ISO/IEC DTR 13211–3:2006
Definite clause grammar rules

Editor: Klaus.Daessler
klaus.daessler@mathint.com

August 20, 2013

Introduction

This technical report (TR) is an optional part of the International Standard for Prolog, ISO/IEC 13211. Prolog manufacturers wishing to implement Definite Clause Grammar rules in a portable way should do so in compliance with this technical report.

Grammar rules provide convenient functionality for parsing and processing text in a variety of languages. They have been implemented in many Prolog processors. This TR is an extension to the ISO/IEC 13211–1 Prolog standard, adopting a similar structure. In particular, this TR adds new sections and subsections to, or modifies existing sections of ISO/IEC 13211–1.

Previous editors and draft documents

- Paulo Moura: *ISO/IEC DTR 13211– 3:2006 Grammar rules in Prolog*, ISO, 2006-10
- Roger Scowen: *N171 — ISO/IEC DTR 13211–3:2004 Grammar rules in Prolog*, ISO, 2004-05
- Tony Dodd: *DCGs in ISO Prolog — A Proposal*, BSI, 1992

Contributors

This list needs to be completed; so far we have only included people present at the ISO meetings collocated with the ICLP (2005, 2006, and 2007), Richard O'Keefe, and the authors of the two drafts cited.

- Bart Demoen (Belgium)
- David Warren (USA)
- Jan Wielemaker, (Netherlands)
- Joachim Schimpf (UK)
- Jonathan Hodgson (USA)
- Jose Morales (Spain)
- Katsuhiko Nakamura (Japan)
- Klaus Daessler (Germany)
- Manuel Carro (Spain)
- Manuel Hermenegildo (Spain)
- Mats Carlsson (Sweden)
- Mike Covington (USA)
- Paulo Moura (Portugal)
- Per Mildner (Sweden)
- Peter Szabo (Hungary)
- Peter Szeredi (Hungary)
- Pierre Deransart (France)
- Richard O'Keefe (NZ)
- Roger Scowen (UK)
- Tony Dodd (UK)
- Ulrich Neumerkel (Austria)
- Victor Santos Costa (Portugal)

1 Scope

This TR is designed to promote the applicability and portability of Prolog grammar rules in data processing systems that support standard Prolog as defined in ISO/IEC 13211–1:1995 and, if supported by the processor, in ISO/IEC 13211–2:2000, and the two Corrigenda of 13211-1: ISO/IEC 13211-1 Technical Corrigendum 1:2007-11, and ISO/IEC 13211-1 Technical Corrigendum 2:2012-02. This TR specifies:

a) The representation, syntax, and constraints of Prolog grammar rules

b) A logical expansion of grammar rules into Prolog clauses

c) A set of built-in predicates for parsing with and expanding grammar rules

NOTE — The limitations, expressed in section 1, Scope, of ISO/IEC 13211–1:1995 apply to this TR.

2 Normative references

The following TR contains provisions which, through reference in this text, constitute provisions of this TR as Part of ISO/IEC 13211.

- ISO/IEC 13211-1:1995
- ISO/IEC 13211-2:2000
- Corrigendum 1 of 13211-1:2006
- Corrigendum 2 of 13211-1:2012

3 Definitions

For the purposes of this TR, the following definitions are added to the ones specified in ISO/IEC 13211–1:

3.1 body (of a grammar-rule): See `grammar-rule-body`

3.2 clause-term: A read-term `T.` in Prolog text where `T` does not have principal functor `(:-)/1` nor principal functor `(-->)/2.` (This definition replaces section 3.33 of ISO/IEC 13211–1).

3.3 comprehensive terminal-sequence: see terminal-sequence, comprehensive.

3.4 cover, a terminal-sequence by a non-terminal resp. a body: A terminal sequence is covered by a non-terminal resp. a body if the non-terminal resp. the body generates the terminal sequence, respectively parses the terminal sequence.

3.5 definite clause grammar: A sequence of grammar-rules.

3.6 expansion (of a grammar-rule): The preparation for execution (cf. ISO/IEC 13211–1, section 7.5.1) of a grammar rule.

3.7 generating (wrt. a definite clause grammar): Producing terminal-sequences of that definite clause grammar, obeying right-hand-contexts, if any.

3.8 grammar-body-alternative: A compound term with principal functor `(;)/2` or with principal functor `('|')` with each argument being a body (of a grammar-rule).

3.9 grammar-body-element: A grammar-body-cut (the atom `!`), or a grammar-body-goal, or a non-terminal, or a terminal-sequence.

3.10 grammar-body-goal: A compound term with principal functor `{}/1` whose argument is a goal.

3.11 grammar-body-not: A compound term with principal functor `(\+)/1` whose argument is a body (of a grammar rule).

3.12 grammar-body-sequence: A compound term with principal functor `(',')/2` and each argument being a body (of a grammar-rule).

3.13 grammar-body-terminals: A terminal-sequence.

3.14 grammar-rule: A compound term with principal functor `(-->)/2`.

3.15 grammar-rule-body: The second argument of a grammar-rule. A grammar-body-sequence, or a grammar-body-alternative, or a grammar-body-not, or a grammar-body-element.

