
**Information technology — Programming
languages — Generic package of complex
elementary functions for Ada**

*Technologies de l'information — Langages de programmation —
Paquetage générique de fonctions élémentaires complexes pour Ada*

Contents

Page

1	Scope	1
2	Normative references	1
3	Subprograms provided	1
4	Instantiations	2
5	Implementations	3
6	Exceptions	4
7	Arguments outside the range of safe numbers	5
8	Method of specification of functions	5
9	Branch cut and domain definitions	5
10	Range definitions	5
11	Accuracy requirements	6
12	Overflow	7
13	Underflow	8
14	Specifications of the functions	9
14.1	SQRT — Square root	9
14.2	LOG — Natural logarithm	10
14.3	EXP — Exponential function, complex argument	10
14.4	EXP — Exponential function, imaginary argument	11

© ISO/IEC 1998

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

ISO/IEC Copyright Office • Case postale 56 • CH-1211 Genève 20 • Switzerland

Printed in Switzerland

14.5	"**" — Exponentiation operator, complex to complex power arguments	12
14.6	"**" — Exponentiation operator, complex to real power arguments	12
14.7	"**" — Exponentiation operator, real to complex power arguments	13
14.8	SIN — Trigonometric sine function	14
14.9	COS — Trigonometric cosine function	15
14.10	TAN — Trigonometric tangent function	16
14.11	COT — Trigonometric cotangent function	16
14.12	ARCSIN — Inverse trigonometric sine function	17
14.13	ARCCOS — Inverse trigonometric cosine function	18
14.14	ARCTAN — Inverse trigonometric tangent function	19
14.15	ARCCOT — Inverse trigonometric cotangent function	19
14.16	SINH — Hyperbolic sine function	20
14.17	COSH — Hyperbolic cosine function	21
14.18	TANH — Hyperbolic tangent function	22
14.19	COTH — Hyperbolic cotangent function	23
14.20	ARCSINH — Inverse hyperbolic sine function	23
14.21	ARCCOSH — Inverse hyperbolic cosine function	24
14.22	ARCTANH — Inverse hyperbolic tangent function	25
14.23	ARCCOTH — Inverse hyperbolic cotangent function	25

Annexes

A	Ada specification for GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS	27
B	Rationale	29
B.1	Introduction and motivation	29
B.2	History	29
B.3	Packaging considerations	29
B.4	Choice of functions	30
B.5	Handling of exceptions	30
B.6	Matters of style	32
B.7	Method of expressing accuracy requirements	32
C	Bibliography	35

Figures

B.1	Result rectangle for a subprogram satisfying a maximum relative component error requirement	33
B.2	Result rectangle for a subprogram satisfying a maximum relative box error requirement	33
B.3	Result circle for a subprogram satisfying a maximum relative vector error requirement	34

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 13814 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee 22, *Programming languages, their environments and system software interfaces*.

Annex A forms an integral part of this International Standard. Annexes B and C are for information only.

Introduction

The generic package described here is intended to provide the basic complex mathematical routines from which portable, reusable applications can be built. This International Standard serves a broad class of applications with reasonable ease of use, while demanding implementations that are of high quality, capable of validation, and also practical given the state of the art.

The specification included in this International Standard is presented as a compilable Ada specification in Annex A, with explanatory text in numbered clauses in the main body of text. The explanatory text is normative, with the exception of the following items:

- notes (under the heading Notes associated with some of the functions); and
- notes (labeled as such) presented at the end of any numbered clause.

The word “may” as used in this International Standard consistently means “is allowed to” (or “are allowed to”). It is used only to express permission, as in the commonly occurring phrase “an implementation may”; other words (such as “can,” “could” or “might”) are used to express ability, possibility, capacity or consequentiality.

IECNORM.COM : Click to view the full PDF of ISO/IEC 13814:1998

Information technology — Programming languages — Generic package of complex elementary functions for Ada

1 Scope

This International Standard defines the specification of a generic package of complex elementary functions called **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS**. It does not provide the body of the package.

This International Standard specifies certain basic complex mathematical routines from which portable, reusable applications can be built. This International Standard serves a broad class of applications with reasonable ease of use, while demanding implementations that are of high quality, capable of validation, and also practical given the state of the art.

This International Standard is applicable to programming environments conforming to ISO/IEC 8652:1987 and is relevant to the revised standard ISO/IEC 8652:1995.

2 Normative references

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

ISO/IEC 8652, *Information technology — Programming languages — Ada*.

ISO/IEC 11430, *Information technology — Programming languages — Generic package of elementary functions for Ada*.

ISO/IEC 11729, *Information technology — Programming languages — Generic package of primitive functions for Ada*.

ISO/IEC 13813, *Information technology — Programming languages — Generic packages of real and complex type declarations and basic operations for Ada (including vector and matrix types)*.

3 Subprograms provided

The following twenty mathematical functions are provided:

SQRT	LOG	EXP	"**"
SIN	COS	TAN	COT
ARCSIN	ARCCOS	ARCTAN	ARCCOT
SINH	COSH	TANH	COTH
ARCSINH	ARCCOSH	ARCTANH	ARCCOTH

These are the square root (**SQRT**), logarithm (**LOG**) and exponential (**EXP**) function and the exponentiation operator (******); the trigonometric functions for sine (**SIN**), cosine (**COS**), tangent (**TAN**) and cotangent (**COT**) and their inverses

(**ARCSIN**, **ARCCOS**, **ARCTAN** and **ARCCOT**); and the hyperbolic functions for sine (**SINH**), cosine (**COSH**), tangent (**TANH**), and cotangent (**COTH**) together with their inverses (**ARCSINH**, **ARCCOSH**, **ARCTANH**, and **ARCCOTH**). These are the same functions by the same names that are in the package **GENERIC_ELEMENTARY_FUNCTIONS** defined in ISO/IEC 11430.

Several variations are provided for the exponentiation operator and the exponential function. All functions have one or more formal parameters of type **COMPLEX** or **IMAGINARY** and return a value of type **COMPLEX**. **COMPLEX** and **IMAGINARY** are generic formal parameters of the generic package **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS**.

4 Instantiations

This International Standard describes a generic package, **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS** which the user must instantiate to obtain a computational capability. The generic package has three required generic formal parameters and many generic formal subprograms with defaults. At instantiation, the user must specify, as the generic actual parameters to be associated with **REAL**, **COMPLEX** and **IMAGINARY** respectively,

- a) a floating-point type,
- b) a private composite type containing real and imaginary components of the type specified by a), (Within this International Standard, the real and imaginary components of the composite type are referred to simply as components.)
- c) a private type of the same base type as a) that is interpreted as the pure imaginary form of b).

Types suitable for b) and c) are exported by the package obtained by instantiating, with the type specified by a) as its generic actual parameter, the generic package **GENERIC_COMPLEX_TYPES** defined in ISO/IEC 13813. (Instantiation of the package **GENERIC_COMPLEX_TYPES** is not a prerequisite for using **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS**; it is merely an option that the user might find convenient to pursue.) In addition the user may specify the many subprogram generic formal parameters as generic actual parameters, or for convenience, all these subprograms are exported by the instantiation of **GENERIC_COMPLEX_TYPES** described above.

Depending on the implementation, the user may or may not be allowed to specify a generic actual type having a range constraint for the first generic actual parameter (see clause 5). The generic actual parameters for the subprograms can be omitted if functions having names and profiles matching those of the corresponding generic formal parameters are visible at the place of the instantiation. An instantiation of **GENERIC_COMPLEX_TYPES**, as described above, can optionally be used to obtain subprograms that satisfy these requirements.

