

Empty Intervals

G. William Walster

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
1 (800) 786.7638
1.512.434.1511

Copyright 2002 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, EJB, EmbeddedJava, Enterprise JavaBeans, Forte, iPlanet, Java, JavaBeans, Java Blend, JavaServer Pages, JDBC, JDK, J2EE, J2SE, and Solaris trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-3(a).

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2002 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, Californie 94303 Etats-Unis. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées des systèmes Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, EJB, EmbeddedJava, Enterprise JavaBeans, Forte, iPlanet, Java, JavaBeans, Java Blend, JavaServer Pages, JDBC, JDK, J2EE, J2SE, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REpondre A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.

Contents

Introduction	1
Distinguishing Between Empty Intervals and Errors	2
Interpreting Empty Interval Results	5
Conclusion	8
Appendix A. References	9

Empty Intervals

Introduction

The “Simple” closed interval arithmetic system is defined in [?]. For an arithmetic system to be closed, any arithmetic operation on members of the system must produce members of the system. For the interval arithmetic system to be closed, the interval $[-\infty, +\infty]$ must be a member. All members of the system must be closed sub-intervals of the set of extended real numbers, $\mathbb{R}^* = \mathbb{R} \cup \{-\infty, +\infty\} = [-\infty, +\infty]$, referred to as the *entire interval*. To include all arithmetic operator-operand combinations, interval results are required that do not exist for points. These include division by zero and indeterminate forms involving combinations of zero and infinity, such as $0/0$, ∞/∞ , and $\infty - \infty$.

Any returned interval must be an enclosure of the set of all possible values that the operation or expression can produce. The smallest set of all possible results is termed the *containment set* of the operation or expression. Enclosing the containment set is the *containment constraint* of interval arithmetic. Any returned interval must satisfy this containment constraint. Given the containment constraint is satisfied, narrow returned intervals are preferred.

To return interval results that are as narrow as possible, the empty interval (which is the empty set) is supported. Whenever possible, features of IEEE floating-point arithmetic are used to facilitate implementing new interval results that do not have

point counterparts, see [?] and [?]. The empty interval naturally arises from the intersection of two disjoint intervals. The empty interval is also returned when interval arguments are bounded away from the domain of the enclosed point function or relation. In this note, justification is provided for returning the empty interval in this case. For an overview of this and other innovations implemented in the Sun Microsystems Inc. Forte™ Developer 6 Fortran 95 Compiler, see [?].

Distinguishing Between Empty Intervals and Errors

In [?], the mathematical foundation is established for returning the empty set when a function or relation, hereafter referred to as an expression, is evaluated at a point that is strictly outside the expression's domain of definition. To avoid containment failures, it is also proved that the topological closure of a relation at points on the boundary of its domain must be included in containment sets. Consequently, no restriction exists on the domain of any extended-real-interval expression. The underlying set-based \mathbb{R}^* -system is closed. When an expression is evaluated at a point that is bounded away from the expression's domain, the natural result is the empty set.

This result has two important consequences: first, if the interval, X , is strictly outside the domain of the expression, f , then the expression's interval evaluation, $f(X)$, is the empty interval. Second, if $X \cap \overline{D}_f \neq \emptyset$ (that is, if X contains at least one point in the closure¹ of domain of f), then any part of X that is outside the closure of f 's domain, \overline{D}_f , is ignored. Thus, for example, $\sqrt{[-1, 1]} = [0, 1]$, and $\sqrt{[-2, -1]} = \emptyset$. In [?], the containment set of an expression is proved to be the expression's topological closure². Therefore, points on the boundary of an expression's domain must be included to prevent containment failures.

Different alternative interpretations are possible for points strictly outside an expression's domain. One alternative system, termed the “*NaRI*-system” treats any point outside a real function's domain of definition to be either an error or a point in the complex plane. In this framework, because some points in $\sqrt{[-1, 1]}$ and

¹ The *closure* of a set is the set, augmented by all limiting values (called accumulation points) of sequences of points in the set.

² An expression's topological closure is augmented by the limiting expression values for argument sequences whose limits equal the given expression argument. Sequence convergence is defined topologically.

Result of:	\mathbb{R}^* -System	
	Extended Real Interval	NaI
$\sqrt{BX - AX} = \sqrt{[-4, 1]}$	$[0, 1]$	C^*
$\sqrt{X(B - A)} = \sqrt{[-2, -1]}$	\mathbb{R}^*	C^*
$\sqrt{AX - BX} = \sqrt{[-1, 4]}$	$[0, 2]$	C^*
$\sqrt{X(A - B)} = \sqrt{[1, 2]}$	$[1, \sqrt{2}]$	$[1, \sqrt{2}]$

TABLE 1 Comparison of *Extended Real Interval* and *Not a Real Interval* Systems.

