z

Quality Management Report

Traffic Collision Avoidance System - AirborneCPS: Decider Project

Prepared By: Anthony Disotell, Mike Palmateer, Alec Goldstein, Alex Voytovich
Position: Project managers
Date: November 2, 2018
Version No: v1.1

Table Of Contents

1. Quality Management Approach

1.1 Component under test

Identification	Attributes	Operations
Decider	recommendationRangeLock positiveRecommendationRange RecommendationRange negativeRecommendationRange kMinGaugeVerticalVelocity_ kMaxGaugeVerticalVelocity_ kProtectionVolumeRadius_ kAlim350_ kAlim400_ kAlim600_ kAlim700_ kAltitudeAlim350Threshold_ kAltitudeAlim400Threshold_ kAltitudeAlim600Threshold_ kVerticalVelocityClimbDescendDelta_ activeConnections thisAircraft tempSense	analyze - critical getThreatClassStr - non-critical determineActionRequired - critical determineThreatClass - critical tauPassesTAThreshold - semi-critical tauPassesRAThreshold - semi-critical determinResolutionSense - non-critical getAlimFt - semi-critical getRAZthrFt - semi-critical getTAZthrFt - semi-critical getRADmodNmi - semi-critical getTADmodNmi - semi-critical getModTauS - semi-critical getRecRangePair - critical getVVelForAlim - critical

1.2 Purpose of Component

The Decider is a crucial part of the Airborne-CPS. Working with other classes, such as Aircraft and Sense, it makes the decision if there is a threat present or not. Along with determining whether there is a threat, it also decides whether to ascend or descend based on the behavior of the detected intruding aircraft.

1.3 Assumptions

- The altitude in this case cannot be lower than zero feet or higher than forty-five thousand feet due to the legal altitude constraint.
- Altitude for a resolution advisory must be over 1,000 ft according to table 2 on page 23 of the specification
- Velocity is calculated in feet per minute based on table 3 on page 32 in the specification.
- Objects have no boundaries, therefore a boundary analysis was not determined.
- Values for 3.8 were derived from table 2 on page 23 and figure 13 on page 25 from the specification
- Values for vertical separation were derived from altitude limit from table 2 in the specification
- Tau values from 3.12 and 3.13 were derived from table 2 in the specification
- Values for vertical velocity in 3.15 are based on specification.

1.4 Glossary of Terms

Term	Definition
TCAS	Traffic Alert and Collision Avoidance System.
RA - Resolution advisory	An indication given by TCAS II to a flight crew that a vertical maneuver should, or in some cases should not, be performed to attain or maintain safe separation from a threat
TA - Traffic Advisory	An indication given by TCAS to the pilot when an aircraft has entered, or is projected to enter, the protected volume around the own aircraft.
VSL	Vertical speed indicator
CAS	Generic term for collision avoidance system
CPA	Closest point of approach as computed from a threat's range and range rate.
DMOD	Distance MODification
ZTHR	fixed altitude threshold
SL - Sensitivity Level	A value used in defining the size of the protected volume around own aircraft
Sense	
ALIM	Altitude Limit
Altitude	a distance measurement, usually in the vertical or "up" direction, between a reference datum and a point or object.
Tau	An approximation of the time, in seconds, to CPA or to the aircraft being at the same altitude
Accuracy	the degree to which the result of a measurement, calculation, or specification conforms to the correct value or a standard
Stability	the quality, state, or degree of being stable
Correctness	the quality or state of being free from error; accuracy

1.5 Project Partners

Partner	Responsibilities
Anthony Disotell Email: adisotel@oswego.edu	Specification Test Derivation, Formatting, Quality Metrics Design, Control flow, Data flow annotation, Test Suitability
Mike Palmateer Email: mpalmate@oswego.edu	Specification Test Derivation, Proofreading,Control flow, Data flow annotation, document formatting
Alec Goldstein Email: agoldste@oswego.edu	Specification Test Derivation; Quality Metrics Design,Control flow, Data flow annotation
Alex Voytovich Email: avoytovi@oswego.edu	Specification Test Derivation, Specification Analysis, Quality Metrics Design,Control flow, Data flow annotation

1.6 Plan for testing

Several different types of testing will be done during the course of the project. One such type testing is Specification-Based Test, using Equivalence Classes and the test cases they derive.

2. Quality Properties, Metrics, and Criteria

Quality Property	Definition	Metric	Criterion
Accuracy	Results from test cases fall within a certain margin of error from the accepted range.	Results of test cases performed with valid equivalence class representatives yield acceptable results based on the RA Detection algorithm.	Accuracy is achieved if: Test involved avoiding crashes will be successful at least 80% of time during testing.
Stability	Results from test cases will reach the same results after multiple test and different variations of testing.	Performing test cases multiple times both under the same conditions and multiple conditions will achieve the same passing results.	Stability is achieved if: Testing segments under different conditions and inputs will result in passing values
Correctness	Results from test cases adhere to acceptable values.	Perform test cases with valid equivalence class representatives to determine whether the calculated vector yields an acceptable thread classification.	Test are correct if: 100% of critical test pass. 50% of semi-critical test pass. 25% of non-critical test pass.

3. Specification-Based Test

3.1 Analyze

3.1.1 Equivalence Classes

[void] *Makes a call to determineActionRequired using passed in Aircraft object as parameter*

Parameter	Equivalence Class ID | Description	Representative
1. intruder	**1.1:** intr is an Aircraft object with first constructor	**1.1:** Aircraft(new Aircraft("blane", "12.4.3.9"))
	1.2: intr is an Aircraft object with second constructor	**1.2:** Aircraft("blah", "127.0.0.1")
	1.3: intr is an Aircraft object with third constructor	**1.3:** Aircraft("klane", "23.45.12.13", new LLA(), new Angle(45.7,AngleUnits.DEGREES), new Velocity(190.3, VelocityUnits.KNOTS))
	1.a: intr is an Aircraft object with invalid constructor	**1.a:** Aircraft(42, true)
	1.b: intr is a number	**1.b:** 235
	1.c: itnr ∈ { }	**1.c:** ∅
	1.d: intr is a char	**1.d:** 'r'
	1.e: intr is a string	**1.e:** "Die Hard"
	1.f: intr is a boolean	**1.f:** true

3.1.2 Test Cases

TC ID	intruder	Covered EC	Exp. Results
TC#1	Aircraft(new Aircraft("blane", "12.4.3.9"))	1.1	PASS
TC#2	Aircraft("blah", "127.0.0.1")	1.2	PASS
TC#3	Aircraft("klane", "23.45.12.13", new LLA(), new Angle(45.7,AngleUnits.DEGREES), new Velocity(190.3, VelocityUnits.KNOTS))	1.3	PASS
TC#4	Aircraft(42, true)	1.a	FAIL
TC#5	235	1.b	FAIL
TC#6	∅	1.c	FAIL

TC#7	‘r’	1.d	FAIL
TC#8	“Aircraft”	1.e	FAIL
TC#9	true	1.f	FAIL

3.2 getThreatClassStr

3.2.1 Equivalence Classes

[String] *returns a string literal of the threat class given an Airplane ThreatClassification*

Parameter	Equivalence Class ID | Description	Representative
1. threatClass	**1.1:** tc is NON_THREAT_TRAFFIC	**1.1:** NON_THREAT_TRAFFIC
	1.2: tc is PROXIMITY_INTRUDER_TRAFFIC	**1.2:** PROXIMITY_INTRUDER_TRAFFIC
	1.3: tc isTRAFFIC_ADVISORY	**1.3:** TRAFFIC_ADVISORY
	1.4: tc is RESOLUTION_ADVISORY	**1.4:** RESOLUTION_ADVISORY
	1.a: tc ∉ { NON_THREAT_TRAFFIC, PROXIMITY_INTRUDER_TRAFFIC, TRAFFIC_ADVISORY, RESOLUTION_ADVISORY }	**1.a:** INVALID_THREAT
	1.b: tc is a boolean	**1.b:** false
	1.c: tc is a number	**1.c:** 460
	1.d: tc ∈ { }	**1.d:** ∅
	1.e: tc is a string	**1.e:** “Threat”
	1.f: tc is a char	**1.f:** ‘T’

3.2.2 Test Cases

TC ID	threatClass	Covered EC	Exp. Results
TC#1	NON_THREAT_TRAFFIC	1.1	"Non-Threat Traffic"
TC#2	PROXIMITY_INTRUDER_TRAFFIC	1.2	"Proximity Intruder Traffic"
TC#3	TRAFFIC_ADVISORY	1.3	"Traffic Advisory"
TC#4	RESOLUTION_ADVISORY	1.4	"Resolution Advisory"
TC#2	INVALID_THREAT	1.a	FAIL
TC#3	false	1.b	FAIL
TC#4	460	1.c	FAIL
TC#5	∅	1.d	FAIL

TC#6	“Threat”	1.e	FAIL
TC#7	‘T’	1.f	FAIL

3.3 getAlimFt

3.3.1 Equivalence Classes

[int] *returns ALIM in feet given altitude in feet as a parameter. If below 1000 ft, returns -1*

Parameter	Equivalence Class ID | Description	Representative
1. altFT	**1.1:** alt ∈ $\mathbb{R}^+$ ≥ 1,000	**1.1:** 2,400
	1.2: alt ∈ $\mathbb{R}^+$ < 45,000	**1.2:** 40,000
	1.a: alt = 0	**1.a:** 0
	1.b: alt ∈ $\mathbb{R}$ < 1,000	**1.b:** -360
	1.c: alt ∈ $\mathbb{R}^+$ ≥ 45,000	**1.c:** 46,000
	1.d: alt ∈ { }	**1.d:** ⌀
	1.e: alt is a char	**1.e:** ‘e’
	1.f: alt is a string	**1.f:** “alim”
	1.g: alt is a boolean	**1.g:** true

3.3.2 Boundary Value Analysis

Parameter	Boundary Values	Test Case IDs
altFT	{999, 1,000, 1,001}	TC#1
	{45,000, 44,999, 44,998}	TC#2

3.3.3 Test Cases

TC ID	Altitude	Covered EC	Exp. Results
TC#1	2,400	1.1	300
TC#2	40,000	1.2	600
TC#2	0	1.a	FAIL
TC#3	-360	1.b	FAIL
TC#4	46,000	1.c	FAIL
TC#5	⌀	1.d	FAIL
TC#6	‘e’	1.e	FAIL
TC#7	“alim”	1.f	FAIL
TC#8	true	1.g	FAIL

3.4 getRADmodNmi

3.4.1 Equivalence Classes

[double] *given altitude as a parameter, returns RA DMOD in nautical miles*

Parameter	Equivalence Class ID | Description	Representative
1. altFt	**1.1:** alt ∈ $\mathbb{R}^+$ ≥ 1,000	**1.1:** 1,200
	1.2: alt ∈ $\mathbb{R}^+$ < 45,000	**1.2:** 19,000
	1.a: alt = 0	**1.a:** 0
	1.b: alt ∈ $\mathbb{R}$ < 1,000	**1.b:** 100
	1.c: alt ∈ $\mathbb{R}^+$ ≥ 45,000	**1.c:** 48,000
	1.d: alt ∈ { }	**1.d:** ∅
	1.e: alt is a char	**1.e:** 'a'
	1.f: alt is a string	**1.f:** "RA DMOD"
	1.g: alt is a boolean	**1.g:** true

3.4.2 Boundary Value Analysis

Parameter	Boundary Values	Test Case IDs
altFt	{999, 1,000, 1,001}	TC#1
	{45,000, 44,999, 44,998}	TC#2

3.4.3 Test Cases

TC ID	Altitude	Covered EC	Exp. Results
TC#1	1,200	1.1	0.20
TC#2	19,000	1.2	0.80
TC#3	0	1.a	FAIL
TC#4	100	1.b	FAIL
TC#5	48,000	1.c	FAIL
TC#6	∅	1.d	FAIL
TC#7	'a'	1.e	FAIL
TC#8	"RA DMOD"	1.f	FAIL
TC#9	true	1.g	FAIL

3.5 getTADmodNmi

3.5.1 Equivalence Classes

[double] *given altitude as a parameter, returns TA DMOD in nautical miles*

Parameter	Equivalence Class ID | Description	Representative
1. altFt	**1.1:** alt ∈ $\mathbb{R}^+$ > 0	**1.1:** 500
	1.2: alt ∈ $\mathbb{R}^+$ < 45,000	**1.2:** 29,000
	1.a: alt = 0	**1.a:** 0
	1.b: alt ∈ $\mathbb{R}^-$ < 0	**1.b:** -200
	1.c: alt ∈ $\mathbb{R}^+$ ≥ 45,000	**1.c:** 50,000
	1.d: alt ∈ { }	**1.d:** ∅
	1.e: alt is a char	**1.e:** 'h'
	1.f: alt is a string	**1.f:** "DMOD"
	1.g: alt is a boolean	**1.g:** false

