Formalization and Analysis of the Separation Minima for the North Atlantic Region: Complete Specification and Analysis Results

Nancy A. Day, University of British Columbia
Jeffrey J. Joyce and Gerry Pelletier, Hughes International Airspace Management Systems

Technical Report 97-12
Department of Computer Science
University of British Columbia
28 Oct 1997

Abstract

This report describes work to formalize and validate a specification of the separation minima for aircraft in the North Atlantic (NAT) region completed by researchers at the University of British Columbia in collaboration with Hughes International Airspace Management Systems. Our formal representation of these separation minima is given in a mixture of a tabular style of specification and textual predicate logic. We analyzed the tables for completeness, consistency and symmetry. This report includes the full specification and complete analysis results.

Contents

1 Introduction

This report describes work to formalize and validate a specification of the separation minima for aircraft in the North Atlantic (NAT) region completed at the University of British Columbia in collaboration with Hughes International Airspace Management Systems. Our formalization is based on a description provided in a source document entitled "Application of Separation Minima for the NAT Region" (3rd edition, effective December 1992) published by Transport Canada on behalf of the ICAO North Atlantic Systems Planning Group. ICAO is the International Civil Aviation Organization with headquarters in Montreal, Canada. Related pseudo code interpretation of this material was also considered. This additional documentation was developed by the COMAG (Communications and ATM Automation Group), now named the CADAG (Communication, Automation and Data Link Applications Group).

The separation minima provides guidance to air traffic controllers managing the region of oceanic airspace between Europe and North America. It is also used as the basis for the development of software based systems that support the management of air traffic in the NAT region. For example, it would be used during the planning of a flight from New York to London to check whether the route is free from separation conflicts with other aircraft expected to be in the NAT region at the same time.

The source document for our formalization is an informal specification that has been scrutinized by the NATSPG (NAT Systems Planning Group) members who are air traffic control (ATC) specialists from the NAT countries, and most of them maintain and use automated systems that implement these rules.

Our formalization and analysis effort directly involved an ATC domain expert (not just formal methods experts) in the process of developing and analyzing a formal representation of this complex real description. The two objectives for formalizing these separation rules were:

- to present them in an understandable, unambiguous format, and
- to detect potential incompleteness or inconsistencies.

Our formal representation of these separation minima is given in a mixture of a tabular style of specification and a textual predicate logic notation called S [JDD94]. This effort required itemizing the variety of English phrases used to describe various conditions to yield a "dictionary" of primitives. These primitives are used in tables to specify the combinations of conditions which result in particular outcomes. It is possible to analyze this formal representation so we are able to determine the scenarios that are either not covered by a table, or are covered by the default case (completeness analysis). We also examined the consistency and symmetry of the tables.

In this project, considerable emphasis was placed on the readability of the specification. Consequently, tools were developed to create an on-line, top-down presentation of the specification with cross-referencing links. This technical report is a printout from a web browser. However the formal specification and analysis results do not necessarily have to be given in this format.

This report presents the complete specification along with an introduction to the formal description technique used for this specification. The full results of the analysis are also included. Details of the technique used to carry out the analysis can be found in [DJP97].

2 Specification Notation and Analysis

This document uses a formal description technique to specify the separation minima for the NAT region. The description is "formal" in the sense that it can be parsed by a software tool into a mathematical representation based on formal logic. Although inherently mathematical, considerable effort has been made in the use of this formal description technique to make the specification understandable to as wide an audience as possible.

The ability to parse this separation minima into a mathematical representation is valuable for several reasons. For instance, software tools based on formal logic can be used to perform a variety of validation checks on the mathematical representation. It may also be possible to use this mathematical representation as input to software tools which automatically generate software components for a software system based on the NAT separation minima.

In this section, we describe the notation used in this specification and briefly introduce the analysis carried out.

2.1 Specification Notation

The formal specification of the separation minima appears in this document as a combination of text such as,

```
WithinOppDirNoLongSepPeriod(A:flight,B:flight,t:time) :=
   let timePeriod := OppDirNoLongSepPeriod(A,B) in
   (StartTime(timePeriod) <= t) AND (t <= EndTime(timePeriod));
```

and tables such as:

Table 1: VerticalSeparation

		1	2	3	4	Default
1	**FlightLevel (A)**	__ <= 280	.	__ > 450	__ > 450	
2	**FlightLevel (B)**	.	__ <= 280	__ > 450	__ > 450	
3	**IsSupersonic (A)**	.	.	__=T	.	
4	**IsSupersonic (B)**	.	.	.	__=T	
	VerticalSeparationRequired (A,B)	1000	1000	4000	4000	2000

The textual parts of the formal specification of the separation minima found in Appendix A are easily distinguished in this document from unformalized text by the use of "typewriter font" for the formalized text, e.g.,

`WithinOppDirNoLongSepPeriod(A:flight,B:flight,t:time) := ...`

The formalized text is written in a formal notation called "S". Although based on formal logic, much of the syntax of this notation is similar to that of well-known programming languages, e.g., "if ... then ... else ...".

The tabular parts of the formal specification of the separation minima are also easy to recognize by the graphical presentation of these tables in this document. The row entries within these tables is based on the syntax of S.

This section gives a brief description of the tabular style of specification. Appendix C provides an description of the "S" notation focusing on the S constructs used in this document.

The formal specification of the separation minima is presented in this document as a sequence of declarations and definitions. Many elements of the formal specification are defined hierarchically in terms of other formally specified elements. For instance, the "top- level" element of this hierarchy is the predicate "AreSeparated"; it is defined in terms of a number of predicates and functions such as "VerticalSeparationRequired", "LatitudeEquivalent", "LateralSeparation RequiredInDegrees", and "LateralSeparation RequiredInMiles". The order of the declarations and definitions for these elements, as they appear in the specification, is approximately "top down" beginning with top level concepts and refining these successively into lower level concepts. Figure 1 in Appendix A shows the hierarchy of definitions for our formal specification of the NAT separation minima.

The bottom (or "leaf") level of definitions in this figure are defined in terms of primitive elements. These primitive elements include objects such as a "flight". They also include terms which correspond to attributes or properties of objects as well as relationships between objects. An example of a primitive term is the "FlightLevel" of a flight. These terms are declared rather than defined. As primitives, they identify the lowest level of abstraction used to specify the NAT separation minima. It can be assumed that the meaning of these primitives is obvious to the intended audience of this specification, namely, individuals familiar with aircraft separation.

2.2 A Brief Introduction to Tabular Style Specification

The tabular style of specification used in this document is quite similar to conventional decision tables used in semi-formal specification methodologies such as Structured Analysis as described by DeMarco [DeM79]. Similar tables are also used in other engineering disciplines such as digital circuit design. A form of decision tables called AND/OR tables were used extensively in the TCAS II (Traffic Collision Avoidance System) formal specification [LHHR94] adopted by the US FAA. Parnas has also advocated the use of tables to represent relations [P92].

A predicate table specifies a condition which is true or false depending on the values of other conditions. Each column of a table represents a set of these subconditions that, when taken together, describe one case. For example, the following table,

Table 2: P1

		1	**2**	**3**	**4**
1	**a**	__=T	__=T	__=F	__=F
2	**b**	__=T	__=F	__=T	__=F
	P1 (a, b)	T	T	F	F

defines a predicate "P1" in terms of two input conditions, "a" and "b". The constants "T" and "F" are used to represent "true" and "false" respectively. An entry in a row such as "__ = T" represents the condition resulting from filling in the blank with the label of the row (an input condition). For example, the entry in row 2, column 3 is "b= T". Each of the four columns corresponds to a separate case. For instance, column 2 specifies that the predicate "P1" is true if condition "a" is true and "b" is false. The value of "P1" for this particular combination of input conditions is indicated by the "T" in the last row of column 2.

The tabular specification of a condition in this manner is an alternative to a textual representation. For instance, an equivalent representation of the condition "P1" using S notation is:

```
P1 := if ((a = T) AND (b = T)) then T
        else if ((a = T) AND (b = F)) then T
        else if ((a = F) AND (b = T)) then F
        else F;
```

2.2.1 Reducing the number of columns in a table

The above example table is "complete" in the sense that every possible combination of the values of the input conditions is represented explicitly by one of the columns in the table. In this case, there are just four possible combinations. However, for more complex tables it is not practical to write out every possible combination of input conditions. Fortunately, the specification notation used in this document provides several mechanisms for reducing the size of tables without changing their meaning.

One such mechanism is the use of "." in a cell of a table which means "don't care" - i.e., the input condition for this row can be any value. This mechanism often allows two or more columns of a table to be merged into one column. For example, Table 2 given above for "P1" can be simplified as follows without changing its meaning:

Table 3: Revised Specification of P1

		1	**2**	**3**
1	**a**	__ = T	__ = F	__ = F
2	**b**	.	__ = T	__ = F
	P1 (a, b)	T	F	F

Column 1 of Table 3 corresponds to Columns 1 and 2 of Table 2. It is possible to merge these columns of Table 2 because the only difference between them is the value of "b".

Another mechanism used in this specification document to reduce the size of tables is the addition of an extra column at the right hand side of the table which specifies the default value of the predicate. This is the value of the predicate if none of the other columns match the table inputs. For example, an even simpler specification of the condition "P1" is given by the following table:

Table 4: Second Revised Specification of P1

		1	**Default**
1	**a**	__ = T	
2	**b**	.	
	P1(a,b)	T	F

2.2.2 Reducing the number of rows in a table

It is also possible to reduce the size of table without changing its meaning by reducing the number of rows. For example, the table,

Table 5: P2

		1	2	Default
1	**x <= 100**	__=T	__=F	
2	**100 < x**	__=F	__=T	
3	**a**	__=F	__=T	
	P2 (x,a)	T	T	F

can be reduced in size by taking advantage of the fact that Rows 1 and 2 are related. In particular, the value in each cell of Row 1 is the logical opposite of the corresponding cell in Row 2 because the conditions labelling the rows are mutually exclusive. These two rows can be merged by using parameterized expressions in a single row other than just "__ = T" or "__ = F". These two rows can be merged by using parameterized expression in the cells of a single row where the parameter corresponds to the value of the input variable "x":

Table 6: Revised Specification of P2

		1	2	Default
1	**x**	__ <= 100	100 < __	
2	**a**	__ = F	__ = T	
	P2 (x, a)	T	T	F

Each cell of the first row of the above table (i.e., the row labelled by "x" on the left hand side of the table) contains either "_ <= 100" or "100 < _". The blank, i.e., "_" stands for the value of the input variable identified at the left hand side of the row, i.e., "x".

The textual parts of the tables which appear in this document are based on the S notation. The only extension to the S syntax used within the tables is the use of "_" to stand for the value of the input variable.

2.2.3 Using Tables to Specify Functions

So far, this section has described how tables are used to describe predicates which are functions that return only true or false. Tables can also be used to specify functions. This is simply a matter of specifying values other than "T" or "F" at the bottom of the table. For instance, the table,

Table 7: F1

		1	2	3	Default
1	**x**	__ <= 100	100 < __	100 < __	
2	**y**	__ < 55	__ = 55	55 < __	
3	**a**	__=T	.	__=F	
	F1(x,y,a)	20	40	60	30

specifies a function "F1" that, for instance, evaluates to "40" when "x" is greater than "100" and "y" is exactly "55". Notice that the input variables are given as parameters to the function specification unless they are global constants.

The equivalent textual representation for "F1" is :

```
F1(x,y,a) :=
        if ((x <= 100) AND (y < 55) AND (a = T)) then 20
        else if ((100 < x) AND (y = 55)) then 40
        else if ((100 < x) AND (55 < y) AND (a = F)) then 60
        else 30;
```

2.3 AllOf and AtLeastOneOf Conditions

Expressions of the form, "AllOf [A;B] P" are frequently used in the formal specification of the separation minima. An expression of this form denotes the condition that both A and B satisfy the predicate P. Hence, "AllOf [A;B] P" is equivalent in meaning to "P (A) AND P (B)". The "AllOf" syntax is used to make the label of a row in a table more concise than its expanded form especially when the name of the predicate is lengthy, for example,

"AllOf [A;B] HavePartOfRouteINMNPSAirspace "

instead of

"HavePartOfRouteINMNPSAirspace (A) AND
HavePartOfRouteINMNPSAirspace (B)"

Similarly the expression "AtLeastOneOf [A;B] P" is equivalent to "P (A) OR P (B)".

2.4 Environmental Assumptions

There are various relationships between the primitive terms in the specification. To both document these relationships and take advantage of them in analysis, they are noted formally as assumptions about the environment (or domain) of the separation minima specification. For example, we documented the constraint that a flight could not satisfy both of the predicates "IsLevel" and "InCruiseClimb" at the same time. This constraint is expressed directly in S notation by the following assertion :

forall (A:flight). NOT (IsLevel A AND InCruiseClimb A);

These assumptions helped reduce the results of the analysis by eliminating some impossible scenarios. They are specified using predicate logic because each assumption is fairly simple.

2.5 Validation

2.5.1 Completeness Checking

A number of routine checks may be performed to validate a tabular specification. One such form of validation is completeness checking.

It is useful to check whether a tabular specification is "complete" in the sense that any possible combination of inputs given by the row entries corresponds to at least one of the columns in the table. For instance, the table,

Table 8: P3

		1	**2**	**3**
1	**a**	__=T	__=T	__=F
2	**b**	__=T	__=F	__=T
	P3 (a, b)	T	T	T

is not complete in the sense that it does not include the case when "a" and "b" are both false. The table,

Table 9: F2

		1	**2**	**3**
1	**x**	__ < 100	100 < __	100 < __
2	**y**	__ < 55	__ = 55	55 < __
3	**a**	__=T	.	__=F
	F2(x,y,a)	20	40	60

is also incomplete because it omits the cases when "x" is equal to "100".

From a logical perspective, it is not necessarily an error if a table is found to be incomplete. It is sometimes desirable to only partially specify a condition or function. However, it would be common in a specification methodology to disallow partial specifications -- which means, in this case, to require every table to be complete.

A table is guaranteed to be complete if it contains a "Default" column. However, it is possible that more cases than intended fall into the "Default" column. When a "Default" column is used in a tabular specification, it is desirable to enumerate the cases included in that column as part of the validation effort.

2.5.2 Consistency Checking

Another form of analysis is consistency checking. For instance, the table,

Table 10: F3

		1	**2**	**3**
1	**x**	__ < 100	100 < __	100 < __
2	**y**	__ < 55	55 < __	55 < __
3	**a**	__=T	.	__=F
	F3(X,Y,a)	20	40	60

is not consistent because it specifies two different values, "40" and "60" for the same case, namely, when "x" is greater than 100, "y" is greater than 55 and "a" is false.

Note that a predicate table can never be inconsistent because all of its columns have the same return value for the function, i.e. "T".

2.5.3 Symmetry Checking

The last validation check that is described here is symmetry checking. All of the tables of the separation minima describe functions over two inputs, namely two flights. Symmetry checking attempts to ensure that these functions do not rely on the order that flights are given as parameters to the function, i.e., for every function "F", "F(flightA,flightB) = F(flightB,flightA)".

2.6 Tool Assisted Validation

Software tools may be used to perform these validation checks on tabular style specifications used in this document. One such tool is called "fusion" which was developed by the first author and used in this project. Others have developed tools for other notations to carry out similar analysis [HC95][HJL96][HL96]. (Note other commerical tools may exist called fusion. Ours is a prototype academic tool.)

Initially fusion typechecks the specification to ensure that the terms are all used consistently and that every object refered to is defined or declared. Then fusion can check the completeness, consistency and symmetry of individual tables.

Completeness checking returns a list of the cases covered by the default column, or those not covered if there is no default column. The tool attempts to shorten the list by determining how output cases can be combined through the use of "don't care" (DC) values. This list of cases can be reviewed manually to determine if these cases are intended to have the default value. While this involves manual effort, the processing performed by fusion streamlines the review process and potentially improves its thoroughness.

Consistency checking returns pairs of columns which return different values for the function but overlap in the set of input values that can satisfy the column. The tool also returns the combination of input values that can satisfy both columns.

If it can not fully determine that it is symmetric, symmetry checking returns a list of conditions which must be satisfied for the table to be symmetric. The addition of some environmental constraints about the symmetry of primitive terms helps this analysis.

3 Analysis Results

The formal specification of the separation minima includes 15 tables which we can analyze for completeness, consistency and symmetry. The results for each of these checks are summarized in the following table.

Table 11: Summary of Analysis Results

Table	# of Rows	# of Cols	Type of Table, Default Col	Completeness (# of missing or default cases)	Consistency (# of overlapping cases)	Definitely Symmetric ?
VerticalSeparationRequired	4	4	F,D	4	0	YES
"LateralSeparation RequiredInDegrees"	8	4	F,D	16	4	YES
"LateralSeparation RequiredInMiles"	8	4	F,D	16	4	YES
LatitudeEquivalent	5	6	P	17	0	NO
LongSameDirSepRequired	2	2	F,D	1	0	NO
"OppDir NoLongSepPeriod"	2	2	F,D	1	0	NO
"ssOppDir NoLongSepPeriod"	1	2	F	0	0	NO
ssSameDirLongSep	4	2	F,D	3	0	YES
ssSubcondition	4	2	P	4	0	YES
"turbojetSameDir LongSep"	2	4	F	0	0	NO
"turbojetOppDir NoLongSepPeriod"	2	4	F	0	0	NO
"MNPSSameDir LongSep"	3	5	F,D	3	0	NO
WATRSCondition	6	2	P	8	0	YES
"genSameDir LongSep"	6	3	F,D	5	0	NO
otherSameDirLongSep	3	2	F,D	2	1	YES

"F" means function table. "P" means predicate table. "D" means there is a default column in the function table. The default column of predicate tables is always F for false. The number of columns does not include the default column in this count.

3.1 Completeness Analysis

Completeness analysis revealed the cases that are covered by the default column of function tables and those cases that return false in the predicate tables. All the function tables that did not have default columns were complete.

For some tables there are a great number of cases covered by the default column but review by our domain expert did not find any errors in the specification from these results.

