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

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

July 9, 2011

Introduction

This technical recommendation (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 shall do so in compliance with this technical recommendation.

Grammar rules provide convenient and simple functionality for parsing and processing text in a variety of languages. They have been implemented in many Prolog systems. As such, they are deemed an worthy extension to the ISO/IEC 13211 Prolog standard.

This TR is an extension to the ISO/IEC 13211–1 Prolog standard, adopting a similar structure. Specifically, this TR either adds new sections and clauses to, or modifies the reading of existing clauses on ISO/IEC 13211–1.

This draft may contain in several places informative text, type-set in *italics*. Such informative text is used for editorial comments deemed useful during the development of this draft and may not be included in the final version.

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) and the authors of the two drafts cited above, and Richard as I have included here some contributions from him that I found on the net. PM.

- Bart Demoen (Belgium)
- Jan Wielemaker, (Netherlands)
- Joachim Schimpf (UK)
- Jonathan Hodgson (USA)
- Jose Morales (Spain)
- Katsuhiko Nakamura (Japan)
- Klaus Daessler (Germany)
- Manuel Carro (Spain)
- Mats Carlsson (Sweden)
- Paulo Moura (Portugal)
- Pierre Deransart (France)
- Péter Szabó (Hungary)
- Péter Szeredi (Hungary)
- Richard O'Keefe (NZ)
- Roger Scowen (UK)
- Tony Dodd (UK)
- Ulrich Neumerkel (Austria)
- Vítor 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. As such, 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

d) Reference implementations and tests for the specified built-in predicates and for a grammar rule translator

NOTE — This part of ISO/IEC 13211 will supplement ISO/IEC 13211–1:1995.

2 Normative references

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

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): The second argument of a grammar-rule. A grammar-body-sequence, or a grammar-body-alternative, or a grammar-body-choice, or a grammar-body-element.

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 clause 3.33 of ISO/IEC 13211–1).

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

3.4 comprehensive terminal-sequence, CTS: see terminal-sequence, comprehensive.

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

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

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

3.8 grammar-body-choice: A compound term with principal functor `(->)/2`. The first argument is a body (of a grammar-rule), and the second argument is a body.

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

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

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

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

3.15 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`, the first argument is a non-terminal (of a grammar), and the second argument is a right-hand-context.

3.16 new variable with respect to a term T: A variable that is not an element of the variable set of `T`.

3.17 non-terminal (of a grammar): An atom or compound term that denotes a non-terminal symbol of the grammar.

3.18 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.

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

3.20 remaining terminal-sequence (RTS): See terminal-sequence, remaining.

3.21 right-hand-context: A terminal-sequence occuring as second argument of a grammar-rule-head, restricting parsing resp. generating after completing this grammar rule.

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

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

3.24 terminal-sequence, comprehensive: Terminal sequence containing as (possibly empty) prefix a terminal-sequence described by a grammar rule body in combination with other grammar rules of a grammar.

3.25 terminal-sequence, remaining: Rest of comprehensive terminal-sequence without the leading terminal-sequence corresponding to a non-terminal.

3.26 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 on 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 on the ISO/IEC 13211–1 Prolog standard.

5.5 Extensions

The corresponding section on 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 on the ISO/IEC 13211–1 Prolog standard.

5.5.2 Predefined operators

Please see section 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 on 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 section 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 , `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 S_i represents the initial terminal-sequence, the variable S_{i+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 on 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

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

6.2.1.2 Clauses

The corresponding section on 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 — Subclauses 7.5 and 7.6 defines how each clause becomes part of the database.

The following section extends, 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 **??** of this TR defines how a grammar rule in Prolog text is expanded into an equivalent clause when Prolog text is prepared for execution.

6.3 Terms

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

7.13 Predicate properties

The following optional property is added to the list of predicate properties:

- `expanded_from(non_terminal, A//N)` — The predicate results from the expansion of a grammar rule for the non-terminal `A//N`

NOTE — the `expanded_from/2` property name was chosen in order to account for other possible, implementation-specific expansions.

7.14 Grammar rules

7.14.1 Terminals and non-terminals

Zero or more terminals are represented by terms contained in lists in order to distinguish them from non-terminals (string notation may be used as an alternative to lists when terminals are characters and the flag "double_quotes" has value "chars"; see sections 6.3.7 and 6.4.6 of ISO/IEC 13211–1). 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 3.18 and 3.22).