3.16 grammar-rule-term: A read-term `T.` where `T` is a grammar-rule.

3.17 head (of a grammar-rule): The first argument of a grammar-rule. Either a non-terminal (of a grammar), or a compound term whose principal functor is `(,)/2`, where the first argument is a non-terminal (of a grammar), and the second argument is a right hand context (cf definition 3.23).

3.18 new variable with respect to a term T: A variable that is not a member of the variable set of `T`.

3.19 non-terminal (of a grammar-rule): A callable term (cf. ISO/IEC 13211–1, Definitions 3.25), i.e. an atom or a compound term, that denotes a non terminal symbol of a grammar rule.

3.20 non-terminal indicator: A compound term `A//N` where `A` is an atom and `N` is a non-negative integer, denoting one particular non-terminal (cf 7.14.4).

3.21 parsing (wrt. a definite clause grammar): Successively accepting or consuming terminal-sequences, assigning them to corresponding non-terminals and obeying right-hand-contexts, if any.

3.22 remaining terminal-sequence: See terminal-sequence, remaining.

3.23 right-hand-context: A terminal-sequence occurring as optional second argument of a grammar-rule-head, constraining parsing, respectively generation, by this grammar rule.

3.24 steadfastness of a goal wrt. an argument Goal `G` is steadfast in argument `n` of its argument list, if for any term `T` that is the `n`th argument in the goal, and the goal `Gnw` that results by replacing `T` by a new variable `Vnw` the execution of `G` and `(Gnw, Vnw=T)` is the same.

3.25 terminal (of a grammar): Any Prolog term that denotes a terminal symbol of the grammar.

3.26 terminal-sequence: A list (cf. ISO/IEC 13211–1, sections 3.99, 6.3.5 and 6.3.1.3) whose first argument, if any, is a terminal (of a grammar), and the second argument, if any, is a terminal-sequence.

3.27 terminal-sequence, comprehensive: Terminal sequence containing a (possibly empty) prefix covered (cf. Definition 3.4) by a grammar-rule-body, i.e. accepted resp. generated by `phrase/3` (cf 8.1.1) .

3.28 terminal-sequence, remaining: Rest of comprehensive terminal-sequence without the leading terminal-sequence covered (cf. Definition 3.4) by a grammar-rule-body.

3.29 variable, new with respect to a term T: See *new variable with respect to a term `T`*.

4 Symbols and abbreviations

NOTE — No changes from the ISO/IEC 13211–1 Prolog standard.

5 Compliance

5.1 Prolog processor

A conforming Prolog processor shall:

a) Correctly prepare for execution Prolog text which conforms to:

   1. the requirements of this TR, and
   2. the requirements of ISO/IEC 13211–1, and
   3. the implementation defined and implementation specific features of the Prolog processor,

b) Correctly execute Prolog goals which have been prepared for execution and which conform to:

   1. the requirements of this TR, and
   2. the requirements of ISO/IEC 13211–1, and
   3. the implementation defined and implementation specific features of the Prolog processor,

c) Reject any Prolog text or read-term whose syntax fails to conform to:

   1. the requirements of this TR, and
   2. the requirements of ISO/IEC 13211–1, and
   3. the implementation defined and implementation specific features of the Prolog processor,

d) Specify all permitted variations from this TR in the manner prescribed by this TR and by the ISO/IEC 13211–1, and

e) Offer a strictly conforming mode which shall reject the use of an implementation specific feature in Prolog text or while executing a goal.

NOTE — This extends corresponding section of ISO/IEC 13211–1.

5.2 Prolog text

NOTE — No changes from the ISO/IEC 13211–1 Prolog standard.

5.3 Prolog goal

NOTE — No changes from the ISO/IEC 13211–1 Prolog standard.

5.4 Documentation

The corresponding section in the ISO/IEC 13211–1 Prolog standard is modified as follows:

A conforming Prolog processor shall be accompanied by documentation that completes the definition of every implementation defined and implementation specific feature specified in this TR and in the ISO/IEC 13211–1 Prolog standard.

5.5 Extensions

The corresponding section in the ISO/IEC 13211–1 Prolog standard is modified as follows:

A processor may support, as an implementation specific feature, any construct that is implicitly or explicitly undefined in this TR or in the ISO/IEC 13211–1 Prolog standard.

A Prolog processor may support additional grammar control constructs, beyond the required ones by this standard (cf. 7.15). These additional control constructs must be treated as non-terminals by a Prolog processor working in a strictly conforming mode (see 5.1e).

NOTE — Examples for additional grammar control constructs include *soft-cuts* and control constructs that enable the use of grammar rules stored on encapsulation units other than modules, such as objects.

5.5.2 Predefined operators

Please see subsection 6.3 for the new predefined operators that this TR adds to the ISO/IEC 13211–1 Prolog standard.

6 Syntax

6.1 Notation

6.1.1 Backus Naur Form

No changes from the ISO/IEC 13211–1 Prolog standard.

