Decimal radices in bits denotations

GNU68-2026-003 (draft)

by Jose E. Marchesi

Copyright © 2026 Jose E. Marchesi.

You can redistribute and/or modify this document under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

Foreword

The following specification has been released under the auspices of the GNU Algol 68 Working Group, and has been scrutinized to ensure that

a. it is strictly upwards-compatible with Algol 68,
b. it is consistent with the philosophy and orthogonal framework of the language, and
c. it fills a clearly discernible gap in the expressive power of that language.

The source of this document can be found at `https://git.sr.ht/~jemarch/gnu68`.

The informal description of this proposal introduces the proposed new language features, providing a rationale and usage examples.

The formal definition of this proposal uses the existing formalism and conventions of the Revised Report, and it is expressed as modifications to the Report.

Finally, the implementation notes of this proposal describes a way in which the features added by this specification can be implemented. No implementer should feel committed to do things as described there; the same language facilities may well be implementable in other ways, more suitable to specific implementations.

1 Informal Description

bits values as unsigned quantities

Algol 68 integral values are always signed. As such, the integral modes **short int**, **int**, **long int** etc, and the arithmetical, relational and other operations defined on them all assume signed quantities.

This, by itself, and assuming enough precision in the underlying machine representation of the quantities, is not a problem, because the domain of signed numbers covers the domain of non-negative numbers, so in principle there is no need to provide a mode just for non-negative numbers. On the contrary, using the same mode avoid unnecessary conversions in many cases, and leads to a more general language with less concepts and less rules. It is an useful feature.

However, once we step out of the immanency of the language and start communicating with the outside world, be it via transput or via procedure calls, the absence of unsigned integral values and modes becomes an actual problem, as it is not possible to cover the domain of an unsigned integral mode with an Algol 68 integral mode of the same precision (measured in number of bits, for example) because the later is signed. We are then forced to use a higher precision signed integral on the Algol 68 side to accommodate an unsigned value with lower precision on the other side, and since the precision has an impact on the interface (higher precision means more storage and possibly even different means of passing arguments) it becomes difficult, if not impossible, to use the foreign interfaces as such in these situations.

For example, suppose we want to call a function defined in C as `void frob (unsigned int n)`. If we are using the GCC Algol 68 compiler, we now that the precision of a C `unsigned int` is the same than the precision of an Algol 68 **int**, so we could wrap the C function using a formal hole **proc(int)void** `myfrob` = **nest C** `"frob"`. However, the Algol 68 **int** parameter does not cover the entire domain of the C `unsigned int` parameter: it is not possible to, for example, pass `max_int+1`. Changing the formal hole so an Algol 68 **long int** is used to pass the parameter does not fix the problem, as it very likely breaks the interface.

The Algol 68 standard modes `SIZETY` **bits** were originally intended in order to efficiently transport the different constituent bits of integral values, understood as truth values, and to operate on them with logical operations. To ease this, the language guarantees that the precision of any

given `SIZETY` **bits** mode is be the same than the precision of the corresponding `SIZETY` **int**. It is therefore possible to convey integral values in **bits** values.

A common technique is therefore to use **bits** values in order to convey unsigned quantities achieving the precision expected externally. In the above example, we would interface the external `frob` with something like **proc(bits)void** `myfrob` = **nest C** `"frob";`.

Decimal radix in bits denotations

The set of allowed radices in **bits** denotations are 2, 4, 8 and 16. This is explicit in the Algol 68 grammar, and it clearly reflects the intention of **bits** values to be used to convey bits and to operate on them: all these radices correspond to numeric bases (binary, octal, hexadecimal, ...) which are often used in contexts in which the bits structure of the denoted quantities should be easily recognizable. In this domain the decimal base is not very useful.

However, as we have seen, it is common to use **bits** values to convey unsigned integral values that fit in some given precision (number of bits) eases interfacing with programs written in other languages that support unsigned integral modes. In these cases, using decimal might be most natural.

This extension allows to specify `SIZETY` **bits** denotations in decimal by using the radix `10r`, such as in **short short bits** `top = 10r255`.

2 Formal Description

The radix `10r` is added to the several allowed radices in bits denotations.

```
8.2 Bits denotations

8.2.1 Syntax

A) RADIX :: radix two ; radix four ; radix eight ; radix ten ; radix sixteen.

c) structured wih row of boolean field
              letter aleph mode denotation{a,b,80a} :
     RADIX{d,e,f,g}, letter r symbol{94a}, RADIX digit{h,i,j,k,n} sequence.

[...]

j) radix eight digit{c,n} :
     radix four digit{i} ; digit four symbol{94b} ;
     digit five symbol{94b} ; digit six symbol{94b} ;
     digit seven symbol{94b}.
n) radix ten digit{c,k} :
     radix eight digit{c} ; digit eight symbol{94b} ;
     digit nine symbol{94b}.
k) radix sixteen digit{c} :
     radix ten digit{n} ; letter a symbol{94a} ;
     letter b symbol{94a} ; letter e symbol{94a} ; letter d symbol{94a} ;
     letter e symbol{94a} ; letter f symbol{94a}.

[...]
```

3 Implementation Notes

It is recommended for the implementation to check and diagnose overflow in **bits** denotations, even if its effect is well defined by the implementation, as unsigned overflow often is.