3.2 Consistency Analysis

The results of analyzing the separation minima revealed that three tables are inconsistent. After consulting the official specification (i.e., not the pseudo code representation), our domain expert concluded that these are cases where the specification is ambiguous.

The table "otherSameDirLongSep" (Appendix A, Table 25) specifies the number of minutes that must exist between two aircraft (that are not both turbojet or both supersonic) flying in the same direction for them to be considered longitudinally separated. The checker identified that, for the case where two aircraft have reported over a common navigation point, are on the same or diverging tracks, and are both on a particular set of routes that have special criteria, the specification is ambiguous as to whether there should be 15 or 20 minutes of separation between them.

The other two tables with inconsistencies describe requirements for lateral separation. These two tables, "LateralSeparation RequiredInDegrees" and "LateralSeparation RequiredInMiles" are the same except that the first returns a value in degrees of latitude and the second returns a value in miles. These tables have eight rows and four columns. The inconsistent cases involve special provisions for particular routes that overlap with the more general criteria. The results clearly reveal cases in the official specification that are ambiguous as to the amount of lateral separation required between aircraft.

3.3 Symmetry Analysis

Symmetry analysis did not reveal any non-symmetric tables. It did highlight information about the primitive terms which might not be known by an implementor of the separation minima in software. For example, to make this analysis more accurate, we added environmental constraints such as:

```
forall (A:flight) (B:flight).
   ReportedOverCommonPoint(A,B) =
      ReportedOverCommonPoint(B,A);
```

When symmetry analysis can not completely determine if a table is symmetric it returns conditions that if satisfied would mean the table is symmetric.

If the return values of a function table are not concrete values, the tool states assumptions about the symmetry of the return values of the function.

Symmetry checking of the "LatitudeEquivalent" table points out that in one row a condition is written "x < 58" and in another row it is written "58 > x". The analysis carried out by the tool is to a large degree based on the syntactical equivalence of primitive terms so it is not able to show these terms are equal. However if the table is amended to use only one form of this expression, the tool would be able to show the table is symmetric.

4 Summary

This project can be considered a success because a formal specification was achieved with little loss of readability of the specification, and analysis of the specification was possible which revealed inconsistencies in the published separation minima.

We believe that this formalization of these separation minima is readable without extensive training in the use of formal specification techniques. A number of potential ambiguities in the source documents were identified and resolved as part of the formalization process. The formal specification contains 15 tables, 18 definitions, 43 uninterpreted constants, 2 uninterpreted types, 1 defined type (location), and 1 type abbreviation (time).

The formal specification technique demonstrated by this example formalization of the NAT Separation Minima bears some similarity with the use of pseudo-code as a means of specification. One common element is the use of a fixed syntax that is potentially parseable by software tools. But unlike pseudo code, the formal specification technique used here provides constructs necessary to avoid the unintentional embedding of design or implementation constraints in the specification. For instance, it does not constrain the order in which conditions are to be evaluated when determining if the condition "AreSeparated" is satisfied. It also does not require full descriptions to be given for all terms used in the specification. The use of uninterpreted constants allows the formalization to maintain a suitable level of abstraction for the separation minima.

The formal specification and analysis techniques demonstrated by example in this report are intended to be widely applicable. We suggest that this approach would be very useful in the specification and analysis of systems with combinations of conditions that produce different outcomes that involve considerable logical complexity. In an informal natural language specification these are conditions that involve considerable use of words such as "or" and "and". As demonstrated here, automated analysis of these specifications, such as the 45 checks carried out in this work, can quickly reveal problems in the specification that are not easily discovered through manual review.

As noted in the introduction of Appendix A, the formalization of the NAT Separation Minima does not include the explanatory text found in the source documents. An interesting next step in the development of this work would be to prototype a new version of the source document incorporating the formalization of the minima based on the approach documented in this report.

5 Acknowledgments

The authors would like to thank Richard Yates of MacDonald Dettwiler for comments on an early draft of the specification. The first author would also like to acknowledge the input of her thesis committee in focusing the ideas used to develop the analysis techniques. Christine Jensen of UBC provided helpful comments on a final draft of this technical report. The university based component of this collaborative research is support by funding from Hughes International Airspace Management Systems, MacDonald Dettwiler, and the BC Advanced Systems Institute.

6 References

[DeM79] Tom DeMarco. *Structured Analysis and System Specification*. Yourdon Press, Englewood Cliffs, New Jersey, 1979.

[DJP97] Nancy A. Day, Jeffrey J. Joyce, and Gerry Pelletier. Formalization and Analysis of the Separation Minima for Aircraft in the North Atlantic Region. In *Lfm97: Fourth NASA Langley Formal Methods Workshop*, NASA Conference Publication 3356, compiled by C. Michael Holloway and Kelly J. Hayhurst, September 1997.

[HC95] D.N. Hoover and Zewei Chen. Tablewise, a Decision Table Tool. In *Proceedings of the Ninth Annual Conference on Computer Assurance (COMPASS'95)*, 97-108, Gaithersburg, MD, June 1995.

[HJL96] Constance L. Heitmeyer, Ralph D. Jeffords, and Bruce G. Labaw. Automated consistency checking of requirements specifications. *ACM Transactions on Software Engineering and Methodology*, 5(3):231-261, July 1996.

[HL96] Mats P.E. Heimdahl, and Nancy G. Leveson. Completeness and consistency in hierarchical state-based requirements. *IEEE Transactions on Software Engineering*, 22(6):363-377, June 1996.

[JDD94] J. Joyce, N. Day, and M. Donat. S: A machine readable specification notation based on higher order logic. In *7th International Workshop on Higher Order Logic Theorem Proving and Its Applications*, 285-299, Valletta, Malta, September 1994.

[LHHR94] Nancy G. Leveson, Mats P.E. Heimdahl, Holly Hildreth, and Jon D. Reese. Requirements specification for process-control systems. *IEEE Transactions on Software Engineering*, 20(9):684-707, September 1994.

[P92] David Lorge Parnas. Tabular representations of relations. Technical Report 260, Communications Research Laboratory, Faculy of Engineering, McMaster University, October 1992.

[P91] L.C. Paulson. *ML for the Working Programmer*. Cambridge University Press, Cambridge, 1991.

Appendix A: Full Specification

1 Introduction

This formal specification of the separation minima for the North Atlantic Region was developed by researchers at the University of British Columbia and Hughes International Airspace Management Systems. Our formalization is based on a description provided in a source document entitled "Application of Separation Minima for the NAT Region" (3rd edition, effective December 1992) published by Transport Canada on behalf of the ICAO North Atlantic Systems Planning Group. ICAO is the International Civil Aviation Organization with headquarters in Montreal, Canada. This separation minima document was developed by the COMAG (Communications and ATM Automation Group), now called the CADAG (Communication, Automation and Data Link Applications Group).

The formalized version of the specification presented here does not include as much explanatory text as would be found in a full specification document. In its current form, this document should be seen as a companion document to the official specification.

This does not represent the views of MacDonald Dettwiller (MDA) or Hughes International Airspace Management Systems. The document that formed the basis for this formal specification is not proprietary.

2 Abbreviated Terms

MNPS - Minimum Navigation Performance Specifications
WATRS - West Atlantic Route System

3 Structure of the Specification

The following diagram illustrates the structure of the specification and the dependencies between functions used in defining the specification. Specification primitives are not included on this map. If viewed using an HTML browsing tool, clicking on a function name in this diagram will take you to the definition of the function.

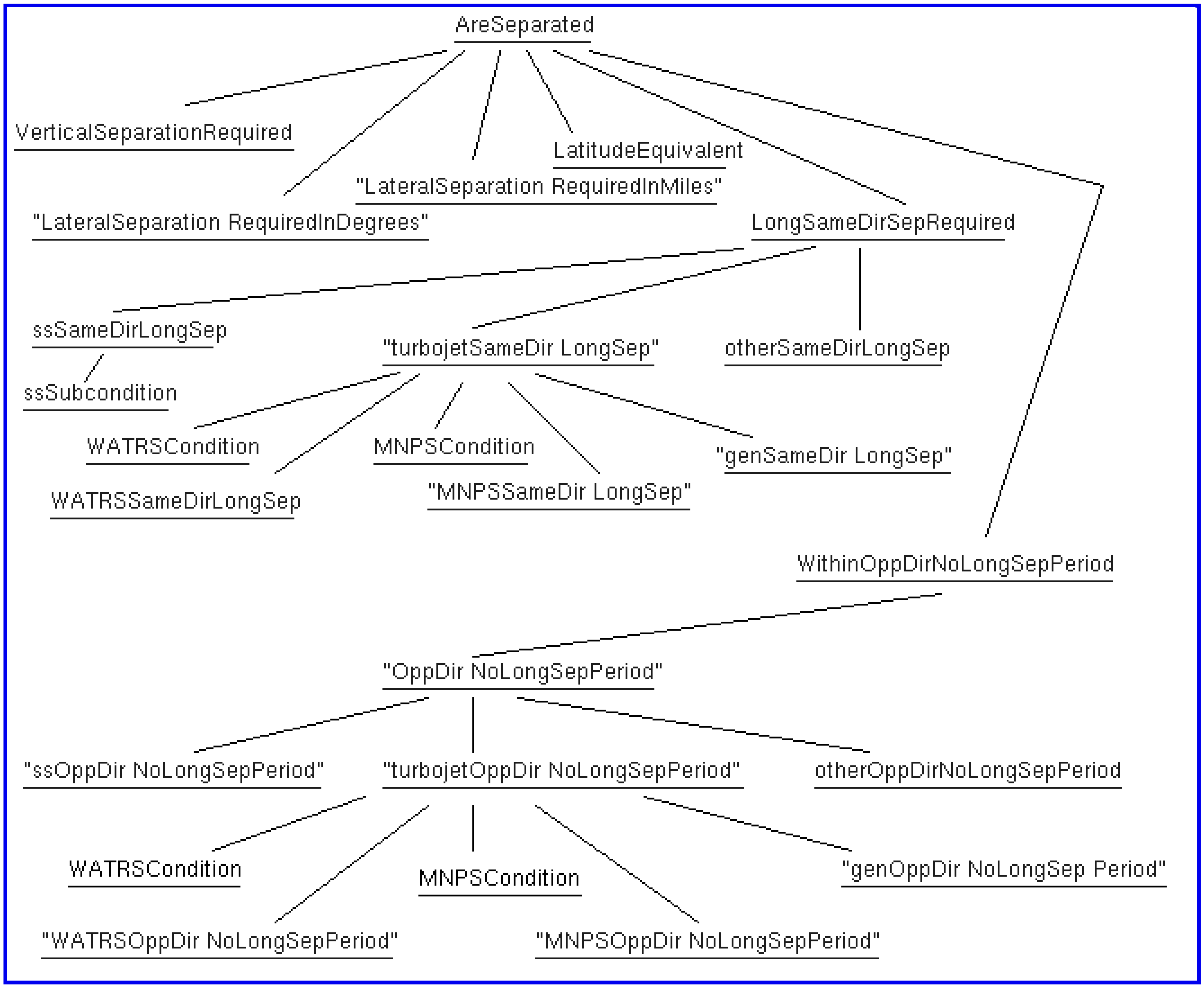

Figure 1. Hierarchy of Definitions.

4 Separation

Two aircraft, A and B, satisfy the separation criteria if they satisfy the rules for at least one of vertical, lateral, or longitudinal separation. This is formalized as:

```
AreSeparated(A:flight,B:flight,t:time) :=
    /* A and B are vertically separated based on flight level */
    (ABS(FlightLevel A - FlightLevel B) > VerticalSeparationRequired(A,B))
    OR

    /* A and B are laterally separated based on either position in degrees
       of latitude or position in miles */
    (if (LatitudeEquivalent(A,B))
    then
      (ABS(LateralPositionInDegrees A - LateralPositionInDegrees B) >
           "LateralSeparation RequiredInDegrees" (A,B))
    else
      (ABS(LateralPositionInMiles A - LateralPositionInMiles B) >
           "LateralSeparation RequiredInMiles" (A,B)))
    OR
```

```
/* A and B are longitudinally separated based on time
   depending on whether the two flights are in the approximate
   same or opposite direction */
(if (AngularDifferenceGreaterThan90Degrees
        (RouteSegment A, RouteSegment B))
then     /* opposite direction */
    NOT (WithinOppDirNoLongSepPeriod(A,B,t))
else     /* same direction */
    ABS(TimeAtPosition A - TimeAtPosition B) >
           LongSameDirSepRequired(A,B));
```

Note that identifiers within quotation marks have no special meaning. The use of quotation marks allows identifiers to contain white space.

5 Vertical Separation

The rules for vertical separation are formalized in the following table:

Table 12: VerticalSeparationRequired

		1	2	3	4	Default
1	**FlightLevel (A)**	__ <= 280	.	__>450	__>450	
2	**FlightLevel (B)**	.	__ <= 280	__ > 450	__>450	
3	**IsSupersonic (A)**	.	.	__=T	.	
4	**IsSupersonic (B)**	.	.	.	__=T	
	VerticalSeparationRequired (A,B)	1000	1000	4000	4000	2000

6 Lateral Separation

The rules for lateral separation have special cases for certain routes. It is assumed that these limits apply to aircraft travelling in either direction on these routes. These routes are described using sets of pairs:

```
Routes1 := {(USA,BDA);(CAN,BDA);(IberianPeninsula, Azores);
(Iceland,Scandinavia);(Iceland, UnitedKingdom)};
```

```
Routes2 := {(USA,Caribbean);(CAN,Caribbean);(BDA, Caribbean)};
```

The following predicate checks whether the departure and destination of a flight fall within a set of routes:

```
IsOnRoute (R:(location#location)set) (X:flight) :=
   ((RouteDeparture (X), RouteDestination (X)) In R) OR
    ((RouteDestination (X), RouteDeparture (X)) In R);
```

The following predicate determines whether the flightlevel of an aircraft is above 275:

```
FlightLevelAbove275 (X:flight) := FlightLevel X > 275;
```

The rules for lateral separation are:

Table 13: "LateralSeparation RequiredInDegrees"

		1	2	3	4	Default
1	**AllOf [A;B] IsOutsideMNPSAirspace**	__=T	__=T	.	.	
2	**AllOf [A;B] (IsOnRoute (Routes1))**	__=T	.	.	.	
3	**AllOf [A;B] (IsOnRoute (Routes2))**	.	__=T	.	.	
4	**AllOf [A;B] IsWestOf55W**	.	__=T	.	.	
5	**AllOf [A;B] IsSupersonic**	.	.	__=T	.	
6	**AllOf [A;B] FlightLevelAbove275**	.	.	__=T	.	
7	**AllOf [A;B] MeetMNPS**	.	.	.	__=T	
8	**AllOf [A;B] HavePartOfRouteInMNPSAirspace**	.	.	.	__=T	
	"LateralSeparation RequiredInDegrees" (A,B)	1.5	1.5	1	1	2

As prescribed by the "AreSeparated" predicate, sometimes it is necessary to calculate the lateral separation required in miles, rather than degrees. The result in miles is given by the following table:

Table 14: "LateralSeparation RequiredInMiles"

		1	2	3	4	Default
1	**AllOf [A;B] IsOutsideMNPSAirspace**	__=T	__=T	.	.	
2	**AllOf [A;B] (IsOnRoute (Routes1))**	__=T	.	.	.	
3	**AllOf [A;B] (IsOnRoute (Routes2))**	.	__=T	.	.	
4	**AllOf [A;B] IsWestOf55W**	.	__=T	.	.	
5	**AllOf [A;B] IsSupersonic**	.	.	__=T	.	
6	**AllOf [A;B] FlightLevelAbove275**	.	.	__=T	.	
7	**AllOf [A;B] MeetMNPS**	.	.	.	__=T	
8	**AllOf [A;B] HavePartOfRouteInMNPSAirspace**	.	.	.	__=T	
	"LateralSeparation RequiredInMiles" (A,B)	90	90	60	60	120

In this formalization, the primitive ``HavePartOfRouteInMNPSAirspace" is true for a flight that has a portion of its route in MNPS airspace. In particular, it only applies to segments of the flight's route that are in, directly above, or directly below MNPS airspace.

Conversely we assume that ``IsOutsideMNPSAirspace" is true for a flight which has no portion of route in MNPS airspace or for those segments of an MNPS flight which are below the level immediately below MNPS airspace and to the segments above the level immediately above MNPS airspace.

The following table determines whether the rules that result in degrees (Table 13) or the rules that result in miles (Table 14) are used:

Table 15: LatitudeEquivalent

		1	2	3	4	5	6	Default
1	**"RouteSegment Degrees" A**	__<=58	.	(58<__) AND (__<70)	.	(70<=__) AND (__<=80)	.	
2	**"RouteSegment Degrees" B**	.	__<=58	.	(__>58) AND (__<70)	.	(70<=__) AND (__<=80)	
3	**AllOf [A;B] "LatChange Per10DLong LessThanOrEq3"**	__=T	__=T	.	.	.	.	
4	**AllOf [A;B] "LatChange Per10DLong LessThanOrEq2"**	.	.	__=T	__=T	.	.	
5	**AllOf [A;B] "LatChange Per10DLong LessThanOrEq1"**	.	.	.	.	__=T	__=T	
	LatitudeEquivalent (A,B)	T	T	T	T	T	T	F

7 Longitudinal Separation

The rules for longitudinal separation are divided into three cases: supersonic aircraft, turbojet aircraft and other aircraft.

Longitudinal Separation does not exist between flights of opposing direction during a certain time period.

```
WithinOppDirNoLongSepPeriod(A:flight,B:flight,t:time) :=
  let timePeriod := "OppDir NoLongSepPeriod"(A,B) in
  (StartTime(timePeriod) <= t) AND (t <= EndTime(timePeriod));
```

Table 16: LongSameDirSepRequired

		1	2	Default
1	**AllOf [A;B] IsSupersonic**	__=T	__=F	
2	**AllOf [A;B] IsTurbojet**	.	__=T	
	LongSameDirSepRequired (A,B)	ssSameDirLongSep (A,B)	"turbojetSameDir LongSep" (A,B)	otherSameDirLongSep (A,B)

Table 17: "OppDir NoLongSepPeriod"

		1	2	Default
1	**AllOf [A;B] IsSupersonic**	__=T	__=F	
2	**AllOf [A;B] IsTurbojet**	.	__=T	
	"OppDir NoLongSepPeriod" (A,B)	"ssOppDir NoLongSepPeriod" (A,B)	"turbojetOppDir NoLongSepPeriod" (A,B)	"otherOppDir NoLongSepPeriod" (A,B)

7.1 Supersonic Aircraft

The rules for the separation period for opposite direction flight route segments is based on the estimated passing time (ept). Opposite direction exists when the intersection angle between route segments of two flights is greater than 90 degrees.