6.1.2 Abstract term syntax

The text near the end of this section in the ISO/IEC 13211–1 Prolog standard is modified as follows:

Prolog text (6.2) is represented abstractly by an abstract list `x` where `x` is:

a) `d.t` where `d` is the abstract syntax for a directive, and `t` is Prolog text, or

b) `g.t` where `g` is the abstract syntax for a grammar rule, and `t` is Prolog text, or

c) `c.t` where `c` is the abstract syntax for a clause, and `t` is Prolog text, or

d) `nil`, the empty list.

The following subsection extends, with the specified number, the corresponding ISO/IEC 13211–1 section.

6.1.3 Variable names convention for terminal-sequences

This TR uses variables named `S0`, `S1`, ..., `S` to represent the terminal-sequences used as arguments when processing grammar rules or when expanding grammar rules into clauses. In this notation, the variables `S0`, `S1`, ..., `S` can be regarded as a sequence of states, with `S0` representing the initial state and the variable `S` representing the final state. Thus, if the variable `Si` represents the terminal-sequence in a given state, the variable `Si+1` will represent the remaining terminal-sequence after processing `Si` with a grammar rule.

6.2 Prolog text and data

The first paragraph of this section on ISO/IEC 13211–1 is modified as follows:

Prolog text is a sequence of read-terms which denote (1) directives, (2) grammar rules, and (3) clauses of user-defined procedures.

6.2.1 Prolog text

The corresponding section in the ISO/IEC 13211–1 is modified as follows:

Prolog text is a sequence of directive-terms, grammar-rule terms, and clause-terms.

	`prolog text =`	`p text`	
Abstract:	*pt*	*pt*	
	`p text =`	`directive term ,`	`p text`
Abstract:	*d.t*	*d*	*t*
	`p text =`	`grammar rule term ,`	`p text`
Abstract:	*g.t*	*g*	*t*
	`p text =`	`clause term ,`	`p text`
Abstract:	*c.t*	*c*	*t*
	`p text =`	`;`	
Abstract:	*nil*		

6.2.1.1 Directives

Syntactically, there are no changes w.r.t. ISO/IEC 13211–1 Prolog standard, with exception of the operator syntax (cf 6.3); for the semantic changes see 7.4.2 of this TR. Whenever directives are applicable to non-terminals, the non-terminal indicators (cf 7.14.4), as arguments of these directives, shall be used like predicate indicators for the predicates, resulting from expanding these non-terminals.

NOTE — The directives `dynamic/1, multifile/1 and discontiguous/1` are applicable to non-terminal indicators.

6.2.1.2 Clauses

The corresponding section in the ISO/IEC 13211–1 is modified as follows:

	`clause term =`	`term, end`
Abstract:	*c*	*c*
Priority:	1201	
Condition:	The principal functor of `c` is not `(:-)/1`	
Condition:	The principal functor of `c` is not `(-->)/2`	

NOTE — Subsections 7.5 and 7.6 define how clauses become part of the database.

The following subsection modifies, with the specified number, the corresponding ISO/IEC 13211–1 section:

6.2.1.3 Grammar rules

	`grammar rule term =`	`term, end`
Abstract:	*gt*	*gt*
Priority:	1201	
Condition:	The principal functor of `gt` is `(-->)/2`	

	`grammar rule =`	`grammar rule term`
Abstract:	*g*	*g*

NOTE — Section 10 of this TR defines how a grammar rule in Prolog text is expanded into an equivalent clause when Prolog text is prepared for execution.

6.2.1.4 Right-hand contexts

	`right hand context term =`	`term`
Abstract:	*rc*	*rc*
Priority:	1201	
Condition:	Rright hand context term is a list	

	`right hand context =`	`right hand context term`
Abstract:	*r*	*r*

NOTE — Section 10 of this TR, rule %02% and rule %04%, defines how a right hand context in a grammar rule is expanded when Prolog text is prepared for execution.

6.3 Terms

Extend the operator table of section 6.3.4.4 of ISO/IEC 13211–1 as follows:

`Priority`	`Specifier`	`Operator(s)`
1105	`xfy`	`'|'`

NOTE — The operator `(-->)/2`, specified in section 6.3.4.4 of the ISO/IEC 13211–1 Prolog standard, is used as the principal functor of grammar rules.

7 Language concepts and semantics

The following subsection extends, with the specified number, the corresponding ISO/IEC 13211–1 section:

7.4 Prolog text

7.4.2 Directives

A non-terminal indicator may appear anywhere that a predicate indicator can appear in the following directives: `dynamic/1`, `multifile/1`, and `discontiguous/1` as specified in section 7.4.2 of the ISO/IEC 13211–1 Prolog standard.

If the Prolog processor supports Prolog Modules as defined in ISO/IEC 13211-2:2000, a non-terminal indicator may appear anywhere that a predicate indicator can in the Module interface directives `export/1, reexport/1, reexport/2` and the Module directives `import/1, import/2` of section 6.24 of ISO/IEC 13211-2:2000.

7.13 Predicate properties

The following subsection extends sections 6.8 and 7.2.2 of ISO/IEC 13211–2 Prolog Modules:
The following property is added to the list of predicate properties of section 6.8 of ISO/IEC 13211–2:

- `expanded_from(A//N)` — The predicate resp. callable with predicate indicator `A/(N+2)` results from the expansion of a grammar rule with non-terminal indicator `A//N`

NOTE — a predicate property is the second argument of the built-in predicate `predicate_property(Callable, Property)`, cf. section 6.8 of ISO/IEC 13211–2.