$\sqrt{[-2, -1]}$ have a complex interpretation, a pair of quiet NaNs, say NaN_{C^*} , could be used to represent the entire complex plane. This could be done, in a real system. In a complex system, a complex result could be returned. An additional pair of quiet NaNs, NaN_{NaI} , could be used to signal an undefined outcome, that does not have a complex interpretation. In the NaI -system:

$$\text{cset}(f, X_0) = \begin{cases} \bar{f}(X_0) & \text{if } X_0 \subseteq \bar{D}_f \\ C^* & \text{if there exists an } x_0 \in X_0 \text{ such that} \\ & \bar{f}(x_0) \cap (\{C^*\} - \{\mathbb{R}^*\}) \neq \emptyset \\ NaI & \text{otherwise.} \end{cases}$$

C^* would be returned anytime there is a complex interpretation of a function result. When no real or complex interpretation exists, NaI (Not an Interval) would be returned. For the purpose of the present development, it is sufficient to introduce the mnemonic “ $NaRI$ ” for the “Not-a-Real-Interval” System to cover the union of both the C^* and NaI cases.

The \mathbb{R}^* - and $NaRI$ -systems are motivated by different views of a compiler’s obligation to produce code in the presence of a *possible* algorithm or coding error. Consider the computation of $\sqrt{AX - BX}$ and $\sqrt{BX - AX}$; with $A = [3, 3]$, $B = [2, 2]$, and $X = [1, 2]$. $AX - BX$ and $BX - AX$ evaluate to $[-1, 4]$ and $[-4, 1]$, respectively. However, $X(A - B)$ and $X(B - A)$ evaluate to the minimum width results: $[1, 2]$ and $[-2, -1]$, respectively. Suppose that an algorithm or programming mistake leads to $BX - AX$ or $X(B - A)$ being coded instead of $AX - BX$ or $X(A - B)$. The following table shows the resulting values under the \mathbb{R}^* and $NaRI$ systems.

Thus, three events can cause an interval argument to be totally or partially outside a function’s domain of definition: an algorithm error, a coding error, or dependence.

With no algorithm or coding error, an interval argument can still be strictly outside an expression's domain of definition. Any interval that is partially outside an expression's domain can be partitioned into two sub-intervals, one of which is strictly outside.

Suppose, for example, that an interval enclosure of the function $f(x) = \sqrt{g(x)}$ is computed using an unsharp interval enclosure of $g(x)$. Let X_0 be the interval over which a bound on the range of f is required. Let $Y_0 = g(X_0)$ be the unsharp enclosure of $\text{cset}(g, X_0)$. If Y_0 is partitioned into $Y_{01} \cup Y_{02}$ with $\text{sup}(Y_{01}) = \text{inf}(Y_{02})$, then it is possible for $\text{sup}(Y_{01})$ to be strictly negative. As a consequence of splitting Y_0 and computing $\sqrt{Y_{01}} \cup \sqrt{Y_{02}}$, the square root of a strictly negative interval can be encountered. The resulting square root of Y_{01} does not imply that an analysis or coding mistake has been made. Therefore, excess interval width, alone, can cause an argument to be strictly or partially outside an expression's domain of definition.

Conversely, neither algorithm nor code correctness are assured when an argument (interval or otherwise) is contained in a expression's domain. There is simply no logical connection between argument values and evidence of an error. Moreover, a heavy price must be paid in defensive code to avoid unnecessary error conditions if any argument partially outside a function's domain is treated as an error. For example, in place of $\text{SQRT}(X)$ the following will need to be written:

$$\text{SQRT}(X \text{ .IX. } [0, \text{INF}]);$$

where in §95, .IX. is the interval intersection operator, see [?].

The negative consequences to programmers from the *NaRI*-system, occur because of the operational difference in behavior between \emptyset and *NaRI*. In arithmetic operations on either \emptyset or *NaRI* and a real interval, X , both \emptyset and *NaRI* propagate. The difference stems from behavior of the interval hull and intersection operators. In the following illustrative examples, let $\underline{\cup}$ and \cap (.IH. and .IX. in §95) denote the interval hull and intersection operators, respectively. The interval hull is the smallest interval that contains the union of two intervals. Therefore, the interval hull of an empty interval and any interval, say X , is simply X . The interval, X , is an element of the set of all possible extended real intervals, denoted, \mathbb{IR}^* . Then, for the interval hull operator:

$$\emptyset \underline{\cup} X \equiv X, \tag{1a}$$

$$C^* \underline{\cup} X \equiv C^*, \tag{1b}$$

$$\text{NaI} \underline{\cup} X \equiv \text{NaI}. \tag{1c}$$

In contrast, for the interval intersection operator:

$$\emptyset \cap X \equiv \emptyset, \quad (2a)$$

$$C^* \cap X \equiv X, \quad (2b)$$

$$NaI \cap X \equiv NaI. \quad (2c)$$

The set-relational operators including set-equality (*.SEQ.*, *.SLE.*, and *.SGE.*) also have truth values that depend on the difference between \emptyset and C^* or NaI . For example:

$$\emptyset \text{ .SEQ. } \emptyset \equiv \text{true}, \text{ and} \quad (3a)$$

$$C^* \text{ .SEQ. } C^* \equiv \text{true}, \text{ whereas} \quad (3b)$$

$$NaI \text{ .SEQ. } NaI \equiv \text{false}. \quad (3c)$$

In every other way, C^* , NaI and \emptyset propagate in the same way.