7.14.2 Format of grammar rules

A grammar rule has the format:

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

A grammar rule is interpreted as stating that its head, `GRHead`, can be rewritten by its body, `GRBody`. The head and the body of grammar rules are constructed from *non-terminals* and *terminals*. 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 on a grammar rule body are described in section 7.14.6. An empty grammar rule body is represented by an empty list:

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

The empty list 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, as an optional second argument of the head of a grammar rule (see 3.15). 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 Syntax

```
GrammarRuleHead    = NonTerminal
GrammarRuleHead    = NonTerminal, RightHandContext
RightHandContext   = TerminalSequence
```

7.14.3.3 Examples

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

```
phrase1(CTS, RTSfinal):-
    phrase2(CTS, RTS2),
    phrase3(RTS2, RTS3),
    RTSfinal = [word | RTS3].

or, respectively

phrase1(CTS, [word |RTS3]) :-
    phrase2(CTS, RTS2),
    phrase3(RTS2, RTS3).
```

Here `CTS` shall denote the comprehensive terminal-sequence for parsing/generating wrt. `phrase1`. `RTS2` and `RTS3` shall denote, respectively, the remaining terminal sequences after application of the nonterminals `phrase2` and `phrase3`.

NOTES

1 In case of parsing, as soon as `phrase2` and `phrase3` are recognized in the comprehensive terminal-sequence (input list) `CTS`, the terminal `word` is prefixed to the remaining terminal-sequence `RTS3` of `phrase1`. `word` is then the first terminal to be taken in respect for further parsing after `phrase1`. Thus the path of further parsing is constrained by the right-hand-context.

2 Sometimes the concept of *comprehensive terminal-sequence* resp. *remaining terminal-sequence* are named *input list* resp. *output list*. This is misleading, because it only takes in respect the case of parsing by a grammar. There a terminal list shall be parsed wrt. nonterminals, and a rest will remain after a parsing step. The inverse case of generating sentences by grammars, where the comprehensive terminal-sequence is the real output list, is ignored by such wording.

3 There are exotic cases, where the remaining terminal-sequence seems not to be trailing part of the comprehensive terminal-sequence, e.g. with the following grammar rule...but it is:

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

which is expanded by preparation for execution to:

```
nt(CTS,[word|RTS]) :-
    CTS = RTS.
```

This nonterminal `nt` represents an empty terminal sequence, but constrains further parsing to take in respect `word` as next token. The comprehensive terminal-sequence is identical with the remaining terminal-sequence for that nonterminal.

7.14.4 Non-terminal indicator

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

The non-terminal indicator `//(A, N)` indicates the grammar rule non-terminal 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`.

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 wherever a predicate indicator as defined in ISO/IEC 13211–1 can appear. Furthermore non-terminal indicators may be used as predicate property (cf. section 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`. Assuming a `public/1` directive for declaring predicate scope, we could write:

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

in order to be possible to use grammar rules for the non-terminal `sentence//0` outside its encapsulation unit.

7.14.5 Prolog goals in grammar rules

7.14.5.1 Description In the body of grammar rules, curly brackets enclose a sequence of Prolog goals that are executed when the grammar rule, prepared for execution, is processed.

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:

```
digit(D) --> [C], {0'0 =< C, C =< 0'9, D is C - 0'0}.
```

This rule recognizes a single terminal as the code of a character representing a digit when the corresponding numeric value can be unified with the non-terminal argument.

7.14.6 Control constructs and built-in predicates supported by grammar rules

The following nonterminals denote control constructs specified in the ISO/IEC 13211–1 Prolog standard, after a special preparation for execution, and may be used in the body of grammar rules:

`(',')//2`, `(';')//2`, `(->)//2`, and `!//0`.

The following nonterminal denotes, after its preparation for execution, the built-in predicate `\+/1` as specified in the ISO/IEC 13211–1 Prolog standard, and may be used in the body of grammar rules:

`\+/1`.

The nonterminal `(:)//2` denotes, after its preparation for execution, the control construct `(:)/2` as specified in the ISO/IEC 13211–2 Prolog Modules standard and may be used as control construct in the body of grammar rules.

For a more precise description of these control constructs wrt. grammar rules see section 7.14.6.1 ff.