7.14 Grammar rules

7.14.1 terminals and non-terminals

One or more terminals are represented by terms directly contained in lists in order to distinguish them from non-terminals. The empty terminal sequence (empty list) is possible. Non-terminals are represented by callable terms.

NOTE — In the context of a grammar rule, *terminals* represent tokens of some language, and *non-terminals* represent sequences of tokens (see, respectively, definitions 3.19 and 3.25).

7.14.1.1 Example

A simple grammar consisting of 11 grammar rules, parsing or generating terminal sequences of the form

```
[the, dog, runs]
[the, dog, barks]
[the, dog, bites]
[the, nice, cat, barks]
```

is given:

```
sentence --> noun_phrase, verb_phrase.
verb_phrase --> verb.
noun_phrase --> article, noun.
noun_phrase --> article, adjective, noun.
article --> [the].
adjective --> [nice].
noun --> [dog].
noun --> [cat].
verb --> [runs].
verb --> [barks].
verb --> [bites].
```

Here the symbols `sentence, verb_phrase, verb` etc. denote non-terminals, whereas `runs, nice, cat` etc. denote terminals.

7.14.2 Format of grammar rules

A grammar rule has the format:

```
GRHead --> GRBody.
```

where `GRHead`, the head(of a grammar rule) (cf. Definition 3.16), can be rewritten by `GRBody`, its body(of a grammar rule) (cf. Definition 3.1). The head and the body of grammar rules are constructed from *non-terminals*, *terminals* and *control constructs*. The head(of a grammar rule) is a non-terminal or the conjunction of a non-terminal and, following, a terminal-sequence (a *right-hand-context*, see 7.14.3):

```
NonTerminal --> GRBody.

NonTerminal, RightHandContext --> GRBody.
```

The control constructs that may be used in a body are described in subsection 7.15. An empty body is represented by an empty terminal sequence:

```
GRHead --> [].
```

This empty terminal sequence cannot be omitted, i.e. there is no `(-->)/1` form for grammar rules.

7.14.3 Right-hand-contexts

7.14.3.1 Description

A *right-hand-context* is a terminal-sequence (see 3.26), as an optional second argument of the head of a grammar rule (see 3.17). A right-hand-context contains terminals that are prefixed to the remaining terminal-sequence after successful application of the grammar rule.

7.14.3.2 Examples

Assume we need rules to *look-ahead* one or two tokens that would be consumed next. This could be accomplished by the following grammar rules:

```
look_ahead(X), [X]      --> [X].

look_ahead(X, Y), [X,Y] --> [X,Y].
```

When used for parsing, procedurally, these grammar rules can be interpreted as, respectively, consuming, and then restoring, one or two terminals.

Another example may be a small grammar rule with right-hand-context:

```
phrase1, [word] --> phrase2, phrase3.
```

After preparation for execution this may occur in the database as follows.

```
phrase1(S0, S):-
    phrase2(S0, S1),
    phrase3(S1, S2),
    S = [word | S2].
```

NOTES

1 In case of parsing, as soon as `phrase2` and `phrase3` have successfully parsed the comprehensive terminal-sequence (input list), the terminal `word` is prefixed to the remaining terminal-sequence. `word` is then the first terminal to be consumed in further parsing after `phrase1`. Thus the path of further parsing is constrained by the right-hand-context.

2 The concepts *comprehensive terminal-sequence* resp. *remaining terminal-sequence* are often named *input list* resp. *output list*. This is misleading, because it only considers the case of parsing using a grammar. There a terminal list shall be parsed wrt. non-terminals, and there will be a remainder after the parsing step. The inverse case, generating sentences by expanding grammars, where the comprehensive terminal-sequence is the real output list, is ignored by such wording.

3 There are cases, where the remaining terminal-sequence is the comprehensive terminal-sequence, e.g. with the following grammar rule, but there is a trailing sequence as the following example shows.

```
nt,[word] --> [].
```

which is expanded by preparation for execution to:

```
nt(S0, [word|S]) :-
    S0 = S.
```

This non-terminal `nt` represents an empty terminal sequence (cf. 7.15.1), but constrains further parsing to take place with `word` as next token.

4 It should be noted that `phrase/2` (cf 8.1.1.3) cannot succeed when applied to a grammar rule, whose head contains a non empty right-hand context, as in

the case above.

5 Some processors allow a cut in the right-hand context; e.g.

```
a, !, [word] --> b.
```

Moving this cut to the end of the grammar body, c.f. `a, [word] --> b, !.` has the identical effect on execution. Thus this TR does not permit a cut in the right hand context.

7.14.4 Non-terminal indicator

A *non-terminal indicator* is a compound term `//(A, N)` where `A` is an atom and `N` is a non-negative integer.

The non-terminal indicator `//(A, N)` indicates the non-terminal of the head of a grammar rule whose functor is `A` and whose arity is `N`.