In addition to the body of the generic package itself, implementers may provide (non-generic) library packages that can be used just like instantiations of the generic package in which the first generic actual parameter is a predefined floating-point type and the remainder are related to the first in appropriate ways. In particular, the name of a package serving as a replacement for an instantiation of **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS** for the predefined type **FLOAT** shall be **COMPLEX_ELEMENTARY_FUNCTIONS** with **REAL** replaced by **FLOAT** in the profiles of the functions that it exports. Similarly, the names of packages serving as replacements for instantiations of **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS** for the predefined types **LONG_FLOAT**, **SHORT_FLOAT**, etc., shall be **LONG_COMPLEX_ELEMENTARY_FUNCTIONS**, **SHORT_COMPLEX_ELEMENTARY_FUNCTIONS**, etc., respectively and **REAL** shall be systematically replaced in the profiles of the functions that they export by **LONG_FLOAT**, **SHORT_FLOAT**, etc. Implementers are responsible for ensuring that a composite type called **COMPLEX** containing a pair of components of types **FLOAT**, **LONG_FLOAT**, **SHORT_FLOAT**, etc., respectively, are available. For example, each of the non-generic library packages could be prefixed by an appropriate context clause, such as:

```
with COMPLEX_TYPES;
package COMPLEX_ELEMENTARY_FUNCTIONS is ...
```

The packages **COMPLEX_TYPES**, **LONG_COMPLEX_TYPES**, etc., described in ISO/IEC 13813, are not a prerequisite for implementation, yet are an option the implementer might find convenient to pursue.

5 Implementations

Portable implementations of **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS** are strongly encouraged. However, implementations are not required to be portable. In particular, an implementation of this International Standard in Ada may use pragma **INTERFACE** or other pragmas, unchecked conversion, machine-code insertions or other machine-dependent techniques as desired. On the other hand, to the extent that generic packages (e.g. **GENERIC_ELEMENTARY_FUNCTIONS** defined in ISO/IEC 11430 and **GENERIC_PRIMITIVE_FUNCTIONS** defined in ISO/IEC 11729) become widely available, portable implementations of **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS** should not be particularly difficult to construct. In particular, it is possible to achieve the accuracy required by this International Standard, when specified, by appropriate use of the package **GENERIC_ELEMENTARY_FUNCTIONS**.

An implementation is allowed to limit the precision it supports (by stating an assumed maximum value for **SYSTEM.MAX_DIGITS**), since portable implementations would not, in general, be possible otherwise. An implementation is also allowed to make other reasonable assumptions about the environment in which it is to be used, but only when necessary in order to match algorithms to hardware characteristics in an economical manner. All such limits and assumptions shall be clearly documented. By convention, an implementation of **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS** is said not to conform to this International Standard in any environment in which its limits or assumptions are not satisfied, and this International Standard does not define its behavior in that environment. In effect, this convention delimits the portability of implementations.

An implementation is allowed to impose a restriction that the generic actual parameter associated with **REAL** shall not have a range constraint that reduces the range of allowable values. If it does impose this restriction, then the restriction shall be documented, and the effects of violating the restriction shall be one of the following:

- a) Compilation of a unit containing an instantiation of **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS** is rejected.
- b) **CONSTRAINT_ERROR** or **PROGRAM_ERROR** is raised during the elaboration of an instantiation of **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS**.

Conversely, if an implementation does not impose the restriction, then it must not allow such a range constraint, when included with the user's actual type, to interfere with the internal computations of the functions; that is, if the components of the argument and the result are within the range of the type, then the implementation shall return the result and shall not raise an exception (such as **CONSTRAINT_ERROR**).

Any of the restrictions discussed above may in fact be inherited from implementations of the packages **GENERIC_ELEMENTARY_FUNCTIONS** defined in ISO/IEC 11430, **GENERIC_PRIMITIVE_FUNCTIONS** defined in ISO/IEC 11729 and **GENERIC_COMPLEX_TYPES** defined in ISO/IEC 13813, if used. The dependence of an implementation on such inherited restrictions should be documented.

An implementation shall function properly in a tasking environment. Apart from the obvious restriction that an implementation of **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS** shall avoid declaring variables that are global to the functions, no special constraints are imposed on implementations. Nothing in this International Standard requires the use of such global variables.

Some hardware and their accompanying Ada implementations have the capability of representing and discriminating between positively and negatively signed zeros as a means, e.g., of preserving the sign of an infinitesimal quantity that has underflowed to zero. This International Standard allows implementations of **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS** to exploit that capability, when available, so as to exhibit continuity in the results of some functions as certain limits are approached. At the same time, it accommodates implementations in which that capability is unavailable. Because a definition of what comprises the capability of representing and distinguishing signed zeros is beyond the scope of this International Standard, implementations are allowed the freedom not to exploit the capability, even when available. An implementation shall exercise its choice consistently, either exploiting signed-zero behavior everywhere or nowhere in this package. The signs of zero results prescribed by this International Standard apply only to implementations that exploit signed zeros; an implementation that does not exploit signed zeros may give any sign to a zero result. In addition an implementation shall document its behavior with respect to signed zeros. In this International Standard, unless otherwise qualified, zero means any of $(\pm 0.0, \pm 0.0)$ and one means either of $(1.0, \pm 0.0)$.

6 Exceptions

The **ARGUMENT_ERROR** exception is raised by a function in the generic package when an argument of the function violates one or more of the conditions given in the function's domain definition (see clause 9). Note that these conditions are related only to the mathematical definition of the function and are therefore implementation independent.

The **ARGUMENT_ERROR** exception is declared as a renaming of the exception of the same name declared in the package **ELEMENTARY_FUNCTIONS_EXCEPTIONS** defined in ISO/IEC 11430. This exception distinguishes neither between different kinds of argument errors, nor between different functions, nor between different instantiations of **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS**, nor between instantiations of **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS** and those of **GENERIC_ELEMENTARY_FUNCTIONS**.

Besides **ARGUMENT_ERROR**, the only exceptions allowed during a call to a function in **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS** are predefined exceptions, as follows:

- a) Virtually any predefined exception is possible during the evaluation of an argument of a function in **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS**. For example, **NUMERIC_ERROR**, **CONSTRAINT_ERROR** or even **PROGRAM_ERROR** could be raised if an argument has an undefined value; and **CONSTRAINT_ERROR** will be raised when the value of an argument (or component of an argument) lies outside the range of the user's generic actual type **REAL**. Additionally **STORAGE_ERROR** could be raised, e.g. if insufficient storage is available to perform the call. All these exceptions are raised before the body of the function is entered and therefore have no bearing on implementations of **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS**.
- b) Also, **CONSTRAINT_ERROR** will be raised when a function in **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS** attempts to construct a complex return value with a component outside the range of the user's generic actual type associated with **REAL**. The exception raised for this reason shall be propagated to the caller of the function.
- c) Whenever the arguments of a function are such that a component of a result permitted by the accuracy requirements would exceed **REAL'SAFE_LARGE** in absolute value, as formalized below in clause 12, an implementation may raise, and shall then propagate to the caller, the exception specified by Ada for signaling overflow.
- d) Whenever the arguments of a function are such that the corresponding mathematical function is infinite (i.e., has an infinite component), an implementation shall raise and propagate to the caller the exception specified by Ada for signaling division by zero.
- e) Once execution of a function has begun, an implementation may propagate **STORAGE_ERROR** to the caller of the function, but only to signal the unexpected exhaustion of storage. Similarly, once execution of a function has begun, an implementation may propagate **PROGRAM_ERROR** to the caller of the function, but only to signal errors made by the user of **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS**.

No exception is allowed during a call to a function in **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS** except those permitted by the foregoing rules. In particular, for arguments for which all components of results satisfying the accuracy requirements remain less than or equal to **REAL'SAFE_LARGE** in absolute value in both real and imaginary parts, a function shall locally handle an overflow occurring during the computation of an intermediate result, if such an overflow is possible, and shall not propagate an exception signaling that overflow to the caller of the function.

The only exceptions allowed during an instantiation of **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS**, including the execution of the optional sequence of statements in the body of the instance, are **CONSTRAINT_ERROR**, **STORAGE_ERROR** and **PROGRAM_ERROR**, and then only for the following reasons. The raising of **CONSTRAINT_ERROR** during instantiation is only allowed when the implementation imposes the restriction that the generic actual type associated with **REAL** shall not have a range constraint, and the user violates that restriction (it may, in fact, be an inescapable consequence of the violation). The raising of **PROGRAM_ERROR** during instantiation is only allowed for the purpose of signaling errors made by the user violating some restriction or limitation of the implementation. The raising of **STORAGE_ERROR** during instantiation is only allowed for the purpose of signaling the exhaustion of storage.

