INTERNATIONAL STANDARD ISO/IEC 13211-1:1995
TECHNICAL CORRIGENDUM 3

Published 2017-07

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION • МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ • ORGANISATION INTERNATIONALE DE NORMALISATION
INTERNATIONAL ELECTROTECHNICAL COMMISSION • МЕЖДУНАРОДНАЯ ЭЛЕКТРОТЕХНИЧЕСКАЯ КОМИССИЯ • COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

Information technology — Programming languages — Prolog —

Part 1: General core

TECHNICAL CORRIGENDUM 3

Technologies de l'information — Langages de programmation — Prolog —

Partie 1: Noyau général

RECTIFICATIF TECHNIQUE 3

Technical Corrigendum 3 to ISO/IEC 13211-1:1995 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

ICS 35.060 **Ref. No. ISO/IEC 13211-1:1995/Cor.3:2017(E)**

Published in Switzerland

Allow explicitly extensions to options in 5.5 Extensions. Add new subclause

5.5.12 Options

A processor may support one or more additional options such as stream-options (7.10.2.11), close-options (7.10.2.12), read-options (7.10.3), and write-options (7.10.4) as an implementation specific feature. An invalid option E shall be associated with only two error conditions: an instantiation error when there is an instance (3.95) of E that is a valid option, and a domain error for the domain *optname*_option when there is no instance of E that is a valid option. Further, an instantiation error may occur in place of the domain error if a component of E is a variable, and an instantiated component is required.

NOTE — A valid option may be associated with other error conditions like 8.11.5.3 l and m.

In 6.2.1 Prolog text, add an optional layout text sequence to the fourth and last production

p text = [layout text sequence (* 6.4.1 *)] ;

In 7.1.6.3 a, Iterated-goal term, replace unifies with *by* has the form

a) If T has the form ^(_, G) then ...

Add term-to-body conversion in 7.8.3.4 Examples. Replace first paragraph by:

Table 21 and 22 show the execution stack before and
after executing the control construct call(G) with *goal*
obtained from G in step 7.8.3.1 f via 7.6.2.

Replace in Table 22 G *by* goal .

N + 1 ((*goal*, *N* – 1), Σ *nil*

In 7.10.3 Read-options list, clarify unification for variables/1, specify left-to-right traversal order in read-options variable_names/1 and singletons/1. Replace the three paragraphs by:

variables(Vars) — After inputting a term, Vars shall be unified with a list of the variables in the term input, in left-to-right traversal order.

variable_names(VN_list) — After inputting a term, VN_list shall be unified with a list of elements where: (1) each element is a term A = V, and
(2) V is a named variable of the term, and (3) A is an
atom whose name is the characters of V, and (4) there is exactly one element for each named variable, and (5) the elements appear in the order of the first occurrence of their variables V in the term input, in left-to-right traversal order.

singletons(VN_list) — After inputting a term,
VN_list shall be unified with a list of elements
where: (1) each element is a term A = V, and (2) V is
a named variable which occurs only once in the term,
and (3) A is an atom whose name is the characters of V,
and (4) there is exactly one element for each named variable occurring only once, and
(5) the elements appear in the order of the first occurrence of their variables V in the
term input, in left-to-right traversal order.

Remove "ground" in 3.206, add write-option variable_names(VN_list) before numbervars(Bool) in analogy to the read-option. Replace "an non-negative" by "a non-negative" in paragraph numbervars(Bool). Add Note 2

7.10.4 Write-options list

variable_names(VN_list) — Each variable V is output as the sequence of characters
defined by the syntax for the atom A iff a term A = V is an element of the list VN_list.
If more than one element applies, the leftmost is used. VN_list is a list of terms A = T
with A an atom and T any term, possibly a variable.

NOTE 2 — Many Prolog processors modified write option numbervars/1 to print arbitrary variable names. The write option variable_names/1 serves this purpose and avoids vulnerabilities.

In 7.10.5, add writing with variable_names/1, correct terminology and writing of {}, lists, extra round brackets. Add as first subclause a1, rename and replace subclause a by a2, e by e1, g by e2, add new subclause e3, replace subclauses f and g:

7.10.5 Writing a term

a1) If Term is a variable and there is an effective write-option variable_names(VN_list)
and there is an element A = Term of the list VN_list with A an atom, then A is output
with effective write-option quoted(false).

a2) Else if Term is a variable, a character sequence repre-
senting that variable is output. The sequence begins
with _ (underscore) and the remaining characters are
implementation dependent. During the execution of write_term/3, the
same character sequence
is used for each occurrence of a particular variable
and a different character sequence is used for each
distinct variable.

e1) If Term has the form '$VAR'(N) for some
non-negative integer N, and there is an effective write-option
numbervars(true), a variable name as defined in
subclause 7.10.4 is output,

e2) Else if Term has the form '.'(Head, Tail), and
there is an effective write-option ignore_ops(false),
then Term is output using list notation, that is:

1) [(open list char) is output.

2) Head is output by recursively applying these
rules. Head is preceded by ((open char)
and followed by) (close char), if the term could not be
re-input correctly with same set of current operators.

3) If Tail has the form '.'(H,T) then , (comma
char) is output, set Head:=H, Tail:=T, and goto (2).