NOTES

1 In Prolog text, including ISO/IEC 13211–1 and this TR, a non-terminal indicator `//(A, N)` is normally written as `A//N` or as `(A)//N` depending on whether or not A is an operator (cf. 7.1.6.6 of 13211–1).

2 The concept of non-terminal indicator is similar to the concept of *predicate indicator* defined in sections 3.131 and 7.1.6.6 of the ISO/IEC 13211–1 Prolog. Non-terminal indicators may be used in exception terms thrown when processing or using grammar rules. In addition, non-terminal indicators may appear at some places, where a predicate indicator as defined in ISO/IEC 13211–1 can appear. See 7.4.2. Furthermore non-terminal indicators may be used in a predicate property (cf. subsection 7.13). In particular, using non-terminal indicators in predicate directives allows the details of the expansion of grammar rules into Prolog clauses to be abstracted.

7.14.4.1 Examples

For example, given the following grammar rule:

```
sentence --> noun_phrase, verb_phrase.
```

The corresponding non-terminal indicator for the grammar rule left-hand side non-terminal is `sentence//0`.

Example:

```
:- export(sentence//0).
```

So the grammar rules of M for the non-terminal `sentence//0` can be used outside module M.

7.14.5 Prolog goals in grammar rules

7.14.5.1 Description

In the body of grammar rules, curly brackets enclose a Prolog goal that is executed when the grammar rule, prepared for execution, is executed.

`{}/1` is a functor of a `grammar-body-goal` {prolog_goal} (cf. Definition 3.10). After expansion `prolog_goal` is unchanged and is handled as a ordinary prolog goal according to section 7.7 of ISO/IEC 13211-1 Prolog.

NOTE — The ISO/IEC 13211–1 Prolog standard defines, in section 6.3.6, a *curly bracketed term* as a compound term with principal functor `'{}'/1`, whose argument may also be expressed by enclosing its argument in curly brackets.

7.14.5.2 Examples

Consider, for example, the following grammar rule:

```
move(Dist) --> step(Dist).
move(Dist) --> step(Dist1), move(Dist2), {Dist is Dist1 + Dist2}.
step(1) --> up.
step(-1)--> down.
```

This grammar generates steps of a robot. The second rule computes a constraining distance for several move patterns.

7.15 Grammar control constructs

This subsection describes the meaning of special non-terminals which are part of grammar rules:

`[]//0`
`('|')//2`
`(',')//2`
`(;)//2 ('|')//2 (alternative)`
`call//1`
`{}//1`
`(\+)//1`
`!//0`
`(->)//2` in an `if-then-else`, and
`phrase//1`.

After preparation for execution of Grammar Rules, named "Grammar Rule expansion" or "expansion" for short, these non-terminals, with exception of phrase//1, result in control constructs, respectively built-in predicates of ISO/IEC 13211-1 Prolog.
Expansion of grammar control constructs is not simply a replacement by Prolog control constructs.

Logical expansion is defined by describing it wrt. the built-in predicate `phrase/3` (see 10 and 8.1.1)

Execution of this predicate serves two goals: Firstly the final expansion(of a grammar rule) (cf. definition 3.6), when this has not taken place earlier, i.e. preparation for execution of its body and arguments; thereafter, secondly, the execution of the resulting Prolog goals.

From the parsing viewpoint `phrase(NT, S)` is true when the terminal sequence `S` is covered (cf. Definition 3.4) by the non-terminal `NT`. Or, from the generation viewpoint it is true when `NT` generates `S`. The Grammar control constructs are described here informally. Section 10, "Prolog Definition of Phrase/3", provides a reference implementation that further defines the semantics of expansion.
The numbering of the Prolog clauses of the procedure `dcg_cbody/4` for the grammar control constructs of this section 10, starting with %071%, is partially synchronized with the following subsection numbers.
In particular this subsection explicates the linkages between the terminal sequences upon expansion of the control constructs.

7.15.1 `[]//0` – empty terminal-sequence

The expansion result of the grammar control construct `empty terminal sequence` (a terminal sequence without contents) unifies the remaining terminal sequence with the comprehensive terminal sequence, i.e. has no effect on parsing resp. generating during execution.
For a formal description of expansion of this form see section 10, clause %071%.

7.15.2 `('|')//2` – list separator

`('|')` used as a non-terminal `('|')//2` separates its first argument, the terminal on its left hand side from the second argument, the terminal sequence on its right hand side.
For a formal description of expansion of this form see section 10, clause %072%.

7.15.3 `(',')//2` – concatenation

In the body of a grammar-rule the non-terminal `(',')//2` acts as principal functor of a `grammar-body-sequence` (cf. Definition 3.12) with a first grammar body GBFirst and a second grammar body GBSecond. Each of them is then subject to subsequent separate expansion - GBFirst first, and then GBSecond.

After being completely expanded, the expansions of GBFirst and GBSecond shall be arguments of a conjunction wrt. section 7.8.5 of ISO/IEC 13211-1 Prolog, which results from expansion of `(',')//2`.
For a formal description of expansion of this form see section 10, clause %073%.

If contained directly in the head(of a grammar rule), `(',')//2` acts simply as main functor `(',')/2` of a term consisting of an expanded grammar-rule-head and a right-hand-context; cf 7.14.3.