NOTE — In ISO/IEC 8652:1995, the exception specified for signaling overflow is **CONSTRAINT_ERROR**.

7 Arguments outside the range of safe numbers

The ISO/IEC 8652:1987 fails to define the result safe interval of any basic or predefined operation of a real subtype when the absolute value of one of its operands exceeds the largest safe number of the operand subtype. (The failure to define a result in this case occurs because no safe interval is defined for the operand in question.) In order to avoid imposing requirements that would, consequently, be more stringent than those of Ada itself, this International Standard likewise does not define the result of a contained function when the absolute value of a component of one of its arguments exceeds `REAL'SAFE_LARGE`. All of the accuracy requirements and other provisions of the following clauses are understood to be implicitly qualified by the assumption that both components of function arguments are less than or equal to `REAL'SAFE_LARGE` in absolute value.

8 Method of specification of functions

One of the functions has three overloaded forms. For each form of each function covered by this International Standard, the function is specified by its parameter and result type profile, a defining equation, the applicable branch cuts, the domain of its argument(s), its range, and the accuracy required of its implementation. The meaning of, and conventions applicable to, these specifications are described below.

The specification of each function has a heading called *Definition*. This is meant to convey the required behavior of the function. The definition is not necessarily an appropriate implementation for the function. The definition is one of the mathematically correct definitions of the function but algebraic simplifications or transformations may not be correct. Transformations may cause the sign of the result to be wrong or cause a gratuitous singularity. The definition is based on exact computation whereas an implementation must take into account the accuracy requirements.

9 Branch cut and domain definitions

The specification of each function covered by this International Standard includes headings of *Branch cut* and *Domain*. The branch cuts or slits are specified to warn the user of discontinuity in the function or one of its derivatives. Under the heading *Domain*, a characterization of the argument values is given for which the function is mathematically defined. The portion of the complex plane over which the function is defined is expressed by inequalities or other conditions which the arguments must satisfy to be valid. The phrase “mathematically unbounded” in a domain definition indicates that all representable values of the argument are valid, i.e., that the domain is the entire complex plane. Whenever the arguments fail to satisfy all the conditions, the implementation shall raise `ARGUMENT_ERROR`. The implementation shall not raise `ARGUMENT_ERROR` if all the domain conditions are satisfied.

Inability to deliver a result for valid arguments because the result overflows, for example, shall not raise `ARGUMENT_ERROR`, but shall be treated in the same way that Ada defines for its predefined floating-point operations (see clause 12).

10 Range definitions

The usual mathematical meaning of the “range” of a function is the set of values into which the function maps the values in its domain. Many of the functions covered by this International Standard are mathematically multivalued, in the sense that a given argument value can be mapped by the function into many different result values. By means of range restrictions, this International Standard imposes a uniqueness requirement on the results of multivalued functions, thereby reducing them to single-valued functions.

The range of each function is shown under the heading *Range* in the specifications. The range definition is expressed by inequalities or other conditions that shall be satisfied by the components of the returned result. An implementation shall not exceed a limit of a range component when that limit is a safe number of `REAL` (like `0.0` or `1.0`). On the other hand, when a range component limit is not a safe number of `REAL` (like π), an implementation is allowed to exceed the range component limit, but it is not allowed to exceed the safe number of `REAL` next beyond the range component limit in the direction away from the interior of the range component; this is in general the best that can be expected

from a portable implementation. Effectively, therefore, range definitions have the added effect of imposing accuracy requirements on implementations above and beyond those presented under the heading *Accuracy* in the specifications (see clause 11).

The phrase “mathematically unbounded” in a range definition indicates that the range of values of the function is not bounded by its mathematical definition.

11 Accuracy requirements

Because they are implemented on digital computers with only finite precision, the functions provided in this generic package can, at best, only approximate the corresponding mathematically defined functions.

The accuracy requirements contained in this International Standard define the latitude that implementations are allowed in approximating the intended precise mathematical result with floating-point computations. Accuracy requirements of two kinds are stated under the heading *Accuracy* in the specifications. Additionally, range definitions stated under the heading *Range* impose requirements that constrain the values implementations may yield, so the range definitions are another source of accuracy requirements (the precise meaning of a range limit that is not a safe number of **REAL**, as an accuracy requirement, is discussed above in clause 10). Every result yielded by a function is subject to all of the function’s applicable accuracy requirements, except in the one case described in clause 13, below. In that case, the result will satisfy a small absolute error requirement in lieu of the other accuracy requirements defined for the function.

The first kind of accuracy requirement used under the heading *Accuracy* in the specifications is a separate bound on the relative error in the components of the computed value of the function, which shall hold (except as provided by the rules in clauses 12 and 13) for all arguments satisfying the conditions in the domain definition, providing the mathematical result is finite and nonzero. For some functions, the relative error bound is stated as a bound on the “relative box error” in each component, while for others it is stated as a bound on the “relative component error” in each component. For a given function f , having an exact result $f(z)$, the relative box error in the components of a computed result $F(z)$ at the argument z is defined as

relative box error in real part of result

$$= \frac{|\operatorname{re}(F(z)) - \operatorname{re}(f(z))|}{\max(|\operatorname{re}(f(z))|, |\operatorname{im}(f(z))|)}$$

relative box error in imaginary part of result

$$= \frac{|\operatorname{im}(F(z)) - \operatorname{im}(f(z))|}{\max(|\operatorname{re}(f(z))|, |\operatorname{im}(f(z))|)}$$

while the relative component error is defined as

relative component error in real part of result

$$= \frac{|\operatorname{re}(F(z)) - \operatorname{re}(f(z))|}{|\operatorname{re}(f(z))|}$$

relative component error in imaginary part of result

$$= \frac{|\operatorname{im}(F(z)) - \operatorname{im}(f(z))|}{|\operatorname{im}(f(z))|}$$

providing the denominators are finite and nonzero. The relative box error in a component of the result is not defined when the mathematical result is zero or either of its components are infinite. The relative component error in a component of the result is not defined when that component of the mathematical result is infinite or zero.

The second kind of accuracy requirement used under the heading *Accuracy* in the specifications is a stipulation, in the form of an equality, that the implementation shall deliver “prescribed results” for certain special arguments. It is used for two purposes: to define a component of the computed result to be zero when the relative error is undefined, i.e., when the mathematical result has a zero component, and to strengthen the accuracy requirements at special argument values. When a component of such a prescribed result is a safe number of **REAL** (like 0.0 or 1.0), an implementation shall deliver a result having that safe number as its component. On the other hand, when a component of a prescribed result is not a safe number of **REAL** (like π or $\pi/2$), an implementation may deliver a result in which the component has any value in the surrounding safe interval. Prescribed results take precedence over maximum relative error requirements but never contravene them. The real and imaginary parts may have any combination of the two kinds of accuracy requirement.

Range definitions, under the heading *Range* in the specifications, are an additional source of accuracy requirements, as stated above in clause 10. As an accuracy requirement, a range definition (other than “mathematically unbounded”) has the effect of eliminating some of the values permitted by the maximum relative error requirements, i.e., those outside the range.

12 Overflow

Floating-point hardware is typically incapable of representing numbers whose absolute value exceeds some implementation-defined maximum. For the type **REAL**, that maximum will be at least **REAL'SAFE_LARGE**. For the functions defined by this International Standard, whenever the maximum error requirements permit a result with a component whose absolute value is greater than **REAL'SAFE_LARGE**, the implementation may

- yield any result permitted by the maximum relative error requirements, or
- raise the exception specified by Ada for signaling overflow.

In addition, some of the functions are allowed to signal overflow for certain arguments for which neither component of the result can overflow. This freedom is granted in the following specific cases:

- a) for **EXP**, when $\text{re } X > \log_e \text{REAL'SAFE_LARGE}$;
- b) for **SIN**, when $\text{im } X > \log_e \text{REAL'SAFE_LARGE} + \log_e 2.0$;
- c) for **COS**, when $\text{im } X > \log_e \text{REAL'SAFE_LARGE} + \log_e 2.0$;
- d) for **SINH**, when $\text{re } X > \log_e \text{REAL'SAFE_LARGE} + \log_e 2.0$;
- e) for **COSH**, when $\text{re } X > \log_e \text{REAL'SAFE_LARGE} + \log_e 2.0$.

