiency is inherent to its semantics. The new form, then, will be a FITS test of the form

  tex FITS *inf* ,

  in which *tex* is a textual-expression and *inf* is a new animal, dubbed an 'input-format'. May some examples suffice instead of a formal definition:

  ```
  "$ 123.45" FITS "$`x EG 0`"
  ```

  will succeed and put the value (number) 123.45 in x. This x is a bound tag, of course: its scope is the part that is reachable only by dint of the success of the test.

  ```
  "f 123.45" FITS "`cur RAW``x EG 0`"
  ```

  succeeds as well, and also puts the text "f " in cur.

  ```
  line FITS "`k RAW`: `i RAW`"
  ```

  has the same meaning as the test in current *B*:

  ```
  SOME k, c, i PARSING line HAS c = ": "
  ```

except, of course, that the dummy target c has gone. Thus, the first occurrence of ": " is used to split the text contained in line. The FITS construct fails if no parsing will make the expression 'fit' the input-format; moreover, the parsing strategy is 'greedy', just as with the present PARSING construct.

If the body of the input-format consists of a single 'input-conversion', the enclosing quotes may be dropped. So, to convert a text to a pair of numbers, the following test suffices:

```
t FITS x, y EG 0, 0
```

There are some murky matters not yet completely cleared up (like whether

```
"12321" FITS "`x EG 0`2`y EG 0"
```

is in error or puts 1, 321 in x, y, or perhaps 123, 1), which is one of the reasons why this might not yet show up in the next release.

- Input-formats are also allowed following READ. In fact, the existing forms READ x EG y and READ x RAW become then special cases of this. The execution of a command like

```
READ "Article: `a RAW` (Code: `c EG 0`)"
```

will prompt the user with

```
Article: ? (Code: ?)
```

Again, this is not expected for the next release.

- The command CHOOSE will be replaced by a monadic function choice, making it possible to do something like

```
TAKE choice exits
```

in a single command. Similarly, DRAW is replaced by a zeroadic function random, allowing the use of expressions like

```
(root(-2*log(1-random)))*sin(2*pi*random)
```

(which has a normal Gaussian distribution).

- The following kind of iteration may be added:

```
FOR [k]: i IN table: ...
```

  to allow an easy way to traverse the keys and associated items of a table together. Although rather nice, this change is still on the waiting list of undecided proposals.

- The `E` in the floating-point number format is going to be replaced by a lowercase `e`. The primary reason for this change is the improved legibility; a secondary reason is the disappearance of an anomaly in the editor to cope with the case of commands with a keyword `E`. Furthermore, on input, numbers in this format will be handled as *exact* numbers; thus, `1e9` evaluates to the same number as `1*10**9`. If an approximate number is intended, the user has to indicate this explicitly, as in `~1e9`.

- We have noticed that errors of the form exemplified by

```
SELECT:
   line|8 = "CONTINUE": GO ON
   line|3 = "BYE": QUIT
   ELSE: UNKNOWN COMMAND
```

  are rather more common than we had anticipated. The problem is that if `line` contains the text `"BYE"`, then the evaluation of `line|8` causes a run-time error. In ABC, `t|n` will be equivalent to `t|min{n; #t}` in current *B*, so that the code fragment above works as expected. Similarly, `t@n` will be equivalent to `t@max{n; 1}`. If `n < 0`, the evaluation of `t|n` will still signal an error, and similarly for `n > #t+1` and `t@n`.

- The evaluation of a list-display like `{1..n}` will no longer signal an error in ABC if `n < 0`; its value is simply the empty list. So, unlike in current *B*, the following is a safe way to compute the intersection of two non-empty ranges of integers `r` and `s`:

```
{max{min r; min s}..min{max r; max s}}
```

  This change, as well as the following, is only ‘Nice’.

- The following will all be valid ABC:

```
{1..i-1; i+1..n}
{-a..a; -b..b}
{1; 2; 4..9; 5; 11}
```

- The priorities of the operators will be slightly changed, allowing, among others, the currently not permitted

```
2 round x >> 7
```

  but ruling out the currently allowed

```
t|2^^n
```

  (and some ambiguities permitted by the Draft Proposal as well).