7.15.4 `(;)//2` – alternative

In the body of a grammar rule `(;)//2` acts as principal functor of a `grammar-body-alternative`. (cf. definition 3.8) with a first grammar body GBFirst and a second grammar body GBSecond. Each of them is subject to subsequent separate expansion - GBFirst first, and then GBSecond. The grammar-body-alternative is expanded to the disjunction `(;)/2` wrt. section 7.8.6 of ISO/IEC 13211-1 Prolog. After being expanded, the expansions of GBFirst and GBSecond shall be arguments of that disjunction, which results from expansion of `(;)//2`.
For a formal description of expansion of this form see section 10, clause %074%.

NOTE — The effect of comma and semicolon, `(',')//2`, `(;)//2`, may be understood best by application of `write_canonical/1` (see section 8.14.2.5 of ISO/IEC 13211–1) on a grammar rule, containing them:

```
?-write_canonical((sentence --> subject, verb, object;
                                object, verb, subject)).

-->(sentence, ;(','(subject, ','(verb, object)),
               (',','(object, ','(verb, subject))))
yes
```

This may lead to the following Prolog clause after preparation for execution:

```
sentence(S0, S) :-
   ( subject(S0, S1),
     verb(S1, S2),
     object(S2, S)
   ; object(S0, S3),
     verb(S3, S4),
     subject(S4, S)
   ).
```

7.15.5 '|'//2 – second form of alternative

'|' used as a non-terminal '|'//2 has the same behaviour as (;)//2, when used for a grammar-body-alternative. See subsection 7.15.4.

NOTE — ('|')//2 is not equivalent to (;)//2, when (;)//2 is used for "if-then-else", as ('|')//2 shall not be used for "if-then-else". See subsection 7.15.10.

For a formal description of expansion of this form see section 10, clause %075%.

7.15.6 call//1

Expanding, i.e. preparing for execution of the non-terminal

```
call//1
```

shall result after the expansion in the goal for the predicate

```
call/3
```

which is required by this TR and defined in 8.15.4 of ISO/IEC 13211-1:1995/Cor.2:2012(E).

For a formal description of expansion of call//1 see section 10, clause %076%.

NOTE — Consider the following example for the correspondence for grammar rules between call//1 and call/3:

```
atom_charsdiff(Atom, Xs0, Xs):-
    atom_chars(Atom, Chars),
    append(Chars, Xs, Xs0).

atomchars(Atom) --> call(atom_charsdiff(Atom)).

at_eos_pred([ ], [ ]).

at_eos --> call(at_eos_pred).
```

7.15.7 {}//1 – grammar-body-goal

The non-terminal {G}, with G a Prolog goal, according to ISO/IEC 13211-1:1995, can stand at any place of a non-terminal inside a grammar-rule-body. After expansion the braces are omitted, the goal G is unchanged. On execution G is executed like any Prolog goal.
If G immediately contains a cut ('!'), this is handled like a grammar-body-cut (cf. 7.15.9).
For a formal description of expansion of ({G}//1 see section 10, clause %077%.

7.15.8 `(\+)//1` – grammar-body-not

In the body of a grammar rule `(\+)//1` is expanded to the principal functor `(\+)/1` of a `grammar-body-not` (cf. Definition 3.11) This functor `(\+)/1` is applied to the expanded argument of the `grammar-body-not`. If the resulting goal succeeds the expanded rule does not change the comprehensive terminal sequence.

For a formal description of expansion of `(\+)//1` see section 10, clause %078%.

Implementations conforming to this TR shall not define or use a predicate `(\+)/3`.

NOTE — The effect of `(\+)//1` can be seen in the following example.

The grammar rule

```
a --> \+ b.
```

may be expanded to:

```
a(S0, S) :- \+ b(S0, _), S0 = S.
```

7.15.9 `!//0` – grammar-body-cut

In the body of a grammar rule `!//0` is a `grammar-body-cut`. After expansion the grammar-body-cut becomes the control construct cut, `!/0`, as in section 7.8.4 of ISO/IEC 13211-1 Prolog. For a formal description of expansion of `!//0` see section 10, clause %079%.

Implementations conforming to this TR shall not define or use a predicate `!/2`.

7.15.10 `(;)//2` and `->//2` – if-then-else

`(;)//2` serves two different functions depending on whether or not its first argument is a compound term with functor `(->)/2`. See 7.15.4 for the use of `(;)//2` for `alternative`, when the first argument of `(;)//2` is not a `(->)/2`.

The grammar control construct `(;)//2` shall have as arguments a grammar body GBFirst and a second grammar body GBSecond. The principal functor of GBFirst is `(->)/2` with arguments GBIf and GBThen. In this if-then-else form the grammar rule bodies GBIf, GBThen, and GBSecond are subject to separate expansion to GBIfExpanded, GBThenExpanded and GBSecondExpanded, respectively.

The result of expansion is an if-then-else `(;)/2` (cf section 7.8.8 of ISO/IEC 13211-1), resulting from expansion of `(;)//2`, with the if-then construct `->(GBIfExpanded, GBThenExpanded)` as first, and `GBSecondExpanded` as second argument.