Permission to signal overflow in these cases recognizes the difficulty of avoiding overflow in the computation of intermediate results and allows the same latitude as specified in the package **GENERIC_ELEMENTARY_FUNCTIONS** defined in ISO/IEC 11430 for real functions **EXP**, **SINH** and **COSH**.

An implementation shall raise the exception specified by Ada for signaling division by zero in the following specific cases:

- a) **LOG(X)** when **X** is zero;
- b) **LEFT ** RIGHT** (clause 14.5) when **LEFT** is zero and $\text{re } \text{RIGHT} < 0.0$;
- c) **LEFT ** RIGHT** (clause 14.6) when **LEFT** is zero and $\text{RIGHT} < 0.0$;
- d) **LEFT ** RIGHT** (clause 14.7) when **LEFT** = 0.0 and $\text{re } \text{RIGHT} < 0.0$;
- e) **COT(X)** when **X** is zero;

- f) **ARCTAN(X)** when **X** is $\pm i$;
- g) **ARCCOT(X)** when **X** is $\pm i$;
- h) **COTH(X)** when **X** is zero;
- i) **ARCTANH(X)** when **X** is either one or negative one; and
- j) **ARCCOTH(X)** when **X** is either one or negative one.

In these cases, the corresponding mathematical functions have a component that is infinite.

NOTES

- 1 The rule concerning overflow permits an implementation to raise an exception, instead of delivering a result, for arguments for which either component of the mathematical result is close to but does not exceed **REAL'SAFE_LARGE** in absolute value. Such arguments must necessarily be very close to an argument for which the component of the mathematical result does exceed **REAL'SAFE_LARGE** in absolute value. In general, this is the best that can be expected from a portable implementation with a reasonable amount of effort.
- 2 The rule is motivated by the behavior prescribed in ISO/IEC 8652:1987 for the predefined operations. That is, when the set of possible results of a predefined operation includes a number whose absolute value exceeds the implementation-defined maximum, the implementation is allowed to raise the exception specified for signaling overflow instead of delivering a result.
- 3 In ISO/IEC 8652:1995, the exception specified for signaling overflow is **CONSTRAINT_ERROR**.

13 Underflow

Floating-point hardware is typically incapable of representing nonzero numbers whose absolute value is less than some implementation-defined minimum. For the type **REAL**, that minimum will be at most **REAL'SAFE_SMALL**. For the functions defined by this International Standard, whenever the maximum relative error requirements permit a result with a component whose absolute value is less than **REAL'SAFE_SMALL** and a prescribed result is not stipulated, the implementation may yield a result in which that component is

- a) any value permitted by the maximum relative error requirements;
- b) any nonzero value less than or equal to **REAL'SAFE_SMALL** in magnitude (and having the correct sign, unless the maximum relative error requirements permit values with either sign); or
- c) zero.

NOTES

- 1 Whenever part b) or c) of this rule takes effect for a component of the result, the maximum relative error requirements for that component are, in general, unachievable and are waived.
- 2 The rule permits an implementation to deliver a result violating the maximum relative error requirements for arguments for which a component of the mathematical result equals or slightly exceeds **REAL'SAFE_SMALL** in absolute value. Such arguments must necessarily be very close to an argument for which that component of the mathematical result is less than **REAL'SAFE_SMALL** in absolute value. In general, this is the best that can be expected from a portable implementation with a reasonable amount of effort.
- 3 The rule is motivated by the behavior prescribed in ISO/IEC 8652:1987 for predefined operations. That is, when the set of possible results of a predefined operation includes a nonzero number whose absolute value is less than the implementation-defined minimum, the implementation is allowed to yield zero or any nonzero number having the correct sign and an absolute value less than or equal to that minimum. An exception is never raised in this case.

14 Specifications of the functions

The equations on the right side of the $\approx$ are mathematical definitions and are not meant to specify how the functions should be implemented. Although some definitions are well known, other definitions are critical to specify the required function, for example the difference in the definition of **ARCTAN** vs. **ARCCOT** and **ARCTANH** vs. **ARCCOTH** in order to get the required values along the branch cut. Yet other definitions provide a basis for accuracy analysis and overflow threshold determination, as in **EXP**, **SIN**, **COS**, **TAN**, **COT**, **SINH**, **COSH**, **TANH** and **COTH**. The mathematical functions $| \cdot |$, $\arg$, re and im denote the modulus (absolute value), argument (angle in $[-\pi, \pi]$), real part and imaginary part respectively. The branch cuts are informational as they may be derived from the definition, domain and range. Branch cuts are important to implementers and users of the specification because branch cuts identify where a function is not analytic.

Under the heading *Definition* in each of the following specifications, the semantics of an Ada call to the function being defined is provided by a mathematical definition in the form of an approximation. The left-hand side (the function call) is set in the fixed-width font used throughout this International Standard for program fragments. The right-hand side is to be interpreted as an exact mathematical formula; as such, it and similar mathematical formulas throughout this International Standard employ standard mathematical symbols, notation and fonts (except for variable names and some real literals, which are set in the fixed-width "program-fragment" font). The degree to which the function call on the left-hand side is allowed to approximate the value of the formula on the right-hand side is, of course, spelled out under the heading *Accuracy*, as discussed in clause 11.

14.1 SQRT — Square root

14.1.1 Declaration

```
function SQRT (X : COMPLEX) return COMPLEX;
```

14.1.2 Definition

$$\text{SQRT}(X) \approx \sqrt{X} = \sqrt{|X|} \cdot e^{i(\arg X)/2}$$

14.1.3 Branch cut

Negative real axis

14.1.4 Domain

Mathematically unbounded

14.1.5 Range

Real part nonnegative, imaginary part mathematically unbounded

14.1.6 Accuracy

- a) Maximum relative component error = `6.0 * REAL 'BASE' EPSILON`
- b) The sign of the real part of the result is positive.
- c) The sign of the imaginary part of the result is the same as the sign of the imaginary part of **X**.
- d) When **X** is zero, the result is zero.
- e) When **X** is negative one, the result is **+i** except that implementations exploiting signed zeros shall conform to case c) above (see clause 5).

14.2 LOG — Natural logarithm**14.2.1 Declaration**

```
function LOG (X : COMPLEX) return COMPLEX;
```

14.2.2 Definition

$$\text{LOG}(X) \approx \log_e X = \log_e |X| + i \arg X$$

14.2.3 Branch cut

Negative real axis

14.2.4 Domain

Mathematically unbounded

14.2.5 Range

Real part mathematically unbounded, imaginary part in $[-\pi, \pi]$

14.2.6 Accuracy

- a) Maximum relative box error of real part = $13.0 \cdot \text{REAL}'\text{BASE}'\text{EPSILON}$
- b) Maximum relative box error of imaginary part = $4.0 \cdot \text{REAL}'\text{BASE}'\text{EPSILON}$
- c) The sign of the imaginary part of the result is the same as the sign of the imaginary part of X .
- d) When X is one, the result is zero.

NOTE — Clause 12 applies when X is zero.

14.3 EXP — Exponential function, complex argument**14.3.1 Declaration**

```
function EXP (X : COMPLEX) return COMPLEX;
```

14.3.2 Definition

$$\text{EXP}(X) \approx e^X = e^{\text{re } X} \cdot \cos(\text{im } X) + i e^{\text{re } X} \cdot \sin(\text{im } X)$$

14.3.3 Branch cut

None

14.3.4 Domain

Mathematically unbounded

14.3.5 Range

Mathematically unbounded

14.3.6 Accuracy

- a) Maximum relative component error = $7.0 \cdot \text{REAL}'\text{BASE}'\text{EPSILON}$ when $|\text{im } X|$ is less than or equal to some documented implementation-dependent threshold, which shall not be less than

$$\text{REAL}'\text{MACHINE_RADIX}^{\lfloor \text{REAL}'\text{MACHINE_MANTISSA}/2 \rfloor}$$

For larger values of $|\text{im } X|$, degraded accuracy is allowed. An implementation shall document its behavior for large $|\text{im } X|$.

- b) When X is zero, the result is one; in this case the sign of the imaginary part of the result is the same as the sign of the imaginary part of X .

