INTERNATIONAL STANDARD

ISO/IEC 13211-1

First edition
1995-06-01

Information technology — Programming languages — Prolog —

Part 1: General core

Technologies de l'information — Langages de programmation — Prolog
Partie 1: Noyau général

Reference number
ISO/IEC 13211-1:1995(E)

Contents

Annex

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75% of the national bodies casting a vote.

International Standard ISO/IEC 13211 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

Annex A of this part of ISO/IEC 13211 is for information only.

Introduction

This is the first International Standard for Prolog, Part 1 (General Core). It was produced on 20 April 1995.

There is no other International Standard for Prolog.

Prolog (Programming in Logic) combines the concepts of logical and algorithmic programming, and is recognized not just as an important tool in AI (Artificial Intelligence) and expert systems, but as a general purpose high-level programming language with some unique properties.

The language originates from work in the early 1970s by Robert A. Kowalski while at Edinburgh University (and ever since at Imperial College, London) and Alain Colmerauer at the University of Aix-Marseilles in France. Their efforts led in 1972 to the use of formal logic as the basis for a programming language. Kowalski's research provided the theoretical framework, while Colmerauer's gave rise to the programming language Prolog. Colmerauer and his team then built the first interpreter, and David Warren at the AI Department, University of Edinburgh, produced the first compiler.

The crucial features of Prolog are unification and backtracking. Unification shows how two arbitrary structures can be made equal, and Prolog processors employ a search strategy which tries to find a solution to a problem by backtracking to other paths if any one particular search comes to a dead end.

Prolog is good for windowing and multimedia because of the ease of building complex data structures dynamically, and also because the concept of backing out of an operation is built into the language.

Prolog is taught in more UK university computing degrees than any other programming language.

This part of ISO/IEC 13211 defines the general core features of Prolog, and part 2 will define modules.

This page intentionally left blank

Information technology — Programming languages — Prolog — Part 1: General core

1 Scope

ISO/IEC 13211 is designed to promote the applicability and portability of Prolog text and data among a variety of data processing systems.

This part of ISO/IEC 13211 specifies:

a) The representation of Prolog text,

b) The syntax and constraints of the Prolog language,

c) The semantic rules for interpreting Prolog text,

d) The representation of input data to be processed by Prolog,

e) The representation of output produced by Prolog, and

f) The restrictions and limits imposed on a conforming Prolog processor.

NOTE — This part of ISO/IEC 13211 does not specify:

a) the size or complexity of Prolog text that will exceed the capacity of any specific data processing system or language processor, or the actions to be taken when the corresponding limits are exceeded;

b) the minimal requirements of a data processing system that is capable of supporting an implementation of a Prolog processor;

c) the methods of activating the Prolog processor or the set of commands used to control the environment in which Prolog text is prepared for execution and executed;

d) the mechanisms by which Prolog text is prepared for use by a data processing system;

e) the typographical representation of Prolog text published for human reading;

f) the user environment (top level loop, debugger, library system, editor, compiler etc.) of a Prolog processor.

This part of ISO/IEC 13211 is intended for use by implementors and knowledgeable programmers, and is not a tutorial.

1.1 Notes

Notes in this part of ISO/IEC 13211 have no effect on the language, Prolog text or Prolog processors that are defined as conforming to this part of ISO/IEC 13211. Reasons for including a note include:

a) Cross references to other clauses and subclauses of this part of ISO/IEC 13211 in order to help readers find their way around,

b) Warnings when a built-in predicate as defined in this part of ISO/IEC 13211 has a different meaning in some existing implementations.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 13211. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 13211 are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 646 : 1991, *Information technology — ISO 7-bit coded character set for information interchange.*

ISO 2382-15 : 1985, *Data processing — Vocabulary — Part 15: Programming languages.*

ISO 8859-1 : 1987, *Information technology — 8-bit single-byte coded graphic character sets — Part 1: Latin alphabet No. 1.*

ISO/IEC 9899 : 1990, *Programming languages — C.*

ISO/IEC TR 10034 : 1990, *Guidelines for the preparation of conformity clauses in programming language standards.*

ISO/IEC 10967-1 : 1994, *Information technology — Language independent arithmetic — Part 1: Integer and floating point arithmetic.*

BS 6154 : 1981, *Method of defining — Syntactic meta-language.*

3 Definitions

This terminology for Prolog has a format modelled on that of ISO 2382.