4) If Tail is [] then a closing bracket] (close list
char) is output,

5) Else a | (head tail separator char) is output,
Tail is output by recursively applying these rules.
Tail is preceded by ((open char)
and followed by) (close char), if the term could not be
re-input correctly with same set of current operators.
And finally,] (close list char) is output.

e3) Else if Term has the form '{}'(Arg), and there is an effective write-option
ignore_ops(false), then Term is output as a curly bracketed term (6.3.6), that is:

1) { (open curly char) is output.

2) Arg is output by recursively applying these rules.

3) } (close curly char) is output.

f) Else if Term has a principal functor which is not
a current operator, or if there is an effective write-
option ignore_ops(true), then the term is output in
functional notation (6.3.3), that is:

1) The atom of the principal functor is output.

2) ((open char) is output.

3) Each argument of the term is output by recursively
applying these rules. The argument is preceded by ((open char) and followed by)
(close char), if the term could not be re-input correctly with the same set of
current operators.

4) , (comma char) is output between each successive
pair of arguments.

5)) (close char) is output.

h) Else if Term has a principal functor which is an operator, and there is an effective write-option ignore_ops(false), then the term is output in operator form, that is:

1) The atom of the principal functor is output in front of its argument (prefix operator), between its arguments (infix operator), or after its argument (postfix operator). In all cases, a space is output to separate an operator from its argument(s) if any ambiguity could otherwise arise.

Operators ',' and '|' are output as , (comma char) and | (bar char) respectively.

2) Each argument of the term is output by recursively applying these rules. An argument is preceded by ((open char) and followed by) (close char) if: (i) the argument's principal functor is an operator whose priority is so high that the term could not be re-input correctly with same set of current operators, or (ii) the argument is an atom which is a current operator, or (iii) the principal functor is output as a prefix operator - and the argument is a non-negative number, or (iv) the principal functor is output as a prefix operator - and the argument is output in infix or postfix operator form.

In 7.12.2 Error classification, add missing type, remove superfluous domain, add comma.

7.12.2 b: Add float to the set ValidType.
7.12.2 c: Remove character_code_list from the set ValidDomain.
7.12.2 e: add comma to PermissionType ∈ { binary_stream, flag

In 8.1.2.1, 8.11.5.2, 8.11.6.2, add _list to types close_options and stream_options. Add 8.1.3 Note 7 for options lists

8.1.3 Errors

7 When a built-in predicate has an argument Options whose type is a list of *optname*-options as input, the argument is always +*optname*_options_list. It is always associated with:

1. an instantiation error, when Options is a partial list, or an element of a list prefix of Options is not a valid option but an instance of the element is a valid option;

2. a type error of the form type_error(list, Options), when Options is neither a partial list nor a list;
3. a domain error of the form domain_error(*optname*_option, E), when an element E of a list prefix of Options is neither a valid option nor any instance of E is a valid option; an instantiation error may occur

   in place of the domain error, if a component is a variable, and an instantiated component is required.

A valid option may be associated with other error conditions.

In 8.5.1.4 Examples, add alternate error to second last example. Replace example by

```
current_prolog_flag(max_arity, A),
    X is A + 1,
    functor(T, foo, X).

  If the Prolog flag max_arity has the value unbounded
    type_error(evaluable, unbounded/0)
  else
    representation_error(max_arity).
```

In 8.9.2.1 e Description (assertz/1), replace B by G.

In 8.10.3.4 example no. 20: undo the change introduced in Cor.1. That is, keep the list [a, b, f(b), f(a)] as it originally was in IS 13211-1:1995.

In 8.11.4.1 add subclause b: b) the goal succeeds.

Replace the four subclauses 8.11.5.3 c, 8.11.6.3 b, 8.14.1.3 b, 8.14.2.3 b respectively:

c/b) Options is a partial list or has a list prefix with an element
E which is a variable or which has a component
which is a variable, and an instantiated component is required.
— instantiation_error.

Replace the four subclauses 8.11.5.3 i, 8.11.6.3 e, 8.14.1.3 e, 8.14.2.3 e using in place of stream-option for the latter three close-option, read-option, and write-option respectively:

i/e) An element E of a list prefix of the Options list is neither a
variable nor a stream-option and there is no instance of E that is a stream-option.
— domain_error(stream_option, E).

In 8.14.1.1 k, replace reference 6.4 *by* 6.2.2.

In 8.14.2.4 Examples, replace second example. Add three further examples

write_canonical([1,2,3]).
Succeeds, outputting the characters
'.'(1,'.'(2,'.'(3,[])))
to the current output stream.

write_term(1,[quoted(non_boolean)]).
domain_error(write_option,quoted(non_boolean)),

write_term(1,[quoted(B)]).
instantiation_error.

B = true, write_term(1,[quoted(B)]).
Succeeds, unifying B with true, and outputting
the character 1.

In 8.17.1.4 Examples, replace domain flag in error in fourth example by prolog_flag:

set_prolog_flag(date, 'July 1988').
domain_error(prolog_flag, date).

In 9.3.1.3 c, replace Y *by* VY.

In 9.3.10.3 e, replace error condition for (^)/2 when resulting value is not an integer but still a real number. Add note.

e) VX and VY are integers and VX is not equal to 1, 0, or –1 and VY is negative.
— type_error(float, VX).

NOTE — Error condition 9.3.10.3 e is satisfied when a float as an argument is needed for a defined result.

Replace and add examples in 9.3.10.4:

2^(-1).
type_error(float, 2).

2.0^(-1).
Evaluates to the value 0.5.