NOTE — clause 12 applies when $\text{re } X > \log_e \text{REAL}'\text{SAFE_LARGE}$, approximately.

14.4 EXP — Exponential function, imaginary argument**14.4.1 Declaration**

function EXP (X : IMAGINARY) return COMPLEX;

14.4.2 Definition

$$\text{EXP}(X) \approx e^X = \cos(\text{im } X) + i \sin(\text{im } X)$$

14.4.3 Branch cut

None

14.4.4 Domain

Mathematically unbounded

14.4.5 Range

Real part in $[-1, 1]$, imaginary part in $[-1, 1]$

14.4.6 Accuracy

- a) Maximum relative component error = $2.0 \cdot \text{REAL}'\text{BASE}'\text{EPSILON}$ when $|X|$ is less than or equal to some documented implementation-dependent threshold, which shall not be less than

$$\text{REAL}'\text{MACHINE_RADIX}^{\lfloor \text{REAL}'\text{MACHINE_MANTISSA}/2 \rfloor}$$

For larger values of $|X|$, degraded accuracy is allowed. An implementation shall document its behavior for large $|X|$.

- b) When X is zero, the result is one; in this case the sign of the imaginary part of the result is the same as the sign of X .

14.5 "**" — Exponentiation operator, complex to complex power arguments

14.5.1 Declaration

function "**" (LEFT, RIGHT : COMPLEX) return COMPLEX;

14.5.2 Definition

$$\text{LEFT} ** \text{RIGHT} \approx \text{LEFT}^{\text{RIGHT}} = e^{\text{RIGHT} \cdot \log_e \text{LEFT}}$$

14.5.3 Branch cut

LEFT along negative real axis

14.5.4 Domain

re RIGHT $\neq$ 0.0 when LEFT is zero

14.5.5 Range

Mathematically unbounded

14.5.6 Accuracy

- a) Maximum relative box error is the error bound that results from an implementation using functions from this International Standard and from the package `GENERIC_ELEMENTARY_FUNCTIONS` defined in ISO/IEC 11430.
- b) When RIGHT is zero and LEFT is nonzero, the result is one.
- c) When LEFT is zero and re RIGHT > 0.0, the result is zero.
- d) When RIGHT is one, the result is LEFT.
- e) When LEFT is one, the result is one.

NOTES

- 1 `EXP(RIGHT*LOG(LEFT))` is an allowed implementation, but note the accuracy requirements.
- 2 clause 12 applies when LEFT is zero and re RIGHT < 0.0 .

14.6 "**" — Exponentiation operator, complex to real power arguments

14.6.1 Declaration

function "**" (LEFT : COMPLEX; RIGHT : REAL) return COMPLEX;

14.6.2 Definition

$$\text{LEFT} ** \text{RIGHT} \approx \text{LEFT}^{\text{RIGHT}} = e^{\text{RIGHT} \cdot \log_e \text{LEFT}}$$

14.6.3 Branch cut

LEFT along negative real axis

14.6.4 Domain

RIGHT $\neq$ 0.0 when LEFT is zero

14.6.5 Range

Mathematically unbounded

14.6.6 Accuracy

- a) Maximum relative box error is the error bound that results from an implementation using functions from this International Standard and from the package **GENERIC_ELEMENTARY_FUNCTIONS** defined in ISO/IEC 11430.
- b) When LEFT is nonzero and RIGHT = 0.0, the result is one.
- c) When LEFT is zero and RIGHT > 0.0, the result is zero.
- d) When RIGHT = 1.0, the result is LEFT.
- e) When LEFT is one, the result is one.

NOTES

- 1 EXP(RIGHT*LOG(LEFT)) is an allowed implementation, but note the accuracy requirements.
- 2 clause 12 applies when LEFT is zero and RIGHT < 0.0.

14.7 "*" — Exponentiation operator, real to complex power arguments**14.7.1 Declaration**

function "*" (LEFT : REAL; RIGHT : COMPLEX) return COMPLEX;

14.7.2 Definition

$$\text{LEFT} ** \text{RIGHT} \approx \text{LEFT}^{\text{RIGHT}} = \begin{cases} e^{\text{RIGHT} \cdot \log_e \text{LEFT}} & , \text{ when } \text{LEFT} \geq 0.0 \\ e^{\text{RIGHT} \cdot (\log_e(-\text{LEFT}) + i\pi)} & , \text{ when } \text{LEFT} < 0.0 \end{cases}$$

14.7.3 Branch cut

None

14.7.4 Domain

re RIGHT $\neq$ 0.0 when LEFT = 0.0

14.7.5 Range

Mathematically unbounded

14.7.6 Accuracy

- a) Maximum relative box error is the error bound that results from an implementation using functions from this International Standard and from the package `GENERIC_ELEMENTARY_FUNCTIONS` defined in ISO/IEC 11430.
- b) When `LEFT` $\neq$ 0.0 and `RIGHT` is zero, the result is one.
- c) When `LEFT` = 0.0 and `reRIGHT` > 0.0, the result is zero.
- d) When `RIGHT` is one, the result is the complex value having `LEFT` as the real part and 0.0 as the imaginary part.
- e) When `LEFT` = 1.0, the result is one.

NOTES

1 `EXP(RIGHT*LOG(LEFT))` when `LEFT` $\geq$ 0.0 and `EXP(RIGHT*(LOG(-LEFT)+i $\pi$ ))` when `LEFT` < 0.0 is an allowed implementation, but note the accuracy requirements.

2 clause 12 applies when `LEFT` = 0.0 and `reRIGHT` < 0.0.

14.8 SIN — Trigonometric sine function**14.8.1 Declaration**

`function SIN (X : COMPLEX) return COMPLEX;`

14.8.2 Definition

$$\text{SIN}(X) \approx \sin X = \sin(\text{re } X) \cdot \cosh(\text{im } X) + i \cos(\text{re } X) \cdot \sinh(\text{im } X)$$

14.8.3 Branch cut

None

14.8.4 Domain

Mathematically unbounded

14.8.5 Range

Mathematically unbounded

14.8.6 Accuracy

- a) Maximum relative component error = $11.0 \cdot \text{REAL}'\text{BASE}'\text{EPSILON}$ when $|\text{re } X|$ is less than or equal to some documented implementation-dependent threshold, which shall not be less than

$$\text{REAL}'\text{MACHINE_RADIX}^{[\text{REAL}'\text{MACHINE_MANTISSA}/2]}$$

For larger values of $|\text{re } X|$, degraded accuracy is allowed. An implementation shall document its behavior for large $|\text{re } X|$.

- b) When X is zero, the result is zero; in this case the sign of each component of the result is the same as the sign of the corresponding component of X .

NOTE — clause 12 applies when $\text{im } X > \log_e \text{REAL}'\text{SAFE_LARGE} + \log_e 2.0$, approximately.

14.9 COS — Trigonometric cosine function

14.9.1 Declaration

function COS (X : COMPLEX) **return** COMPLEX;

14.9.2 Definition

$$\text{COS}(X) \approx \cos X = \cos(\text{re } X) \cdot \cosh(\text{im } X) - i \sin(\text{re } X) \cdot \sinh(\text{im } X)$$

14.9.3 Branch cut

None

14.9.4 Domain

Mathematically unbounded

14.9.5 Range

Mathematically unbounded

14.9.6 Accuracy

- a) Maximum relative component error = $11.0 \cdot \text{REAL}'\text{BASE}'\text{EPSILON}$ when $|\text{re } X|$ is less than or equal to some documented implementation-dependent threshold, which shall not be less than

$$\text{REAL}'\text{MACHINE_RADIX}^{[\text{REAL}'\text{MACHINE_MANTISSA}/2]}$$

For larger values of $|\text{re } X|$, degraded accuracy is allowed. An implementation shall document its behavior for large $|\text{re } X|$.

- b) When X is zero, the result is one; in this case the sign of the imaginary part of the result is negative when the sign of $\text{re } X$ is the same as the sign of $\text{im } X$, and positive otherwise.

NOTE — clause 12 applies when $\text{im } X > \log_e \text{REAL}'\text{SAFE_LARGE} + \log_e 2.0$, approximately.

