Information technology - Programming languages - Prolog - Part 1: General Core

DRAFT TECHNICAL CORRIGENDUM 1

Draft technical corrigendum 1 to International Standard 13211-1:1995 (E) was prepared by Joint Technical Committee ISO/IEC JTC 1, Information technology

----

NOTE - Some text in Mathematical font is expressed using Latex convention, i.e. surrounded with '$' signs.

3.106 mapping

'mapping' is used with a second meaning in the standard: add a second definition

A function from a value of one type T to a value of another type R denoted by $T \rightarrow R$

3.108 most general unifier (MGU)

Replace 'instance' by 'example' because 'instance' is not being used with the meaning defined in 3.95.

3.125 partial list

Replace 'A variable' by 'A *variable*'.

Replace 'second argument' by 'second *argument*'.

3.148 read-term

Replace 'end token.' by 'end token'.

4.1.3.5 Axiom

Replace:

Axiom: if $x > 0$ then $\sqrt{x}$ is the positive square root of x else **undefined**.

by

Axiom: if $x \geq 0$ then $\sqrt{x}$ is the non-negative square root of x else **undefined**.

6.3.7 Term -- double quoted list notation

If a double quoted list represents an atom (i.e. the Prolog flag 'double_quotes' has value 'atom'), the priority of the term should depend on whether or not the atom is an operator as in 6.3.1.3. ISO/IEC 13211-1 states that the priority of an atom represented by a double quoted list is always zero.

Replace the syntax rule by the four syntax rules:

```
term = double quoted list ;
```
Abstract: *l* *dql*
Priority: 0
Condition: Prolog flag `double_quotes` has value `chars`

```
term = double quoted list ;
```
Abstract: *l* *dql*
Priority: 0
Condition: Prolog flag `double_quotes` has value `code`

```
atom = double quoted list ;
```
Abstract: *a* *dql*
Priority: *n*
Condition: Prolog flag `double_quotes` has value `atom`
Condition: *a* is an operator

```
atom = double quoted list ;
```
Abstract: *a* *dql*
Priority: 0
Condition: Prolog flag `double_quotes` has value `atom`
Condition: *a* is not an operator

7.2.5 c) 2)

Replace

2) if XN is the ...

by

2) XN is the ...

7.8.5.4

Replace the first sentence:

Tables 27 and 28 show the execution stack before and after executing the control construct `','(First, second)`.

by

Tables 27 and 28 show the execution stack before and after executing the control construct `','(First, Second)`.

Table 35 line 2

Replace

```
(else(W), CP)
```

by

```
(Else, CP)
```

7.8.8.4 last example

Replace

```
';'('->'(!,fail), true), true).
```

by

```
';'(('->'(!,fail), true), true).
```

7.9.2

Add additional errors:

i) The value of an argument `Culprit` is not a member of the set I

- `type_error(integer, Culprit)`

j) The value of an argument `Culprit` is not a member of the set F

- `type_error(float, Culprit)`

9.1.7 example no. 35 shows these errors are required.

7.12.2 i)

Twice replace

`imp_dep_atom`

by

`Imp_dep_atom`

8.8.1.1 d)

Replace
Chooses the first element of the list L
by
Chooses the first element of the list L, unifies it with the term `clause(Head,Body)`

Similarly for f).

8.9.4.1 abolish/1: Description

In the note, replace ‘procedures identified’ by ‘procedure identified’.

8.10.3.4 example no. 20

Replace

```
[a, b, f(b), f(a)]
```

by

```
[a, b, f(a), f(b)]
```

8.13.3.4 put_byte/1

Replace

```
put_byte(84).
  If the current output stream contains
    [..., 113,119,101,114]
  Succeeds, and leaves that stream
    [..., 113,119,101,114,116]

put_byte(st_o, 84).
  If the stream associated with st_o contains
    [..., 113,119,101,114]
  Succeeds, and leaves that stream
    [..., 113,119,101,114,116]
```

by

```
put_byte(84).
  If the current output stream contains
    [..., 113,119,101,114]
  Succeeds, and leaves that stream
    [..., 113,119,101,114,84]

put_byte(st_o, 116).
  If the stream associated with st_o contains
    [..., 113,119,101,114]
  Succeeds, and leaves that stream
    [..., 113,119,101,114,116]
```

8.14.1.4 examples no. 2 and 3

Replace

`st_o`

by

`st_i`

8.14.1.4 example no. 6 (last)

Replace

The current input stream is left with position past-end-of-stream.

by

The current input stream is left in an undefined state.