Table 18: "ssOppDir NoLongSepPeriod"

		1	2
1	**ReportedOverCommonPoint(A,B)**	__=T	__=F
	"ssOppDir NoLongSepPeriod" (A,B)	(ept(A,B),ept(A,B)+10)	(ept(A,B)-15,ept(A,B)+15)

The following table shows the longitudinal separation requirements for same direction supersonic flights. Same direction here applies to route segments with an intersection angle of 90 degrees or less.

Table 19: ssSameDirLongSep

		1	2	Default
1	**ssSubcondition(A,B)**	__=T	__=T	
2	**"SameOr Diverging Tracks"(A,B)**	__=T	__=T	
3	**ReportedOverCommonPoint(A,B)**	__=T	.	
4	**"Appropriate TimeSep AtCommon Point"(A,B)**	.	__=T	
	ssSameDirLongSep(A,B)	10	10	15

Table 20: ssSubcondition

		1	2	Default
1	**AllOf [A;B] IsLevel**	__=T	.	
2	**SameMachNumber (A,B)**	__=T	.	
3	**SameType(A,B)**	.	__=T	
4	**AllOf [A;B] InCruiseClimb**	.	__=T	
	ssSubcondition(A,B)	T	T	F

7.2 Turbojet Aircraft

The required longitudinal separation for turbojet aircraft is the minimum value obtained by three possible calculations. If certain conditions apply then the MNPS calculation is relevant; likewise certain conditions limit when the WATRS calculation is applicable. These conditions are specified at the beginning of the MNPS and WATRS sections. The calculations for the general longitudinal separation for turbojet aircraft always apply. The following table specifies how the different calculation produce the minimum longitudinal separation for turbojet aircraft travelling in the same direction.

Table 21: "turbojetSameDir LongSep"

		1	2	3	4
1	**MNPSCondition (A,B)**	__=T	__=F	__=T	__=F
2	**WATRSCondition (A,B)**	__=T	__=T	__=F	__=F
	"turbojetSameDir LongSep" (A,B)	MinAll (A,B)	Min { "WATRSSameDir LongSep" (A,B); "genSameDir LongSep" (A,B)}	Min { "MNPSSameDir LongSep" (A,B); "genSameDir LongSep" (A,B)}	"genSameDir LongSep" (A,B)

This table relies on the following definition which takes the minimum value of all the possible calculations:

```
MinAll(A,B) :=
    Min {
         "MNPSSameDir LongSep"(A,B);
         "WATRSSameDir LongSep"(A,B);
         "genSameDir LongSep"(A,B)};
```

The separation period for which turbojet aircraft flying in opposing directions are not considered separated also depends on multiple calculations as given by the following table:

Table 22: "turbojetOppDir NoLongSepPeriod"

		1	2	3	4
1	**MNPSCondition (A,B)**	__=T	__=F	__=T	__=F
2	**WATRSCondition (A,B)**	__=T	__=T	__=F	__=F
	"turbojetOppDir NoLongSepPeriod" (A,B)	UnionAll (A,B)	UnionOfRange { "WATRSOppDir NoLongSepPeriod" (A,B); "genOppDir NoLongSep Period" (A,B)}	UnionOfRange { "MNPSOppDir NoLongSepPeriod" (A,B); "genOppDir NoLongSep Period" (A,B)}	"genOppDir NoLongSep Period" (A,B)

The resulting time period in which the aircraft do not have longitudinal separation is the minimum of all the start times given by the relevant calculations and the maximum of all the possible end times, as given by the following two definitions:

```
UnionOfRange (periods) :=
        (MinEarliestTime (periods), MaxLatestTime (periods));


UnionAll (A,B) :=
    let periods :=
             {"MNPSOppDir NoLongSepPeriod"(A,B);
              "WATRSOppDir NoLongSepPeriod"(A,B);
              "genOppDir NoLongSep Period"(A,B)} in
    (MinEarliestTime (periods), MaxLatestTime (periods));
```

7.2.1 MNPS Rules

The conditions for using the MNPS rules are:

```
MNPSCondition(A,B) :=
(AllOf [A;B] MeetMNPS ) AND
(AllOf [A;B] HavePartOfRouteInMNPSAirspace );
```

The time period during which turbojet aircraft on opposing tracks do not have separation is the same as that given for supersonic aircraft.

```
"MNPSOppDir NoLongSepPeriod"(A,B) := "ssOppDir NoLongSepPeriod"(A,B);
```

The following table defines the separation required in minutes for turbojet aircraft flying in the same direction using the MNPS calculation.

Table 23: "MNPSSameDir LongSep"

		1	**2**	**3**	**4**	**5**	**Default**
1	**"Appropriate TimeSep AtCommon Point" (A,B)**	__=T	__=T	__=T	__=T	__=T	
2	**"SameOr Diverging Tracks" (A,B)**	__=T	__=T	__=T	__=T	__=T	
3	**Mach (FirstAircraft (A,B)) - Mach (SecondAircraft (A,B))**	(__>0.06)	((0.06>=__) AND (__>0.05))	((0.05>=__) AND (__>0.04))	((0.04>=__) AND (__>0.03))	((0.03>=__) AND (__>0.02))	
	"MNPSSameDir LongSep" (A,B)	5	6	7	8	9	10

7.2.2 WATRS Rules

The following table gives the conditions for using the WATRS rules:

Table 24: WATRSCondition

		1	2	Default
1	**AllOf [A;B] EnterWATRSAirspaceAtSomeTime**	__=T	__=T	
2	**AllOf [A;B] IsWestOf60W**	__=T	.	
3	**AllOf [A;B] InWATRSAirspace**	.	__=T	
4	**AllOf [A;B] MachTechniqueUsed**	__=T	__=T	
5	**AllOf [A;B] OnPublishedRoute**	__=T	__=T	
6	**"SameOr Diverging Tracks" (A,B)**	__=T	__=T	
	WATRSCondition(A,B)	T	T	F

A completed specification should provide detailed specifications of the same and opposite direction longitudinal separation required in the WATRS airspace. However, due to problems in the WATRS rules in the source document, we could not proceed with an accurate definition. Our interim solution is to introduce these constants as uninterpreted primitives.

```
"WATRSSameDir LongSep" : (flight#flight) -> num;
```

```
"WATRSOppDir NoLongSepPeriod": (flight#flight) -> (time#time);
```

7.2.3 General Turbojet Rules

The time period during which turbojet aircraft flying in opposing directions do not have longitudinal separation is the same as that given by the MNPS calculations.

```
"genOppDir NoLongSep Period"(A,B) := "MNPSOppDir NoLongSepPeriod"(A,B);
```

The general calculation for the separation required by turbojet aircraft flying in the same direction is given by the following table:

Table 25: "genSameDir LongSep"

		1	2	3	Default
1	**"SameOr Diverging Tracks" (A,B)**	__=T	__=T	__=T	
2	**AllOf [A;B] MachTechniqueUsed**	__=F	__=T	__=T	
3	**AtLeastOneOf [A;B] InCruiseClimb**	__=F	__=F	__=F	
4	**ReportedOverCommonPoint (A,B)**	__=T	.	.	
5	**"Appropriate TimeSep AtCommon Point" (A,B)**	.	__=T	__=T	
6	**Mach (FirstAircraft(A,B)) - Mach (SecondAircraft(A,B))**	.	(__>0.6)	(0.6>=__) AND (__>0.3)	
	"genSameDir LongSep" (A,B)	15	5	10	20

8 Other Aircraft

The rules for calculating the time period during which flights on opposing tracks are not longitudinally separated for aircraft other than turbojets and supersonics are the same as those for the general turbojet case:

```
"otherOppDir NoLongSepPeriod" (A,B) := "genOppDir NoLongSep Period"(A,B);
```

There is a special case for certain routes:

```
Routes3 :=
        {(USA,Caribbean);(CAN,Caribbean);
        (BDA, Caribbean);(USA,BDA); (CAN,BDA)};
```

Table 26: otherSameDirLongSep

		1	2	Default
1	**ReportedOverCommonPoint(A,B)**	__=T	.	
2	**"SameOr Diverging Tracks"(A,B)**	__=T	.	
3	**AllOf [A;B] (IsOnRoute Routes3)**	.	__=T	
	otherSameDirLongSep (A,B)	15	20	30

9 Environmental Assumptions

There are various relationships between the primitive terms used in this specification. These are documented as environmental assumptions.

Often it is the case that two conditions can not both be true at the same time. However, both conditions can be false. "env1" specifies constraints of this sort.

```
env1 :=
(forall (A:flight).  NOT (IsLevel (A) AND InCruiseClimb (A)))
AND
(forall (A:flight).NOT (IsOnRoute (Routes1) (A) AND IsOnRoute (Routes2) (A)));
```

In order to check the symmetry of the tables, environmental assumptions are needed about the symmetry of some primitive terms.

```
env2 :=
(forall (A:flight) (B:flight).
        ReportedOverCommonPoint(A,B) = ReportedOverCommonPoint(B,A))
AND
(forall (A:flight) (B:flight).
        SameMachNumber(A,B) = SameMachNumber(B,A))
AND
(forall (A:flight) (B:flight).
        SameType(A,B) = SameType(B,A))
AND
(forall (A:flight) (B:flight).
        "SameOr Diverging Tracks"(A,B) = "SameOr Diverging Tracks"(B,A))
AND
(forall (A:flight) (B:flight).
        "Appropriate TimeSep AtCommon Point"(A,B) =
                                "Appropriate TimeSep AtCommon Point"(B,A)) ;
```

There are also some numeric relationships between the rows used in the LatitudeEquivalent table.

```
env3 :=
(forall A.
        if "LatChange Per10DLong LessThanOrEq2" (A)
```

```
        then "LatChange Per10DLong LessThanOrEq3" (A))
AND
(forall A.
        if "LatChange Per10DLong LessThanOrEq1" (A)
        then "LatChange Per10DLong LessThanOrEq2" (A))
AND
(forall A.
        if "LatChange Per10DLong LessThanOrEq1" (A)
        then "LatChange Per10DLong LessThanOrEq3" (A));
```

All environmental assumptions are given by:

```
env := env1 AND env2 AND env3;
```

10 Specification Primitives

The primitives specified in this section are assumed to be terms in the common domain knowledge of the intended users of this specification. This list should also include an informal description of each term to more thoroughly document the meaning of these terms.

10.1 Primitive Types

- `:flight;`
- `:location := Azores | BDA | CAN | Caribbean | IberianPeninsula | Iceland | Scandinavia | UnitedKingdom | USA ;`
- `:segment;`
- `:time == num;`

10.2 Primitive Constants

Constants include functions, and predicates in addition to data values.

Static attributes of an object of type "flight":

- `IsSupersonic :(flight -> bool);`
- `IsTurbojet :(flight -> bool);`
- `HavePartOfRouteInMNPSAirspace : (flight -> bool) ;`
- `MeetMNPS :(flight -> bool);`
- `OnPublishedRoute :(flight -> bool);`
- `RouteDeparture :(flight -> location);`
- `RouteDestination :(flight -> location);`
- `MachTechniqueUsed : flight ->bool;`

Time-varying attributes of an object of type "flight" at the current time:

- `FlightLevel:(flight -> num);`
- `InCruiseClimb:(flight -> bool);`
- `InWATRSAirspace :(flight -> bool);`
- `IsLevel:(flight -> bool);`
- `IsOutsideMNPSAirspace :(flight -> bool);`
- `IsWestOf60W :(flight -> bool);`
- `IsWestOf55W:(flight -> bool);`
- `"LatChange Per10DLong LessThanOrEq1" :flight->bool;`
- `"LatChange Per10DLong LessThanOrEq2" :flight->bool;`
- `"LatChange Per10DLong LessThanOrEq3" :flight->bool;`
- `LateralPositionInDegrees :(flight -> num);`
- `LateralPositionInMiles :(flight -> num);`
- `Mach : (flight->num);`
- `RouteSegment :(flight -> segment);`
- `"RouteSegment Degrees" :(flight -> num);`
- `TimeAtPosition :(flight->time);`

Static relationships between pairs of objects of type "flight":

- `SameType :((flight # flight) -> bool);`

Time varying relationships between pairs of objects of type "flight" at the current time:

- `SameMachNumber :((flight # flight) -> bool);`
- `FirstAircraft :((flight # flight) -> flight);`
- /* ept is the estimated passing time of two flights */

  `ept : ((flight # flight)->time);`
- `"SameOr Diverging Tracks" :((flight # flight) -> bool);`
- `SecondAircraft :((flight # flight) -> flight);`

Historical relationships between pairs of objects of type "flight":

- `ReportedOverCommonPoint :((flight # flight) -> bool);`
- `"Appropriate TimeSep AtCommon Point":((flight # flight) -> bool);`

- `EnterWATRSAirspaceAtSomeTime : (flight -> bool);`

Static relationships between pairs of other types of objects:

- `AngularDifferenceGreaterThan90Degrees:(segment # segment)->bool;`

Miscellaneous Application Specific Primitives :

- /* given a time period return the beginning of the time period */

  `StartTime : (time # time) -> time;`

- /* given a time period return the end of the time period */

  `EndTime : (time # time) -> time;`

- /* for a set of pairs of times, return the minimum value of the first element of the pairs */

  `MinEarliestTime : (time # time)set -> time;`

- /* for a set of pairs of times, return the maximum value of the second element of the pairs */

  `MaxLatestTime : (time # time)set -> time;`

Application Independent Primitives:

- /* absolute value of a number */

  `ABS : num -> num;`

- /* determine if an element is in a set */

  `(:ty) (_ In _) : (ty) -> ((ty)set) ->bool;`

- /* return the minimum number of a set of numbers */

  `Min: ((num)set -> num);`

- This specification also makes use of common arithmetic operators, such as "<" and ">=", and the common Boolean operators AND, OR , and NOT.

This specification is dependent upon the following files:

```
%include startup.s
%include table.s
```

Appendix B: Full Analysis Results

1 Introduction to the Analysis Results

This appendix presents the full analysis results carried out in this work. For each table in the specification, we present the tool's output for completeness, consistency and symmetry analysis. The constant "DC" is used to represent a "." or "don't care" case.

1.1 VerticalSeparationRequired Completeness Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%comp VerticalSeparationRequired env

VerticalSeparationRequired is:
(
  (Table
    [
      ((Row (FlightLevel A))
        [(\x.(x <= 280));DC;(\x.(x > 450));(\x.(x > 450))])
      ;
      ((Row (FlightLevel B))
        [DC;(\x.(x <= 280));(\x.(x > 450));(\x.(x > 450))])
      ;((Row (IsSupersonic A)) [DC;DC;TRUE;DC]);
      ((Row (IsSupersonic B)) [DC;DC;DC;TRUE])])
  [1000;1000;4000;4000;2000])

Invoking interval checker...

Interval checker partitions the range into:
((FlightLevel A) > 450)
((280 < (FlightLevel A)) AND ((FlightLevel A) <= 450))
((FlightLevel A) <= 280)

Invoking interval checker...