For a formal description of expansion of if-then-else see section 10, clause %074% and %0710%.

7.15.11 `phrase//1`

`phrase//1` is not included in the allowed constructs of the body (of a grammar-rule). It is designed as grammar-built-in predicate for using the built-in predicate `phrase/3`.

Expanding, i.e. preparing for execution of the non-terminal

```
phrase//1
```

with argument

```
G
```

shall result in a goal for the built-in predicate

```
phrase/3
```

with first argument `G`.

For a definition of the built-in predicate `phrase/3` see section 8.1.1.

7.16 Executing procedures expanded from grammar rules

If a grammar rule to be prepared for execution has a non-terminal indicator N//A, and N is the name of the predicate indicator N/A', with A' = A + 2, of a built-in predicate, existing in the complete database, the result of expansion and the behaviour of the prepared grammar rule on execution is implementation dependent. This does not hold for the required non-terminals expanding to built-in predicates defined in 7.15.

When the database does not contain a grammar rule, prepared for execution, with non-terminal indicator N//A during execution of a non-terminal with non-terminal indicator N//A , the behaviour of the processor shall be as follows.

If the error handling of the processor is standard conforming as specified in section 7.7.7 of ISO/IEC 13211–1, then the error term as specified in section 7.7.7b of ISO/IEC 13211–1 when the flag `unknown` is set to `error` shall be:

```
existence_error(procedure, N//A)
```

If the error handling of the processor supports definite clause grammar errors, then the error term shall be:

```
existence_error(grammar_rule, N//A)
```

In other cases the behaviour shall be implementation specific.

NOTES

1 Prolog Processors shall report errors resulting from execution of grammar rules at the same abstraction level as grammar rules whenever possible.

2 Parsing resp. generating of terminal sequences using grammar rules is defined in subsection 8.1.1. Grammar rules are expanded there into Prolog clauses during preparation for execution, which maps the parsing or generating with a grammar-rule-body into executing a goal given a sequence of predicate clauses. See section 7.7 of ISO/IEC 13211–1 for details.

8 Built-in predicates

8.1 Grammar rule built-in predicates

8.1.1 phrase/3, phrase/2

8.1.1.1 Description

`phrase(GRBody, S0, S)` is true iff `S0` either unifies with the concatenation of the grammar-body-sequence of `GRBody`, if any, (cf. Definition 3.12) with the remaining terminal-sequence `S`, or with the concatenation of a terminal-sequence resulting from generation by the non-terminal, if any, of `GRBody` w.r.t. the current Grammar rules with the remaining terminal-sequence `S`.

NOTE 1 — An `A` of a `B` means, construct `A` is directly contained in construct `B`. This is general standard wording for programming languages.

NOTE 2 — The simple grammar of example 7.14.1.1 may be prepared for execution.

Then with

```
GRBody: non-terminal:  noun_phrase
S0: comprehensive terminal-sequence: [the, dog, barks]
S:  remaining terminal-sequence: [barks]
```

`phrase(noun_phrase, [the, dog, barks], [barks])` is true.

If the non-terminal of `GRBody`, if any, is followed by a right-hand-context (cf. Definition 3.23), then the right-hand-context shall be prefixed to the remaining terminal sequence after having been parsed resp. generated wrt. the non-terminal of `GRBody`.
Procedurally, `phrase(GRBody, S0, S)` is executed by calling the Prolog goal

corresponding to the expansion of the grammar-rule-body `GRBody`, given the terminal-sequences `S0` and `S`, according to the logical expansion of grammar rules described in section 10. See in particular the clauses for `dcg_rule/4`.

`phrase(GRBody, S0, S)` shall be steadfast in its third argument S (cf. Definition 3.24).

8.1.1.2 Template and modes

```
phrase(+grammar-rule-body, ?comprehensive-terminal-sequence, ?remaining-terminal-sequence)
```

8.1.1.3 Bootstrapped built-in predicates

The built-in predicate `phrase/2` provides similar functionality to `phrase/3`. The goal `phrase(GRBody, S0)` is true when all terminals in the terminal-sequence S0 are consumed and accepted respectively generated.

```
phrase(GRBody, S0) :-
    phrase(GRBody, S0, []).
```

8.1.1.4 Errors

a) `GRBody` is a variable
— `instantiation_error`

b) `GRBody` is neither a variable nor a callable term
— `type_error(callable, GRBody)`

The following two errors are implementation defined if applied to `phrase/3`, i.e. no error checking is required on S0 and S by this TR for `phrase/3`. If, however, a Prolog processor offers them, their form and consequence must be the following:

c) `S0` is not a terminal-sequence
— `type_error(terminal_sequence, S0)`

For `phrase/2` error clause `c` is required.

d) `S` is not a terminal-sequence
— `type_error(terminal_sequence, S)`

NOTE — This relaxation is allowed because handling these errors could overburden a Prolog processor.