14.10 TAN — Trigonometric tangent function**14.10.1 Declaration**

function TAN (X : COMPLEX) return COMPLEX;

14.10.2 Definition

$$\text{TAN}(X) \approx \tan X = \frac{\sin(\text{re } X) \cdot \cos(\text{re } X)}{\cos(\text{re } X)^2 + \sinh(\text{im } X)^2} + i \frac{\sinh(\text{im } X) \cdot \cosh(\text{im } X)}{\cos(\text{re } X)^2 + \sinh(\text{im } X)^2}$$

14.10.3 Branch cut

None

14.10.4 Domain

Mathematically unbounded

14.10.5 Range

Mathematically unbounded

14.10.6 Accuracy

- a) Maximum relative component error = $35.0 \cdot \text{REAL}'\text{BASE}'\text{EPSILON}$ when $|\text{re } X|$ is less than or equal to some documented implementation-dependent threshold, which shall not be less than

$$\text{REAL}'\text{MACHINE_RADIX}^{[\text{REAL}'\text{MACHINE_MANTISSA}/2]}$$

For larger values of $|\text{re } X|$, degraded accuracy is allowed. An implementation shall document its behavior for large $|\text{re } X|$.

- b) The sign of the imaginary part of the result is the same as the sign of the imaginary part of X .
- c) When X is zero, the result is zero; in this case the sign of the real part of the result is the same as the sign of the real part of X .

14.11 COT — Trigonometric cotangent function**14.11.1 Declaration**

function COT (X : COMPLEX) return COMPLEX;

14.11.2 Definition

$$\text{COT}(X) \approx \cot X = \frac{\sin(\text{re } X) \cdot \cos(\text{re } X)}{\sin(\text{re } X)^2 + \sinh(\text{im } X)^2} - i \frac{\sinh(\text{im } X) \cdot \cosh(\text{im } X)}{\sin(\text{re } X)^2 + \sinh(\text{im } X)^2}$$

14.11.3 Branch cut

None

14.11.4 Domain

Mathematically unbounded

14.11.5 Range

Mathematically unbounded

14.11.6 Accuracy

- a) Maximum relative component error = $35.0 \cdot \text{REAL}'\text{BASE}'\text{EPSILON}$ when $|\text{re } \mathbf{X}|$ is less than or equal to some documented implementation-dependent threshold, which shall not be less than

$$\text{REAL}'\text{MACHINE_RADIX}^{|\text{REAL}'\text{MACHINE_MANTISSA}|/2}$$

For larger values of $|\text{re } \mathbf{X}|$, degraded accuracy is allowed. An implementation shall document its behavior for large $|\text{re } \mathbf{X}|$.

- b) The sign of the imaginary part of the result is the same as the sign of the imaginary part of $\mathbf{X}$.

NOTE — clause 12 applies when $\mathbf{X}$ is zero.

14.12 ARCSIN — Inverse trigonometric sine function**14.12.1 Declaration**

```
function ARCSIN (X : COMPLEX) return COMPLEX;
```

14.12.2 Definition

$$\text{ARCSIN}(\mathbf{X}) \approx \arcsin \mathbf{X} = -i \log_e(i \mathbf{X} + \sqrt{1 - \mathbf{X}^2})$$

14.12.3 Branch cutReal axis outside the open interval $(-1.0, 1.0)$ **14.12.4 Domain**

Mathematically unbounded

14.12.5 RangeImaginary part mathematically unbounded, real part in $[-\pi/2, \pi/2]$

14.12.6 Accuracy

- a) Maximum relative component error = $14.0 \cdot \text{REAL}'\text{BASE}'\text{EPSILON}$
- b) The sign of the real part of the result is the same as the sign of the real part of X .
- c) The sign of the imaginary part of the result is the same as the sign of the imaginary part of X .
- d) When X is zero, the result is zero.
- e) When X is one, the result is $\pi/2$.
- f) When X is negative one, the result is $-\pi/2$.

NOTE — $\pi/2$ and $-\pi/2$ are not safe numbers of **REAL**. Accordingly, an implementation may exceed the range limits, but only slightly; see clause 10 for a precise statement of the requirements. Similarly, when accuracy requirement e) or f) applies, an implementation may approximate the prescribed result, but only within narrow limits; see clause 11 for a precise statement of the requirements.

14.13 ARCCOS — Inverse trigonometric cosine function**14.13.1 Declaration**

```
function ARCCOS (X : COMPLEX) return COMPLEX;
```

14.13.2 Definition

$$\text{ARCCOS}(X) \approx \arccos X - i \log_e(X + i \sqrt{1 - X^2})$$

14.13.3 Branch cut

Real axis outside the open interval $(-1.0, 1.0)$

14.13.4 Domain

Mathematically unbounded

14.13.5 Range

Imaginary part mathematically unbounded, real part in $[0.0, \pi]$

14.13.6 Accuracy

- a) Maximum relative component error = $14.0 \cdot \text{REAL}'\text{BASE}'\text{EPSILON}$
- b) The sign of the imaginary part of the result is the opposite of the sign of the imaginary part of X , unless the imaginary part of the result is zero and the implementation does not exploit signed zeros.
- c) The sign of the real part of the result is positive.
- d) When X is one, the result is zero.
- e) When X is zero, the result is $\pi/2$.

- f) When X is negative one, the result is π .

NOTE — $\pi/2$ and π are not safe numbers of REAL. Accordingly, an implementation may exceed the range limits, but only slightly; see clause 10 for a precise statement of the requirements. Similarly, when accuracy requirement e) or f) applies, an implementation may approximate the prescribed result, but only within narrow limits; see clause 11 for a precise statement of the requirements.

14.14 ARCTAN — Inverse trigonometric tangent function

14.14.1 Declaration

```
function ARCTAN (X : COMPLEX) return COMPLEX;
```

14.14.2 Definition

$$\text{ARCTAN}(X) \approx \arctan X = -\frac{i}{2}(\log_e(1 + i X) - \log_e(1 - i X))$$

14.14.3 Branch cut

Imaginary axis outside the open interval $(-i, i)$

14.14.4 Domain

Mathematically unbounded

14.14.5 Range

Imaginary part mathematically unbounded, real part in $[-\pi/2, \pi/2]$

14.14.6 Accuracy

- Maximum relative component error = $14.0 \cdot \text{REAL}'\text{BASE}'\text{EPSILON}$
- The sign of the real part of the result is the same as the sign of the real part of X .
- The sign of the imaginary part of the result is the same as the sign of the imaginary part of X .
- When X is zero, the result is zero.

NOTES

1 $-\pi/2$ and $\pi/2$ are not safe numbers of REAL. Accordingly, an implementation may exceed the range limits, but only slightly; see clause 10 for a precise statement of the requirements.

2 clause 12 applies when X is $\pm i$.

14.15 ARCCOT — Inverse trigonometric cotangent function

14.15.1 Declaration

```
function ARCCOT (X : COMPLEX) return COMPLEX;
```

14.15.2 Definition

$$\text{ARCCOT}(X) \approx \text{arccot } X = \frac{i}{2} \log_e \frac{X-i}{X+i}$$

14.15.3 Branch cut

Imaginary axis in the closed interval $[-i, i]$

14.15.4 Domain

Mathematically unbounded

14.15.5 Range

Imaginary part mathematically unbounded, real part in $[0.0, \pi]$

14.15.6 Accuracy

- a) Maximum relative component error = `14.0 · REAL 'BASE' EPSILON`
- b) The sign of the imaginary part of the result is the opposite of the sign of the imaginary part of X , unless the imaginary part of the result is zero and the implementation does not exploit signed zeros.
- c) The sign of the real part of the result is positive.
- d) When X is zero, the result is $\pi/2$.

NOTES

1 π and $\pi/2$ are not safe numbers of `REAL`. Accordingly, an implementation may exceed the range limits, but only slightly; see clause 10 for a precise statement of the requirements. Similarly, when accuracy requirement d) applies, an implementation may approximate the prescribed result, but only within narrow limits; see clause 11 for a precise statement of the requirements.

2 clause 12 applies when X is $\pm i$.