Interval checker partitions the range into:
((FlightLevel B) > 450)
((280 < (FlightLevel B)) AND ((FlightLevel B) <= 450))
((FlightLevel B) <= 280)

The following cases
yield the default value of 2000
Case 1
Row 1 : ((280 < (FlightLevel A)) AND ((FlightLevel A) <= 450))
Row 2 : ((FlightLevel B) > 450)
Row 3 : DC
Row 4 : DC

Case 2
Row 1 : ((FlightLevel A) > 450)
Row 2 : ((280 < (FlightLevel B)) AND ((FlightLevel B) <= 450))
Row 3 : DC
Row 4 : DC

Case 3
Row 1 : ((280 < (FlightLevel A)) AND ((FlightLevel A) <= 450))
Row 2 : ((280 < (FlightLevel B)) AND ((FlightLevel B) <= 450))
Row 3 : DC
Row 4 : DC

Case 4
Row 1 : ((FlightLevel A) > 450)
Row 2 : ((FlightLevel B) > 450)
Row 3 : ((IsSupersonic A) = F)
Row 4 : ((IsSupersonic B) = F)


Stats for VerticalSeparationRequired completeness checking:
Number of cases identified: 4
Processor time: user: 0 sec; system: 0 sec
```

```
 ------------------------------------------------------------
>
Fusion session over.
```

1.2 VerticalSeparationRequired Consistency Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%cons VerticalSeparationRequired env

VerticalSeparationRequired is:
(
  (Table
    [
      ((Row (FlightLevel A))
        [(\x.(x <= 280));DC;(\x.(x > 450));(\x.(x > 450))])
      ;
      ((Row (FlightLevel B))
        [DC;(\x.(x <= 280));(\x.(x > 450));(\x.(x > 450))])
      ;((Row (IsSupersonic A)) [DC;DC;TRUE;DC]);
      ((Row (IsSupersonic B)) [DC;DC;DC;TRUE])])
  [1000;1000;4000;4000;2000])

Invoking interval checker...

Interval checker partitions the range into:
((FlightLevel A) > 450)
((280 < (FlightLevel A)) AND ((FlightLevel A) <= 450))
((FlightLevel A) <= 280)

Invoking interval checker...

Interval checker partitions the range into:
((FlightLevel B) > 450)
((280 < (FlightLevel B)) AND ((FlightLevel B) <= 450))
((FlightLevel B) <= 280)

No inconsistencies
were found in the table.

Stats for VerticalSeparationRequired consistency checking:
Number of cases identified: 0
Processor time: user: 0 sec; system: 0 sec
 ------------------------------------------------------------
>
Fusion session over.
```

1.3 VerticalSeparationRequired Symmetry Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
```

```
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%sym VerticalSeparationRequired env

VerticalSeparationRequired is:
(
  (Table
    [
      ((Row (FlightLevel A))
        [(\x.(x <= 280));DC;(\x.(x > 450));(\x.(x > 450))])
      ;
      ((Row (FlightLevel B))
        [DC;(\x.(x <= 280));(\x.(x > 450));(\x.(x > 450))])
      ;((Row (IsSupersonic A)) [DC;DC;TRUE;DC]);
      ((Row (IsSupersonic B)) [DC;DC;DC;TRUE])])
  [1000;1000;4000;4000;2000])


The table is symmetric.
Processor time: user: 0 sec; system: 0 sec
 ------------------------------------------------------------
>
Fusion session over.
```

1.4 LateralSeparation RequiredInDegrees Completeness Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%comp "LateralSeparation RequiredInDegrees" env

"LateralSeparation RequiredInDegrees" is:
(
  (Table
    [
      ((Row ((AllOf [A;B]) IsOutsideMNPSAirspace))
        [TRUE;TRUE;DC;DC]);
      ((Row ((AllOf [A;B]) (IsOnRoute Routes1)))
        [TRUE;DC;DC;DC]);
      ((Row ((AllOf [A;B]) (IsOnRoute Routes2)))
        [DC;TRUE;DC;DC]);
      ((Row ((AllOf [A;B]) IsWestOf55W)) [DC;TRUE;DC;DC]);
      ((Row ((AllOf [A;B]) IsSupersonic)) [DC;DC;TRUE;DC]);
      ((Row ((AllOf [A;B]) FlightLevelAbove275))
        [DC;DC;TRUE;DC]);
      ((Row ((AllOf [A;B]) MeetMNPS)) [DC;DC;DC;TRUE]);
      ((Row ((AllOf [A;B]) HavePartOfRouteInMNPSAirspace))
        [DC;DC;DC;TRUE])]) [1.5;1.5;1;1;2])

The following cases
yield the default value of 2
Case 1
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = F)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : DC
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = F)
Row 6 : DC
Row 7 : (((AllOf [A;B]) MeetMNPS) = F)
Row 8 : DC

Case 2
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = F)
Row 2 : DC
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = F)
```

```
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = F)
Row 6 : DC
Row 7 : (((AllOf [A;B]) MeetMNPS) = F)
Row 8 : DC

Case 3
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = F)
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = F)
Row 6 : DC
Row 7 : (((AllOf [A;B]) MeetMNPS) = F)
Row 8 : DC

Case 4
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = T)
Row 4 : (((AllOf [A;B]) IsWestOf55W) = F)
Row 5 : (((AllOf [A;B]) IsSupersonic) = F)
Row 6 : DC
Row 7 : (((AllOf [A;B]) MeetMNPS) = F)
Row 8 : DC

Case 5
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = F)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : DC
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = T)
Row 6 : (((AllOf [A;B]) FlightLevelAbove275) = F)
Row 7 : (((AllOf [A;B]) MeetMNPS) = F)
Row 8 : DC

Case 6
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = F)
Row 2 : DC
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = F)
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = T)
Row 6 : (((AllOf [A;B]) FlightLevelAbove275) = F)
Row 7 : (((AllOf [A;B]) MeetMNPS) = F)
Row 8 : DC

Case 7
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = F)
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = T)
Row 6 : (((AllOf [A;B]) FlightLevelAbove275) = F)
Row 7 : (((AllOf [A;B]) MeetMNPS) = F)
Row 8 : DC

Case 8
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = T)
Row 4 : (((AllOf [A;B]) IsWestOf55W) = F)
Row 5 : (((AllOf [A;B]) IsSupersonic) = T)
Row 6 : (((AllOf [A;B]) FlightLevelAbove275) = F)
Row 7 : (((AllOf [A;B]) MeetMNPS) = F)
Row 8 : DC

Case 9
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = F)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : DC
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = F)
Row 6 : DC
Row 7 : (((AllOf [A;B]) MeetMNPS) = T)
Row 8 : (((AllOf [A;B]) HavePartOfRouteInMNPSAirspace) = F)

Case 10
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = F)
Row 2 : DC
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = F)
Row 4 : DC
```

```
Row 5 : (((AllOf [A;B]) IsSupersonic) = F)
Row 6 : DC
Row 7 : (((AllOf [A;B]) MeetMNPS) = T)
Row 8 : (((AllOf [A;B]) HavePartOfRouteInMNPSAirspace) = F)

Case 11
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = F)
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = F)
Row 6 : DC
Row 7 : (((AllOf [A;B]) MeetMNPS) = T)
Row 8 : (((AllOf [A;B]) HavePartOfRouteInMNPSAirspace) = F)

Case 12
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = T)
Row 4 : (((AllOf [A;B]) IsWestOf55W) = F)
Row 5 : (((AllOf [A;B]) IsSupersonic) = F)
Row 6 : DC
Row 7 : (((AllOf [A;B]) MeetMNPS) = T)
Row 8 : (((AllOf [A;B]) HavePartOfRouteInMNPSAirspace) = F)

Case 13
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = F)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : DC
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = T)
Row 6 : (((AllOf [A;B]) FlightLevelAbove275) = F)
Row 7 : (((AllOf [A;B]) MeetMNPS) = T)
Row 8 : (((AllOf [A;B]) HavePartOfRouteInMNPSAirspace) = F)

Case 14
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = F)
Row 2 : DC
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = F)
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = T)
Row 6 : (((AllOf [A;B]) FlightLevelAbove275) = F)
Row 7 : (((AllOf [A;B]) MeetMNPS) = T)
Row 8 : (((AllOf [A;B]) HavePartOfRouteInMNPSAirspace) = F)

Case 15
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = F)
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = T)
Row 6 : (((AllOf [A;B]) FlightLevelAbove275) = F)
Row 7 : (((AllOf [A;B]) MeetMNPS) = T)
Row 8 : (((AllOf [A;B]) HavePartOfRouteInMNPSAirspace) = F)

Case 16
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = T)
Row 4 : (((AllOf [A;B]) IsWestOf55W) = F)
Row 5 : (((AllOf [A;B]) IsSupersonic) = T)
Row 6 : (((AllOf [A;B]) FlightLevelAbove275) = F)
Row 7 : (((AllOf [A;B]) MeetMNPS) = T)
Row 8 : (((AllOf [A;B]) HavePartOfRouteInMNPSAirspace) = F)


Stats for "LateralSeparation RequiredInDegrees" completeness checking:
Number of cases identified: 16
Processor time: user: 3 sec; system: 0 sec
 ------------------------------------------------------------
>
Fusion session over.
```

1.5 LateralSeparation RequiredInDegrees Consistency Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997
```

```
Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%cons "LateralSeparation RequiredInDegrees" env

"LateralSeparation RequiredInDegrees" is:
(
  (Table
    [
      ((Row ((AllOf [A;B]) IsOutsideMNPSAirspace))
        [TRUE;TRUE;DC;DC]);
      ((Row ((AllOf [A;B]) (IsOnRoute Routes1)))
        [TRUE;DC;DC;DC]);
      ((Row ((AllOf [A;B]) (IsOnRoute Routes2)))
        [DC;TRUE;DC;DC]);
      ((Row ((AllOf [A;B]) IsWestOf55W)) [DC;TRUE;DC;DC]);
      ((Row ((AllOf [A;B]) IsSupersonic)) [DC;DC;TRUE;DC]);
      ((Row ((AllOf [A;B]) FlightLevelAbove275))
        [DC;DC;TRUE;DC]);
      ((Row ((AllOf [A;B]) MeetMNPS)) [DC;DC;DC;TRUE]);
      ((Row ((AllOf [A;B]) HavePartOfRouteInMNPSAirspace))
        [DC;DC;DC;TRUE])]) [1.5;1.5;1;1;2])

Columns 1 and 3 conflict in the following:
Case 1
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = T)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = F)
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = T)
Row 6 : (((AllOf [A;B]) FlightLevelAbove275) = T)
Row 7 : DC
Row 8 : DC

Columns 1 and 4 conflict in the following:
Case 2
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = T)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = F)
Row 4 : DC
Row 5 : DC
Row 6 : DC
Row 7 : (((AllOf [A;B]) MeetMNPS) = T)
Row 8 : (((AllOf [A;B]) HavePartOfRouteInMNPSAirspace) = T)

Columns 2 and 3 conflict in the following:
Case 3
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = T)
Row 4 : (((AllOf [A;B]) IsWestOf55W) = T)
Row 5 : (((AllOf [A;B]) IsSupersonic) = T)
Row 6 : (((AllOf [A;B]) FlightLevelAbove275) = T)
Row 7 : DC
Row 8 : DC

Columns 2 and 4 conflict in the following:
Case 4
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = T)
Row 4 : (((AllOf [A;B]) IsWestOf55W) = T)
Row 5 : DC
Row 6 : DC
Row 7 : (((AllOf [A;B]) MeetMNPS) = T)
Row 8 : (((AllOf [A;B]) HavePartOfRouteInMNPSAirspace) = T)


Stats for "LateralSeparation RequiredInDegrees" consistency checking:
Number of cases identified: 4
Processor time: user: 2 sec; system: 0 sec
 -----------------------------------------------------------
```

```
>
Fusion session over.
```

1.6 LateralSeparation RequiredInDegrees Symmetry Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%sym "LateralSeparation RequiredInDegrees" env

"LateralSeparation RequiredInDegrees" is:
(
  (Table
    [
      ((Row ((AllOf [A;B]) IsOutsideMNPSAirspace))
        [TRUE;TRUE;DC;DC]);
      ((Row ((AllOf [A;B]) (IsOnRoute Routes1)))
        [TRUE;DC;DC;DC]);
      ((Row ((AllOf [A;B]) (IsOnRoute Routes2)))
        [DC;TRUE;DC;DC]);
      ((Row ((AllOf [A;B]) IsWestOf55W)) [DC;TRUE;DC;DC]);
      ((Row ((AllOf [A;B]) IsSupersonic)) [DC;DC;TRUE;DC]);
      ((Row ((AllOf [A;B]) FlightLevelAbove275))
        [DC;DC;TRUE;DC]);
      ((Row ((AllOf [A;B]) MeetMNPS)) [DC;DC;DC;TRUE]);
      ((Row ((AllOf [A;B]) HavePartOfRouteInMNPSAirspace))
        [DC;DC;DC;TRUE])]) [1.5;1.5;1;1;2])


The table is symmetric.
Processor time: user: 1 sec; system: 0 sec
 ------------------------------------------------------------
>
Fusion session over.
```

1.7 LateralSeparation RequiredInMiles Completeness Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%comp "LateralSeparation RequiredInMiles" env

"LateralSeparation RequiredInMiles" is:
(
  (Table
    [
      ((Row ((AllOf [A;B]) IsOutsideMNPSAirspace))
        [TRUE;TRUE;DC;DC]);
      ((Row ((AllOf [A;B]) (IsOnRoute Routes1)))
        [TRUE;DC;DC;DC]);
```

```
      ((Row ((AllOf [A;B]) (IsOnRoute Routes2)))
        [DC;TRUE;DC;DC]);
      ((Row ((AllOf [A;B]) IsWestOf55W)) [DC;TRUE;DC;DC]);
      ((Row ((AllOf [A;B]) IsSupersonic)) [DC;DC;TRUE;DC]);
      ((Row ((AllOf [A;B]) FlightLevelAbove275))
        [DC;DC;TRUE;DC]);
      ((Row ((AllOf [A;B]) MeetMNPS)) [DC;DC;DC;TRUE]);
      ((Row ((AllOf [A;B]) HavePartOfRouteInMNPSAirspace))
        [DC;DC;DC;TRUE])]) [90;90;60;60;120])

The following cases
yield the default value of 120
Case 1
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = F)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : DC
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = F)
Row 6 : DC
Row 7 : (((AllOf [A;B]) MeetMNPS) = F)
Row 8 : DC

Case 2
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = F)
Row 2 : DC
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = F)
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = F)
Row 6 : DC
Row 7 : (((AllOf [A;B]) MeetMNPS) = F)
Row 8 : DC

Case 3
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = F)
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = F)
Row 6 : DC
Row 7 : (((AllOf [A;B]) MeetMNPS) = F)
Row 8 : DC

Case 4
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = T)
Row 4 : (((AllOf [A;B]) IsWestOf55W) = F)
Row 5 : (((AllOf [A;B]) IsSupersonic) = F)
Row 6 : DC
Row 7 : (((AllOf [A;B]) MeetMNPS) = F)
Row 8 : DC

Case 5
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = F)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : DC
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = T)
Row 6 : (((AllOf [A;B]) FlightLevelAbove275) = F)
Row 7 : (((AllOf [A;B]) MeetMNPS) = F)
Row 8 : DC

Case 6
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = F)
Row 2 : DC
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = F)
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = T)
Row 6 : (((AllOf [A;B]) FlightLevelAbove275) = F)
Row 7 : (((AllOf [A;B]) MeetMNPS) = F)
Row 8 : DC

Case 7
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = F)
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = T)
Row 6 : (((AllOf [A;B]) FlightLevelAbove275) = F)
Row 7 : (((AllOf [A;B]) MeetMNPS) = F)
Row 8 : DC
```

```
Case 8
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = T)
Row 4 : (((AllOf [A;B]) IsWestOf55W) = F)
Row 5 : (((AllOf [A;B]) IsSupersonic) = T)
Row 6 : (((AllOf [A;B]) FlightLevelAbove275) = F)
Row 7 : (((AllOf [A;B]) MeetMNPS) = F)
Row 8 : DC

Case 9
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = F)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : DC
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = F)
Row 6 : DC
Row 7 : (((AllOf [A;B]) MeetMNPS) = T)
Row 8 : (((AllOf [A;B]) HavePartOfRouteInMNPSAirspace) = F)

Case 10
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = F)
Row 2 : DC
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = F)
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = F)
Row 6 : DC
Row 7 : (((AllOf [A;B]) MeetMNPS) = T)
Row 8 : (((AllOf [A;B]) HavePartOfRouteInMNPSAirspace) = F)

Case 11
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = F)
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = F)
Row 6 : DC
Row 7 : (((AllOf [A;B]) MeetMNPS) = T)
Row 8 : (((AllOf [A;B]) HavePartOfRouteInMNPSAirspace) = F)

Case 12
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = T)
Row 4 : (((AllOf [A;B]) IsWestOf55W) = F)
Row 5 : (((AllOf [A;B]) IsSupersonic) = F)
Row 6 : DC
Row 7 : (((AllOf [A;B]) MeetMNPS) = T)
Row 8 : (((AllOf [A;B]) HavePartOfRouteInMNPSAirspace) = F)

Case 13
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = F)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : DC
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = T)
Row 6 : (((AllOf [A;B]) FlightLevelAbove275) = F)
Row 7 : (((AllOf [A;B]) MeetMNPS) = T)
Row 8 : (((AllOf [A;B]) HavePartOfRouteInMNPSAirspace) = F)

Case 14
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = F)
Row 2 : DC
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = F)
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = T)
Row 6 : (((AllOf [A;B]) FlightLevelAbove275) = F)
Row 7 : (((AllOf [A;B]) MeetMNPS) = T)
Row 8 : (((AllOf [A;B]) HavePartOfRouteInMNPSAirspace) = F)

Case 15
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = F)
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = T)
Row 6 : (((AllOf [A;B]) FlightLevelAbove275) = F)
Row 7 : (((AllOf [A;B]) MeetMNPS) = T)
Row 8 : (((AllOf [A;B]) HavePartOfRouteInMNPSAirspace) = F)
```

```
Case 16
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = T)
Row 4 : (((AllOf [A;B]) IsWestOf55W) = F)
Row 5 : (((AllOf [A;B]) IsSupersonic) = T)
Row 6 : (((AllOf [A;B]) FlightLevelAbove275) = F)
Row 7 : (((AllOf [A;B]) MeetMNPS) = T)
Row 8 : (((AllOf [A;B]) HavePartOfRouteInMNPSAirspace) = F)


Stats for "LateralSeparation RequiredInMiles" completeness checking:
Number of cases identified: 16
Processor time: user: 4 sec; system: 0 sec
 ------------------------------------------------------------
>
Fusion session over.
```

1.8 LateralSeparation RequiredInMiles Consistency Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%cons "LateralSeparation RequiredInMiles" env

"LateralSeparation RequiredInMiles" is:
(
  (Table
    [
      ((Row ((AllOf [A;B]) IsOutsideMNPSAirspace))
        [TRUE;TRUE;DC;DC]);
      ((Row ((AllOf [A;B]) (IsOnRoute Routes1)))
        [TRUE;DC;DC;DC]);
      ((Row ((AllOf [A;B]) (IsOnRoute Routes2)))
        [DC;TRUE;DC;DC]);
      ((Row ((AllOf [A;B]) IsWestOf55W)) [DC;TRUE;DC;DC]);
      ((Row ((AllOf [A;B]) IsSupersonic)) [DC;DC;TRUE;DC]);
      ((Row ((AllOf [A;B]) FlightLevelAbove275))
        [DC;DC;TRUE;DC]);
      ((Row ((AllOf [A;B]) MeetMNPS)) [DC;DC;DC;TRUE]);
      ((Row ((AllOf [A;B]) HavePartOfRouteInMNPSAirspace))
        [DC;DC;DC;TRUE])]) [90;90;60;60;120])

Columns 1 and 3 conflict in the following:
Case 1
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = T)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = F)
Row 4 : DC
Row 5 : (((AllOf [A;B]) IsSupersonic) = T)
Row 6 : (((AllOf [A;B]) FlightLevelAbove275) = T)
Row 7 : DC
Row 8 : DC

Columns 1 and 4 conflict in the following:
Case 2
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = T)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = F)
Row 4 : DC
Row 5 : DC
Row 6 : DC
Row 7 : (((AllOf [A;B]) MeetMNPS) = T)
Row 8 : (((AllOf [A;B]) HavePartOfRouteInMNPSAirspace) = T)

Columns 2 and 3 conflict in the following:
```

```
Case 3
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = T)
Row 4 : (((AllOf [A;B]) IsWestOf55W) = T)
Row 5 : (((AllOf [A;B]) IsSupersonic) = T)
Row 6 : (((AllOf [A;B]) FlightLevelAbove275) = T)
Row 7 : DC
Row 8 : DC

Columns 2 and 4 conflict in the following:
Case 4
Row 1 : (((AllOf [A;B]) IsOutsideMNPSAirspace) = T)
Row 2 : (((AllOf [A;B]) (IsOnRoute Routes1)) = F)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes2)) = T)
Row 4 : (((AllOf [A;B]) IsWestOf55W) = T)
Row 5 : DC
Row 6 : DC
Row 7 : (((AllOf [A;B]) MeetMNPS) = T)
Row 8 : (((AllOf [A;B]) HavePartOfRouteInMNPSAirspace) = T)


Stats for "LateralSeparation RequiredInMiles" consistency checking:
Number of cases identified: 4
Processor time: user: 3 sec; system: 0 sec
 ------------------------------------------------------------
>
Fusion session over.
```