- The name of the function `atan` will be changed to `arctan`. Possibly, also `arcsin` and `arccos` will be added. Instead of dyadic `x atan y`, there will be `angle(x, y)`, and then a Nice addition is its natural companion `radius(x, y)`, computing the same as `root(x**2+y**2)`, but offering for many applications a far more convenient diction. In particular, if coordinates `(x, y)` are stored as table items, say, you can use expressions like `angle t[i]` without having to unpack the compound first. The value of `angle(0, 0)`, unlike that of `0 atan 0` in unrevised *B*, is not undefined, but zero.
- Another proposal in the Nice category: to facilitate computation with degrees instead of with radians, dyadic versions of all trigonometric and cyclometric functions are added in ABC. The first parameter is the value of a full circular arc expressed in the units desired. For example, since a full circle is 360°, use `360 sin 15` to compute the sine of 15°.
- Some new functions will simplify text handling, especially, but not only, user-supplied interactive input text. To convert to lower and upper case, we

have in ABC:

```
lower "E.E. Cummings"
upper "L19: cvtbl *-12(fp),r0"
```

returning respectively:

```
"e.e. cummings"
"L19: CVTBL *-12(FP),R0"
```

The function `stripped` will return its argument with all leading and trailing blanks removed. To see if the reply to a question is any of `"yes"`, `"Y"`, `"  yep!!"`, etc., it suffices to test if

```
(stripped lower reply)|1 = "y"
```

Nice would be a function `words`, or `split`, that takes a text and splits it (on the blanks) into a sequence of words.

- The formula `n th'of s`, with its attendant abominations like `2 th'of s` and the frequent mistake `n'th of s`, will be traded in for `s item n`. The order of the parameters is more in line with other selections like `s[n]` and `s@n|1`.

We have also designed several extensions to ABC, for example for matrix/vector functions (see issue 4 of The *B* Newsletter) and for graphics. These will not be in the next release. In fact, if they will be offered in the future, they will be possibly be separate additions with their own manuals that people may or may not acquire.

Going beyond this, we are also considering extensions that are primarily intended for applications designers. Among these are a facility for dealing with data entry (a generalisation of the `FITS` test), and also a possibility for adding new, user- (or in fact designer-) defined, types (that are atomic to the end-user).

There is a designed addition that we have not yet classified; maybe it should be incorporated in 'general' ABC and not be part of an extension. It is the addition of a new sequence type. Although it is possible to use tables to model sequences, some natural operations on sequence are difficult to handle; in particular, joining two sequences and extracting a (contiguous) subsequence. The proposed addition would introduce sequences as arbitrarily long, ordered but not sorted, collections

of items of the same type. A possible syntax for sequence-displays is that of

```
{0;; 1;; 0;; 2;; 0;; 1;; 0;; 3},
```

that is, like a list-display except that the semicolons are doubled. Not an obvious candidate for a beauty contest, but we haven't been able to think of anything nicer that does not conflict with existing syntax. The notation `{}`, already doing double duty for empty lists *and* tables, would acquire a third meaning: that of an empty sequence. The displays `{a}` and `{a..b}`, currently only lists, would also get an extra meaning: they would also be acceptable sequence-displays. The static type-finding algorithm is easily amended to cope with this. The generic operations on texts, lists and tables (iterating with `FOR` or with `SOME`/`EACH`/`NO`, the function `item` and the monadic and dyadic versions of `#`, `min` and `max`) would extend to sequences. Other operations on sequences would be: the join function `^` and the repeat function `^^`, and the trims `@` and `|` (now only defined on texts). A new monadic function `items` can be used to turn a text, list or table into a sequence; the effect of `FOR i IN tlt: ...` is the same as that of `FOR i IN items tlt: ...`. Like `keys`, this function takes constant time, independent of the size of its parameter. (Using `items` on a sequence is also permitted; the result is the same as the parameter supplied.)

Although an obvious possibility to consider, it will probably not be the case that texts are a special case of sequences. There is a difference between `"abc"` and `{"a";; "b";; "c"}` since identifying these would imply that `"a"` is the same as `{"a"}`, which looks undesirable. But this needs further study; maybe there is a satisfactory and consistent solution to this.