(Cf. 8.14.1.1 NOTE 2)

8.14.4.1 d)

Replace

Chooses a member of Set_Op and the goal succeeds

by

Chooses a member of Set_Op, unifies it with `(Priority, Op_specifier, Operator)`, and the goal succeeds

8.16.4 atom_chars/2

The sixth example in 8.16.4.4 is

```
atom_chars('North', ['N’ | X]).
  Succeeds, unifying X with
    ['o', ‘r', ‘t', ‘h'].
```

but the procedural description does not permit this.

Replace 8.16.4.1(c) by:

c) Else if `Atom` is an atom whose name is the sequence of characters Seq and `List` unifies with a list `L` such that every element of `L` is the one-char atom whose name is the corresponding element of Seq, then the goal succeeds,

8.16.5 atom_codes/2

The error noted in 8.16.4 implies a similar change in this procedure. Replace 8.16.5.1(c) by:

c) Else if `Atom` is an atom whose name is the sequence of characters Seq and `List` unifies with a list `L` such that every element of `L` is the character code of the corresponding element of Seq, then the goal succeeds,

9.1.4.1

Add a note pointing to the definition of F* (7.1.3.1).

9.1.7 example no. 21

Replace

```
'/'(7, 35)
```

by

```
'//'(7, 35)
```

9.1.7 example no. 23

Replace

```
'/'(140, 3+11)
```

by

```
'//'(140, 3+11)
```

9.1.7 example no. 24

Replace

```
14.200
```

by

```
1.4200
```

9.1.7 example no. 48

Replace

```
float(5/3)
```

by

```
float(5//3)
```

9.3.5.4 example no. 2 9.3.6.4 example no. 2

Replace

```
2.7818
```

by

```
2.71828
```

9.4.1.4 example no. 5, 9.4.2.4 example no. 5. 9.4.3.4 example no. 6, 9.4.4.4 example no. 6

Replace

```
type(integer,foo)
```

by

```
type_error(evaluable,foo/0)
```

Annex A

(informative)

Issues still to be resolved

The following points indicate possible problems with the standard, but do not specify corrections or cures.

6.4.2.1 Quoted characters

The text does not define what character is denoted by a ‘double quote char’ or ‘back quote char’ which is a ‘single quoted character’.

Similar omissions exist for ‘double quoted character’ and ‘back quoted character’.

6.5

Replace

It shall be implementation defined for each extended character whether it is a graphic char, or an alphanumeric char, or a solo char,

This classification is incomplete in the following sense:

- It is not enough to classify an extended character as an <<alphanumeric char>>, one has to also tell if it is a <<small letter char>>, <<capital letter char>> (e.g. NOTE 2 of 6.5 speaks of <<extended small letter char>>).

- The notion of <<solo char>> is unfortunate: it contains characters that form a <<name token>> alone (! and ;), and punctuation characters, which cannot be part of an unquoted <<name token>>. Classifying an extended character as a <<solo char>> does not make this character usable as a token alone, because there is no syntactic rule for this (! and ; are included explicitly in <<name token>>).

Along the lines of the Quintus/SICStus Prolog syntax, perhaps the category of <<solo char>> should be changed to mean only <<cut char>> and <<semicolon char>>, and a new category <<punctuation char>> is introduced to contain all other characters presently classified as solo. And there should be a non-terminal <<solo token>>, replacing <<semicolon token>> and <<cut token>> in <<name token>>, which is defined as:

solo token = solo char ;

The standard should let the user classify an extended character by placing it in exactly one of the following character categories:

graphic char
small letter char
capital letter char
solo char
layout char

In addition to these, the notion of <<char>> covers the categories <<decimal digit char>>, <<underline char>>, <<punctuation char>> and <<meta char>> - It may not make sense to allow the user to classify extended characters into one of these categories. Perhaps it may be useful to have a category <<other char>>, initially empty; any extended character classified as such would only be allowed in (double, back) quoted tokens.