1.9 LateralSeparation RequiredInMiles Symmetry Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%sym "LateralSeparation RequiredInMiles" env

"LateralSeparation RequiredInMiles" is:
(
  (Table
    [
      ((Row ((AllOf [A;B]) IsOutsideMNPSAirspace))
        [TRUE;TRUE;DC;DC]);
      ((Row ((AllOf [A;B]) (IsOnRoute Routes1)))
        [TRUE;DC;DC;DC]);
      ((Row ((AllOf [A;B]) (IsOnRoute Routes2)))
        [DC;TRUE;DC;DC]);
      ((Row ((AllOf [A;B]) IsWestOf55W)) [DC;TRUE;DC;DC]);
      ((Row ((AllOf [A;B]) IsSupersonic)) [DC;DC;TRUE;DC]);
      ((Row ((AllOf [A;B]) FlightLevelAbove275))
        [DC;DC;TRUE;DC]);
      ((Row ((AllOf [A;B]) MeetMNPS)) [DC;DC;DC;TRUE]);
      ((Row ((AllOf [A;B]) HavePartOfRouteInMNPSAirspace))
        [DC;DC;DC;TRUE])]) [90;90;60;60;120])


The table is symmetric.
Processor time: user: 1 sec; system: 0 sec
 ------------------------------------------------------------
>
Fusion session over.
```

1.10 LatitudeEquivalent Completeness Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%comp LatitudeEquivalent env

LatitudeEquivalent is:
(PredicateTable
  [
    ((Row ("RouteSegment Degrees" A))
      [(\x.(x <= 58));DC;(\x.((58 < x) AND (x < 70)));DC;
        (\x.((70 <= x) AND (x <= 80)));DC]);
    ((Row ("RouteSegment Degrees" B))
      [DC;(\x.(x <= 58));DC;(\x.((x > 58) AND (x < 70)));DC
        ;(\x.((70 <= x) AND (x <= 80)))]);
    ((Row ((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq3"))
      [TRUE;TRUE;DC;DC;DC;DC]);
    ((Row ((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq2"))
      [DC;DC;TRUE;TRUE;DC;DC]);
    ((Row ((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq1"))
      [DC;DC;DC;DC;TRUE;TRUE])])

Invoking interval checker...

Interval checker partitions the range into:
(80 < ("RouteSegment Degrees" A))
((70 <= ("RouteSegment Degrees" A)) AND
  (("RouteSegment Degrees" A) <= 80))
((58 < ("RouteSegment Degrees" A)) AND
  (("RouteSegment Degrees" A) < 70))
(("RouteSegment Degrees" A) <= 58)

Invoking interval checker...

Interval checker partitions the range into:
(80 < ("RouteSegment Degrees" B))
((70 <= ("RouteSegment Degrees" B)) AND
  (("RouteSegment Degrees" B) <= 80))
((("RouteSegment Degrees" B) > 58) AND
  (("RouteSegment Degrees" B) < 70))
(("RouteSegment Degrees" B) <= 58)

The predicate is false
for the following cases:

Case 1
Row 1 : (80 < ("RouteSegment Degrees" A))
Row 2 : (80 < ("RouteSegment Degrees" B))
Row 3 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq3") = T)
Row 4 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq2") = T)
Row 5 : DC

Case 2
Row 1 : ((70 <= ("RouteSegment Degrees" A)) AND
  (("RouteSegment Degrees" A) <= 80))
Row 2 : (80 < ("RouteSegment Degrees" B))
Row 3 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq3") = T)
Row 4 : DC
Row 5 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq1") = F)

Case 3
Row 1 : (80 < ("RouteSegment Degrees" A))
Row 2 : ((70 <= ("RouteSegment Degrees" B)) AND
  (("RouteSegment Degrees" B) <= 80))
Row 3 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq3") = T)
Row 4 : DC
Row 5 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq1") = F)

Case 4
```

```
Row 1 : ((70 <= ("RouteSegment Degrees" A)) AND
  (("RouteSegment Degrees" A) <= 80))
Row 2 : ((70 <= ("RouteSegment Degrees" B)) AND
  (("RouteSegment Degrees" B) <= 80))
Row 3 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq3") = T)
Row 4 : DC
Row 5 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq1") = F)

Case 5
Row 1 : (80 < ("RouteSegment Degrees" A))
Row 2 : (80 < ("RouteSegment Degrees" B))
Row 3 : DC
Row 4 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq2") = F)
Row 5 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq1") = F)

Case 6
Row 1 : ((58 < ("RouteSegment Degrees" A)) AND
  (("RouteSegment Degrees" A) < 70))
Row 2 : (80 < ("RouteSegment Degrees" B))
Row 3 : DC
Row 4 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq2") = F)
Row 5 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq1") = F)

Case 7
Row 1 : ((58 < ("RouteSegment Degrees" A)) AND
  (("RouteSegment Degrees" A) < 70))
Row 2 : ((70 <= ("RouteSegment Degrees" B)) AND
  (("RouteSegment Degrees" B) <= 80))
Row 3 : DC
Row 4 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq2") = F)
Row 5 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq1") = F)

Case 8
Row 1 : (80 < ("RouteSegment Degrees" A))
Row 2 : ((("RouteSegment Degrees" B) > 58) AND
  (("RouteSegment Degrees" B) < 70))
Row 3 : DC
Row 4 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq2") = F)
Row 5 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq1") = F)

Case 9
Row 1 : ((70 <= ("RouteSegment Degrees" A)) AND
  (("RouteSegment Degrees" A) <= 80))
Row 2 : ((("RouteSegment Degrees" B) > 58) AND
  (("RouteSegment Degrees" B) < 70))
Row 3 : DC
Row 4 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq2") = F)
Row 5 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq1") = F)

Case 10
Row 1 : ((58 < ("RouteSegment Degrees" A)) AND
  (("RouteSegment Degrees" A) < 70))
Row 2 : ((("RouteSegment Degrees" B) > 58) AND
  (("RouteSegment Degrees" B) < 70))
Row 3 : DC
Row 4 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq2") = F)
Row 5 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq1") = F)

Case 11
Row 1 : (("RouteSegment Degrees" A) <= 58)
Row 2 : DC
Row 3 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq3") = F)
Row 4 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq2") = F)
Row 5 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq1") = F)

Case 12
Row 1 : ((70 <= ("RouteSegment Degrees" A)) AND
  (("RouteSegment Degrees" A) <= 80))
Row 2 : (80 < ("RouteSegment Degrees" B))
Row 3 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq3") = F)
Row 4 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq2") = F)
Row 5 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq1") = F)

Case 13
Row 1 : (80 < ("RouteSegment Degrees" A))
Row 2 : ((70 <= ("RouteSegment Degrees" B)) AND
  (("RouteSegment Degrees" B) <= 80))
Row 3 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq3") = F)
Row 4 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq2") = F)
Row 5 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq1") = F)
```

```
Case 14
Row 1 : ((70 <= ("RouteSegment Degrees" A)) AND
  (("RouteSegment Degrees" A) <= 80))
Row 2 : ((70 <= ("RouteSegment Degrees" B)) AND
  (("RouteSegment Degrees" B) <= 80))
Row 3 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq3") = F)
Row 4 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq2") = F)
Row 5 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq1") = F)

Case 15
Row 1 : (80 < ("RouteSegment Degrees" A))
Row 2 : (("RouteSegment Degrees" B) <= 58)
Row 3 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq3") = F)
Row 4 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq2") = F)
Row 5 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq1") = F)

Case 16
Row 1 : ((70 <= ("RouteSegment Degrees" A)) AND
  (("RouteSegment Degrees" A) <= 80))
Row 2 : (("RouteSegment Degrees" B) <= 58)
Row 3 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq3") = F)
Row 4 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq2") = F)
Row 5 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq1") = F)

Case 17
Row 1 : ((58 < ("RouteSegment Degrees" A)) AND
  (("RouteSegment Degrees" A) < 70))
Row 2 : (("RouteSegment Degrees" B) <= 58)
Row 3 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq3") = F)
Row 4 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq2") = F)
Row 5 : (((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq1") = F)


Stats for LatitudeEquivalent completeness checking:
Number of cases identified: 17
Processor time: user: 2 sec; system: 0 sec
 --------------------------------------------------------------
>
Fusion session over.
```

1.11 LatitudeEquivalent Consistency Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%cons LatitudeEquivalent env

LatitudeEquivalent is:
(PredicateTable
  [
    ((Row ("RouteSegment Degrees" A))
      [(\x.(x <= 58));DC;(\x.((58 < x) AND (x < 70)));DC;
        (\x.((70 <= x) AND (x <= 80)));DC]);
    ((Row ("RouteSegment Degrees" B))
      [DC;(\x.(x <= 58));DC;(\x.((x > 58) AND (x < 70)));DC
        ;(\x.((70 <= x) AND (x <= 80)))]);
    ((Row ((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq3"))
      [TRUE;TRUE;DC;DC;DC;DC]);
    ((Row ((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq2"))
      [DC;DC;TRUE;TRUE;DC;DC]);
    ((Row ((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq1"))
      [DC;DC;DC;DC;TRUE;TRUE])])

By definition a predicate table can not be inconsistent,
since it returns "T" for all columns.
Processor time: user: 0 sec; system: 0 sec
```

```
 -----------------------------------------------------------
>
Fusion session over.
```

1.12 LatitudeEquivalent Symmetry Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%sym LatitudeEquivalent env

LatitudeEquivalent is:
(PredicateTable
  [
    ((Row ("RouteSegment Degrees" A))
      [(\x.(x <= 58));DC;(\x.((58 < x) AND (x < 70)));DC;
        (\x.((70 <= x) AND (x <= 80)));DC]);
    ((Row ("RouteSegment Degrees" B))
      [DC;(\x.(x <= 58));DC;(\x.((x > 58) AND (x < 70)));DC
        ;(\x.((70 <= x) AND (x <= 80)))]);
    ((Row ((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq3"))
      [TRUE;TRUE;DC;DC;DC;DC]);
    ((Row ((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq2"))
      [DC;DC;TRUE;TRUE;DC;DC]);
    ((Row ((AllOf [A;B]) "LatChange Per10DLong LessThanOrEq1"))
      [DC;DC;DC;DC;TRUE;TRUE])])

The table is symmetric if the following condition is true:

((58 < ("RouteSegment Degrees" A)) OR
  ((NOT (58 < ("RouteSegment Degrees" A))) AND
    (("RouteSegment Degrees" B) > 58)))
  =
((58 < ("RouteSegment Degrees" B)) OR
  ((NOT (58 < ("RouteSegment Degrees" B))) AND
    (("RouteSegment Degrees" A) > 58)))

Processor time: user: 2 sec; system: 0 sec
 -----------------------------------------------------------
>
Fusion session over.
```

1.13 LongSameDirSepRequired Completeness Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%comp LongSameDirSepRequired env

LongSameDirSepRequired is:
```

```
(
  (Table
    [((Row ((AllOf [A;B]) IsSupersonic)) [TRUE;FALSE]);
      ((Row ((AllOf [A;B]) IsTurbojet)) [DC;TRUE])])
  [(ssSameDirLongSep (A , B));("turbojetSameDir LongSep" (A , B));
    (otherSameDirLongSep (A , B))])

The following cases
yield the default value of (otherSameDirLongSep (A , B))
Case 1
Row 1 : (((AllOf [A;B]) IsSupersonic) = F)
Row 2 : (((AllOf [A;B]) IsTurbojet) = F)


Stats for LongSameDirSepRequired completeness checking:
Number of cases identified: 1
Processor time: user: 0 sec; system: 0 sec
 -------------------------------------------------------------
>
Fusion session over.
```

1.14 LongSameDirSepRequired Consistency Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%cons LongSameDirSepRequired env

LongSameDirSepRequired is:
(
  (Table
    [((Row ((AllOf [A;B]) IsSupersonic)) [TRUE;FALSE]);
      ((Row ((AllOf [A;B]) IsTurbojet)) [DC;TRUE])])
  [(ssSameDirLongSep (A , B));("turbojetSameDir LongSep" (A , B));
    (otherSameDirLongSep (A , B))])

No inconsistencies
were found in the table.

Stats for LongSameDirSepRequired consistency checking:
Number of cases identified: 0
Processor time: user: 0 sec; system: 0 sec
 -------------------------------------------------------------
>
Fusion session over.
```

1.15 LongSameDirSepRequired Symmetry Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
```

```
Closing onlinespec.hpp
>%sym LongSameDirSepRequired env

LongSameDirSepRequired is:
(
  (Table
    [((Row ((AllOf [A;B]) IsSupersonic)) [TRUE;FALSE]);
      ((Row ((AllOf [A;B]) IsTurbojet)) [DC;TRUE])])
  [(ssSameDirLongSep (A , B));("turbojetSameDir LongSep" (A , B));
    (otherSameDirLongSep (A , B))])

Assumption:
(ssSameDirLongSep (A , B))
  =
(ssSameDirLongSep (B , A))

Assumption:
("turbojetSameDir LongSep" (A , B))
  =
("turbojetSameDir LongSep" (B , A))

Assumption:
(otherSameDirLongSep (A , B))
  =
(otherSameDirLongSep (B , A))

The table is symmetric.
Processor time: user: 1 sec; system: 0 sec
 ------------------------------------------------------------
>
Fusion session over.
```

1.16 OppDir NoLongSepPeriod Completeness Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%comp "OppDir NoLongSepPeriod" env

"OppDir NoLongSepPeriod" is:
(
  (Table
    [((Row ((AllOf [A;B]) IsSupersonic)) [TRUE;FALSE]);
      ((Row ((AllOf [A;B]) IsTurbojet)) [DC;TRUE])])
  [("ssOppDir NoLongSepPeriod" (A , B));
    ("turbojetOppDir NoLongSepPeriod" (A , B));
    ("otherOppDir NoLongSepPeriod" (A , B))])

The following cases
yield the default value of ("otherOppDir NoLongSepPeriod" (A , B))
Case 1
Row 1 : (((AllOf [A;B]) IsSupersonic) = F)
Row 2 : (((AllOf [A;B]) IsTurbojet) = F)


Stats for "OppDir NoLongSepPeriod" completeness checking:
Number of cases identified: 1
Processor time: user: 1 sec; system: 0 sec
 ------------------------------------------------------------
>
Fusion session over.
```

1.17 OppDir NoLongSepPeriod Consistency Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%cons "OppDir NoLongSepPeriod" env

"OppDir NoLongSepPeriod" is:
(
  (Table
    [((Row ((AllOf [A;B]) IsSupersonic)) [TRUE;FALSE]);
      ((Row ((AllOf [A;B]) IsTurbojet)) [DC;TRUE])])
  [("ssOppDir NoLongSepPeriod" (A , B));
    ("turbojetOppDir NoLongSepPeriod" (A , B));
    ("otherOppDir NoLongSepPeriod" (A , B))])

No inconsistencies
were found in the table.