3.5.2 Boundary Value Analysis

Parameter	Boundary Values	Test Case IDs
1. altFt	{0, 1, 2}	TC#1
	{45,000, 44,999, 44,998}	TC#2

3.5.3 Test Cases

TC ID	Altitude	Covered EC	Exp. Results
TC#1	500	1.1	0.30
TC#2	29,000	1.2	1.30
TC#3	0	1.a	FAIL
TC#4	-200	1.b	FAIL
TC#5	50,000	1.c	FAIL
TC#6	∅	1.d	FAIL
TC#7	'h'	1.e	FAIL
TC#8	"DMOD"	1.f	FAIL
TC#9	false	1.g	FAIL

3.6 getRAZthrFt

3.6.1 Equivalence Classes

[int] *given altitude as a parameter, returns RA ZTHR in feet. Returns -1 if below 1000 ft*

Parameter	Equivalence Class ID | Description	Representative
1. altFt	**1.1:** alt ∈ $\mathbb{R}^+$ ≥ 1,000	**1.1:** 15,000
	1.2: alt ∈ $\mathbb{R}^+$ < 45,000	**1.2:** 21,000
	1.a: alt = 0	**1.a:** 0
	1.b: alt ∈ $\mathbb{R}$ < 1,000	**1.b:** 250
	1.c: alt ∈ $\mathbb{R}^+$ ≥ 45,000	**1.c:** 45,001
	1.d: alt ∈ { }	**1.d:** ∅
	1.e: alt is a char	**1.e:** 'f'
	1.f: alt is a string	**1.f:** "ZTHR"
	1.g: alt is a boolean	**1.g:** true

3.6.2 Boundary Value Analysis

Parameter	Boundary Values	Test Case IDs
altFt	{999, 1,000, 1,001}	TC#1
	{45,000, 44,999, 44,998}	TC#2

3.6.3 Test Cases

TC ID	Altitude	Covered EC	Exp. Results
TC#1	15,000	1.1	600
TC#2	21,000	1.2	700
TC#3	0	1.a	FAIL
TC#4	250	1.b	FAIL
TC#5	45,001	1.c	FAIL
TC#6	∅	1.d	FAIL
TC#7	'f'	1.e	FAIL
TC#8	"ZTHR"	1.f	FAIL
TC#9	true	1.g	FAIL

3.7 getTAZthrFt

3.7.1 Equivalence Classes

[int] *given altitude as a parameter, returns TA ZTHR in feet. Returns -1 if below 1000 ft*

Parameter	Equivalence Class ID | Description	Representative
1. altFt	**1.1:** alt ∈ $\mathbb{R}^+$ > 0	**1.1:** 5,450
	1.2: alt ∈ $\mathbb{R}^+$ < 45,000	**1.2:** 6,000
	1.a: alt = 0	**1.a:** 0
	1.b: alt ∈ $\mathbb{R}^-$ < 0	**1.b:** -750
	1.c: alt ∈ $\mathbb{R}^+$ ≥ 45,000	**1.c:** 49,000
	1.d: alt ∈ { }	**1.d:** ∅
	1.e: alt is a char	**1.e:** 'e'
	1.f: alt is a string	**1.f:** "Hello"
	1.g: alt is a boolean	**1.g:** false

3.7.2 Boundary Value Analysis

Parameter	Boundary Values	Test Case IDs
altFT	{0, 1, 2}	TC#1
	{45,000, 44,999, 44,998}	TC#2

3.7.3 Test Cases

TC ID	Altitude	Covered EC	Exp. Results
TC#1	5,450	1.1	850
TC#2	6,000	1.2	850
TC#3	0	1.a	FAIL
TC#4	-750	1.b	FAIL
TC#5	49,000	1.c	FAIL
TC#6	∅	1.d	FAIL
TC#7	'e'	1.e	FAIL
TC#8	"Hello"	1.f	FAIL
TC#9	false	1.g	FAIL

3.8 getModTauS

3.8.1 Equivalence Classes

[double] *calculates modified tau*

Parameter	Equivalence Class ID | Description	Representative
1. rangeNmi	**1.1:** rangeNmi ∈ $\mathbb{R}^+ > 0$	**1.1:** 5
	1.2: rangeNmi ∈ $\mathbb{R}^+ \leq 14$	**1.2:** 14
	1.a: rangeNmi = 0	**1.a:** 0
	1.b: rangeNmi ∈ { }	**1.b:** ⌀
	1.c: rangeNmi is a char	**1.c:** 'q'
	1.d: rangeNmi is a string	**1.d:** "Steve"
	1.e: rangeNmi is a boolean	**1.e:** true
2. closureRateKnots	**2.1:** closureRateKnots ∈ $\mathbb{R}^+ > 0$	**2.1:** 100
	2.2: closureRateKnots ∈ $\mathbb{R}^+ \leq 1{,}200$	**2.2:** 1,100
	2.a: closureRateKnots ∈ { }	**2.a:** ⌀
	2.b: closureRateKnots is a char	**2.b:** 'd'
	2.c: closureRateKnots is a string	**2.c:** "pi"
	2.d: closureRateKnots is a boolean	**2.d:** false
	2.e: closureRateKnots = 0	**2.e:** 0
3. dmodNmi	**3.1:** dmodNmi ∈ $\mathbb{R}^+ \geq 0.20$	**3.1:** 0.34
	3.2: dmodNmi ∈ $\mathbb{R}^+ \leq 1.30$	**3.2:** 1.12
	3.a: dmodNmi ∈ { }	**3.a:** ⌀
	3.b: dmodNmi is a char	**3.b:** 's'
	3.c: dmodNmi is a string	**3.c:** "string"
	3.d: dmodNmi is boolean	**3.d:** false
	3.e: dmodNmi = 0	**3.e:** 0

3.8.2 Boundary Value Analysis

Parameter	Boundary Values	Test Case IDs
1. rangeNmi	{0, 1, 2}	TC#1
	{15, 14, 13}	TC#2
2. closureRateKnots	{0, 1, 2}	TC#1
	{1,201, 1,200, 1,999}	TC#2

3. dmodNmi	{0.19, 0.20, 0.21} {1.31, 1.30, 1.29}	TC#1 TC#2

3.8.3 Test Cases

TC ID	rangeNmi	closureRateKnots	dmodNmi	Covered EC	Exp. Results
TC#1	5	100	0.34	1.1, 2.1, 3.1	169.21392
TC#2	14,	1,100	1.12	1.2, 2.2, .3.2	42.71884
TC#3	0	0	0	1.a, 2.e, 3.e	FAIL
TC#4	∅	∅	∅	1.b,2.a,3.a	FAIL
TC#5	q	d	s	1.c,2.b,3.b	FAIL
TC#6	“Steve”	“pi”	“string”	1.d,2.c,3.c	FAIL
TC#7	true	false	false	1.e, 2.d,3.d	FAIL

3.9 getVvelForALIM

3.9.1 Equivalence Classes

[double] *returns vertical velocity needed to achieve ALIM*

Parameter	Equivalence Class ID | Description	MN<Kvvb NM'sbn Representative
1. sense	**1.1:** sense is UPWARDS	**1.1:** UPWARDS
	1.2: sense is DOWNWARDS	**1.2:** DOWNWARDS
	1.3: sense is UNKNOWN	**1.3:** UNKNOWN
	1.a: sense ∈ ℝ	**1.a:** 9
	1.b: sense is a char	**1.b:** 'k'
	1.c: sense is a string	**1.c:** "DOWNWARDS"
	1.d: sense ∈ { }	**1.d:** ⌀
	1.e: sense is a boolean	**1.e:** true
2. altFt	**2.1:** alt ∈ $\mathbb{R}^+$ > 1,000	**2.1:** 6,100
	2.2: alt ∈ $\mathbb{R}^+$ < 45,000	**2.2:** 23,500
	2.a: alt = 0	**2.a:** 0
	2.b: alt ∈ ℝ ≤ 1000	**2.b:** -2,700
	2.c: alt ∈ $\mathbb{R}^+$ ≥ 45,000	**2.c:** 55,000
	2.d: alt ∈ { }	**2.d:** ⌀
	2.e: alt is a char	**2.e:** 'w'
	2.f: alt is a string	**2.f:** "down"
	2.g: alt is a boolean	**2.g:** false
3. vsepAtCpaFt	**3.1:** vsep ∈ $\mathbb{R}^+$ ≥ 300	**3.1:** 400
	3.2: vsep ∈ ℝ ≤ 700	**3.2:** 600
	3.a: vsep = 0	**3.a:** 0
	3.b: vsep ∈ ℝ < 300	**3.b:** 200
	3.c: vsep ∈ $\mathbb{R}^+$ > 700	**3.c:** 900
	3.d: vsep ∈ { }	**3.d:** ⌀
	3.e: vsep is a char	**3.e:** 't'
	3.f: vsep is a string	**3.f:** "help"
	3.g: vsep is a boolean	**3.g:** false
4. intrProjAltFt	**4.1:** alt ∈ $\mathbb{R}^+$ > 1,000	**4.1:** 1,200
	4.2: ∈ $\mathbb{R}^+$ < 45,000	**4.2:** 36,000

	4.a: alt = 0 **4.b:** alt $\in \mathbb{R} \leq 1000$ **4.c:** alt $\in \mathbb{R}^+ \geq 45{,}000$ **4.d:** alt $\in \{\}$ **4.e:** alt is a char **4.f:** alt is a string **4.g:** alt is a boolean	**4.a:** 0 **4.b:** 800 **4.c:** 60,000 **4.d:** ⌀ **4.e:** 'p' **4.f:** "black" **4.g:** true
5. rangeTauS	**5.1:** tau $\in \mathbb{R}^+ \geq 15$ **5.2:** tau $\in \mathbb{R}^+ \leq 48$ **5.a:** tau = 0 **5.b:** tau $\in \mathbb{R} < 15$ **5.c:** tau $\in \mathbb{R}^+ > 48$ **5.d:** tau $\in \{\}$ **5.e:** tau is a char **5.f:** tau is a string **5.g:** tau is a boolean	**5.1:** 17 **5.2:** 22 **5.a:** 0 **5.b:** 11 **5.c:** 59 **5.d:** ⌀ **5.e:** 'n' **5.f:** "time" **5.g:** true

3.9.2 Boundary Value Analysis

Parameter	Boundary Values	Test Case IDs
1. altFt	{-1, 0, 1} {46,000, 44,999, 44,990}	TC#1 TC#2
2. vsepAtCpaFt	{299,300,301} {800,700,600}	TC#1 TC#2
3. intrProjAltFt	{999, 1,000, 1,001} {46,000, 44,999, 44,990}	TC#1 TC#2
4. rangeTauS	{14, 15, 17} {46, 48, 49}	TC#1 TC#2

3.9.3 Test Cases

TC ID	Sense	altFt	vsepAtCpaFt	intrProjAltFt	rangeTauS	Covered EC	Exp. Results
TC#1	UPWARDS	6,100	400	1,200	17	1.1,2.1,3.1,4.1,5.1	705.8823
TC#2	DOWNWARDS	23,500	600	36,000	22	1.2,2.2,3.2,4.2,5.2	-272.7273
TC#3	UNKNOWN	6,000	400	0	0	1.3,2.1,3.a,4.a,5.a	0
TC#4	9	-2,700	200	800	11	2.b,3.b,4.b,5.b	FAIL
TC#5	9	55,000	900	60,000	59	2.c,3.c,4.c,5.c	FAIL
TC#6	∅	∅	∅	∅	∅	1.e,2.d,3.d,4.d5.d	FAIL
TC#7	k	w	t	p	n	1.c,2.e,3.e,4.e,5.e	FAIL
TC#8	“DOWNWARDS”	“down”	“help”	“black”	“time”	1.d,2.f,3.f,4.f,5.f	FAIL
TC#9	true	false	false	true	true	1.f,2.g,3.g,4.g,5.g	FAIL

3.10 determineActionRequired

3.10.1 Equivalence Classes

[void] *given an intruder, determines if the intruder is a threat or not*

Parameter	Equivalence Class ID | Description	Representative
1. intruder	**1.1:** intr is an Aircraft object with first constructor	**1.1:** Aircraft(new Aircraft(“craft”, “12.34.25.47”))
	1.2: intr is an Aircraft object with second constructor	**1.2:** Aircraft(“plane”, “129.25.54.235”)
	1.3: intr is an Aircraft object with third constructor	**1.3:** Aircraft(“Intr”, “473.23.354.45”, new LLA(), new Angle(80.4,AngleUnits.DEGREES), new Velocity(23.0, VelocityUnits.KNOTS))
	1.a: intr is an Aircraft object with invalid constructor	**1.a:** Aircraft(“Some plane”)
	1.b: intr is a number	**1.b:** 500
	1.c: itnr ∈ { }	**1.c:** ∅
	1.d: intr is a char	**1.d:** ‘s’
	1.e: intr is a string	**1.e:** ‘Intruder’
	1.f: intr is a boolean	**1.f:** false