A wide character extension to SICStus Prolog allows the user to plug in arbitrary character sets and define, through C hook functions, the character-type mapping. This is a function taking a character code, and returning a constant denoting one of the above character categories. This extension is part of the SICStus Prolog 3.8 release.

I hope the term <<implementation defined>> in the Prolog standard does allow such delegation of definitions to the user through hook functions.

6.5 and 7.1.4.1

The distinction between PCS (Processor character set) and C (the set of characters) is quite confusing at first reading. It does make sense, for example, in SICStus Prolog there is a member of PCS with character_code 0, which is not a character (not a member of C) - as there cannot be a one-char atom representing a ‘character’ with code 0.

Giving an example of this kind would help the uninitiated reader. Also replacing the name <<character>> by a qualified form, e.g. <<Prolog character>> would help. This is because currently the <<processor character set>> has members which are <<character>>s, and other members which are not - how would you call the letter ‘things"? For example in the previous paragraph, referring to the ‘character’ with code 0 was quite cumbersome, because formally the thing with code 0 is not a character.

7.2.2, 7.2.3 and 7.2.5

The use of the built-in predicates ‘<'/2 and ‘=='/2 in defining the term order is not very elegant. Perhaps using the mathematical relation <, and the notion of <<identical terms>> would be preferable.

7.4.2

A reference to the definition of <<predicate indicator sequence>> and <<predicate indicator list>> in 7.1.6 would be quite helpful here. Also perhaps these notions should be included in chapter 3.

7.6

Perhaps it would make sense to explicitly mention that an implementation may add other conversion steps, as an extension. The goal_expansion mechanism of SICStus Prolog is an example of such extensions.

7.7.12 a)

Where is the definition and purpose of the <<callable term representing the built-in predicate BP>>. Also the <<MGU>> produced here is never used.

Table 17

Why is the cut-parent of true shown to be equal to N-2?

7.8.3.3 c)

It might benefit the user if the Culprit in the error term here could be more precise than the whole argument of call/1. Maybe call((...,1)) raising type_error(callable, 1) should be allowed by the standard.

7.8.3.4 example no. 6

This example should not output ‘3’.

Table 25, Table 26

The cut-parent of the first subgoal of the execution state with index CP is shown as CP - this should be different.

7.8.8.1 o)

The introduction of a local cut in front of the Else part is artificial.

7.8.10 c)

This says that a throw without an applicable catch causes a system error. This error condition is different from other errors, because in this situation it does not make sense to apply the rules concerning the effect of an error: 7.12.1 says that the current goal should be replaced by the goal

```
throw(error(system_error,Imp_def)).
```

Perhaps it should be left implementation defined what should happen with uncaught exceptions.

7.8.10.1 e)

It does not seem right to replace CP by the cut-parent of *each* execution state.

7.8.10.1 g)

The text

```
R
```

is used without definition.

7.10.4 NOTE

Replace

The current operators do not affect output when there is a write option numbervars(true).

This seems to be garbled.

7.10.5 Writing a term

No method is provided for writing a curly bracketed term (6.3.6) with curly brackets.

7.10.5 d)

Some implementations, e.g. SICStus, write <<.>> as <<'.'>> when quoted(true) is in force, to avoid confusion with end token. It might be good if the standard allowed this.

7.12.2 NOTE 4 (b)

See the remark to 7.8.10 c)

8.1.3

Replace

A list of the error conditions and associated error terms for the built-in predicate

Is the order of this list relevant? It should not be. If two or more error conditions apply, any of them could be raised by a standard-conforming processor. Perhaps this issue is worth clarifying.