Stats for "OppDir NoLongSepPeriod" consistency checking:
Number of cases identified: 0
Processor time: user: 0 sec; system: 0 sec
 ------------------------------------------------------------
>
Fusion session over.
```

1.18 OppDir NoLongSepPeriod Symmetry Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%sym "OppDir NoLongSepPeriod" env

"OppDir NoLongSepPeriod" is:
(
  (Table
    [((Row ((AllOf [A;B]) IsSupersonic)) [TRUE;FALSE]);
      ((Row ((AllOf [A;B]) IsTurbojet)) [DC;TRUE])])
  [("ssOppDir NoLongSepPeriod" (A , B));
    ("turbojetOppDir NoLongSepPeriod" (A , B));
    ("otherOppDir NoLongSepPeriod" (A , B))])

Assumption:
("ssOppDir NoLongSepPeriod" (A , B))
  =
("ssOppDir NoLongSepPeriod" (B , A))

Assumption:
("turbojetOppDir NoLongSepPeriod" (A , B))
  =
("turbojetOppDir NoLongSepPeriod" (B , A))

Assumption:
```

```
("otherOppDir NoLongSepPeriod" (A , B))
  =
("otherOppDir NoLongSepPeriod" (B , A))

The table is symmetric.
Processor time: user: 0 sec; system: 0 sec
 -------------------------------------------------------------
>
Fusion session over.
```

1.19 ssOppDir NoLongSepPeriod Completeness Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%comp "ssOppDir NoLongSepPeriod" env

"ssOppDir NoLongSepPeriod" is:
(
  (Table [((Row (ReportedOverCommonPoint (A , B))) [TRUE;FALSE])]
    )
  [((ept (A , B)) , ((ept (A , B)) + 10));
    (((ept (A , B)) - 15) , ((ept (A , B)) + 15))])

The table is complete.
Stats for "ssOppDir NoLongSepPeriod" completeness checking:
Number of cases identified: 0
Processor time: user: 0 sec; system: 0 sec
 -------------------------------------------------------------
>
Fusion session over.
```

1.20 ssOppDir NoLongSepPeriod Consistency Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%cons "ssOppDir NoLongSepPeriod" env

"ssOppDir NoLongSepPeriod" is:
(
  (Table [((Row (ReportedOverCommonPoint (A , B))) [TRUE;FALSE])]
    )
  [((ept (A , B)) , ((ept (A , B)) + 10));
    (((ept (A , B)) - 15) , ((ept (A , B)) + 15))])

No inconsistencies
were found in the table.

Stats for "ssOppDir NoLongSepPeriod" consistency checking:
```

```
Number of cases identified: 0
Processor time: user: 0 sec; system: 0 sec
 ------------------------------------------------------------
>
Fusion session over.
```

1.21 ssOppDir NoLongSepPeriod Symmetry Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%sym "ssOppDir NoLongSepPeriod" env

"ssOppDir NoLongSepPeriod" is:
(
  (Table [((Row (ReportedOverCommonPoint (A , B))) [TRUE;FALSE])]
    )
  [((ept (A , B)) , ((ept (A , B)) + 10));
    (((ept (A , B)) - 15) , ((ept (A , B)) + 15))])

Assumption:
((ept (A , B)) , ((ept (A , B)) + 10))
  =
((ept (B , A)) , ((ept (B , A)) + 10))

Assumption:
(((ept (A , B)) - 15) , ((ept (A , B)) + 15))
  =
(((ept (B , A)) - 15) , ((ept (B , A)) + 15))


The table is symmetric.
Processor time: user: 0 sec; system: 0 sec
 ------------------------------------------------------------
>
Fusion session over.
```

1.22 ssSameDirLongSep Completeness Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%comp ssSameDirLongSep env

ssSameDirLongSep is:
(
  (Table
    [((Row (ssSubcondition (A , B))) [TRUE;TRUE]);
      ((Row ("SameOr Diverging Tracks" (A , B))) [TRUE;TRUE]);
      ((Row (ReportedOverCommonPoint (A , B))) [TRUE;DC]);
```

```
      ((Row ("Appropriate TimeSep AtCommon Point" (A , B))) [DC;TRUE]
        )]) [10;10;15])

The following cases
yield the default value of 15
Case 1
Row 1 : ((ssSubcondition (A , B)) = F)
Row 2 : DC
Row 3 : DC
Row 4 : DC

Case 2
Row 1 : ((ssSubcondition (A , B)) = T)
Row 2 : (("SameOr Diverging Tracks" (A , B)) = F)
Row 3 : DC
Row 4 : DC

Case 3
Row 1 : ((ssSubcondition (A , B)) = T)
Row 2 : (("SameOr Diverging Tracks" (A , B)) = T)
Row 3 : ((ReportedOverCommonPoint (A , B)) = F)
Row 4 : (("Appropriate TimeSep AtCommon Point" (A , B)) = F)


Stats for ssSameDirLongSep completeness checking:
Number of cases identified: 3
Processor time: user: 1 sec; system: 0 sec
 -----------------------------------------------------------
>
Fusion session over.
```

1.23 ssSameDirLongSep Consistency Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%cons ssSameDirLongSep env

ssSameDirLongSep is:
(
  (Table
    [((Row (ssSubcondition (A , B))) [TRUE;TRUE]);
      ((Row ("SameOr Diverging Tracks" (A , B))) [TRUE;TRUE]);
      ((Row (ReportedOverCommonPoint (A , B))) [TRUE;DC]);
      ((Row ("Appropriate TimeSep AtCommon Point" (A , B))) [DC;TRUE]
        )]) [10;10;15])

No inconsistencies
were found in the table.

Stats for ssSameDirLongSep consistency checking:
Number of cases identified: 0
Processor time: user: 0 sec; system: 0 sec
 -----------------------------------------------------------
>
Fusion session over.
```

1.24 ssSameDirLongSep Symmetry Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997
```

```
Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%sym ssSameDirLongSep env

ssSameDirLongSep is:
(
  (Table
    [((Row (ssSubcondition (A , B))) [TRUE;TRUE]);
      ((Row ("SameOr Diverging Tracks" (A , B))) [TRUE;TRUE]);
      ((Row (ReportedOverCommonPoint (A , B))) [TRUE;DC]);
      ((Row ("Appropriate TimeSep AtCommon Point" (A , B))) [DC;TRUE]
        )]) [10;10;15])


The table is symmetric.
Processor time: user: 4 sec; system: 0 sec
 ------------------------------------------------------------
>
Fusion session over.
```

1.25 ssSubcondition Completeness Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%comp ssSubcondition env

ssSubcondition is:
(PredicateTable
  [((Row ((AllOf [A;B]) IsLevel)) [TRUE;DC]);
    ((Row (SameMachNumber (A , B))) [TRUE;DC]);
    ((Row (SameType (A , B))) [DC;TRUE]);
    ((Row ((AllOf [A;B]) InCruiseClimb)) [DC;TRUE])])

The predicate is false
for the following cases:

Case 1
Row 1 : (((AllOf [A;B]) IsLevel) = F)
Row 2 : DC
Row 3 : ((SameType (A , B)) = F)
Row 4 : DC

Case 2
Row 1 : (((AllOf [A;B]) IsLevel) = F)
Row 2 : DC
Row 3 : DC
Row 4 : (((AllOf [A;B]) InCruiseClimb) = F)

Case 3
Row 1 : (((AllOf [A;B]) IsLevel) = T)
Row 2 : ((SameMachNumber (A , B)) = F)
Row 3 : DC
Row 4 : (((AllOf [A;B]) InCruiseClimb) = F)

Case 4
```

```
Row 1 : (((AllOf [A;B]) IsLevel) = F)
Row 2 : DC
Row 3 : ((SameType (A , B)) = T)
Row 4 : (((AllOf [A;B]) InCruiseClimb) = F)


Stats for ssSubcondition completeness checking:
Number of cases identified: 4
Processor time: user: 1 sec; system: 0 sec
 -------------------------------------------------------------
>
Fusion session over.
```

1.26 ssSubcondition Consistency Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%cons ssSubcondition env

ssSubcondition is:
(PredicateTable
  [((Row ((AllOf [A;B]) IsLevel)) [TRUE;DC]);
    ((Row (SameMachNumber (A , B))) [TRUE;DC]);
    ((Row (SameType (A , B))) [DC;TRUE]);
    ((Row ((AllOf [A;B]) InCruiseClimb)) [DC;TRUE])])

By definition a predicate table can not be inconsistent,
since it returns "T" for all columns.
Processor time: user: 0 sec; system: 0 sec
 -------------------------------------------------------------
>
Fusion session over.
```

1.27 ssSubcondition Symmetry Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%sym ssSubcondition env

ssSubcondition is:
(PredicateTable
  [((Row ((AllOf [A;B]) IsLevel)) [TRUE;DC]);
    ((Row (SameMachNumber (A , B))) [TRUE;DC]);
    ((Row (SameType (A , B))) [DC;TRUE]);
    ((Row ((AllOf [A;B]) InCruiseClimb)) [DC;TRUE])])

The table is symmetric
Processor time: user: 1 sec; system: 0 sec
```

```
 -----------------------------------------------------------
>
Fusion session over.
```

1.28 turbojetSameDir LongSep Completeness Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%comp "turbojetSameDir LongSep" env

"turbojetSameDir LongSep" is:
(
  (Table
    [((Row (MNPSCondition (A , B))) [TRUE;FALSE;TRUE;FALSE]);
      ((Row (WATRSCondition (A , B))) [TRUE;TRUE;FALSE;FALSE])]
    )
  [(MinAll (A , B));
    (Min
      {("WATRSSameDir LongSep" (A , B));
        ("genSameDir LongSep" (A , B))});
    (Min
      {("MNPSSameDir LongSep" (A , B));("genSameDir LongSep" (A , B))
        });("genSameDir LongSep" (A , B))])

The table is complete.
Stats for "turbojetSameDir LongSep" completeness checking:
Number of cases identified: 0
Processor time: user: 1 sec; system: 0 sec
 -----------------------------------------------------------
>
Fusion session over.
```

1.29 turbojetSameDir LongSep Consistency Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%cons "turbojetSameDir LongSep" env

"turbojetSameDir LongSep" is:
(
  (Table
    [((Row (MNPSCondition (A , B))) [TRUE;FALSE;TRUE;FALSE]);
      ((Row (WATRSCondition (A , B))) [TRUE;TRUE;FALSE;FALSE])]
    )
  [(MinAll (A , B));
    (Min
      {("WATRSSameDir LongSep" (A , B));
        ("genSameDir LongSep" (A , B))});
```

```
    (Min
      {("MNPSSameDir LongSep" (A , B));("genSameDir LongSep" (A , B))
        });("genSameDir LongSep" (A , B))])

No inconsistencies
were found in the table.

Stats for "turbojetSameDir LongSep" consistency checking:
Number of cases identified: 0
Processor time: user: 1 sec; system: 0 sec
 ------------------------------------------------------------
>
Fusion session over.
```

1.30 turbojetSameDir LongSep Symmetry Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%sym "turbojetSameDir LongSep" env

"turbojetSameDir LongSep" is:
(
  (Table
    [((Row (MNPSCondition (A , B))) [TRUE;FALSE;TRUE;FALSE]);
      ((Row (WATRSCondition (A , B))) [TRUE;TRUE;FALSE;FALSE])]
    )
  [(MinAll (A , B));
    (Min
      {("WATRSSameDir LongSep" (A , B));
        ("genSameDir LongSep" (A , B))});
    (Min
      {("MNPSSameDir LongSep" (A , B));("genSameDir LongSep" (A , B))
        });("genSameDir LongSep" (A , B))])

Assumption:
(MinAll (A , B))
  =
(MinAll (B , A))

Assumption:
(Min
  {("WATRSSameDir LongSep" (A , B));("genSameDir LongSep" (A , B))}
  )
  =
(Min
  {("WATRSSameDir LongSep" (B , A));("genSameDir LongSep" (B , A))}
  )

Assumption:
(Min
  {("MNPSSameDir LongSep" (A , B));("genSameDir LongSep" (A , B))})
  =
(Min
  {("MNPSSameDir LongSep" (B , A));("genSameDir LongSep" (B , A))})

Assumption:
("genSameDir LongSep" (A , B))
  =
("genSameDir LongSep" (B , A))


The table is symmetric.
Processor time: user: 8 sec; system: 0 sec
 ------------------------------------------------------------
>
```

```
Fusion session over.
```

1.31 turbojetOppDir NoLongSepPeriod Completeness Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%comp "turbojetOppDir NoLongSepPeriod" env

"turbojetOppDir NoLongSepPeriod" is:
(
  (Table
    [((Row (MNPSCondition (A , B))) [TRUE;FALSE;TRUE;FALSE]);
      ((Row (WATRSCondition (A , B))) [TRUE;TRUE;FALSE;FALSE])]
    )
  [(UnionAll (A , B));
    (UnionOfRange
      {("WATRSOppDir NoLongSepPeriod" (A , B));
        ("genOppDir NoLongSep Period" (A , B))});
    (UnionOfRange
      {("MNPSOppDir NoLongSepPeriod" (A , B));
        ("genOppDir NoLongSep Period" (A , B))});
    ("genOppDir NoLongSep Period" (A , B))])

The table is complete.
Stats for "turbojetOppDir NoLongSepPeriod" completeness checking:
Number of cases identified: 0
Processor time: user: 1 sec; system: 0 sec
 -------------------------------------------------------------
>
Fusion session over.
```

1.32 turbojetOppDir NoLongSepPeriod Consistency Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%cons "turbojetOppDir NoLongSepPeriod" env

"turbojetOppDir NoLongSepPeriod" is:
(
  (Table
    [((Row (MNPSCondition (A , B))) [TRUE;FALSE;TRUE;FALSE]);
      ((Row (WATRSCondition (A , B))) [TRUE;TRUE;FALSE;FALSE])]
    )
  [(UnionAll (A , B));
    (UnionOfRange
      {("WATRSOppDir NoLongSepPeriod" (A , B));
        ("genOppDir NoLongSep Period" (A , B))});
    (UnionOfRange
```

```
      {("MNPSOppDir NoLongSepPeriod" (A , B));
        ("genOppDir NoLongSep Period" (A , B))});
    ("genOppDir NoLongSep Period" (A , B))])

No inconsistencies
were found in the table.

Stats for "turbojetOppDir NoLongSepPeriod" consistency checking:
Number of cases identified: 0
Processor time: user: 0 sec; system: 0 sec
 ------------------------------------------------------------
>
Fusion session over.
```

1.33 turbojetOppDir NoLongSepPeriod Symmetry Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%sym "turbojetOppDir NoLongSepPeriod" env

"turbojetOppDir NoLongSepPeriod" is:
(
  (Table
    [((Row (MNPSCondition (A , B))) [TRUE;FALSE;TRUE;FALSE]);
      ((Row (WATRSCondition (A , B))) [TRUE;TRUE;FALSE;FALSE])]
    )
  [(UnionAll (A , B));
    (UnionOfRange
      {("WATRSOppDir NoLongSepPeriod" (A , B));
        ("genOppDir NoLongSep Period" (A , B))});
    (UnionOfRange
      {("MNPSOppDir NoLongSepPeriod" (A , B));
        ("genOppDir NoLongSep Period" (A , B))});
    ("genOppDir NoLongSep Period" (A , B))])

Assumption:
(UnionAll (A , B))
  =
(UnionAll (B , A))

Assumption:
(UnionOfRange
  {("WATRSOppDir NoLongSepPeriod" (A , B));
    ("genOppDir NoLongSep Period" (A , B))})
  =
(UnionOfRange
  {("WATRSOppDir NoLongSepPeriod" (B , A));
    ("genOppDir NoLongSep Period" (B , A))})

Assumption:
(UnionOfRange
  {("MNPSOppDir NoLongSepPeriod" (A , B));
    ("genOppDir NoLongSep Period" (A , B))})
  =
(UnionOfRange
  {("MNPSOppDir NoLongSepPeriod" (B , A));
    ("genOppDir NoLongSep Period" (B , A))})

Assumption:
("genOppDir NoLongSep Period" (A , B))
  =
("genOppDir NoLongSep Period" (B , A))


The table is symmetric.
```

```
Processor time: user: 9 sec; system: 0 sec
 ------------------------------------------------------------
>
Fusion session over.
```

1.34 MNPSSameDir LongSep Completeness Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%comp "MNPSSameDir LongSep" env

"MNPSSameDir LongSep" is:
(
  (Table
    [
      ((Row ("Appropriate TimeSep AtCommon Point" (A , B)))
        [TRUE;TRUE;TRUE;TRUE;TRUE]);
      ((Row ("SameOr Diverging Tracks" (A , B)))
        [TRUE;TRUE;TRUE;TRUE;TRUE]);
      (
        (Row
          ((Mach (FirstAircraft (A , B))) -
            (Mach (SecondAircraft (A , B)))))
        [(\x.(x > 0.06));(\x.((0.06 >= x) AND (x > 0.05)));
          (\x.((0.05 >= x) AND (x > 0.04)));
          (\x.((0.04 >= x) AND (x > 0.03)));
          (\x.((0.03 >= x) AND (x > 0.02)))])])
  [5;6;7;8;9;10])

Invoking interval checker...

Interval checker partitions the range into:
(
  ((Mach (FirstAircraft (A , B))) - (Mach (SecondAircraft (A , B)))
    ) > 0.06)
(
  (0.06 >=
    ((Mach (FirstAircraft (A , B))) -
      (Mach (SecondAircraft (A , B))))) AND
  (
    ((Mach (FirstAircraft (A , B))) -
      (Mach (SecondAircraft (A , B)))) > 0.05))
(
  (0.05 >=
    ((Mach (FirstAircraft (A , B))) -
      (Mach (SecondAircraft (A , B))))) AND
  (
    ((Mach (FirstAircraft (A , B))) -
      (Mach (SecondAircraft (A , B)))) > 0.04))
(
  (0.04 >=
    ((Mach (FirstAircraft (A , B))) -
      (Mach (SecondAircraft (A , B))))) AND
  (
    ((Mach (FirstAircraft (A , B))) -
      (Mach (SecondAircraft (A , B)))) > 0.03))
(
  (0.03 >=
    ((Mach (FirstAircraft (A , B))) -
      (Mach (SecondAircraft (A , B))))) AND
  (
    ((Mach (FirstAircraft (A , B))) -
      (Mach (SecondAircraft (A , B)))) > 0.02))
(
  ((Mach (FirstAircraft (A , B))) - (Mach (SecondAircraft (A , B)))
```

```
    ) <= 0.02)

The following cases
yield the default value of 10
Case 1
Row 1 : (("Appropriate TimeSep AtCommon Point" (A , B)) = F)
Row 2 : DC
Row 3 : DC

Case 2
Row 1 : (("Appropriate TimeSep AtCommon Point" (A , B)) = T)
Row 2 : (("SameOr Diverging Tracks" (A , B)) = F)
Row 3 : DC

Case 3
Row 1 : (("Appropriate TimeSep AtCommon Point" (A , B)) = T)
Row 2 : (("SameOr Diverging Tracks" (A , B)) = T)
Row 3 : (
  ((Mach (FirstAircraft (A , B))) - (Mach (SecondAircraft (A , B)))
    ) <= 0.02)


Stats for "MNPSSameDir LongSep" completeness checking:
Number of cases identified: 3
Processor time: user: 1 sec; system: 0 sec
 ------------------------------------------------------------
>
Fusion session over.
```

1.35 MNPSSameDir LongSep Consistency Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%cons "MNPSSameDir LongSep" env

"MNPSSameDir LongSep" is:
(
  (Table
    [
      ((Row ("Appropriate TimeSep AtCommon Point" (A , B)))
        [TRUE;TRUE;TRUE;TRUE;TRUE]);
      ((Row ("SameOr Diverging Tracks" (A , B)))
        [TRUE;TRUE;TRUE;TRUE;TRUE]);
      (
        (Row
          ((Mach (FirstAircraft (A , B))) -
            (Mach (SecondAircraft (A , B)))))
        [(\x.(x > 0.06));(\x.((0.06 >= x) AND (x > 0.05)));
          (\x.((0.05 >= x) AND (x > 0.04)));
          (\x.((0.04 >= x) AND (x > 0.03)));
          (\x.((0.03 >= x) AND (x > 0.02)))])])
  [5;6;7;8;9;10])

Invoking interval checker...