3.10.2 Test Cases

TC ID	Altitude	Covered EC	Exp. Results
TC#1	Aircraft(new Aircraft(“craft”, “12.34.25.47”))	1.1	PASS
TC#2	Aircraft(“plane”, “129.25.54.235”)	1.2	PASS
TC#3	Aircraft(“Intr”, “473.23.354.45”, new LLA(), new Angle(80.4,AngleUnits.DEGREES), new Velocity(23.0, VelocityUnits.KNOTS))	1.3	PASS
TC#4	Aircraft(“Some plane”)	1.a	FAIL
TC#5	500	1.b	FAIL
TC#6	∅	1.c	FAIL
TC#7	‘s’	1.d	FAIL
TC#8	“Intruder”	1.e	FAIL
TC#9	false	1.f	FAIL

3.11 determineThreatClass

3.11.1 Equivalence Classes

[Aircraft::ThreatClassification] *returns threat classification*

Parameter	Equivalence Class ID | Description	Representative
1. intrCopy	**1.1:** intr is an Aircraft object with first constructor	**1.1:** Aircraft(new Aircraft(“plane”, “129.0.7.10”))
	1.2: intr is an Aircraft object with second constructor	**1.2:** Aircraft(“air”, “136.23.87.10”)
	1.3: intr is an Aircraft object with third constructor	**1.3:** Aircraft(“flane”, “126.23.6.10”, new LLA(13.0, -80.0, 1439.89, AngelUnits.DEGREE, DistanceUnits.METERS), new Angle(47.9, AngleUnits.DEGREES), new Velocity(13.4, Velocity.KNOTS))

	1.a: intr is an Aircraft object with invalid constructor **1.b:** intr is a number **1.c:** itnr ∈ { } **1.d:** intr is a char **1.e:** intr is a string **1.f:** intr is a boolean	**1.a:** Aircraft(false) **1.b:** 235 **1.c:** ∅ **1.d:** 'r' **1.e:** 'Aircraft' **1.f:** true
2. conn	**2.1:** conn is a ResolutionConnection object **2.a:** conn is a ResolutionConnection object with an invalid constructor **2.b:** conn is a number **2.c:** conn ∈ { } **2.d:** conn is a char **2.e:** conn is a string **2.f:** conn is a boolean	**2.1:** ResolutionConnection("AB:CD:EF:12:34:56" "BA:DC:FE:21:43:65", "198.0.10.16", 1234, Aircraft("guy", "198.0.10.16") **2.a:** ResolutionConnection(45, true, 'c') **2.b:** 300 **2.c:** ∅ **2.d:** 'c' **2.e:** "Connect" **2.f:** true

3.11.2 Test Cases

TC ID	intrCopy	conn	Covered EC	Exp. Results
TC#1	Aircraft(new Aircraft("plane", "129.0.7.10"))	ResolutionConnection("AB:CD:EF:12:34:56" "BA:DC:FE:21:43:65", "198.0.10.16", 1234, Aircraft("guy", "198.0.10.16")	1.1, 2.1	TRAFFIC_ ADVISORY

TC#2	Aircraft(“air”, “136.23.87.10”)	ResolutionConnection(“AB:CD:EF:12:34:56” “BA:DC:FE:21:43:65”, “198.0.10.16”, 1234, Aircraft(“guy”, “198.0.10.16”)	1.1, 2.2	RESOLUTION_ ADVISORY
TC#3	Aircraft(“flane”, “126.23.6.10”, new LLA(13.0, -80.0, 1439.89, AngelUnits.DEGREE, DistanceUnits.METERS), new Angle(47.9, AngleUnits.DEGREES), new Velocity(13.4, Velocity.KNOTS))	ResolutionConnection(“AB:CD:EF:12:34:56” “BA:DC:FE:21:43:65”, “198.0.10.16”, 1234, Aircraft(“guy”, “198.0.10.16”)	1.1, 2.3	TRAFFIC_ ADVISORY
TC#4	Aircraft(false)	ResolutionConnection(45, true, ‘c’)	1.a, 2.a	FAIL
TC#5	235	‘c’	1.b, 2.d	FAIL
TC#6	∅	300	1.c, 2.b	FAIL
TC#7	‘r’	∅	1.d, 2.c	FAIL
TC#8	“Aircraft”	true	1.e, 2.f	FAIL
TC#9	true	“Connect”	1.f, 2.e	FAIL

3.12 TauPassesTAThreshold

3.12.1 Equivalence Classes

[bool] *Returns whether the supplied taus trigger a TA at this altitude*

Parameter	Equivalence Class ID | Description	Representative
1. altFt	**1.1:** alt ∈ $\mathbb{R}^+$ > 0	**1.1:** 5,300
	1.2: alt ∈ $\mathbb{R}^+$ < 45,000	**1.2:** 42,500
	1.a: alt = 0	**1.a:** 0
	1.b: alt ∈ $\mathbb{R}^-$ < 0	**1.b:** 2,000
	1.c: alt ∈ $\mathbb{R}^+$ ≥ 45,000	**1.c:** 48,000
	1.d: alt ∈ { }	**1.d:** ∅
	1.e: alt is a char	**1.e:** 'Y'
	1.f: alt is a string	**1.f:** "Altitude"
	1.g: alt is a boolean	**1.g:** false
2. modTauS	**2.1:** modTau ∈ $\mathbb{R}^+$ ≥ 20	**2.1:** 23
	2.2: modTau ∈ $\mathbb{R}^+$ ≤ 48	**2.2:** 45
	2.a: modTau = 0	**2.a:** 0
	2.b: modTau ∈ $\mathbb{R}^-$ < 20	**2.b:** 15
	2.c: modTau ∈ $\mathbb{R}^-$ > 48	**2.c:** 58
	2.d: modTau ∈ { }	**2.d:** ∅
	2.e: modTau is a char	**2.e:** 'P'
	2.f: modTau is a string	**2.f:** "Modified"
	2.g: modTau is a boolean	**2.g:** true
3. vertTauS	**3.1:** modTau ∈ $\mathbb{R}^+$ ≥ 20	**3.1:** 32
	3.2: modTau ∈ $\mathbb{R}^+$ ≤ 48	**3.2:** 42
	3.a: modTau = 0	**3.a:** 0
	3.b: modTau ∈ $\mathbb{R}^-$ < 20	**3.b:** 4
	3.c: modTau ∈ $\mathbb{R}^-$ > 48	**3.c:** 65
	3.d: modTau ∈ { }	**3.d:** ∅
	3.e: modTau is a char	**3.e:** 'G'
	3.f: modTau is a string	**3.f:** "Vertical"
	3.g: modTau is a boolean	**3.g:** false
4. vSepFt	**4.1:** vSep ∈ $\mathbb{R}^+$ ≥ 850	**4.1:** 940

	4.2: vSep ∈ $\mathbb{R}^+ \leq 1{,}200$ **4.a:** vSep = 0 **4.b:** vSep ∈ $\mathbb{R} < 850$ **4.c:** vSep ∈ $\mathbb{R} > 1{,}200$ **4.d:** vSep ∈ { } **4.e:** vSep is a char **4.f:** vSep is a string **4.g:** vSep is a boolean	**4.2:** 1,000 **4.a:** 0 **4.b:** 423 **4.c:** 1,450 **4.d:** ∅ **4.e:** 'L' **4.f:** "Vertical" **4.g:** true

3.12.2 Boundary Value Analysis

Parameter	Boundary Values	Test Case IDs
1. altFt	{0, 1, 2} {45,000, 44,999, 44,998}	TC#1 TC#2
2. modTauS	{19, 20, 21} {47, 48, 49}	TC#1 TC#2
3. vertTauS	{18, 20, 22} {47, 48, 50}	TC#1 TC#2
4. vSepFt	{849, 850, 851} {1,199, 1,200, 1,201}	TC#1 TC#2

3.12.3 Test Cases

TC ID	altFt	modTauS	vertTauS	vSeptFt	Covered EC	Exp. Results
TC#1	5,300	23	32	940	1.1, 2.1, 3.1, 4.1	true
TC#2	42,500	45	42	1,000	1.2, 2.2, 3.2, 4.2	true
TC#3	0	0	0	0	1.a, 2.a, 3.a, 4.a	FAIL
TC#4	2,000	15	4	423	1.b, 2.b, 3.b, 4.b	FAIL
TC#5	48,000	58	65	1,450	1.c, 2.c, 3.c, 4.c	FAIL
TC#6	∅	∅	∅	∅	1.d, 2.d, 3.d, 4.d	FAIL
TC#7	Y	P	G	L	1.e, 2.e, 3.e, 4.e	FAIL
TC#8	"Altitude"	"Modified"	"Vertical"	'Vertical"	1.f, 2.f, 3.f, 4.f	FAIL
TC#9	false	true	false	true	1.g, 2.g, 3.g, 4.g	FAIL

3.13 TauPassesRAThreshold

3.13.1 Equivalence Classes

[bool] *Returns whether the supplied taus trigger a TA at this altitude*

Parameter	Equivalence Class ID | Description	Representative
1. altFt	**1.1:** alt ∈ $\mathbb{R}^+$ > 0	**1.1:** 48
	1.2: alt ∈ $\mathbb{R}^+$ < 45,000	**1.2:** 42,000
	1.a: alt = 0	**1.a:** 0
	1.b: alt ∈ $\mathbb{R}$ < 0	**1.b:** -4,400
	1.c: alt ∈ $\mathbb{R}^+$ ≥ 45,000	**1.c:** 63,000
	1.d: alt ∈ { }	**1.d:** ⌀
	1.e: alt is a char	**1.e:** 'O'
	1.f: alt is a string	**1.f:** "Feet"
	1.g: alt is a boolean	**1.g:** false
2. modTauS	**2.1:** modTau ∈ $\mathbb{R}^+$ ≥ 15	**2.1:** 17
	2.2: modTau ∈ $\mathbb{R}^+$ ≤ 35	**2.2:** 32
	2.a: modTau = 0	**2.a:** 0
	2.b: modTau ∈ $\mathbb{R}$ < 15	**2.b:** 9
	2.c: modTau ∈ $\mathbb{R}$ > 35	**2.c:** 42
	2.d: modTau ∈ { }	**2.d:** ⌀
	2.e: modTau is a char	**2.e:** 'R'
	2.f: modTau is a string	**2.f:** "Tau"
	2.g: modTau is a boolean	**2.g:** false
3. vertTauS	**3.1:** modTau ∈ $\mathbb{R}^+$ ≥ 15	**3.1:** 19
	3.2: modTau ∈ $\mathbb{R}^+$ ≤ 35	**3.2:** 29
	3.a: modTau = 0	**3.a:** 0
	3.b: modTau ∈ $\mathbb{R}$ < 15	**3.b:** 12
	3.c: modTau ∈ $\mathbb{R}$ > 35	**3.c:** 89
	3.d: modTau ∈ { }	**3.d:** ⌀
	3.e: modTau is a char	**3.e:** 'S'
	3.f: modTau is a string	**3.f:** "vertTau"
	3.g: modTau is a boolean	**3.g:** true

4. vSepFt	**4.1:** vSep ∈ $\mathbb{R}^+ \geq 600$ **4.2:** vSep ∈ $\mathbb{R}^+ \leq 800$ **4.a:** vSep = 0 **4.b:** vSep ∈ $\mathbb{R} < 600$ **4.c:** vSep ∈ $\mathbb{R} > 800$ **4.d:** vSep ∈ { } **4.e:** vSep is a char **4.f:** vSep is a string **4.g:** vSep is a boolean	**4.1:** 670 **4.2:** 720 **4.a:** 0 **4.b:** 230 **4.c:** 920 **4.d:** ∅ **4.e:** 'W' **4.f:** "SepFt" **4.g:** true

3.13.2 Boundary Value Analysis

Parameter	Boundary Values	Test Case IDs
1. altFt	{999, 1,000, 1,001} {45,000, 44,999, 44,998}	TC#1 TC#2
2. modTauS	{14, 15, 16} {34, 35, 36}	TC#1 TC#2
3. vertTauS	{14, 15, 16} {34, 35, 36}	TC#1 TC#2
4. vSepFt	{599, 600, 601} {799, 800, 801}	TC#1 TC#2

3.13.3 Test Cases

TC ID	altFt	modTauS	vertTauS	vSepFt	Covered EC	Exp. Results
TC#1	48	17	19	670	1.1, 2.1, 3.1, 4.1	false
TC#2	42,000	32	29	720	1.2, 2.2, 3.2, 4.2	true
TC#3	0	0	0	0	1.a, 2.a, 3.a, 4.a	FAIL
TC#4	-4,400	9	12	230	1.b, 2.b, 3.b, 4.b	FAIL
TC#5	63,000	42	89	920	1.c, 2.c, 3.c, 4.c	FAIL
TC#6	∅	∅	∅	∅	1.d, 2.d, 3.d, 4.d	FAIL
TC#7	'O'	'R'	'S'	'W'	1.e, 2.e, 3.e, 4.e	FAIL
TC#8	"Feet"	"Tau"	"vertTau"	"SepFt"	1.f, 2.f, 3.f, 4.f	FAIL
TC#9	false	false	true	true	1.g, 2.g, 3.g, 4.g	FAIL