14.16 SINH — Hyperbolic sine function**14.16.1 Declaration**

```
function SINH (X : COMPLEX) return COMPLEX;
```

14.16.2 Definition

$$\text{SINH}(X) \approx \sinh X = \sinh(\text{re } X) \cdot \cos(\text{im } X) + i \cosh(\text{re } X) \cdot \sin(\text{im } X)$$

14.16.3 Branch cut

None

14.16.4 Domain

Mathematically unbounded

14.16.5 Range

Mathematically unbounded

14.16.6 Accuracy

- a) Maximum relative component error = $11.0 \cdot \text{REAL}'\text{BASE}'\text{EPSILON}$ when $|\text{im } X|$ is less than or equal to some documented implementation-dependent threshold, which shall not be less than

$$\text{REAL}'\text{MACHINE_RADIX}^{\lfloor \text{REAL}'\text{MACHINE_MANTISSA}/2 \rfloor}$$

For larger values of $|\text{im } X|$, degraded accuracy is allowed. An implementation shall document its behavior for large $|\text{im } X|$.

- b) When X is zero, the result is zero; in this case the sign of each component of the result is the same as the sign of the corresponding component of X .

NOTE — clause 12 applies when $\text{re } X > \log_e \text{REAL}'\text{SAFE_LARGE} + \log_e 2.0$, approximately.

14.17 COSH — Hyperbolic cosine function**14.17.1 Declaration**

```
function COSH (X : COMPLEX) return COMPLEX;
```

14.17.2 Definition

$$\text{COSH}(X) \approx \cosh X = \cosh(\text{re } X) \cdot \cos(\text{im } X) + i \sinh(\text{re } X) \cdot \sin(\text{im } X)$$

14.17.3 Branch cut

None

14.17.4 Domain

Mathematically unbounded

14.17.5 Range

Mathematically unbounded

14.17.6 Accuracy

- a) Maximum relative component error = $11.0 \cdot \text{REAL}'\text{BASE}'\text{EPSILON}$ when $|\text{im } X|$ is less than or equal to some documented implementation-dependent threshold, which shall not be less than

$$\text{REAL}'\text{MACHINE_RADIX}^{[\text{REAL}'\text{MACHINE_MANTISSA}/2]}$$

For larger values of $|\text{im } X|$, degraded accuracy is allowed. An implementation shall document its behavior for large $|\text{im } X|$.

- b) When X is zero, the result is one; in this case the sign of the imaginary part of the result is positive when the signs of the real and imaginary parts of X agree and negative otherwise.

NOTE — clause 12 applies when $\text{re } X > \log_e \text{REAL}'\text{SAFE_LARGE} + \log_e 2.0$, approximately.

14.18 TANH — Hyperbolic tangent function**14.18.1 Declaration**

function TANH (X : COMPLEX) return COMPLEX;

14.18.2 Definition

$$\text{TANH}(X) \approx \tanh X = \frac{\sinh(\text{re } X) \cdot \cosh(\text{re } X)}{\sinh(\text{re } X)^2 + \cos(\text{im } X)^2} + i \frac{\sin(\text{im } X) \cdot \cos(\text{im } X)}{\sinh(\text{re } X)^2 + \cos(\text{im } X)^2}$$

14.18.3 Branch cut

None

14.18.4 Domain

Mathematically unbounded

14.18.5 Range

Mathematically unbounded

14.18.6 Accuracy

- a) Maximum relative component error = $35.0 \cdot \text{REAL}'\text{BASE}'\text{EPSILON}$ when $|\text{im } X|$ is less than or equal to some documented implementation-dependent threshold, which shall not be less than

$$\text{REAL}'\text{MACHINE_RADIX}^{[\text{REAL}'\text{MACHINE_MANTISSA}/2]}$$

For larger values of $|\text{im } X|$, degraded accuracy is allowed. An implementation shall document its behavior for large $|\text{im } X|$.

- b) The sign of the real part of the result is the same as the sign of the real part of X .
- c) When X is zero, the result is zero; in this case the sign of the imaginary part of the result is the same as the sign of the imaginary part of X .

14.19 COTH — Hyperbolic cotangent function**14.19.1 Declaration**

```
function COTH (X : COMPLEX) return COMPLEX;
```

14.19.2 Definition

$$\text{COTH}(X) \approx \coth X = \frac{\sinh(\text{re } X) \cdot \cosh(\text{re } X)}{\sinh(\text{re } X)^2 + \sin(\text{im } X)^2} - i \frac{\sin(\text{im } X) \cdot \cos(\text{im } X)}{\sinh(\text{re } X)^2 + \sin(\text{im } X)^2}$$

14.19.3 Branch cut

None

14.19.4 Domain

Mathematically unbounded

14.19.5 Range

Mathematically unbounded

14.19.6 Accuracy

- a) Maximum relative component error = $35.0 \cdot \text{REAL}'\text{BASE}'\text{EPSILON}$ when $|\text{im } X|$ is less than or equal to some documented implementation-dependent threshold, which shall not be less than

$$\text{REAL}'\text{MACHINE_RADIX}^{\lfloor \text{REAL}'\text{MACHINE_MANTISSA}/2 \rfloor}$$

For larger values of $|\text{im } X|$, degraded accuracy is allowed. An implementation shall document its behavior for large $|\text{im } X|$.

- b) The sign of the real part of the result is the same as the sign of the real part of X .

NOTE — clause 12 applies when X is zero.

14.20 ARCSINH — Inverse hyperbolic sine function**14.20.1 Declaration**

```
function ARCSINH (X : COMPLEX) return COMPLEX;
```

14.20.2 Definition

$$\text{ARCSINH}(X) \approx \text{arcsinh } X = \log_e(X + \sqrt{1 + X^2})$$

14.20.3 Branch cutImaginary axis outside the open interval $(-i, i)$

14.20.4 Domain

Mathematically unbounded

14.20.5 Range

Real part mathematically unbounded, imaginary part in $[-\pi/2, \pi/2]$

14.20.6 Accuracy

- a) Maximum relative component error = $14.0 \cdot \text{REAL}'\text{BASE}'\text{EPSILON}$
- b) The sign of the real part of the result is the same as the sign of the real part of X .
- c) The sign of the imaginary part of the result is the same as the sign of the imaginary part of X .
- d) When X is zero, the result is zero.

NOTE — $-\pi/2$ and $\pi/2$ are not safe numbers of **REAL**. Accordingly, an implementation may exceed the range limits, but only slightly; see clause 10 for a precise statement of the requirements.

14.21 ARCCOSH — Inverse hyperbolic cosine function**14.21.1 Declaration**

function ARCCOSH (X : **COMPLEX**) return **COMPLEX**;

14.21.2 Definition

$$\text{ARCCOSH}(X) \approx \text{arccosh } X = \log_e(X + \sqrt{X-1} \cdot \sqrt{X+1})$$

14.21.3 Branch cut

Real axis less than or equal 1.0

14.21.4 Domain

Mathematically unbounded

14.21.5 Range

Real part ≥ 0.0 and imaginary part in $[-\pi, \pi]$

14.21.6 Accuracy

- a) Maximum relative component error = $14.0 \cdot \text{REAL}'\text{BASE}'\text{EPSILON}$
- b) The sign of the imaginary part of the result is the same as the sign of the imaginary part of X .
- c) When X is one, the result is zero; in this case the sign of the real part of the result is positive.

NOTE — $-\pi$ and π are not safe numbers of **REAL**. Accordingly, an implementation may exceed the range limits, but only slightly; see clause 10 for a precise statement of the requirements.

14.22 ARCTANH — Inverse hyperbolic tangent function

14.22.1 Declaration

`function ARCTANH (X : COMPLEX) return COMPLEX;`

14.22.2 Definition

$$\text{ARCTANH}(X) \approx \text{arctanh } X = \frac{1}{2}(\log_e(1+X) - \log_e(1-X))$$

14.22.3 Branch cut

Real axis outside the open interval $(-1.0, 1.0)$

14.22.4 Domain

Mathematically unbounded

14.22.5 Range

Real part mathematically unbounded, imaginary part in $[-\pi/2, \pi/2]$

14.22.6 Accuracy

- a) Maximum relative component error = $14.0 \cdot \text{REAL}'\text{BASE}'\text{EPSILON}$
- b) The sign of the real part of the result is the same as the sign of the real part of X .
- c) The sign of the imaginary part of the result is the same as the sign of the imaginary part of X .
- d) When X is zero, the result is zero.