An entry consists of a phrase (in **bold type**) being defined, followed by its definition. Words and phrases defined in the glossary are printed in *italics* when they are used in other entries. When a definition contains two words or phrases defined in separate entries directly following each other (or separated only by a punctuation sign), * (an asterisk) separates them.

Words and phrases not defined in this glossary are assumed to have the meaning given in ISO 2382-15; if they do not appear in ISO 2382-15, then they are assumed to have their usual meaning.

For the purposes of ISO/IEC 13211, the following definitions apply:

3.1 A**:** The set of *atoms* (see 6.1.2 b, 7.1.4).

3.2 activation: The process of *executing* an *activator*.

3.3 activator: The result of preparing a *goal* for *execution* (see 7.7.3).

3.4 algorithm, Herbrand: See 3.85 — **Herbrand algorithm**.

3.5 alias: An *atom* associated with an open *stream* (see 7.10.2.2).

The standard input *stream* has the alias `user_input`, and the standard output *stream* has the alias `user_output` (see 7.10.2.3).

NOTE — A *stream* can have many aliases, but an *atom* can be the *alias* of at most one *stream*.

3.6 anonymous variable: A *variable* (represented in a *term* or *Prolog text* by _) which differs from every other *variable* (and anonymous variable) (see 6.1.2, 6.4.3).

3.7 argument: A *term* which is associated with a *predication* or *compound term*.

3.8 arithmetic data type: A *data type* whose values are members of $\mathcal{Z}$ or $\mathcal{R}$.

3.9 arity: The number of *arguments* of a *compound term*. Syntactically, a non-negative integer associated with a *functor* or *predicate*.

3.10 assert, to: To assert a *clause* is to add it to the *user-defined procedure* in the *database* defined by the *predicate* of that *clause*.

NOTE — It is unnecessary for the *user-defined procedure* to already exist.

3.11 associativity (of an *operator***):** Property of being non-associative, right-associative, or left-associative (see 6.3.4, table 4).

3.12 atom: A basic object, denoted by an *identifier* (see 6.1.2 b, 7.1.4).

3.13 atom, null: See 3.117 — **null atom**.

3.14 atom, one-char: See 3.119 — **one-char atom**.

3.15 atomic term: An *atom* or a *number*.

3.16 axiom: A rule satisfied by an operation and all values of the *data type* to which the operation belongs.

3.17 backtrack, to: To return to the *choicepoint* of the current *goal* in order to attempt to *re-execute* it (see 7.7.8).

3.18 bias, exponent: See 3.68 — **exponent bias**.

3.19 body: A *goal*, distinguished by its context as part of a *rule* (see 3.154).

3.20 bootstrapped (built-in predicate): Defined as a special case of a more general *built-in predicate* (see 8.1.5).

3.21 built-in predicate: A *procedure* whose *execution* is implemented by the *processor* (see 8).

3.22 byte: An integer in the range [0..255] (see 7.1.2.1).

3.23 C**:** The set of *characters* (see 7.1.4.1).

3.24 callable term: An *atom* or a *compound term*.

3.25 CC: The set of character codes (see 7.1.2.2).

3.26 character: A member of C — an *implementation defined* character set (see 6.5, 7.1.4.1).

3.27 character, quoted: See 3.144 — **quoted character**.

3.28 character, unquoted: See 3.194 — **unquoted character**.

3.29 character-conversion mapping: A *mapping* on the set of *characters*, C, which specifies that, in some *Prolog text* units and *sources*, some *characters* are intended to be equivalent to other *characters*, and *converted* to those *characters* (see 3.46, 7.4.2.5, 8.14.5).

3.30 choicepoint: A state during *execution* from which a *goal* can be *executed* in more than one way.

3.31 class (of an *operator*): The class of an *operator* defines whether it is a prefix, infix, or postfix *operator* (see 6.3.4).

3.32 clause: A *fact* or a *rule*. It has two parts: a *head*, and a *body*.

NOTE — In ISO/IEC International Standards "clause" has the meaning: one of the numbered paragraphs of a standard. In this part of ISO/IEC 13211, the context distinguishes the two meanings.

3.33 clause-term: A *read-term* `T`. in *Prolog text* where `T` does not have *principal functor* `(:-)/1` (see 6.2.1.2).

3.34 collating sequence: An *implementation defined* ordering defined on the set C of *characters* (see 6.6).

3.35 complete database: The set of *procedures* with respect to which *execution* is performed (see 7.5).