3.14 determineResolutionSense

3.14.1 Equivalence Classes

[Sense] *Determines the sense (Sense::UPWARDS or Sense::DOWNWARDS) that the user's aircraft should use when resolving an RA with the details of the supplied intruding aircraft*

Parameter	Equivalence Class ID | Description	Representative
1. userAltFt	**1.1:** altSelf ∈ $\mathbb{R}^+$ ≥ 1000	**1.1:** 16,000
	1.2: altSelf ∈ $\mathbb{R}^+$ < 45000	**1.2:** 40,000
	1.a: altSelf = 0	**1.a:** 0
	1.b: altSelf ∈ $\mathbb{R}$ < 1000	**1.b:** 999
	1.c: altSelf ∈ { }	**1.c:** ∅
	1.d: altSelf is a char	**1.d:** 's'
	1.e: altSelf is a string	**1.e:** "Self"
	1.f: altSelf is a boolean	**1.f:** false
2. intrAltFt	**2.1:** altIntr ∈ $\mathbb{R}^+$ ≥ 1000	**2.1:** 25,000
	2.2: altIntr ∈ $\mathbb{R}^+$ < 45000	**2.2:** 35,000
	2.a: altIntr = 0	**2.a:** 0
	2.b: altIntr ∈ $\mathbb{R}$ < 1000	**2.b:** 800
	2.c: altIntr ∈ { }	**2.c:** ∅
	2.d: altIntr is a char	**2.d:** 'i'
	2.e: altIntr is a string	**2.e:** "Intruder"
	2.f: alt is a boolean	**2.f:** true

3.14.2 Boundary Value Analysis

Parameter	Boundary Values	Test Case IDs
1. userAltFt	{999, 1,000, 1,001}	TC#1
	{45,000, 44,999, 44,998}	TC#2
2. intrAltFt	{999, 1,000, 1,001}	TC#1
	{45,000, 44,999, 44,998}	TC#2

3.14.3 Test Cases

TC ID	Self Altitude	Intruder Altitude	Covered EC	Exp. Results
TC#1	16,000	25,000	1.1, 2.1	DOWNWARDS
TC#2	21,000	35,000	1.2, 2.2	UPWARDS
TC#3	0	0	1.a, 2.a	FAIL
TC#3	999	800	1.b, 2.b	FAIL
TC#4	∅	∅	1.c, 2.c	FAIL
TC#5	‘s’	‘i’	1.d, 2.d	FAIL
TC#6	“Self”	“Intruder”	1.e, 2.e	FAIL
TC#7	false	true	1.f, 2.f	FAIL

3.15 getRecRangePair

3.15.1 Equivalence Classes

[RecommendationRangePair] *Returns a pair of recommendation ranges as for a Resolution Advisory*

Parameter	Equivalence Class ID | Description	Representative
1. sense	**1.1:** sense is UPWARD	**1.1:** UPWARD
	1.2: sense is DOWNWARD	**1.2:** DOWNWARD
	1.a: sense is UNKNOWN	**1.a:** UNKNOWN
	1.b: sense ∈ ℝ	**1.b:** 500
	1.c: sense is a char	**1.c:** 's'
	1.d: sense is a string	**1.d:** "UPWARD"
	1.e: sense ∈ { }	**1.e:** ∅
	1.f: sense is a boolean	**1.f:** false
2. userVvelFtM	**2.1:** userVvel ∈ ℝ ≥ -4,400	**2.1:** -3,500
	2.2: userVvel ∈ ℝ ≤ 4,400	**2.2:** 4,000
	2.a: userVvel = 0	**2.a:** 0
	2.b: userVvel ∈ $\mathbb{R}^-$ < -4,400	**2.b:** -5,000
	2.c: userVvel ∈ $\mathbb{R}^+$ > 4,000	**2.c:** 5,000
	2.d: userVvel ∈ { }	**2.d:** ∅
	2.e: userVvel is a char	**2.e:** 's'
	2.f: userVvel is a string	**2.f:** "User"
	2.g: userVvel is a boolean	**2.g:** false
3. IntrVvelFtm	**3.1:** intrVvel ∈ ℝ ≥ -4,400	**3.1:** -4,000
	3.2: intrVvel ∈ ℝ ≤ 4,400	**3.2:** 3,000
	3.a: intrVvel = 0	**3.a:** 0
	3.b: intrVvel ∈ $\mathbb{R}^-$ < -4,400	**3.b:** -5,000
	3.c: intrVvel ∈ $\mathbb{R}^+$ > 4,00	**3.c:** -4,500
	3.d: intrVvel ∈ { }	**3.d:** ∅
	3.e: intrVvel is a char	**3.e:** 'i'
	3.f: intrVvel is a string	**3.f:** "Other"
	3.g: intrVvel is a boolean	**3.g:** true
4. userAltFt	**1.1:** alt ∈ $\mathbb{R}^+$ > 0	**4.1:** -5

	1.2: alt ∈ $\mathbb{R}^+$ < 45,000 **1.a:** alt = 0 **1.b:** alt ∈ $\mathbb{R}^-$ < 0 **1.c:** alt ∈ $\mathbb{R}^+$ ≥ 45,000 **1.d:** alt ∈ { } **1.e:** alt is a char **1.f:** alt is a string **1.g:** alt is a boolean	**4.2:** 49,000 **4.a:** 0 **4.b:** -10 **4.c:** 60,000 **4.d:** ∅ **4.e:** 'v' **4.f:** "Self Velocity" **4.g:** true
5. intrAltFt	**1.1:** alt ∈ $\mathbb{R}^+$ > 0 **1.2:** alt ∈ $\mathbb{R}^+$ < 45,000 **1.a:** alt = 0 **1.b:** alt ∈ $\mathbb{R}^-$ < 0 **1.c:** alt ∈ $\mathbb{R}^+$ ≥ 45,000 **1.d:** alt ∈ { } **1.e:** alt is a char **1.f:** alt is a string **1.g:** alt is a boolean	**5.1:** 11,800 **5.2:** 44,000 **5.a:** 0 **5.b:** -1,000 **5.c:** 51,000 **5.d:** ∅ **5.e:** 'g' **5.f:** "Intruder" **5.g:** false
6. rangeTauS	**6.1:** tau ∈ $\mathbb{R}^+$ > 0 **6.2:** tau ∈ $\mathbb{R}^+$ ≤ 40 **6.a:** tau = 0 **6.b:** tau ∈ $\mathbb{R}$ ≤ 0 **6.c:** tau ∈ $\mathbb{R}^+$ > 40 **6.d:** tau ∈ { } **6.e:** tau is a char **6.f:** tau is a string **6.g:** tau is a boolean	**6.1:** 10 **6.2:** 40 **6.a:** 0 **6.b:** -2 **6.c:** 60 **6.d:** ∅ **6.e:** 'e' **6.f:** "Seconds" **6.g:** true

3.15.2 Boundary Value Analysis

Parameter	Boundary Values	Test Case IDs
userVvelFtm	{-4,399, -4,400, -4,401} {4,399, 4,400, 4,401}	TC#1 TC#2
IntrVvelFtm	{-4,399, -4,400, -4,401} {4,399, 4,400, 4,401}	TC#1 TC#2
userAltFt	{0, 1, 2} {45,000, 44,999, 44,998}	TC#1 TC#2
intrAltFt	{0, 1, 2} {45,000, 44,999, 44,998}	TC#1 TC#2
rangeTauS	{0, 1, 2} {39, 40, 41}	TC#1 TC#2

3.15.3 Test Cases

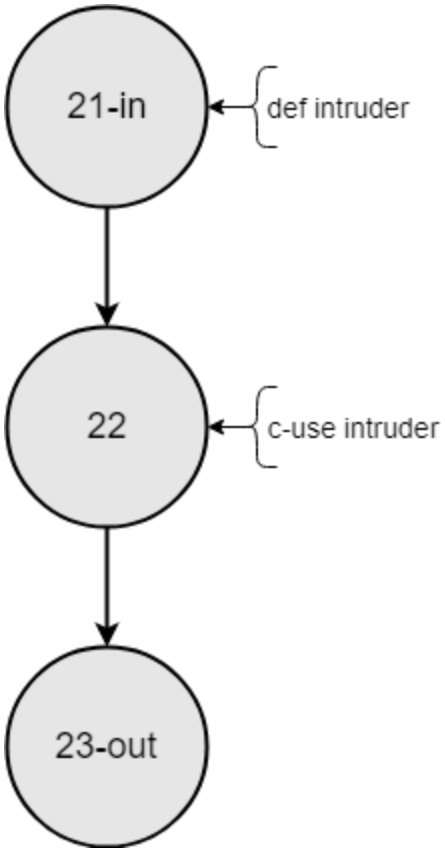
TC ID	sense	userVvel FtM	IntrVvel Ftm	userAltFt	intrAltFt	rangeTauS	Covered EC	Exp. Results
TC#1	UPWARD	-3,500	-4,000	-5	11,800	10	1.1, 2.1, 3.1, 4.1, 5.1, 6.1	PASS
TC#2	DOWNWARD	4,000	3,000	49,000	46,000	40	1.2, 2.2, 3.2, 4.2, 5.2, 6.2	PASS
TC#3	UNKNOWN	0	0	0	0	0	1.a, 2.a, 3.a, 4.a, 5.a, 6.a	FAIL
TC#4	500	-5,000	-5,000	-1,000	-1,000	-2	1.b, 2.b, 3.b, 4.b, 5.b, 6.b	FAIL
TC#5	‘s’	5,000	-4,500	60,000	51,000	60	1.c, 2.c, 3.c, 4.c, 5.c, 6.c	FAIL
TC#6	“UPWARD”	∅	∅	∅	∅	∅	1.d, 2.d, 3.d, 4.d, 5.d, 6.d	FAIL
TC#7	∅	‘s’	‘i’	‘v’	‘g’	‘e’	1.e, 2.e, 3.e, 4.e, 5.e, 6.e	FAIL

TC#8	false	“User”	“Other”	“Self Velocity”	“Intruder”	“Seconds”	1.f, 2.f, 3.f, 4.f, 5.f, 6.f	FAIL
TC#9	∅	false	true	true	false	true	2.g, 3.g, 4.g, 5.g, 6.g	FAIL

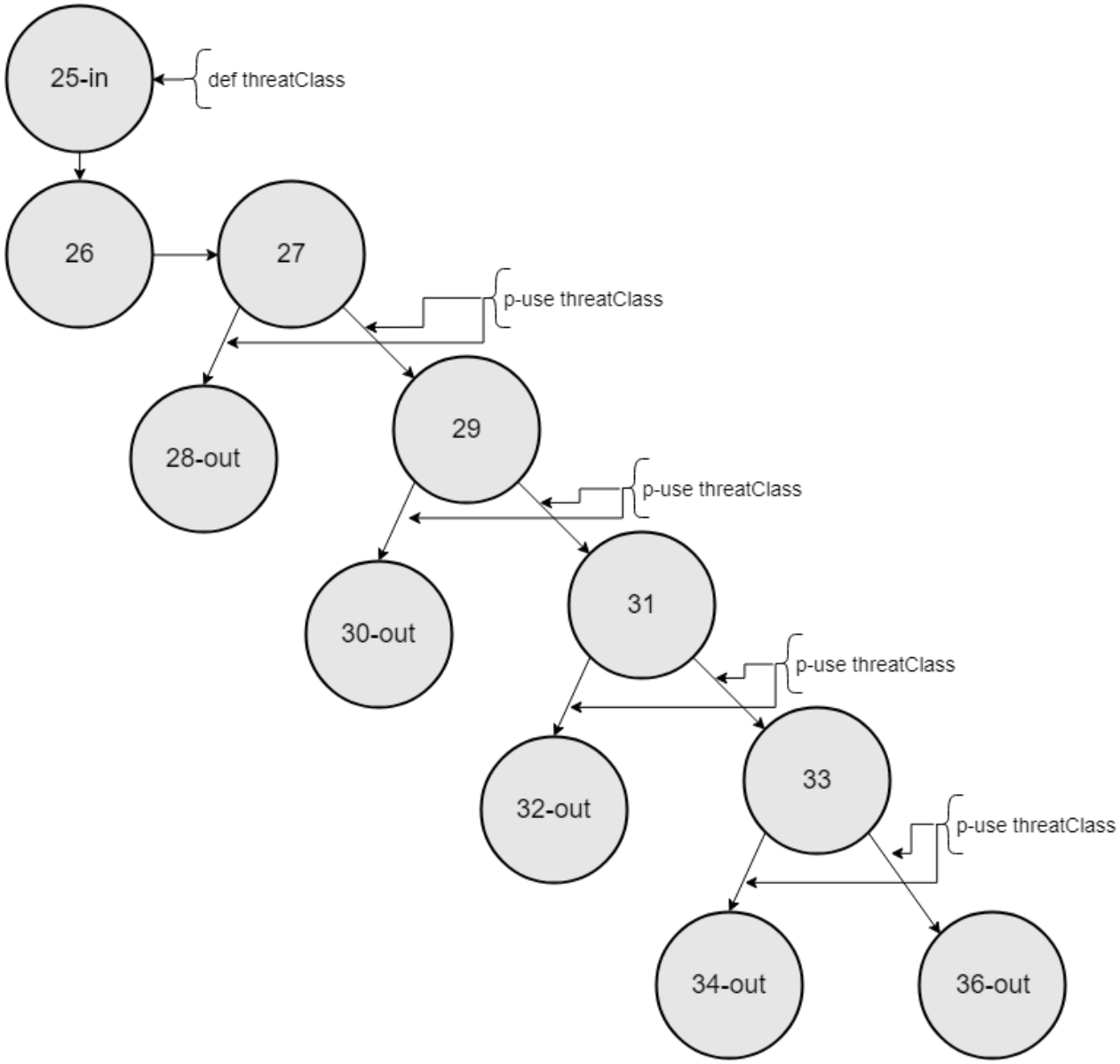
4. Data-Flow Annotated Control Flow Graphs