The problem with returning C^* or NaI is that this implies an error of some kind has taken place. As shown above, this is not necessarily the case. Conversely, an empty outcome, as with any result, may be the product of an error. With an empty outcome, context can be used to identify and/or preclude some of the possible error causes.

Interpreting Empty Interval Results

Interpreting empty interval results requires care, because depending on the context, an empty result may or may not be an error indicator.

Intervals have a dual numeric and set interpretation. In a set context, empty results and operations on empty intervals are natural. For example, the empty interval in a set context is a valid outcome of the intersection (*.IX.* in §95) of two disjoint intervals.

Arithmetic operations can also produce empty intervals. When they do, it may or may not indicate a programming or algorithm error has occurred. For example, as seen above in Table 1, if an interval is computed that must be non-negative, and it turns out to be strictly negative, say $[-2, -1]$, this is an error because a containment failure has occurred. In §95, this error can be detected in a number of ways. For example an explicit test can be made using the certainly-less-than relational operator (*.CLT.* in §95):

`IF(X .CLT. 0)`

See [?] for documentation of all interval-specific relational operators.

Alternatively, if an intrinsic function with a non-negative domain is given a strictly negative interval argument, an empty interval result may be an error indicator. For example, if X is the complete result of a calculation that must be non-negative, X may contain some negative values, because of rounding and/or dependence. See the examples in Table 1, above. However, X must contain some non-negative values. Therefore if X is strictly negative, `SQRT(X)` returning `[empty]` is an error indicator. However, if X is only partially negative and X is partitioned into two sub-intervals, $X1$ and $X2$, with $X1$ strictly negative,

`SQRT(X1)` returning `[empty]`

is not an error. If $X1$ is strictly negative, the empty interval is the required result for the following expression to return a valid interval that does not violate the containment constraint:

`SQRT(X1) .IH. SQRT(X2)`

Context is necessary to identify situations where an empty result is an error indicator.

Empty interval results propagate through arithmetic operations, and therefore will not be lost. However, there are four ways an erroneous empty intermediate result can be lost and therefore go undetected:

- An empty result is an operand of an interval hull operation with a non-empty operand. This can be prevented by checking for interval hull operands that may be unexpectedly empty.
- An empty result is an operand of an intersection operation, the result of which, is legitimately empty. To prevent losing an erroneous empty result, check to make sure arguments of the intersection operator are not spuriously empty.
- Dead code. (There is no path to dead code.) To prevent losing an empty result, check for, and eliminate, dead code.

- An empty result is part of an incomplete set of relational tests. Complete sequences of relational tests can be used to prevent the loss of an empty interval. For example, the code in Example 1 can hide the fact that B is empty. If A is empty, X must be also from the assignment in line 2. However, if B is empty, X is set to $A * 2$ in line 4. If B is erroneously empty and never used in subsequent code, the error that caused B to be empty will not be detected.

Example 1

```

IF(A .CLT .B)THEN
X = A                ! line 2
ELSE
X = A**2            ! line 4
ENDIF

```

Affirmative relational tests (any order relation other than certainly- or possibly-not-equal) are false if either operand is empty. To prevent empty results from being hidden by incomplete sequences of relational tests, either check if A or B is empty before executing the code in Example 1, or use a complete set of affirmative relational tests, as shown in Example 2.

Example 2

```

IF(A .CLT .B)THEN
X = A
ELSEIF(A .PGE .B)THEN
X = A**2
ELSE
!code to deal with A or B being empty
ENDIF

```

Coding complete sequences of relational test is facilitated with the full set of certainly-, possibly-, and set-relational operators. See [?].

Compiler support can be provided to alert programmers when each of the four empty-hiding mechanisms may exist. For example:

- flag any interval hull operator containing an operand that results from an arithmetic operation;
- flag any incomplete relational test sequence;

- flag any intersection operator containing an operand resulting from an arithmetic operation; and,
- identify dead code.

These mechanisms have not yet been implemented in §95. They remain quality of implementation opportunities.

Conclusion

The concepts of Not-an-Interval, *NaI*, Not-a-Real-Interval, *NaRI*, and the entire extended complex plane, C^* , are neither required nor desired in the implementation of an extended real interval system. In a real interval system, returning anything but the empty interval is undesirable because without context it is not possible to know if C^* or *NaI* is the correct value to return when an argument is strictly outside a real expression's domain. There is no need to enclose complex values in a real system and it is always possible to encounter intervals that are either partially or totally outside the domain of expressions for which interval enclosures are computed.

References

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303

1 (800) 786.7638
1.512.434.1511

<http://www.sun.com>

April 2002