The Draft Formal Definition of Ada ®

The Extent of the Trial Dynamic Semantics Definition

DATE : *Pisa, 29 January , 1985*

AUTHOR : *Claus Bendix Nielsen, DDC*
Alessandro Fantechi, I.E.I. - C.N.R.
Franco Mazzanti, CRAI
Jan Storbank Pedersen, DDC

REPORT No :

WORKPACKAGE : *E (Trial Definition)*

DISTRIBUTION : *Internal Use Only*

This work is partially funded by the Commission of the European Communities under the Multi-Annual Programme in the field of Data-Processing, proj. No. 782. "The Draft Formal Definition of ANSI/MIL-STD 1815A Ada".

TABLE OF CONTENTS

1 THE EXTENT OF THE ADA DYNAMIC SEMANTICS IN THE Ada FD PROJECT

In the Formal Definition of the Ada language, we are interested in giving the semantics of a syntactic object which is built following an Ada grammar (i.e. it is syntactically correct) which does not contain any error that an implementation conforming to the standard is required to detect (i.e. it is legal); we will call this syntactic object an Ada user-program.
However, this statement requires some clarification because the definitions of legality and executability of a program may in general be implementation dependent concepts; in particular, we have found two large classes of restrictions to be imposed on Ada user-programs in order to ensure their legality and executability in every implementation:

a) Restrictions on legality imposed by implementation dependent parts of the predefined environment.

The first class of restrictions is related to the following dependences of the legality on the particular predefined environment of the implementation:

1) When a program makes use of some additional (implementation defined) numeric types defined within the package STANDARD:

```
Es.   ...                                -- the definition of the type SHORT_INTEGER
      X: SHORT_INTEGER := 27283;         -- is neither provided by the user, nor part
      ...                                -- of the "standard" part of the package
      ...                                -- STANDARD
```

In this case the legality of a program depends on the structure of the package STANDARD.

2) When a program makes use of the internal structure of the type ADDRESS defined within the predefined package SYSTEM:

A : **constant** ADDRESS := (12333,33473)   -- it might be a pair (segment, offset)

In this case the legality of a program depends on the structure of the package SYSTEM.

3) When a program makes use of the types defined in the package LOW_LEVEL_IO :

```
Dev : LOW_LEVEL_IO.DEVICE := 37;                -- some device information
Inf : LOW_LEVEL_IO.DATA := (561,934894);        -- some data kind
  ....
LOW_LEVEL_IO.SEND_CONTROL (Dev,Inf);            -- some interaction with the device
```

In this case the legality of a program depends on the structure of the package LOW_LEVEL_IO.

4) When a program makes use of the types defined in the package MACHINE_CODE:

```
procedure SET_MASK is
   use MACHINE_CODE;
begin
    SI_FORMAT'(CODE => SSM, B => BaseReg, D => Display);
      -- the type SI_FORMAT is defined in the MACHINE_CODE package
end;
```

In this case the legality of a program depends on the structure of the package MACHINE CODE.

5) When a program makes use of some implementation defined attribute (to be specified in the appendix "F" of the manual):

```
M: MASK;
BaseReg : INTEGER := M'BASE_REG;
Display : INTEGER := M'DISP;
```

In this case the legality of a program depends on the existence of implementation defined attributes.

b) Semantic restrictions imposed by an implementation.

The second class is related with the semantic restrictions (on the implementation of some features) that an implementation is allowed to impose.
This occurs in three cases:

2 THE TRIAL DEFINITION DYNAMIC SEMANTICS SUBSET

Prior to carrying out the Trial Definition, a subset of the Ada language to be formally described had to be identified. The purpose of the subset is to enable the Trial Definition to demonstrate the feasibility of the models and techniques to be employed in the full formal definition.

The choice of Ada constructs has been made with the above purpose of the subset in mind; this in particular means that:

- the resulting language is not intended as a "programming language", i.e. a language intended for programming.
- the constructs included must be suitable to serve as a basis for an assessment of the model and techniques used.

The dynamic semantics subset is described in the report "Dynamic Semantics Example Ada Subset" [Christensen et al. 85].

The primary general principle behind the choice of the subset language is that it should contain all difficult and/or interesting Ada concepts. Hence, a major part of the simplifications introduced involve the removal of simple aspects related to constructs which are otherwise considered important. Moreover, simple aspects are removed if they would require a lot of tedious work, while simple aspects that are simple to model are often included - also to give to the subset a minimal flavour of a programming language.

Summarizing, the main features of the subset, divided by manual chapters, are the following:

- predefined integer types, subtypes, one-dimensional array types, record types with (one) discriminant and variant part, access types are included;
- very limited sets of operators and attributes are included;
- all statements (except case and code statements) are included, with some limitation on if and loop statements;
- both procedures and functions are included, but only in and inout parameters and positional parameter association are permitted;
- packages are included, with private types;
- all of tasking is included, except that task objects are not allowed as subcomponents, there are no entry families and no when conditions;
- subunits are not included;
- all of exceptions is included;
- the only form of generic units are generic packages with private type parameters;
- restricted representation clauses and representation attributes, a simplified package

SYSTEM and unchecked programming are included;

- a simplified package DIRECT_IO is included;
- a simplified package STANDARD is included.

3 CONCLUSION

In the end, the dynamic semantics Trial Definition formally defines the language which is the intersection of what is defined by the two largely independent sections above: the general aspects of modeling Ada dynamic semantics and the selected subset.
Moreover, the fact that certain Ada constructs are absent in the subset has introduced two (later discovered) inconsistencies in the subset itself:

- since enumeration types are not in the subset, the type BOOLEAN, defined in the package STANDARD as an enumeration type, causes some problems. The solution adopted in the Trial Definition involves not having the Ada type BOOLEAN, but using the booleans of the metalanguage whenever boolean values are needed.
- the lack of enumeration types also implies that the type STRING cannot be defined as an array of characters. However, STRINGs are still needed to express the NAME and FORM parameters for input-output. In [Christensen et al. 85] an attempt to solve this problem was proposed, but a more straightforward approach has been chosen for the Trial Definition: to define STRING as a subtype of INTEGER.

4 REREFERNCES

[Christensen et al. 85] P. Christensen, J.Storbank Pedersen, A.Fantechi, A.Giovini, G.Reggio, F.Mazzanti, "Dynamic Semantics Example Ada Subset", Deliverable 6B of The Draft Formal Definition of ANSI/MIL-STD 1815A Ada, 6 Aug 1985.