*Annotation in red means the variable is not global, and used only in the node
*Annotation in purple means the variable is created, but never used
*Variables in red and purple are not reported in section 6

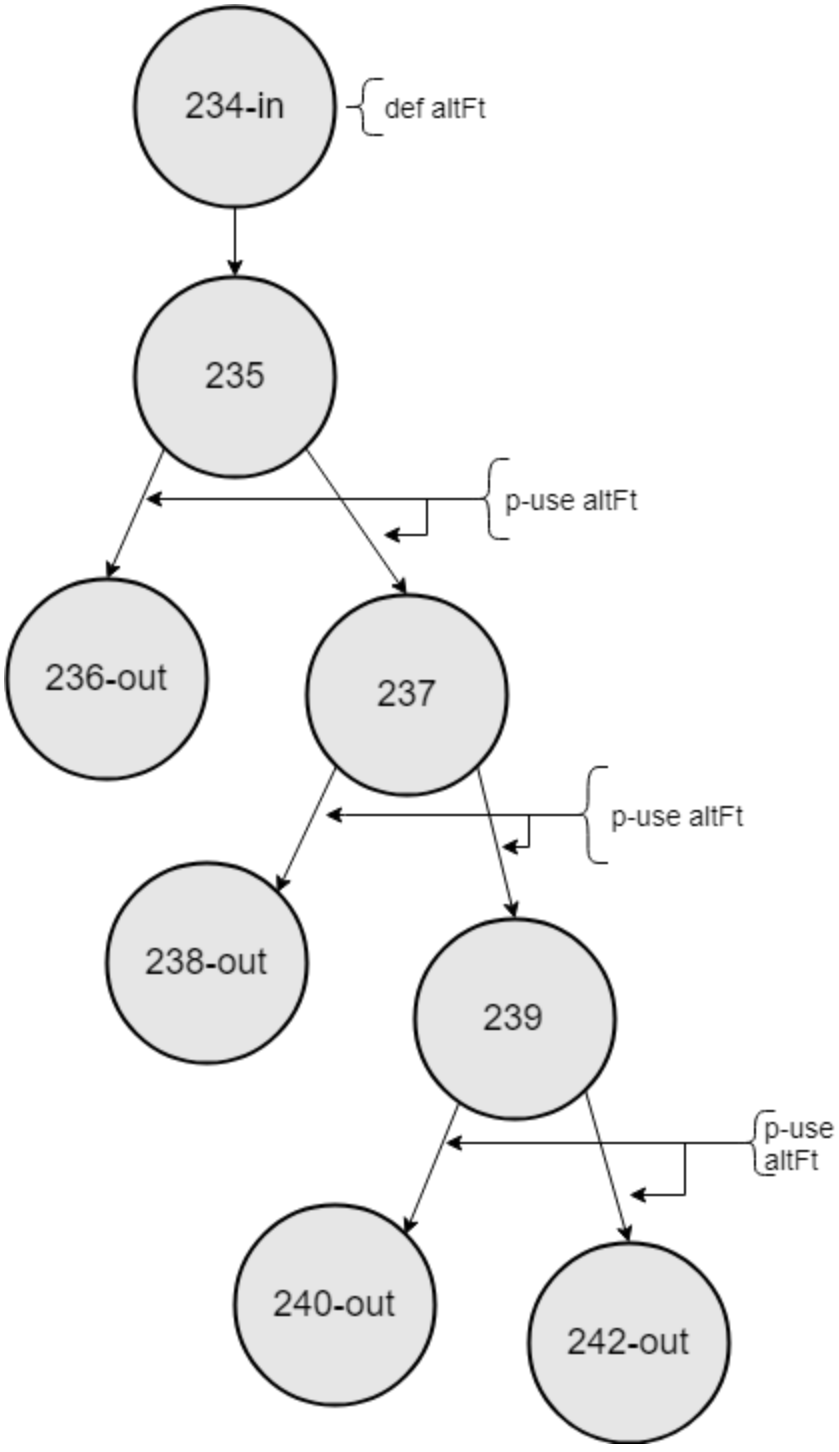
4.1 Analyze



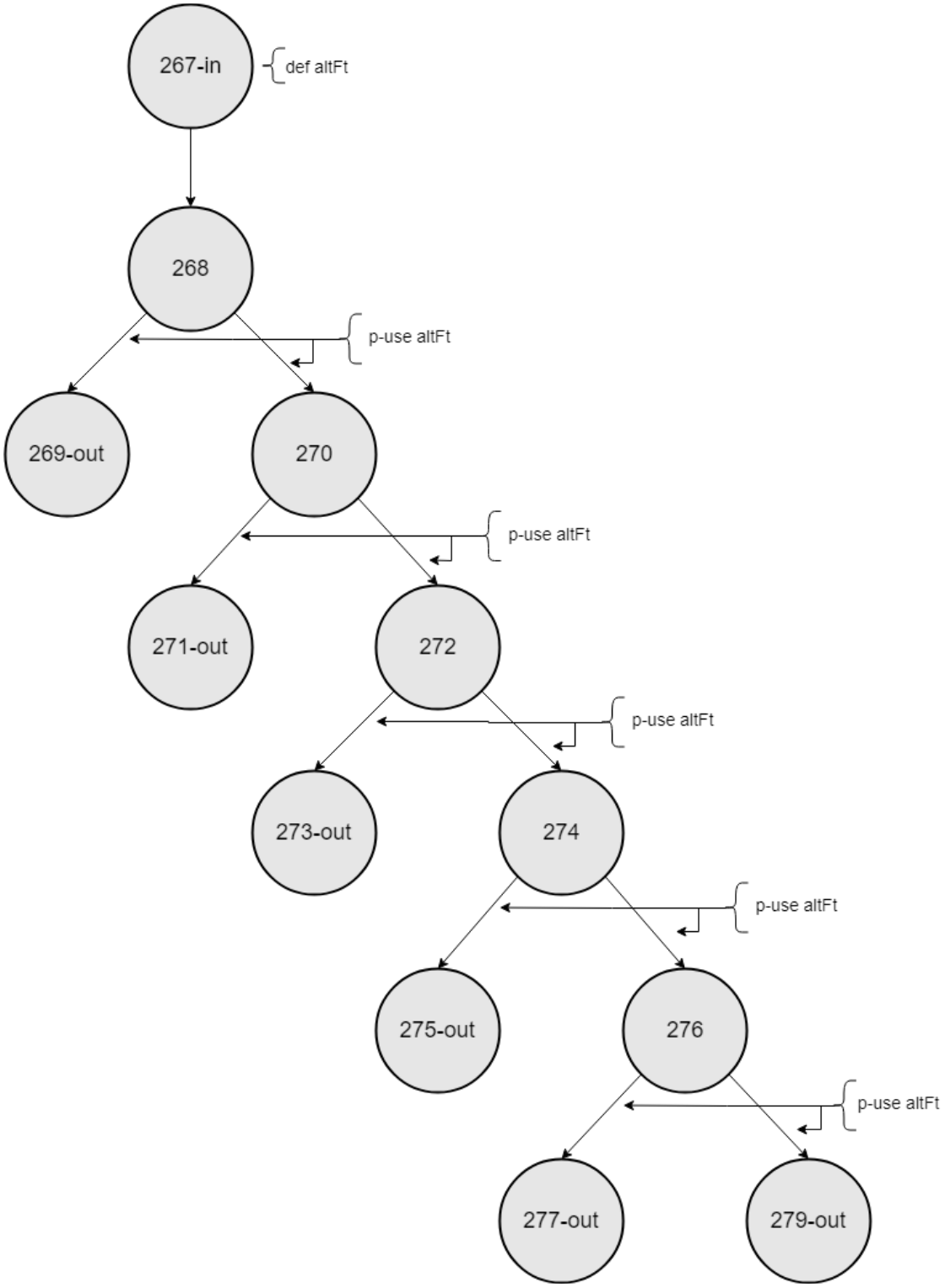
4.2 getThreatClassStr



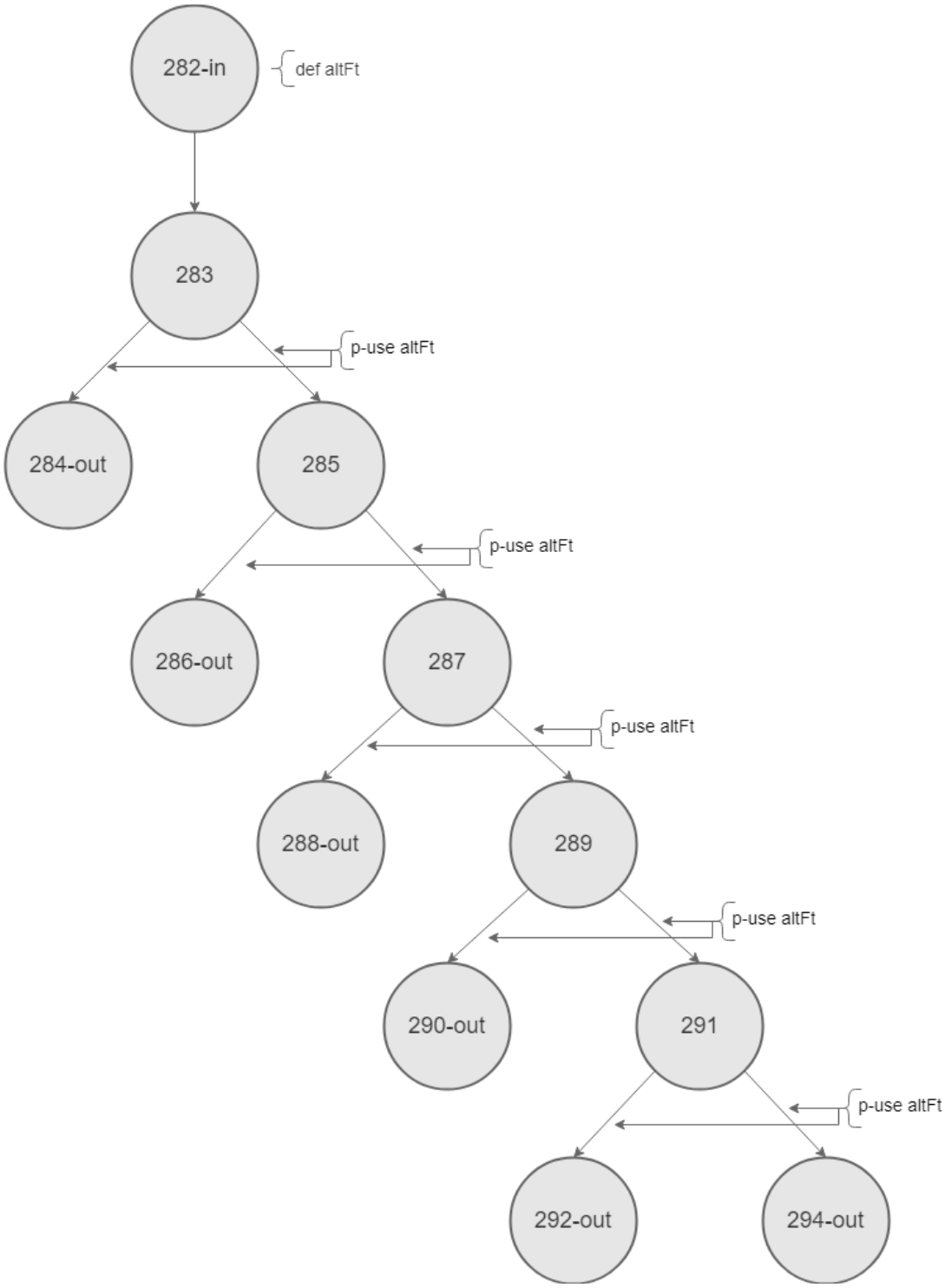
4.3 getAlimFt



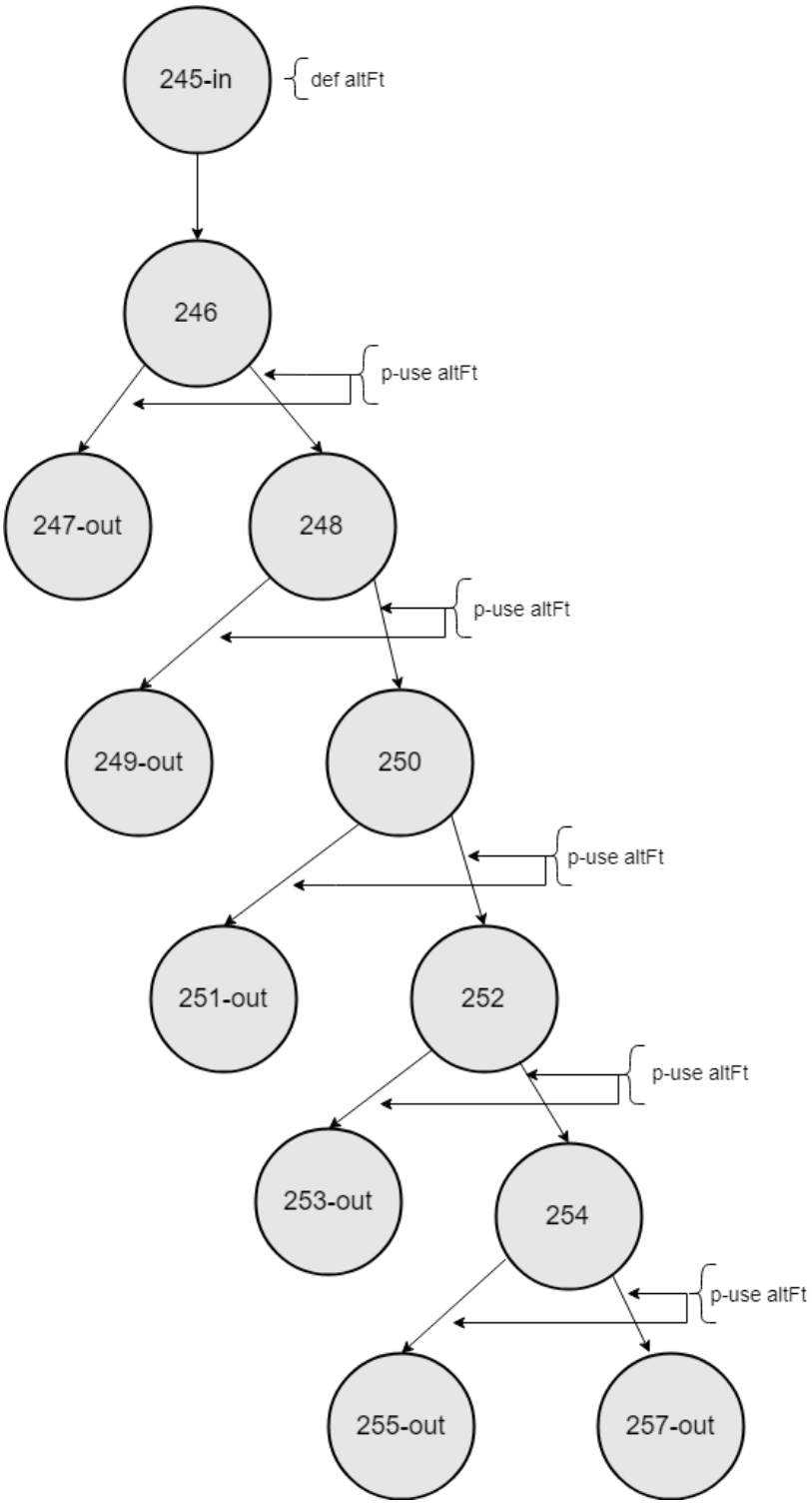
4.4 getRADmodNm



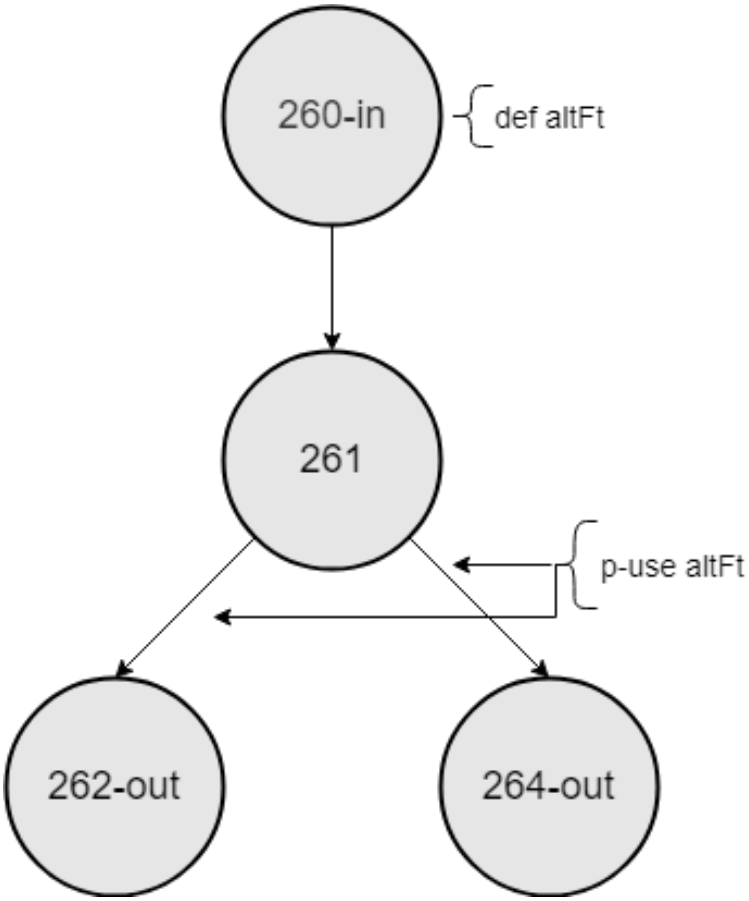
4.5 getTADmodNmi



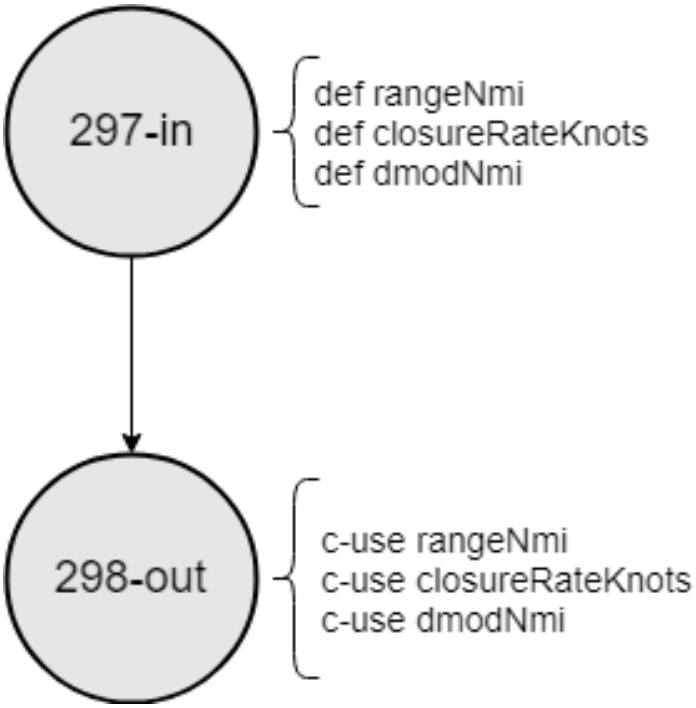
4.6 getRAZthrFt



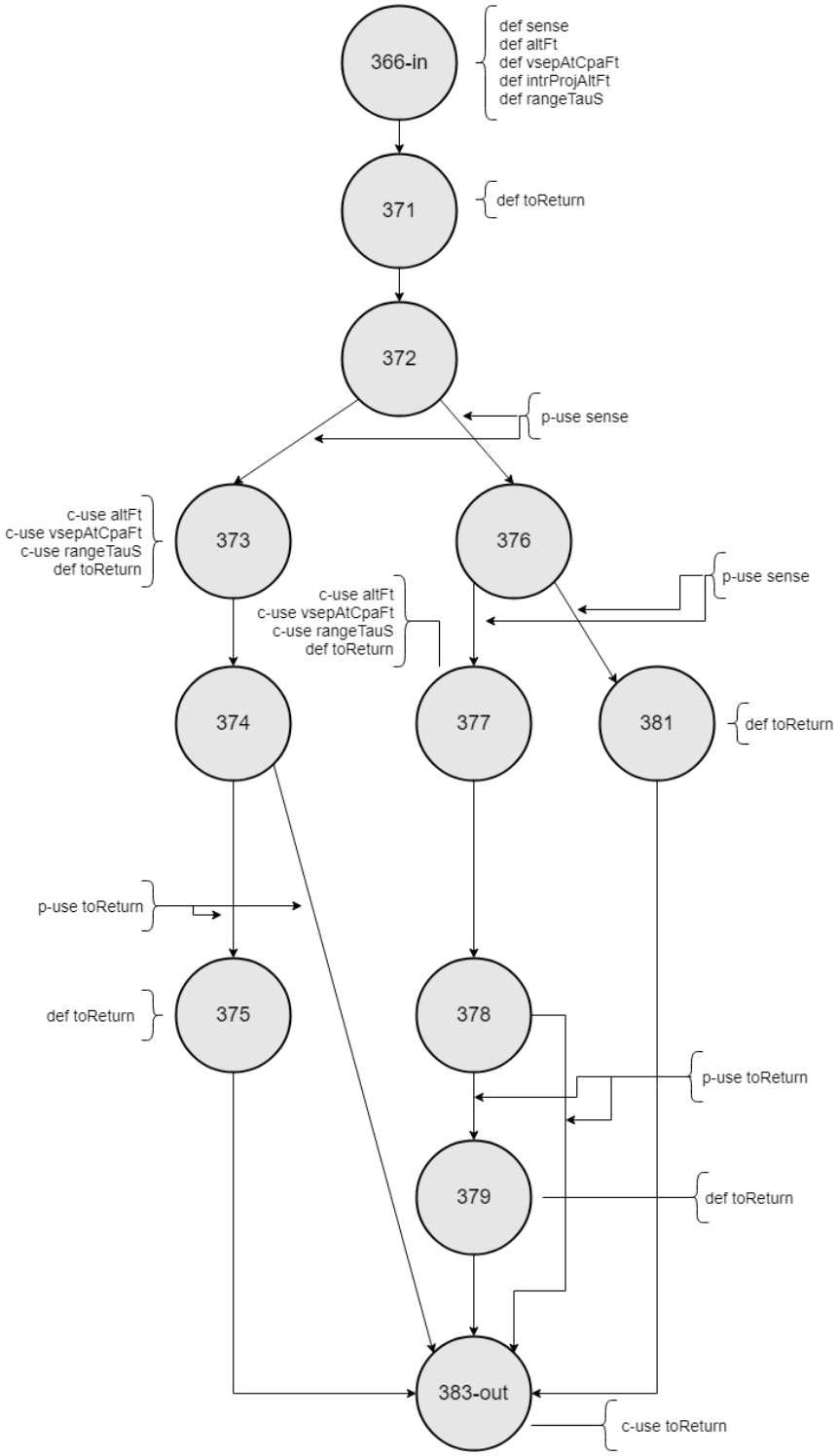
4.7 getTAZthrFt



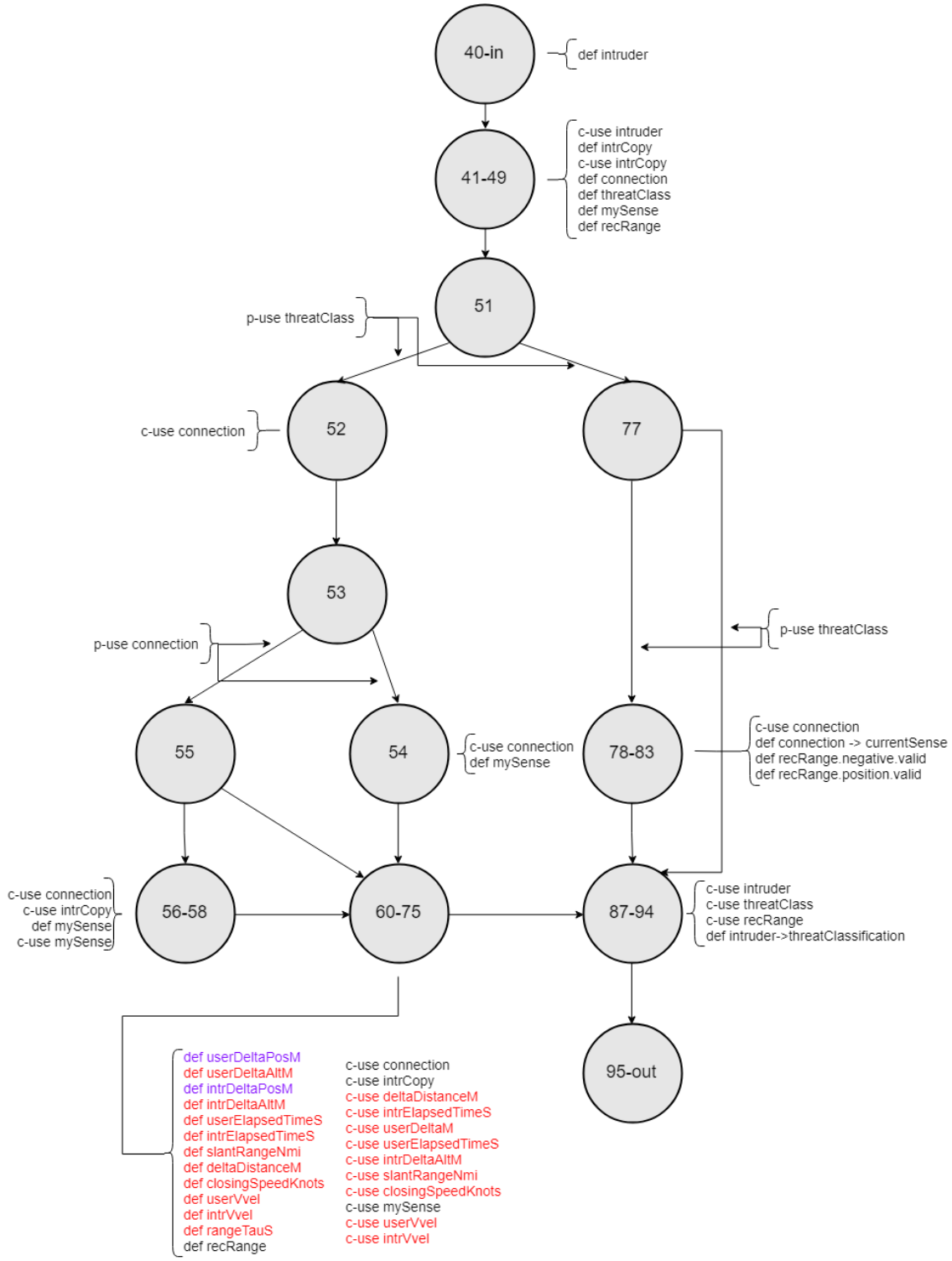
4.8 getModTauS



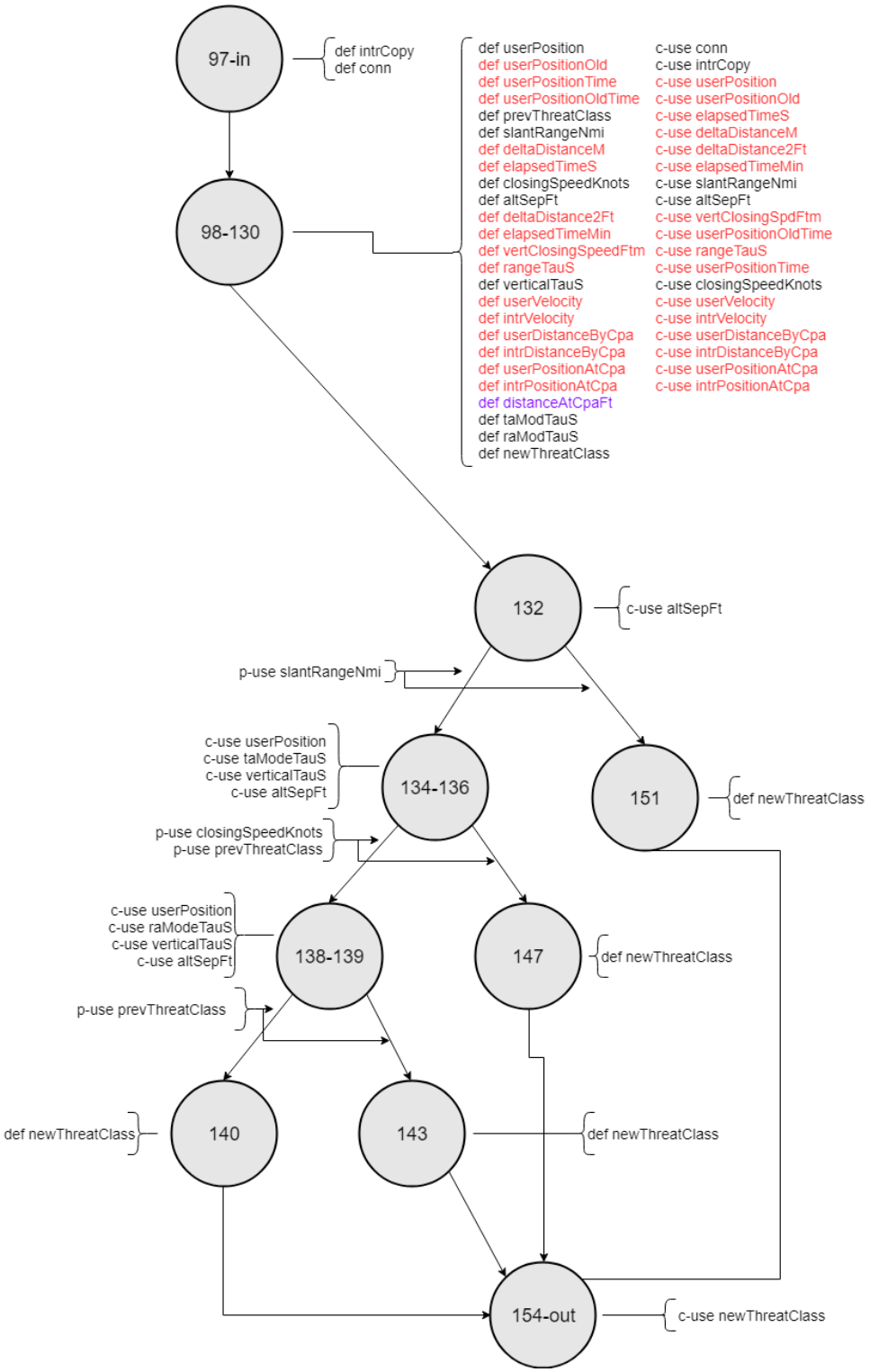
4.9 getVvelForAlim



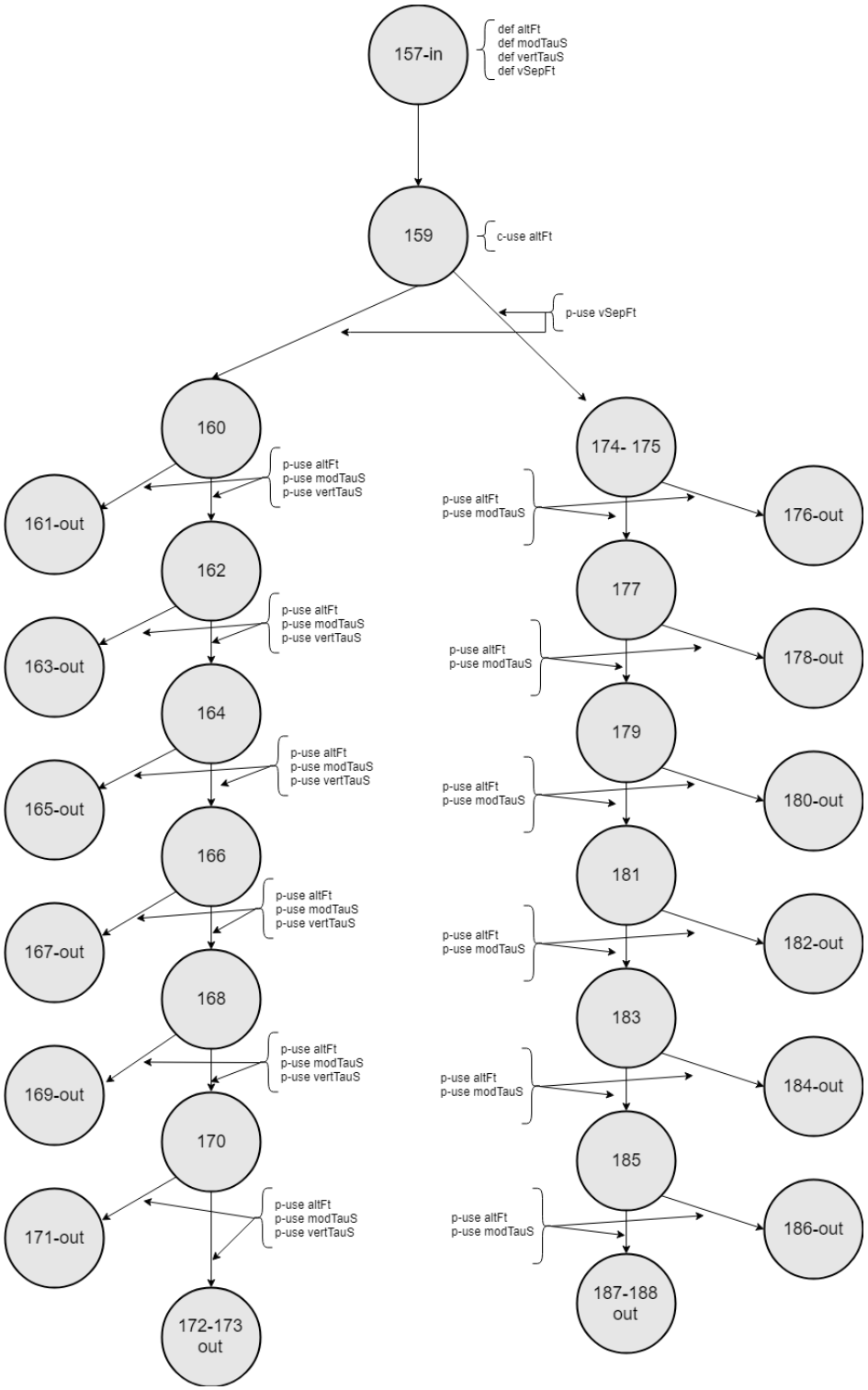
4.10 determineActionRequired



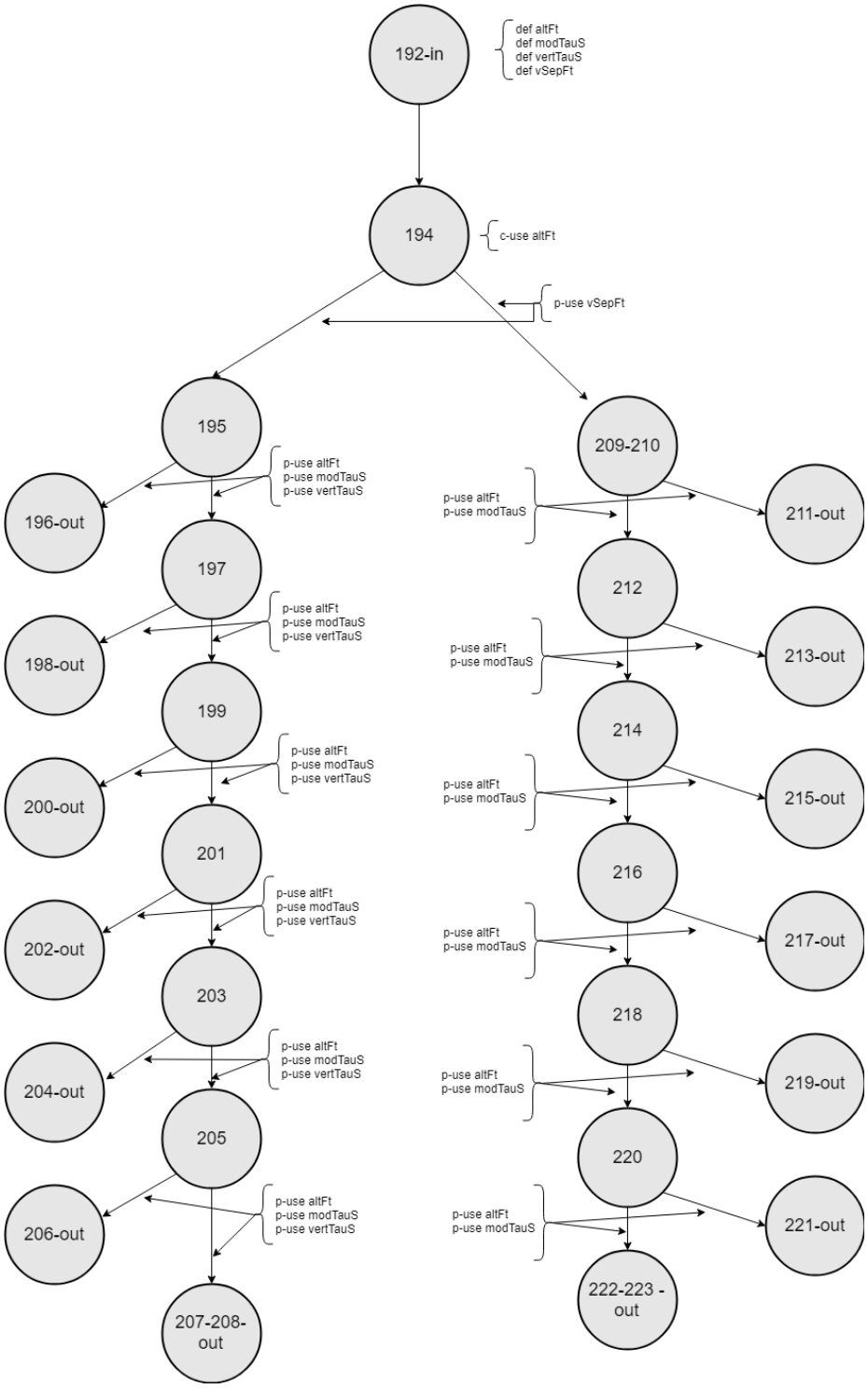
4.11 determineThreatClass



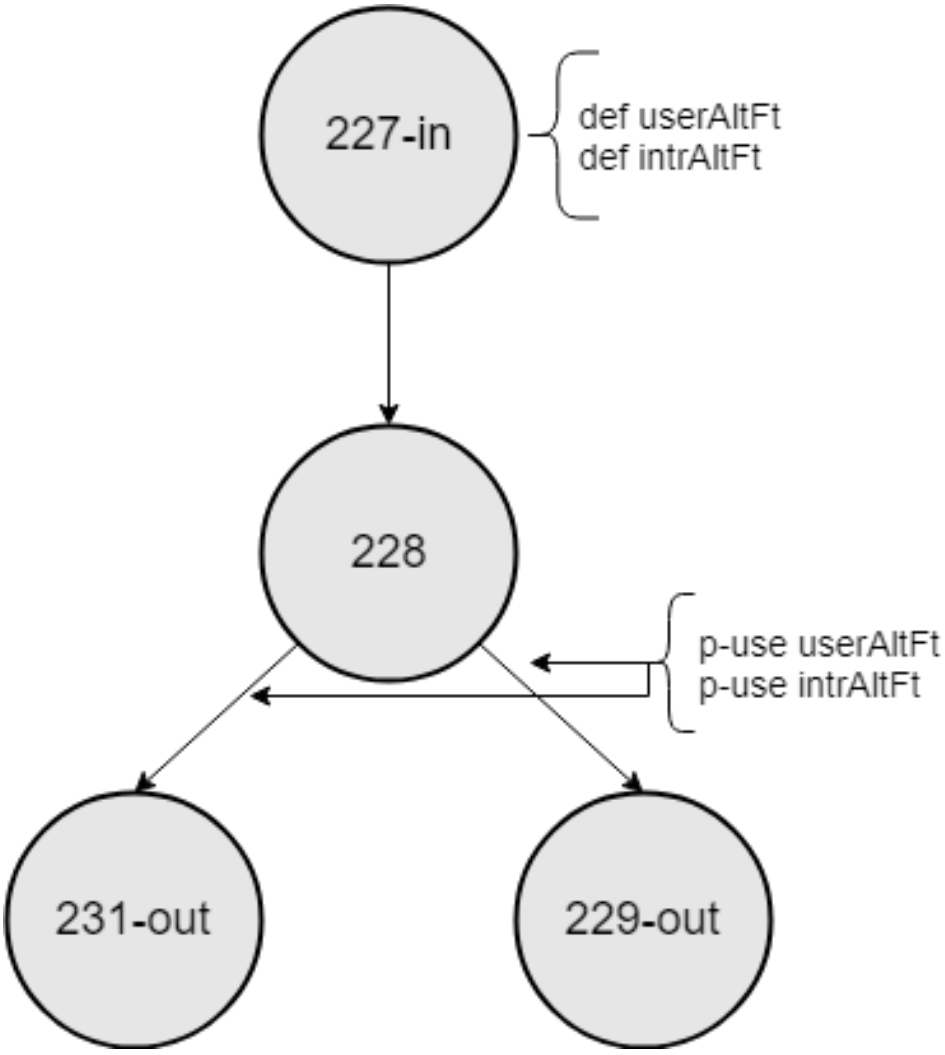
4.12 TauPassesTAThreshold