8.1.1.5 Examples

These examples assume that the following grammar rules has been correctly prepared for execution and are part of the complete database:

```
determiner --> [the].
determiner --> [a].

noun --> [boy].
noun --> [girl].

verb --> [likes].
verb --> [scares].

noun_phrase --> determiner, noun.
noun_phrase --> noun.

verb_phrase --> verb.
verb_phrase --> verb, noun_phrase.

sentence --> noun_phrase, verb_phrase.
```

Some example calls of `phrase/2` and `phrase/3`:

```
| ?- phrase([the], [the]).

yes

| ?- phrase(sentence, [the, girl, likes, the, boy]).

yes
 | ?- phrase(sentence, [the, girl, likes, the, boy, today]).

no

| ?- phrase(sentence, [the, girl, likes]).

yes

| ?- phrase(sentence, Sentence).

Sentence = [the, boy, likes]
yes

| ?- phrase(noun_phrase, [the, girl, scares, the, boy], Rest).

Rest = [scares, the, boy]
yes
```

9 Evaluable functors

NOTE — No changes from the ISO/IEC 13211–1 Prolog standard.

10 Logical Expansion: Prolog Definition of phrase/3

```
% To avoid name clashes: if phrase/3 exists already on the processor,
% the word "phrase" may be replaced by "iso_phrase" or else at two places.
% Missing prerequisite definitions as append/3 shall be defined by
% the "Prolog Prologue" (ISO/IEC JTC1 SC22 WG17 N235)

:- op(1105,xfy,'|').


%01%
phrase(GRBody, S0, S) :-
   dcg_body(GRBody, S0, S, Goal),
   call(Goal).

%02%
% dcg_rule(DCGrule, S0, S, Expansion).
% Translates a DCG rule into a Prolog rule, when no error condition applies.

dcg_rule((NonTerminal, Terminals --> GRBody), S0, S, (Head :- Body)) :-
   dcg_non_terminal(NonTerminal, S0, S, Head),
   dcg_body(GRBody, S0, S1, Goal1),
   dcg_terminals(Terminals, S, S1, Goal2),
   Body = (Goal1, Goal2).

dcg_rule((NonTerminal --> GRBody), S0, S, (Head :- Body)) :-
   NonTerminal \= (_,_),
   dcg_non_terminal(NonTerminal, S0, S, Head),
   dcg_body(GRBody, S0, S, Body).

%03%
% translates a grammar goal non-terminal:

dcg_non_terminal(NonTerminal, S0, S, Goal) :-
   NonTerminal =.. NonTerminalUniv,
   append(NonTerminalUniv, [S0, S], GoalUniv),
   Goal =.. GoalUniv.

%04%
% translates a terminal-sequence:
```

```
dcg_terminals(Terminals, S0, S, S0 = List) :-
   append(Terminals, S, List).

%05%
% translates a grammar-rule-body:

dcg_body(Var, S0, S, Body) :-
   var(Var),
   Body = phrase(Var, S0, S).

dcg_body(GRBody, S0, S, Body) :-
   nonvar(GRBody),
   dcg_constr(GRBody),
   dcg_cbody(GRBody, S0, S, Body).

dcg_body(NonTerminal, S0, S, Goal) :-
   nonvar(NonTerminal),
   \+ dcg_constr(NonTerminal),
   NonTerminal \= (_->_),
   dcg_non_terminal(NonTerminal, S0, S, Goal).

%06%
dcg_constr([]).
dcg_constr([_|_]).
dcg_constr((_,_)).
dcg_constr((_;_)).
dcg_constr((_'|'_)).
dcg_constr(call(_)).
dcg_constr({_}).
dcg_constr(\+_).
dcg_constr(!).

%071%
dcg_cbody([], S0, S, (S0=S)).

%072%
dcg_cbody([T|Ts], S0, S, Goal) :-
   dcg_terminals([T|Ts], S0, S, Goal).

%073%
dcg_cbody(( GRFirst , GRSecond ), S0, S, ( First, Second )) :-
   dcg_body(GRFirst, S0, S1, First),
   dcg_body(GRSecond, S1, S, Second).

%074%
dcg_cbody(( GREither ; GROr ), S0, S, ( Either ; Or )) :-
```

```
   subsumes_term((_->_),GREither),
   dcg_cbody(GREither, S0, S, Either),
   dcg_body(GROr, S0, S, Or).


dcg_cbody(( GREither ; GROr ), S0, S, ( Either ; Or )) :-
   \+ subsumes_term((_->_),GREither),
   dcg_body(GREither, S0, S, Either),
   dcg_body(GROr, S0, S, Or).

%075%
dcg_cbody(( GREither '|' GROr ), S0, S, ( Either ; Or )) :-
   dcg_body(GREither, S0, S, Either),
   dcg_body(GROr, S0, S, Or).

%076%
dcg_cbody(call(Cont), S0, S, call(Cont, S0, S)).

%077%
dcg_cbody({Goal}, S0, S, (Goal, S0 = S)).

%078%
dcg_cbody(\+ GRBody, S0, S, (\+ Goal, S0 = S)) :-
   dcg_body(GRBody, S0, _, Goal).

%079%
dcg_cbody(!, S0, S, (!, S0 = S)).

%0710%
dcg_cbody(( GRIf -> GRThen ), S0, S, ( If -> Then )) :-
   dcg_body(GRIf, S0, S1, If),
   dcg_body(GRThen, S1, S, Then).
```