NOTES

1 $-\pi/2$ and $\pi/2$ are not safe numbers of **REAL**. Accordingly, an implementation may exceed the range limits, but only slightly; see clause 10 for a precise statement of the requirements.

2 clause 12 applies when X is either one or negative one.

14.23 ARCCOTH — Inverse hyperbolic cotangent function

14.23.1 Declaration

`function ARCCOTH (X : COMPLEX) return COMPLEX;`

14.23.2 Definition

$$\text{ARCCOTH}(X) \approx \text{arccoth } X = \frac{1}{2} \log_e \frac{X+1}{X-1}$$

14.23.3 Branch cut

Real axis in the closed interval $[-1.0, 1.0]$

14.23.4 Domain

Mathematically unbounded

14.23.5 Range

Real part mathematically unbounded, imaginary part in $[0.0, \pi]$

14.23.6 Accuracy

- a) Maximum relative component error = $14.0 \cdot \text{REAL}'\text{BASE}'\text{EPSILON}$
- b) The sign of the real part of the result is the same as the sign of the real part of X .
- c) The sign of the imaginary part of the result is positive.
- d) When X is zero, the result is $i\pi/2$.

NOTES

1 π and $\pi/2$ are not safe numbers of **REAL**. Accordingly, an implementation may exceed the range limits, but only slightly; see clause 10 for a precise statement of the requirements. Similarly, when accuracy requirement d) applies, an implementation may approximate the prescribed result, but only within narrow limits; see clause 11 for a precise statement of the requirements.

2 clause 12 applies when X is either one or negative one.

Annex A

(normative)

Ada specification for GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS

with ELEMENTARY_FUNCTIONS_EXCEPTIONS;

generic

```
type REAL is digits <>;
type COMPLEX is private;
type IMAGINARY is private;
```

```
with function RE (X : COMPLEX) return REAL is <>;
with function IM (X : COMPLEX) return REAL is <>;
with function IM (X : IMAGINARY) return REAL is <>;
with procedure SET_RE (X : in out COMPLEX; RE : REAL) is <>;
with procedure SET_IM (X : in out COMPLEX; IM : REAL) is <>;
with function COMPOSE_FROM_CARTESIAN (RE, IM : REAL) return COMPLEX is <>;
with function MODULUS (X : COMPLEX) return REAL is <>;
with function "-" (RIGHT : COMPLEX) return COMPLEX is <>;
with function "+" (LEFT, RIGHT : COMPLEX) return COMPLEX is <>;
with function "-" (LEFT, RIGHT : COMPLEX) return COMPLEX is <>;
with function "*" (LEFT, RIGHT : COMPLEX) return COMPLEX is <>;
with function "/" (LEFT, RIGHT : COMPLEX) return COMPLEX is <>;
with function "+" (LEFT : REAL; RIGHT : COMPLEX) return COMPLEX is <>;
with function "-" (LEFT : REAL; RIGHT : COMPLEX) return COMPLEX is <>;
with function "-" (LEFT : COMPLEX; RIGHT : REAL) return COMPLEX is <>;
with function "*" (LEFT : REAL; RIGHT : COMPLEX) return COMPLEX is <>;
with function "/" (LEFT : COMPLEX; RIGHT : REAL) return COMPLEX is <>;
```

package GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS is

```
function SQRT (X : COMPLEX) return COMPLEX;
function LOG (X : COMPLEX) return COMPLEX;
function EXP (X : COMPLEX) return COMPLEX;
function EXP (X : IMAGINARY) return COMPLEX;

function "***" (LEFT, RIGHT : COMPLEX) return COMPLEX;
function "***" (LEFT : COMPLEX; RIGHT : REAL) return COMPLEX;
function "***" (LEFT : REAL; RIGHT : COMPLEX) return COMPLEX;

function SIN (X : COMPLEX) return COMPLEX;
function COS (X : COMPLEX) return COMPLEX;
function TAN (X : COMPLEX) return COMPLEX;
function COT (X : COMPLEX) return COMPLEX;

function ARCSIN (X : COMPLEX) return COMPLEX;
function ARCCOS (X : COMPLEX) return COMPLEX;
function ARCTAN (X : COMPLEX) return COMPLEX;
function ARCCOT (X : COMPLEX) return COMPLEX;

function SINH (X : COMPLEX) return COMPLEX;
function COSH (X : COMPLEX) return COMPLEX;
function TANH (X : COMPLEX) return COMPLEX;
function COTH (X : COMPLEX) return COMPLEX;
```

```
function ARCSINH (X : COMPLEX) return COMPLEX;  
function ARCCOSH (X : COMPLEX) return COMPLEX;  
function ARCTANH (X : COMPLEX) return COMPLEX;  
function ARCCOTH (X : COMPLEX) return COMPLEX;
```

```
ARGUMENT_ERROR : exception renames ELEMENTARY_FUNCTIONS_EXCEPTIONS.ARGUMENT_ERROR;
```

```
end GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS;
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 13814:1998

Annex B

(informative)

Rationale

B.1 Introduction and motivation

This International Standard is the fourth in a series of secondary standards for mathematical packages for Ada 83, the first three being ISO/IEC 11430 [12], ISO/IEC 11729 [13] and ISO/IEC 13813 [14]. This International Standard adds to the functionality of complex types as defined in the generic package **GENERIC_COMPLEX_TYPES** of ISO/IEC 13813 by providing complex analogs of the real elementary functions defined in the generic package **GENERIC_ELEMENTARY_FUNCTIONS** of ISO/IEC 11430.

Several application areas depend on the use of complex arithmetic and the complex elementary functions. Complex fast Fourier transforms are used, for example, in conjunction with radar, sonar and electro-optical sensors; conformal mapping uses complex arithmetic in fluid-flow problems, including the analysis of velocity fields around radomes, torpedo noses and airfoils; and a/c circuit analysis is classically modeled in terms of complex exponentials.

B.2 History

This International Standard was developed concurrently with ISO/IEC 13813, several years after the other two standards in the series. The bulk of the design work was performed by the SIGAda Numerics Working Group, reporting to the WG9 Numerics Rapporteur Group; contributions were also made by the Ada-Europe Numerics Working Group.

Similar capabilities have been incorporated into the Numerics Annex of Ada 95 [11]. Originally, the Ada 95 versions of the complex arithmetic packages were patterned after working drafts of the Ada 83 secondary standards for complex arithmetic, except that types and operations for vectors and matrices of complex components were never proposed for Ada 95. The complex types package in Ada 95 underwent a significant redesign as a result of the much larger number of people monitoring and contributing to its design. The two secondary standards for complex arithmetic in Ada 83 benefitted from the redesign of their Ada 95 analogs and are in general agreement with the latter.

B.3 Packaging considerations

Since the complex type is defined elsewhere (e.g., by an instantiation of **GENERIC_COMPLEX_TYPES** or by some alternative thereto), **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS** must be generic, so that it can “import” the complex type to be used in the parameter and result profiles of the functions it defines. It is also necessary to import the floating-point type used in the definition of the complex type (i.e., for its components), as well as the pure-imaginary type defined in terms of that floating-point type. The generic formal parameters for these imported types are called **COMPLEX**, **REAL** and **IMAGINARY**, respectively; they define a generic formal private type, a generic formal floating-point type and another generic formal private type. The order of the parameters is **REAL**, **COMPLEX**, and **IMAGINARY**. A typical instantiation of **GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS** will be as follows:

```
with GENERIC_COMPLEX_TYPES;
with GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS;
procedure APPLICATION is
  type REAL is digits ...;
  package COMPLEX_TYPES is new GENERIC_COMPLEX_TYPES (REAL);
  use COMPLEX_TYPES;
  package COMPLEX_ELEMENTARY_FUNCTIONS is
    new GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS (REAL, COMPLEX, IMAGINARY);
  use COMPLEX_ELEMENTARY_FUNCTIONS;
  ...
begin
  ...
end APPLICATION;
```