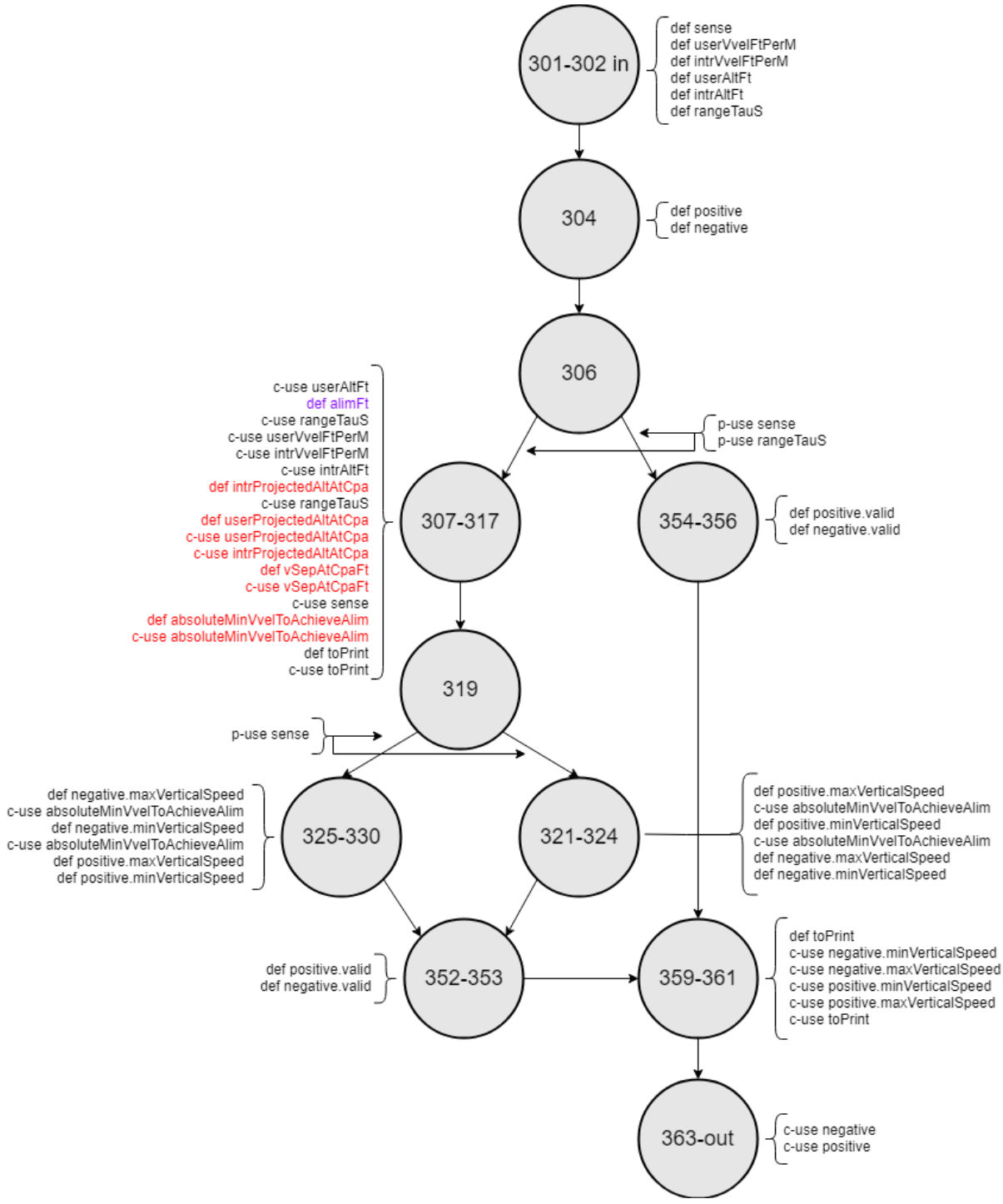
4.13 TauPassesRAThreshold



4.14 determineResolutionSense



4.15 getRecRangePair



5.Control Flow Based Tests

5.1 Analyze

Test Case ID	Parameters	Coverage	
	intruder	path	%
TC#1	Aircraft(“blah”, “127.0.0.1”)	21-in, 22, 23-out	100%

Test cases sufficient for path and statement coverage
C_2 and C_0 met
C_1 and C_{3b} not necessary due to lack of control flow

5.2 getThreatClassStr

Test Case ID	case NON_THREAT_TRAFFIC
TC #1	T
TC #2	F

Test Case ID	case PROXIMITY_INTRUDER_TRAFFIC
TC #3	T
TC #4	F

Test Case ID	case TRAFFIC_ADVISORY
TC #5	T
TC #6	F

Test Case ID	case RESOLUTION_ADVISORY
TC #7	T
TC #8	F

Test Case ID	Parameters	Coverage	
	threatClass	path	%
TC #9	NON_TRAFFIC_TRAFFIC	25-in, 26, 27, 28-out	30%
TC #10	PROXIMITY_INTRUDER_TRAFFIC	25-in, 26, 27, 29, 30-out	50%
TC #11	TRAFFIC_ADVISORY	25-in, 26, 27, 29, 31, 32-out	70%
TC #12	RESOLUTION_ADVISORY	25-in, 26, 27, 29, 31, 33, 34-out	90%