Interval checker partitions the range into:
(
  ((Mach (FirstAircraft (A , B))) - (Mach (SecondAircraft (A , B)))
    ) > 0.06)
(
  (0.06 >=
    ((Mach (FirstAircraft (A , B))) -
      (Mach (SecondAircraft (A , B))))) AND
  (
    ((Mach (FirstAircraft (A , B))) -
```

```
      (Mach (SecondAircraft (A , B)))) > 0.05))
(
  (0.05 >=
    ((Mach (FirstAircraft (A , B))) -
      (Mach (SecondAircraft (A , B))))) AND
  (
    ((Mach (FirstAircraft (A , B))) -
      (Mach (SecondAircraft (A , B)))) > 0.04))
(
  (0.04 >=
    ((Mach (FirstAircraft (A , B))) -
      (Mach (SecondAircraft (A , B))))) AND
  (
    ((Mach (FirstAircraft (A , B))) -
      (Mach (SecondAircraft (A , B)))) > 0.03))
(
  (0.03 >=
    ((Mach (FirstAircraft (A , B))) -
      (Mach (SecondAircraft (A , B))))) AND
  (
    ((Mach (FirstAircraft (A , B))) -
      (Mach (SecondAircraft (A , B)))) > 0.02))
(
  ((Mach (FirstAircraft (A , B))) - (Mach (SecondAircraft (A , B)))
    ) <= 0.02)

No inconsistencies
were found in the table.

Stats for "MNPSSameDir LongSep" consistency checking:
Number of cases identified: 0
Processor time: user: 0 sec; system: 0 sec
 ------------------------------------------------------------
>
Fusion session over.
```

1.36 MNPSSameDir LongSep Symmetry Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%sym "MNPSSameDir LongSep" env

"MNPSSameDir LongSep" is:
(
  (Table
    [
      ((Row ("Appropriate TimeSep AtCommon Point" (A , B)))
        [TRUE;TRUE;TRUE;TRUE;TRUE]);
      ((Row ("SameOr Diverging Tracks" (A , B)))
        [TRUE;TRUE;TRUE;TRUE;TRUE]);
      (
        (Row
          ((Mach (FirstAircraft (A , B))) -
            (Mach (SecondAircraft (A , B)))))
        [(\x.(x > 0.06));(\x.((0.06 >= x) AND (x > 0.05)));
          (\x.((0.05 >= x) AND (x > 0.04)));
          (\x.((0.04 >= x) AND (x > 0.03)));
          (\x.((0.03 >= x) AND (x > 0.02)))])])
  [5;6;7;8;9;10])


The table is symmetric if the following condition(s) hold
(some conditions may overlap):

Condition 1:
```

```
(
  (
    ((Mach (FirstAircraft (A , B))) -
      (Mach (SecondAircraft (A , B)))) > 0.06) =
  (
    ((Mach (FirstAircraft (B , A))) -
      (Mach (SecondAircraft (B , A)))) > 0.06))

Condition 2:
(
  (
    (0.06 >=
      ((Mach (FirstAircraft (A , B))) -
        (Mach (SecondAircraft (A , B))))) AND
    (
      ((Mach (FirstAircraft (A , B))) -
        (Mach (SecondAircraft (A , B)))) > 0.05)) =
  (
    (0.06 >=
      ((Mach (FirstAircraft (B , A))) -
        (Mach (SecondAircraft (B , A))))) AND
    (
      ((Mach (FirstAircraft (B , A))) -
        (Mach (SecondAircraft (B , A)))) > 0.05)))

Condition 3:
(
  (
    (0.05 >=
      ((Mach (FirstAircraft (A , B))) -
        (Mach (SecondAircraft (A , B))))) AND
    (
      ((Mach (FirstAircraft (A , B))) -
        (Mach (SecondAircraft (A , B)))) > 0.04)) =
  (
    (0.05 >=
      ((Mach (FirstAircraft (B , A))) -
        (Mach (SecondAircraft (B , A))))) AND
    (
      ((Mach (FirstAircraft (B , A))) -
        (Mach (SecondAircraft (B , A)))) > 0.04)))

Condition 4:
(
  (
    (0.04 >=
      ((Mach (FirstAircraft (A , B))) -
        (Mach (SecondAircraft (A , B))))) AND
    (
      ((Mach (FirstAircraft (A , B))) -
        (Mach (SecondAircraft (A , B)))) > 0.03)) =
  (
    (0.04 >=
      ((Mach (FirstAircraft (B , A))) -
        (Mach (SecondAircraft (B , A))))) AND
    (
      ((Mach (FirstAircraft (B , A))) -
        (Mach (SecondAircraft (B , A)))) > 0.03)))

Condition 5:
(
  (
    (0.03 >=
      ((Mach (FirstAircraft (A , B))) -
        (Mach (SecondAircraft (A , B))))) AND
    (
      ((Mach (FirstAircraft (A , B))) -
        (Mach (SecondAircraft (A , B)))) > 0.02)) =
  (
    (0.03 >=
      ((Mach (FirstAircraft (B , A))) -
        (Mach (SecondAircraft (B , A))))) AND
    (
      ((Mach (FirstAircraft (B , A))) -
        (Mach (SecondAircraft (B , A)))) > 0.02)))

Processor time: user: 3 sec; system: 0 sec
 -----------------------------------------------------------
>
Fusion session over.
```

1.37 WATRSCondition Completeness Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%comp WATRSCondition env

WATRSCondition is:
(PredicateTable
  [
    ((Row ((AllOf [A;B]) EnterWATRSAirspaceAtSomeTime))
      [TRUE;TRUE]);
    ((Row ((AllOf [A;B]) IsWestOf60W)) [TRUE;DC]);
    ((Row ((AllOf [A;B]) InWATRSAirspace)) [DC;TRUE]);
    ((Row ((AllOf [A;B]) MachTechniqueUsed)) [TRUE;TRUE]);
    ((Row ((AllOf [A;B]) OnPublishedRoute)) [TRUE;TRUE]);
    ((Row ("SameOr Diverging Tracks" (A , B))) [TRUE;TRUE])])

The predicate is false
for the following cases:

Case 1
Row 1 : (((AllOf [A;B]) EnterWATRSAirspaceAtSomeTime) = F)
Row 2 : DC
Row 3 : DC
Row 4 : DC
Row 5 : DC
Row 6 : DC

Case 2
Row 1 : (((AllOf [A;B]) EnterWATRSAirspaceAtSomeTime) = T)
Row 2 : (((AllOf [A;B]) IsWestOf60W) = F)
Row 3 : (((AllOf [A;B]) InWATRSAirspace) = F)
Row 4 : DC
Row 5 : DC
Row 6 : DC

Case 3
Row 1 : (((AllOf [A;B]) EnterWATRSAirspaceAtSomeTime) = T)
Row 2 : (((AllOf [A;B]) IsWestOf60W) = T)
Row 3 : DC
Row 4 : (((AllOf [A;B]) MachTechniqueUsed) = F)
Row 5 : DC
Row 6 : DC

Case 4
Row 1 : (((AllOf [A;B]) EnterWATRSAirspaceAtSomeTime) = T)
Row 2 : (((AllOf [A;B]) IsWestOf60W) = F)
Row 3 : (((AllOf [A;B]) InWATRSAirspace) = T)
Row 4 : (((AllOf [A;B]) MachTechniqueUsed) = F)
Row 5 : DC
Row 6 : DC

Case 5
Row 1 : (((AllOf [A;B]) EnterWATRSAirspaceAtSomeTime) = T)
Row 2 : (((AllOf [A;B]) IsWestOf60W) = T)
Row 3 : DC
Row 4 : (((AllOf [A;B]) MachTechniqueUsed) = T)
Row 5 : (((AllOf [A;B]) OnPublishedRoute) = F)
Row 6 : DC

Case 6
Row 1 : (((AllOf [A;B]) EnterWATRSAirspaceAtSomeTime) = T)
Row 2 : (((AllOf [A;B]) IsWestOf60W) = F)
Row 3 : (((AllOf [A;B]) InWATRSAirspace) = T)
Row 4 : (((AllOf [A;B]) MachTechniqueUsed) = T)
```

```
Row 5 : (((AllOf [A;B]) OnPublishedRoute) = F)
Row 6 : DC

Case 7
Row 1 : (((AllOf [A;B]) EnterWATRSAirspaceAtSomeTime) = T)
Row 2 : (((AllOf [A;B]) IsWestOf60W) = T)
Row 3 : DC
Row 4 : (((AllOf [A;B]) MachTechniqueUsed) = T)
Row 5 : (((AllOf [A;B]) OnPublishedRoute) = T)
Row 6 : (("SameOr Diverging Tracks" (A , B)) = F)

Case 8
Row 1 : (((AllOf [A;B]) EnterWATRSAirspaceAtSomeTime) = T)
Row 2 : (((AllOf [A;B]) IsWestOf60W) = F)
Row 3 : (((AllOf [A;B]) InWATRSAirspace) = T)
Row 4 : (((AllOf [A;B]) MachTechniqueUsed) = T)
Row 5 : (((AllOf [A;B]) OnPublishedRoute) = T)
Row 6 : (("SameOr Diverging Tracks" (A , B)) = F)


Stats for WATRSCondition completeness checking:
Number of cases identified: 8
Processor time: user: 1 sec; system: 0 sec
 -------------------------------------------------------------
>
Fusion session over.
```

1.38 WATRSCondition Consistency Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%cons WATRSCondition env

WATRSCondition is:
(PredicateTable
  [
    ((Row ((AllOf [A;B]) EnterWATRSAirspaceAtSomeTime))
      [TRUE;TRUE]);
    ((Row ((AllOf [A;B]) IsWestOf60W)) [TRUE;DC]);
    ((Row ((AllOf [A;B]) InWATRSAirspace)) [DC;TRUE]);
    ((Row ((AllOf [A;B]) MachTechniqueUsed)) [TRUE;TRUE]);
    ((Row ((AllOf [A;B]) OnPublishedRoute)) [TRUE;TRUE]);
    ((Row ("SameOr Diverging Tracks" (A , B))) [TRUE;TRUE])])

By definition a predicate table can not be inconsistent,
since it returns "T" for all columns.
Processor time: user: 0 sec; system: 0 sec
 -------------------------------------------------------------
>
Fusion session over.
```

1.39 WATRSCondition Symmetry Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima
```

```
>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%sym WATRSCondition env

WATRSCondition is:
(PredicateTable
  [
    ((Row ((AllOf [A;B]) EnterWATRSAirspaceAtSomeTime))
      [TRUE;TRUE]);
    ((Row ((AllOf [A;B]) IsWestOf60W)) [TRUE;DC]);
    ((Row ((AllOf [A;B]) InWATRSAirspace)) [DC;TRUE]);
    ((Row ((AllOf [A;B]) MachTechniqueUsed)) [TRUE;TRUE]);
    ((Row ((AllOf [A;B]) OnPublishedRoute)) [TRUE;TRUE]);
    ((Row ("SameOr Diverging Tracks" (A , B))) [TRUE;TRUE])])

The table is symmetric
Processor time: user: 1 sec; system: 0 sec
 -------------------------------------------------------------
>
Fusion session over.
```

1.40 genSameDir LongSep Completeness Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%comp "genSameDir LongSep" env

"genSameDir LongSep" is:
(
  (Table
    [((Row ("SameOr Diverging Tracks" (A , B))) [TRUE;TRUE;TRUE])
      ;
      ((Row ((AllOf [A;B]) MachTechniqueUsed))
        [FALSE;TRUE;TRUE]);
      ((Row ((AtLeastOneOf [A;B]) InCruiseClimb))
        [FALSE;FALSE;FALSE]);
      ((Row (ReportedOverCommonPoint (A , B))) [TRUE;DC;DC]);
      ((Row ("Appropriate TimeSep AtCommon Point" (A , B)))
        [DC;TRUE;TRUE]);
      (
        (Row
          ((Mach (FirstAircraft (A , B))) -
            (Mach (SecondAircraft (A , B)))))
        [DC;(\x.(x > 0.6));(\x.((0.6 >= x) AND (x > 0.3)))])
      ]) [15;5;10;20])

Invoking interval checker...

Interval checker partitions the range into:
(
  ((Mach (FirstAircraft (A , B))) - (Mach (SecondAircraft (A , B)))
    ) > 0.6)
(
  (0.6 >=
    ((Mach (FirstAircraft (A , B))) -
      (Mach (SecondAircraft (A , B))))) AND
  (
    ((Mach (FirstAircraft (A , B))) -
      (Mach (SecondAircraft (A , B)))) > 0.3))
(
```

```
  ((Mach (FirstAircraft (A , B))) - (Mach (SecondAircraft (A , B)))
    ) <= 0.30)

The following cases
yield the default value of 20
Case 1
Row 1 : (("SameOr Diverging Tracks" (A , B)) = F)
Row 2 : DC
Row 3 : DC
Row 4 : DC
Row 5 : DC
Row 6 : DC

Case 2
Row 1 : (("SameOr Diverging Tracks" (A , B)) = T)
Row 2 : DC
Row 3 : (((AtLeastOneOf [A;B]) InCruiseClimb) = T)
Row 4 : DC
Row 5 : DC
Row 6 : DC

Case 3
Row 1 : (("SameOr Diverging Tracks" (A , B)) = T)
Row 2 : (((AllOf [A;B]) MachTechniqueUsed) = F)
Row 3 : (((AtLeastOneOf [A;B]) InCruiseClimb) = F)
Row 4 : ((ReportedOverCommonPoint (A , B)) = F)
Row 5 : DC
Row 6 : DC

Case 4
Row 1 : (("SameOr Diverging Tracks" (A , B)) = T)
Row 2 : (((AllOf [A;B]) MachTechniqueUsed) = T)
Row 3 : (((AtLeastOneOf [A;B]) InCruiseClimb) = F)
Row 4 : DC
Row 5 : (("Appropriate TimeSep AtCommon Point" (A , B)) = F)
Row 6 : DC

Case 5
Row 1 : (("SameOr Diverging Tracks" (A , B)) = T)
Row 2 : (((AllOf [A;B]) MachTechniqueUsed) = T)
Row 3 : (((AtLeastOneOf [A;B]) InCruiseClimb) = F)
Row 4 : DC
Row 5 : (("Appropriate TimeSep AtCommon Point" (A , B)) = T)
Row 6 : (
  ((Mach (FirstAircraft (A , B))) - (Mach (SecondAircraft (A , B)))
    ) <= 0.30)


Stats for "genSameDir LongSep" completeness checking:
Number of cases identified: 5
Processor time: user: 1 sec; system: 0 sec
 ------------------------------------------------------------
>
Fusion session over.
```

1.41 genSameDir LongSep Consistency Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%cons "genSameDir LongSep" env

"genSameDir LongSep" is:
(
  (Table
    [((Row ("SameOr Diverging Tracks" (A , B))) [TRUE;TRUE;TRUE])
```

```
      ;
      ((Row ((AllOf [A;B]) MachTechniqueUsed))
        [FALSE;TRUE;TRUE]);
      ((Row ((AtLeastOneOf [A;B]) InCruiseClimb))
        [FALSE;FALSE;FALSE]);
      ((Row (ReportedOverCommonPoint (A , B))) [TRUE;DC;DC]);
      ((Row ("Appropriate TimeSep AtCommon Point" (A , B)))
        [DC;TRUE;TRUE]);
      (
        (Row
          ((Mach (FirstAircraft (A , B))) -
            (Mach (SecondAircraft (A , B)))))
        [DC;(\x.(x > 0.6));(\x.((0.6 >= x) AND (x > 0.3)))])
      ]) [15;5;10;20])

Invoking interval checker...

Interval checker partitions the range into:
(
  ((Mach (FirstAircraft (A , B))) - (Mach (SecondAircraft (A , B)))
    ) > 0.6)
(
  (0.6 >=
    ((Mach (FirstAircraft (A , B))) -
      (Mach (SecondAircraft (A , B))))) AND
  (
    ((Mach (FirstAircraft (A , B))) -
      (Mach (SecondAircraft (A , B)))) > 0.3))
(
  ((Mach (FirstAircraft (A , B))) - (Mach (SecondAircraft (A , B)))
    ) <= 0.30)

No inconsistencies
were found in the table.