8.1.5

Consider

The error conditions and examples for a bootstrapped built-in predicate are included in the appropriate clauses of the general built-in predicate.

This is quite cumbersome, especially that error conditions use the argument names which appear only later, and only as part of the Prolog definitions. I would suggest to list the heads of the bootstrapped predicates earlier, so that the argument naming becomes more visible.

An example of this: 8.12.1.3.j) mentions the name <<Code>> which only appears buried in Prolog code on the next page in 8.12.1.5.

8.9.3.1 c)

Maybe replace 2) and 3) with

2) `(H:-B)` unifies with `(Head:-Body)`.

8.10.2.4 examples no. 5, 10 and 12

Mention that the order of solutions is undefined.

8.10.3.4 examples no. 22, 23 and 24 (last three)

Replacing <<Xs>> by <<Ys>> might make these examples easier to understand.

8.12.1.5

get_code/1 should be defined in terms of get_code/2, just as it is done for get_char:

```
get_code(Code) :-
   current_input(S),
   get_code(S, Code).
```

8.12.3.3 d) and j)

Error condition j) makes little sense, given error condition d).

8.13.3.4 example no. 4

The example is not fortunate, as the error permission_error(output, text_stream, user_output) would be also a valid outcome (see also the remark to 8.1.3).

8.14.4.3

I suggest to add an error condition, so that

```
current_op(_, 0, _)
```

raises the error type_error(atom, 0).

8.16.5.3

Maybe an error condition has to be added so that if atom_codes is called, e.g. as atom_codes(X, [foo]), it should raise a type_error(integer, foo).

8.16.4 -- 8.16.8

The error conditions of atom_codes and number_codes seem not to be in sync, and similarly for ..._chars. Example:

```
| ?- catch(atom_codes(f, [foo]), error(E,_),
true).
    no.
| ?- catch(number_codes(1, [foo]),
error(E,_), true).
    E = type_error(integer,foo) ?
    yes
```

9.1.4.3

Explain the need for the last sentence before the NOTE: The approximate-addition function should satisfy ...

9.1.6.1 last but one line

Replace
round_R->Z(x) = entier(x+1/2)>>

Is it not the case that the IEEE FP standard allows other forms of rounding, e.g. rounding an integer+0.5 to the nearest *even* integer? If so, could this be allowed in standard Prolog?

9.1.7 examples no. 5 10 15 20 28 34 45 50 55

These should raise the type_error(evaluable, foo/0) error, according to 7.9.2 c).

9.1.7 examples no. 25 and 26

Replace

```
'/'(7, -3)
```

by

```
'//'(7, -3)
```

And replace

```
'/'(-7, 3)
```

by

```
'//'(-7, 3)
```

Perhaps it is worth adding extending the text 'Evaluates to an implementation dependent value which is either -2 or -3'.

Annex B

(informative)

Editorial notes

Unusual characters

Check the following characters are correctly printed.

4.1.3.5 Square root symbol: √

4.1.3.5 Greater than or equal symbol: ≥

Background

These faults were noted after preparing for publication the text of ISO/IEC 13211-1:1995 Prolog: Part 1 - General Core, and subsequent lists of errors noted by Roger Scowen, Peter Szeredi, Pierre Deransart, and Ali Ed-Dbali.

Roger Scowen (editor)
9 Birchwood Grove, Hampton, Middlesex
United Kingdom TW12 3DU
Telephone: +44 (0) 20 8979 7429
E-Mail: roger.scowen@npl.co.uk
October - November 2005

Document history

2006 April 15: Two further corrections are noted (3.125, 3.148). The structure is amended to be based on ISO/IEC 1539-1:2004/Cor.1:2005 (E). Posted to Jonathan Hodgson for distribution and balloting within SC22.

December 1: Posted to Jonathan Hodgson for distribution to, and discussion within JTC 1 SC 22 WG17.

2005 October 26: Copied from c:\rs0\prolog\da58.tex, and stored in c:\rs5\standard\st77.doc. Examples and sources are removed.