Test cases not sufficient for achieving path coverage
C_2 cannot be achieved
C_{3b} cannot be achieved
Node 36 not reachable as a default for switch case - 100% path or minimal condition coverage unachievable

5.3 getAlimFt

Test Case ID	altFt < 1,000
TC #1	T
TC #2	F

Test Case ID	altFt < 20,000
TC #3	T
TC #4	F

Test Case ID	altFt < 42,000
TC #5	T
TC #6	F

Test Case ID	Parameters	Coverage	
	altFt	path	%
TC#7	800	234-in, 235, 236-out	25%
TC#8	3,000	234-in, 235, 237, 238-out	50%
TC#9	32,000	234-in, 235, 237, 239, 240-out	75%
TC#10	43,000	234-in, 235, 237, 239, 242-out	100%

Test cases sufficient for path coverage and minimal condition coverage
C_2 and C_{3b} met
C_2 subsumes C_1
C_1 subsumes C_0

5.4 getRADmodNmi

Test Case ID	altFt < 1,000
TC #1	T
TC #2	F

Test Case ID	altFt < 2,350
TC #3	T
TC #4	F

Test Case ID	altFt < 5,000
TC #5	T
TC #6	F

Test Case ID	altFt < 10,000
TC #7	T
TC #8	F

Test Case ID	altFt < 20,000
TC #9	T
TC #10	F

Test Case ID	Parameters	Coverage	
	altFt	path	%