3.36 composition (of two *substitutions*): The mapping resulting from the application of the first *substitution* followed by the application of the second. Composition of the *substitutions* σ_1 and σ_2 is denoted $\sigma_1 \circ \sigma_2$. When the composition acts on a *term* t, it is denoted by $t\sigma_1\sigma_2$, with the meaning $((t\sigma_1)\sigma_2)$.

3.37 compound term: A *functor* of *arity* `N`, `N` positive, together with a sequence of `N` *arguments* (see 6.1.2 e, 7.1.5).

3.38 configuration: Host and target computers, any operating system(s) and software used to operate a *processor*.

3.39 conforming processor: A *processor* which conforms to all the compliance clauses (see 5.1) for *processors* in this part of ISO/IEC 13211.

3.40 conforming Prolog data: Sequences of *characters* and *bytes* that conform to all the compliance clauses for *Prolog data* in this part of ISO/IEC 13211 (see 5, 6.2.2).

3.41 conforming Prolog text: A sequence of *characters* that conforms to all the compliance clauses for *Prolog text* in this part of ISO/IEC 13211 (see 5, 6.2).

3.42 construct, control: See 3.45 — **control construct**.

3.43 constructor, list: See 3.100 — **list constructor**.

3.44 contain, to: A *term* `T1` contains another *term* `T2` if either `T1` and `T2` are *identical terms*, or `T1` is a *compound term*, one of whose *arguments* contains `T2`.

3.45 control construct: A *procedure* whose definition is part of the Prolog *processor* (see 7.8).

3.46 $Conv_C$: The *character-conversion mapping* on C (the set of *characters*) which specifies that, in some *Prolog text* units and *sources*, some *characters* are *converted* to other *characters* (see 3.29, 7.4.2.5, 8.14.5).

The initial value of $Conv_C$ shall be $identity\_mapping_C$.

NOTES

1 A directive or goal `char_conversion(In, Out)` (7.4.2.5, 8.14.5) replaces $Conv_C$ by $update\_mapping_C$(`In, Out,` $Conv_C$).

2 Any unquoted character `C` that is part of a read-term which is input by `read_term/3` (8.14.1) or as Prolog text is replaced by $apply\_mapping_C$(`C,` $Conv_C$).

3 $Conv_C$ can be inspected by calling `current_char_conversion/2` (8.14.6).

4 The rationale for providing this facility is because some extended character sets (for example, Japanese JIS character sets) are used with the basic character set and contain the characters equivalent to those in the basic character set with different encoding. In such cases, users will often wish the meaning of characters in Prolog data and Prolog text to be the same regardless of the encoding.

3.47 convert (from type A to type B): An operation whose *signature* is

$convert_{A \rightarrow B} : A \rightarrow B \cup \{\mathbf{error}\}$

which converts a value of *type* A to *type* B. It shall be an error if the conversion cannot be made.

For example, see converting a *term* to a *clause* and vice versa (7.6), character-conversion (3.29, 7.4.2.5, 8.14.5), and converting a *floating point value* to an *integer value* and vice versa (9.1.6).

3.48 copy, renamed (of a term): See 3.150 — **renamed copy (of a term)**.

3.49 CT**:** The set of *compound terms* (see 6.1.2 e, 7.1.5).

3.50 cut: A *control construct* whose effect is to remove all *choicepoints* back to a deeper execution state defined by its cutparent (see 7.7.2, 7.8.4).

3.51 data, conforming Prolog: See 3.40 — **conforming Prolog data**.

3.52 database: The set of *user-defined procedures* which currently exist during *execution* (see 7.5).

3.53 database, complete: See 3.35 — **complete database**.

3.54 data type: A set of values and a set of operations that manipulate those values.

3.55 data type, arithmetic: See 3.8 — **arithmetic data type**.

3.56 denormalized value: A *floating point value* of type F providing less than the full precision allowed by F (see F_D, 7.1.3).

3.57 directive: A *term* `D` which affects the meaning of *Prolog text* (see 7.4.2), and is denoted in that *Prolog text* by a *directive-term* `:-(D)`.

3.58 directive-term: A *read-term* `T.` in *Prolog text* where `T` has *principal functor* `(:-)/1` (see 6.2.1.1).

3.59 dynamic (of a *procedure*): A dynamic *procedure* is one whose *clauses* can be inspected or altered during *execution*, for example by *asserting* or *retracting* $*$ *clauses* (see 7.5.2).