The following nonterminals, occuring in grammar rules, shall NOT, with exception of `call//1`, denote corresponding Prolog control constructs resp. corresponding built-in predicates after their preparation for execution:

`true//0`, `fail//0`, `call//1`, `catch//3`, and `throw//1`.

7.14.6.1 The nonterminal (',')//2

In the body of a grammar rule `(',')/2` is principal functor of a `grammar-body-sequence` (cf. Definition 3.11). If occuring in the head of a grammar rule, `(',')//2` is main functor of a term consisting of a nonterminal and a right-hand-context.

7.14.6.2 The nonterminal (';')//2

In the body of a grammar rule `(';')/2` is principal functor of a `grammar-body-alternative`.

NOTE — The effect of comma and semicolon, `(',')//2`, `(';')//2`, maybe 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 leads to the following Prolog clause during preparation for execution :

```
sentence(CTS, RTS) :-
            subject(CTS, ITS1),
            verb(ITS1, ITS2),
            object(ITS2, RTS)
      ;
            object(CTS, ITS3),
            verb(ITS3, ITS4),
            subject(ITS4, RTS).
```

`ITS1` .. `ITS4` are the respective intermediate terminal sequences, arising during continued parsing resp. generating the comprehensive terminal sequence `CTS` with two alternative forms.

7.14.6.3 The nonterminal `('->')//2` together with `(';')//2` (choice)

In the body of a grammar rule `('->')/2` is principal functor of a `grammar-body-choice`. See section **??**.

7.14.6.4 The nonterminal ('!')//2

During execution of a grammar rule `('!')//2` commits parsing resp. generation to that grammar rule; no other grammar rule with same non-terminal indicator can be applied during execution of the respective comprehensive terminal sequence.
If occuring in the right-hand-context of the head of a grammar rule, `('!')//2` is prefixed to the remaining terminal sequence of that nonterminal, where it may commit further parsing or generating. The actual nonterminal is not influenced. See section **??**.

7.14.7 The nonterminal call//1

7.14.7.1 Description

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

```
call//1
```

shall lead to an expansion result

```
call/3
```

which is a legal goal for the control construct `call/3` which is required by this DTR and defined in 7.14.8.
A Prolog processor may support additional control constructs. Examples include *soft-cuts* and control constructs that enable the use of grammar rules stored on encapsulation units other than modules, such as objects. These additional control constructs must be treated as non-terminals by a Prolog processor working on a strictly conforming mode (see 5.1e).

NOTE — Consider the following example for the correlation of Grammar

Rules, `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).
```

7.14.8 The control construct `call/3`

7.14.8.1 Description

`call(G, A1, A2)` is true iff G is a goal which is true when activated using the implementation defined arguments A1 and A2. For the definition of G, and Error cases restricted to G, see section 7.8.3 of ISO/IEC 13211-1.
The definitions of the arguments A1 and A2, examples and the error cases of `call/3` shall be supplied in the implementation specific documentation.

7.14.8.2 Template

```
call(+callable_term, ?argument1, ?argument2)
```

7.14.9 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' of a built-in predicate 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 built-in predicates defined in 7.14.6.

When the database does not contain a grammar rule with non-terminal indicator N//A during execution of a non-terminal with non-terminal indicator N//A , the error term as specified in clause 7.7.7b of ISO/IEC 13211–1 when the flag `unknown` is set to `error` shall be:

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

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 texts with grammar rules is defined in section 8.1.1. Grammar rules are expanded 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 the comprehensive terminal sequence `S0` unifies with the concatenation of either a terminal sequence of the grammar rule body `GRBody` or a terminal sequence resulting from generation by the non-terminal of `GRBody` w.r.t. the current Grammar rules, both extended by the remaining terminal sequence, where `S` unifies with the remaining terminal sequence.
If the nonterminal of `GRBody` is followed by a right-hand-context (cf. definition 3.2), then the right-hand-context shall be prefixed to the remaining terminal sequence after having parsed resp. generated wrt. the nonterminal 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 **??**.

8.1.1.2 Template and modes

`phrase(+terminal-sequence, ?terminal-sequence, ?terminal-sequence)`

8.1.1.3 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, i.e. if a Prolog processor offers them, their form must be the following:

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

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.

8.1.1.4 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 recognized resp. generated:

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

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

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

verb_phrase --> verb.
verb_phrase --> verb, noun_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]).
 no

| ?- phrase(sentence, Sentence).

Sentence = [the, girl, likes, the, boy]
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.