Stats for "genSameDir LongSep" consistency checking:
Number of cases identified: 0
Processor time: user: 0 sec; system: 0 sec
 ------------------------------------------------------------
>
Fusion session over.
```

1.42 genSameDir LongSep Symmetry Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%sym "genSameDir LongSep" env

"genSameDir LongSep" is:
(
  (Table
    [((Row ("SameOr Diverging Tracks" (A , B))) [TRUE;TRUE;TRUE])
      ;
      ((Row ((AllOf [A;B]) MachTechniqueUsed))
        [FALSE;TRUE;TRUE]);
      ((Row ((AtLeastOneOf [A;B]) InCruiseClimb))
        [FALSE;FALSE;FALSE]);
      ((Row (ReportedOverCommonPoint (A , B))) [TRUE;DC;DC]);
      ((Row ("Appropriate TimeSep AtCommon Point" (A , B)))
        [DC;TRUE;TRUE]);
      (
        (Row
          ((Mach (FirstAircraft (A , B))) -
            (Mach (SecondAircraft (A , B)))))
        [DC;(\x.(x > 0.6));(\x.((0.6 >= x) AND (x > 0.3)))])
```

```
      ]) [15;5;10;20])

The table is symmetric if the following condition(s) hold
(some conditions may overlap):

Condition 1:
(
  (
    ((Mach (FirstAircraft (A , B))) -
      (Mach (SecondAircraft (A , B)))) > 0.6) =
  (
    ((Mach (FirstAircraft (B , A))) -
      (Mach (SecondAircraft (B , A)))) > 0.6))

Condition 2:
(
  (
    (0.6 >=
      ((Mach (FirstAircraft (A , B))) -
        (Mach (SecondAircraft (A , B))))) AND
    (
      ((Mach (FirstAircraft (A , B))) -
        (Mach (SecondAircraft (A , B)))) > 0.3)) =
  (
    (0.6 >=
      ((Mach (FirstAircraft (B , A))) -
        (Mach (SecondAircraft (B , A))))) AND
    (
      ((Mach (FirstAircraft (B , A))) -
        (Mach (SecondAircraft (B , A)))) > 0.3)))

Processor time: user: 2 sec; system: 0 sec
 ------------------------------------------------------------
>
Fusion session over.
```

1.43 otherSameDirLongSep Completeness Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%comp otherSameDirLongSep env

otherSameDirLongSep is:
(
  (Table
    [((Row (ReportedOverCommonPoint (A , B))) [TRUE;DC]);
      ((Row ("SameOr Diverging Tracks" (A , B))) [TRUE;DC]);
      ((Row ((AllOf [A;B]) (IsOnRoute Routes3))) [DC;TRUE])]
    ) [15;20;30])

The following cases
yield the default value of 30
Case 1
Row 1 : ((ReportedOverCommonPoint (A , B)) = F)
Row 2 : DC
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes3)) = F)

Case 2
Row 1 : ((ReportedOverCommonPoint (A , B)) = T)
Row 2 : (("SameOr Diverging Tracks" (A , B)) = F)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes3)) = F)


Stats for otherSameDirLongSep completeness checking:
```

```
Number of cases identified: 2
Processor time: user: 0 sec; system: 0 sec
 -----------------------------------------------------------
>
Fusion session over.
```

1.44 otherSameDirLongSep Consistency Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%cons otherSameDirLongSep env

otherSameDirLongSep is:
(
  (Table
    [((Row (ReportedOverCommonPoint (A , B))) [TRUE;DC]);
      ((Row ("SameOr Diverging Tracks" (A , B))) [TRUE;DC]);
      ((Row ((AllOf [A;B]) (IsOnRoute Routes3))) [DC;TRUE])]
    ) [15;20;30])

Columns 1 and 2 conflict in the following:
Case 1
Row 1 : ((ReportedOverCommonPoint (A , B)) = T)
Row 2 : (("SameOr Diverging Tracks" (A , B)) = T)
Row 3 : (((AllOf [A;B]) (IsOnRoute Routes3)) = T)


Stats for otherSameDirLongSep consistency checking:
Number of cases identified: 1
Processor time: user: 1 sec; system: 0 sec
 -----------------------------------------------------------
>
Fusion session over.
```

1.45 otherSameDirLongSep Symmetry Check

```
Fusion - Version 1.0 Sep 26 1997 15:22:26
Copyright University of British Columbia, 1996, 1997

Type "%include " or type in S paragraphs directly
Type "%help" to see list of % commands.
search path: . /isd/usr/day/src/fusion /isd/usr/day/examples/SeparationMinima


>%include minima.s
Including /isd/usr/day/examples/SeparationMinima/minima.s
Including /isd/usr/day/src/fusion/startup.s
Closing startup.s
Including /isd/usr/day/src/fusion/table.s
Closing table.s
Closing onlinespec.hpp
>%sym otherSameDirLongSep env

otherSameDirLongSep is:
(
  (Table
    [((Row (ReportedOverCommonPoint (A , B))) [TRUE;DC]);
      ((Row ("SameOr Diverging Tracks" (A , B))) [TRUE;DC]);
      ((Row ((AllOf [A;B]) (IsOnRoute Routes3))) [DC;TRUE])]
    ) [15;20;30])
```

```
The table is symmetric.
Processor time: user: 0 sec; system: 0 sec
 ----------------------------------------------------------
>
Fusion session over.
```

Appendix C: A Brief Introduction to the S Notation

S[JDD94] is an ASCII-based formal description notation developed at the University of British Columbia to support the industrial application of formal methods. It is also used to support a variety of research initiatives in the area of formal methods such as the development of the analysis techniques and tools demonstrated by the example documented in this report.

1 Similarities to Code

Although based upon the formalism of typed predicate logic, much of the syntax of S is similar to the syntax of many general-purpose programming languages. One such example is the "if ... then ... else ..." construct provided in S for conditional expressions. The association of values denoted by S expressions with types is very similar to the association of data values with types in strongly typed programming languages such as Pascal, Ada and C++. The syntax and type system of S bears a particularly strong similarity to aspects of Standard ML [P91].

Just as software developers often use a compiler to check that their code conforms to the syntax and typechecking rules of the programming language, a formal description written in S can be parsed and checked automatically by a software tool for conformance to the syntax and typechecking rules of S. This function is performed by a tool named "Fuss" developed at the University of British Columbia. Getting an S specification to be parsed by Fuss without errors is very much like getting a software package to "clean compile". "Fusion", the analysis tool used to generate the results reported in Appendix B, is an extension of Fuss developed by Nancy Day at the University of British Columbia.

There is a limited sense in which a formal description can evaluated or partially evaluated in a manner conceptually similar to the evaluation of expressions by an evaluation function for a functional programming language, e.g., a Lisp interpreter. Limitations of this sense of evaluation are related, in part, to some of the fundamental differences between S and "code" as discussed below. This includes limitations on the evaluation of S expressions that express quantification over infinite types. A subset of S could be used as a very simple, but useful, functional programming language. Various forms of evaluating formal descriptions written in S are being investigated at the University of British Columbia. One such approach is built into the "Fusion" tool used to generate the results reported in Appendix B.

2 Differences from Code

In spite of these similarities with various programming languages, S is a mathematical notation with fundamental differences from most general-purpose programming languages.

Unlike programming languages (except a specialized family of "purely functional" programming languages such as variants of Lisp), there is no sense of dynamics in S such as variables whose values may change over time. There are well-known methods of using predicate logic based notations to formally describe systems with dynamic behaviour, but this is a matter of how the formal description is interpreted as the description of the dynamic behaviour of a system. Discussion of how S may be used to specify dynamic behaviour is outside the scope of this brief description of S since the formalization of the NAT separation minima is ultimately a matter of defining a static condition represented by the predicate "AreSeparated" rather than dynamic behaviour.

Another fundamental difference from most general-purpose programming languages is the ability in S to express "quantification" (e.g., "for all", "there exists") over all values of a given type. The distinction between requirements specification and design is often characterized as the difference between describing *what* is required and *how* this requirement is to be realized. Quantification is often invaluable as a means of expressing "what" rather than "how". Quantification is also valuable for expressing global properties or constraints at the requirements level. In the formalization of the NAT separation minima, quantification is used to express environment assumptions as global constraints.

A third fundamental way in which S differs from most general-purpose programming languages is its ability to support "uninterpreted" types and constants. For most general-purpose programming languages, software is a description of a series of completely defined operations on patterns of bits. However, it is often useful, such as when specifying the requirements of system, to develop a formal description on top of a set of uninterpreted types and constants. In most programming languages, a user-defined data type must be defined in such a way that the compiler or interpreter can realize instances of this data type as a patterns of bits. But in S, as well as several other formal description notations, it is possible to simply introduce the name of a user-defined type without details of its realization. Similarly, it is possible in S, to introduce a constant by providing its name and its type without details of its realization.

Uninterpreted types and uninterpreted constants are used in the formalization of the NAT separation minima to match the level of abstraction used in the source documents. In principle, an implementation of the NAT separation minima could be produced by a translation of the formalization into the notation of a programming language along with instantiations of the uninterpreted types and constants. The uninterpreted types and uninterpreted constants used in the NAT separation minima are regarded as common vocabulary for users of this formalization. They are regarded as common vocabulary in the sense that it is assumed that its users would know how to correctly instantiate the uninterpreted types and uninterpreted constants with suitable implementations.

3 Declaration and Definition of Types

A *type declaration* in S may be used to introduce one or more new types. For example, the paragraph,

```
: flight, segment;
```

is an example of a type declaration. This example introduces two new types, "flight" and "segment". Types introduced by means of a type declaration are "uninterpreted" in the sense that nothing is said about their composition. As mentioned earlier, an uninterpreted type represents part of what is assumed to be the common vocabulary of the intended users of the specification.

In addition to type declarations, it is possible to define new types. A *type definition* in S is similar to, but more general than, the notion of an enumerated type in some programming languages such as Ada. The formalization of the NAT separation minima contains just one type definition which looks very much like an enumerated type in a programming language:

```
: location := Azores | BDA | CAN | Caribbean | IberianPenisula
            | Iceland | Scandinavia | UnitedKingdom | USA ;
```

This type definition is used to introduce a new type named "location" and a set of constants corresponding to the names of all of locations explicitly mentioned in the separation minima.

More advanced forms of type definition allow a type to be defined as the range of a set of functions called "constructors". Under certain conditions, a type definition may be recursive allowing recursive data types such as a "binary tree" to be defined. Declared and defined types may also be parameterized by type parameters. These more advanced forms of type definition are not used in the formalization of the NAT separation minima.

In addition to type declarations and type definitions, S provides a construct for the introduction of *type abbreviations*. In this case, a new type is not declared or defined - instead, a new name is introduced as an abbreviation for a potentially more complicated type expression or else as an alias for the name of very general type. For example, the type abbreviation,

```
: time == num;
```

results in the introduction of "time" as an alias for "num". Type abbreviations are used mainly to enhance the readability of a specification.

4 Type Expressions

Type expressions are used within S specifications to indicate the types of constants and variables. A type expression may be the name of a previously declared or defined type or one of the built-in types of S such as "bool" (Boolean values) or "num" (natural numbers). A type expression may also be the name of a type parameter; this is similar to the concept of a generic type in some programming languages such as Ada. Otherwise, a type expression is the application of a type operator to one or more simpler type expressions.

The formalization of the NAT separation minima uses three different type operators, namely, "->", "#" and "set".

The infix type operator "->" is used to specify function types. For instance, the type expression "flight -> bool" is used to denote the type of a function whose domain and range are the uninterpreted type "flight" and the built-in type "bool" respectively.

The infix type operator "#" is used to specify Cartesian products, that is, types whose members are pairs of elements. For instance, the type expression "flight # flight" is used in the formalization of the NAT separation minima to denote a type whose members are pairs of flights.

The postfix type operator "set" is used to denote a type whose members are sets of elements of some other type. For example, the type expression "(num) set" denotes the type whose members are sets of whole numbers, i.e., ... -2, -1, 0, 1, 2, ...

Type expressions may be combined using type operators to make more complex type expressions. For example, the type expression "(flight # flight) -> bool" denotes a function type whose domain is a Cartesian product, namely, "flight # flight", and whose range is "bool". Another example is the type expression "(location # location) set" which denotes a type whose values are sets of pairs of locations.

5 Declaration and Definition of Constants

In the description of many programming languages, the term "constant" is usually used to refer to the name associated with a fixed data value. In the definition of S, as in the definition of many other notations based on predicate logic, the term "constant" is used in a more general way to refer to the name of any fixed value including values which are functions types, i.e., values whose type is denoted by a type expression of the form "... -> ...".

Thus, the term "constant" in a description of S refers is used more generally than for just names of "data values". The term "function" is often used to refer to constants whose values are function types. The term "predicate" is used to refer specifically to functions whose range is "bool". In other words, functions are a particular class of constants and predicates are a particular class of functions.

The formalization of the NAT separation minima introduces a number of constants. Many of these constants are introduced in the form of tables. The name of each table in Appendix B is the name of a constant.

A number of other constants are also introduced in this formalization directly in S notation - either in the form of a constant declaration or a constant definition.

A *constant declaration* is used to introduce an uninterpreted constant. For example, the constant declaration,

```
IsSupersonic: (flight -> bool);
```

introduces "IsSupersonic" as the name of a function whose domain and range are "flight" and "bool" respectively. This is an example of a predicate since the range of this function is "bool".

As mentioned before, an uninterpreted constant represents part of what is assumed to be the common vocabulary of the intended users of the specification. In the case of "IsSupersonic", the introduction of this function as an uninterpreted constant is based on an assumption that the intended users of the NAT separation minima all understand that the value of the attribute "IsSupersonic" can be determined *somehow* from the representation of a flight -- but the details of the representation of the flight and how this attribute may be obtained is a design detail beneath the desired level of abstraction for this specification.

Similarly, the constant declaration,

```
RouteDeparture : (flight -> location);
```

introduces what may be regarded informally as the name of an attribute of a flight, i.e., "RouteDeparture". As in the case of "IsSupersonic", details of how the value of this attribute is obtained from the representation of a flight is intentionally excluded from this formalization.

Other constants are introduced in the formalization by means of *constant definitions*. For example,

```
Routes2 := {(USA,Caribbean);(Can,Caribbean);(BDA,Caribbean)};
```

defines the constant "Routes2" in terms of a set of pairs of locations. Another instance of a constant definition,

```
IsOnRoute (R:(location#location) set) (X:flight) :=
    ((RouteDeparture (X), RouteDestination (X)) In R) OR
```

```
        (RouteDestination (X), RouteDeparture (X)) In R);
```

introduces a function which may be used to determine if the departure location and destination location of a flight (the second parameter) is within a particular set of routes (the first parameter). The above definition uses an infix predicate named "In" to determine if an element is a member of set, i.e., set membership. The above definition also uses the built-in infix logical operator, "OR".

The above definition of "IsOnRoute" illustrates a feature of S which allows the parameters of functions to be separated so that they can be "provided one at a time". The first parameter of "IsOnRoute", namely, "R", is separated from its second parameter, "X". Functions of this form are known as "curried functions". This is different than grouping the parameters of a function together , e.g., "(R,X)", so that they must be "provided all at once". While this is not available in most programming languages, Standard ML allows for the definition of curried functions.

Curried functions are particularly useful in situations where a general-purpose function is partially evaluated to yield a function for a more specific purpose. For instance, the expressions "IsOnRoutes Routes1" and "IsOnRoutes Routes2" denotes two different functions. The former is a function for determining whether a flight falls within to the set of routes denoted by the constant "Routes1" while the latter corresponds to a function for determining whether a flight falls within to the set of routes denoted by "Routes2".

S allows constants to be introduced as infix functions. While the formalization of the NAT separation minima uses several "built-in" infix functions of S (e.g., "OR"), it does not include the introduction of any application specific infix functions.

Constants parameterized by types may also be declared or defined in a S specification. Several built-in polymorphic constants (i.e., constants parameterized by one or more types) are used in the formalization of the NAT separation minima; however, this formalization does not include the declaration or definition of any such constants.

Constants may be defined recursively (based on the constructors of a recursively defined type) but this feature of S is not used in the formalization of the NAT separation minima for any application specific constants.

6 S Expression Syntax

A very basic kind of S expression is the *prefix application* of a function (called the "operator") to another expression (called the "operand"). This is expressed by the juxtaposition of the expression denoting the operator with expression denoting the operand. For example, "MaxEarliestTime periods" denotes the application of the operator "MaxEarliestTime" to the operand "periods". Unlike most programming languages, it is not always necessary to enclose the operand within parentheses, i.e., "MaxEarliestTime (periods)" can be written as just "MaxEarliestTime periods". Matching parentheses may be placed around any S expression, but they are only necessary where required to eliminate ambiguity in the parsing of expressions. For instance, the parentheses in the expression "ABS (2 - 1)" are necessary for this expression to be parsed and typechecked successfully.

If a function has been introduced as an infix function, e.g., "+", then it may be applied to a pair of expressions in the form of an *infix application*, e.g., "1 + 2".

A third kind of function application is *post-fix application*. In the definition of "IsOnRoute", the expression "RouteDeparture (X)" could have been written "X.RouteDeparture". This expression may informally be understood as a reference to the "RouteDeparture" attribute of the parameter "X". Actually, this expression is merely the post-fix application of the function "RouteDeparture" to this parameter. It is semantically equivalent to the prefix application of this function to "X", i.e., "RouteDeparture (X)". Post-fix function application is merely "syntactic sugar" but it has been found to be helpful as a means of improving the readability of a formal specification especially for readers of the specification who are familiar with programming notations that use "dot notation" to refer to components of records or objects.

Another form of S expression is *universal quantification* which is used to express the condition that a specific property is true for every value of a specific type. For example, the environmental assumptions section of the formalization includes the assertion,

forall (A:flight). NOT (IsLevel (A) and InCruiseClimb (A))

that all flights cannot both satisfy (simultaneously) the conditions represented by the predicates "IsLevel" and "InCruiseClimb". S also provides a complementary form of quantification called *existential quantification* to assert that at least one value of the specified type satisfies the property.

S includes special syntax for denoting pairs, e.g., "(1,2)" and sets "{1;2;3}". This illustrated by the definition of "Routes2" where the right hand side of this definition is a set of pairs of locations.

To allow S specifications to be given a code-like appearance, the syntax of S also includes special syntax for conditional expressions, i.e., "if ... then ... else" and "if ... then ...".

Another code-like construct is the let-definition construct illustrated below by the definition of the predicate "WithinOppDirNoLongSepPeriod",

```
WithinOppDirNoLongSepPeriod (A,B,t) :=
    let timePeriod := "OppDir NoLongSepPeriod" (A,B) in
    (StartTime(timePeriod) <= t) AND (t <= EndTime(timePeriod));
```

where the right hand side of the definition of "WithinOppDirNoLongSepPeriod" is semantically equivalent to the following expression:

```
(StartTime("OppDir NoLongSepPeriod" (A,B)) <= t) AND
(t <= EndTime("OppDir NoLongSepPeriod" (A,B)));
```

7 Names of Types, Variables and Constants

Another useful feature of S is the ability to use phrases such as "this is a very long name" as the names of types, variables and constants, in addition to non-quoted identifiers. The use of this feature in the NAT separation minima is illustrated above by the use of the constant "OppDir NoLongSepPeriod" whose name includes a space character. The spaces in the names of these constants have no logical significance -- they are just names -- but these spaces are used, in part, to make the specification more readable. This feature of S has proven very useful as a means of achieving a higher degree of consistency in appearance between a formal specification expressed in S and other specification-related documentation.

8 Built-in Types and Constants

The formalization of the NAT separation minima involves a modest number of types and constants which, for the purposes of this report, may be described as "built-in" elements of S.

Technically, only a very small number of types and constants are fundamental elements of the notation. Most of the types and constants described here as "built-in" are really just elements of a standard library of very basic types and constants that underlies the NAT separation minima.

The dependence of the NAT separation minima formalization on this standard library is revealed by the second last line of Appendix A, where the library file "startup.s" is referenced by a "%include" directive. The other "%include" directive refers to some additional constants and types which support the tabular style of specification used in the formalization.

The only built-in types of S used in the formalization of the NAT separation minima are "bool" and "num".

Several built-in constants of S denoting arithmetic/logical functions are used with this formalization. These include "-" (subtraction), "+", ">", "<=", "<", "NOT", "OR" and "AND".