3.60 effect, side: See 3.157 — **side effect**.

3.61 element (of a *list*): An element of a *non-empty list* is either the *head* of the *list* or an element of the *tail* of the *list*. The *empty list* has no elements.

3.62 empty list: The *atom* `[]` (nil).

3.63 error: A special circumstance which causes the normal process of *execution* to be interrupted (see 7.12).

3.64 evaluable functor: The *principal functor* of an *expression* (see 7.9, 9).

3.65 evaluate: To reduce an *expression* to its value. (see 7.9, 8.6.1, 9).

3.66 exceptional value: A non-numeric value of an *expression*: **float_overflow**, **int_overflow**, **underflow**, **zero_divisor**, or **undefined** (see 7.9).

NOTE — It is an `evaluation_error(E)` when the value of an *expression* is an exceptional value.

3.67 execution (verb: to execute): The process by which a Prolog *processor* tries to *satisfy* a *goal* (see 7.7.1).

3.68 exponent bias: A number added to the exponent of a floating point number, usually to convert the exponent to an unsigned integer.

3.69 expression: An *atomic term* or a *compound term* which may be *evaluated* to produce a value (see 8.6.1, 9).

3.70 extension: A facility provided by the *processor* that is not specified in this part of ISO/IEC 13211 but that would not cause any ambiguity or contradiction if added to this part of ISO/IEC 13211.

3.71 F**:** The set of *floating point values* (see 6.1.2 d, 7.1.3).

3.72 fact: A *clause* whose *body* is the *goal* `true`.

NOTE — A fact can be represented in *Prolog text* by a *term* whose *principal functor* is neither `(:-)/1` nor `(:-)/2`.

3.73 fail, to: *Execution* of a *goal* fails if it is not *satisfied*.

3.74 file name: An *implementation defined* $*$ *ground term* which identifies to the *processor* a file which will be used for input/output during the *execution* of the *Prolog text*.

3.75 flag: An *atom* which is associated with an *implementation defined* or user-defined value (see 7.11).

3.76 floating point value: A member of the set F (see 6.1.2 d, 7.1.3).

3.77 functor: An *identifier* together with an *arity*.

3.78 functor name: The *identifier* of a *functor*.

3.79 function, rounding: See 3.153 — **rounding function**.

3.80 functor, principal: See 3.134 — **principal functor**.

3.81 goal: A *predication* which is to be *executed* (see *body*, *query*, and 7.7.3).

3.82 ground term: An *atomic term* or a *compound term* whose *arguments* are all ground. A *term* is ground with respect to a *substitution* if application of the *substitution* yields a ground term.

3.83 head (of a *list***):** The first *argument* of a *non-empty list*.

3.84 head (of a *rule***):** A *predication*, distinguished by its context.

3.85 Herbrand algorithm: An algorithm which computes the *most general unifier MGU* of a set of equations (see 7.3.2).

3.86 I**:** The set of integers (see 6.1.2 c, 7.1.2).

3.87 identical terms: Two *terms* are identical if they have the same abstract syntax (see 6.1.2).

3.88 identifier: A basic unstructured object used to denote an *atom*, *functor name* or *predicate name*.

3.89 iff: If and only if.

3.90 implementation defined: Defined partly by this part of ISO/IEC 13211, and partly by the documentation accompanying a *processor* (see 5).

3.91 implementation dependent: An implementation dependent feature is dependent on the *processor*.

NOTE — This part of ISO/IEC 13211 does not require an implementation dependent feature to be defined in the accompanying *processor* documentation.

3.92 implementation specific: *Undefined* by this part of ISO/IEC 13211 but supported by a *conforming processor*.

NOTE — This part of ISO/IEC 13211 does not require an implementation specific feature to be supported by a conforming processor, but it preserves the syntax and semantics of a strictly conforming Prolog text which does not use it, for example, defining a term order on variables, or defining unification for terms which are *STO* (3.165).

3.93 indicator, predicate: See 3.131 — **predicate indicator**.

3.94 input/output mode: An *atom* which represents an attribute of a *stream*. A *processor* shall support the *input/output modes*: `read`, `write`, `append` (see 8.11.5, 7.10.1.1).

3.95 instance (of a *term***):** The result of applying a *substitution* to the *term*.

If t is a term and σ a *substitution*, the instance of t by σ is denoted $t\sigma$.