TC#7	800	267-in, 268, 269-out	18%
TC#8	1,500	267-in, 268, 270, 271-out	36%
TC#9	4,000	267-in, 268, 270, 272, 273-out	55%
TC#10	8,000	267-in, 268, 270, 272, 274, 275-out	73%
TC#11	19,000	267-in, 268, 270, 272, 274, 276, 277-out	91%
TC#12	24,000	267-in, 268, 270, 272, 274, 276, 279-out	100%

Test cases sufficient for path coverage and minimal condition coverage
C_2 and C_{3b} met
C_2 subsumes C_1
C_1 subsumes C_0

5.5 getTADmodNmi

Test Case ID	altFt < 1000
TC #1	T
TC #2	F

Test Case ID	altFt < 2,350
TC #3	T
TC #4	F

Test Case ID	altFt < 5,000
TC #5	T
TC #6	F

Test Case ID	altFt < 10,000
TC #7	T
TC #8	F

Test Case ID	altFt < 20,000
TC #9	T
TC #10	F

Test Case ID	Parameters	Coverage	
	altFt	path	%
TC#7	750	282-in, 283, 284-out	18%
TC#8	2,000	282-in, 283, 285, 286-out	36%
TC#9	3,500	282-in, 283, 285, 287, 288-out	55%
TC#10	8,500	282-in, 283, 285, 287, 289, 290-out	73%
TC#11	15,000	282-in, 283, 285, 287, 289, 291, 292-out	91%
TC#12	21,000	282-in, 283, 285, 287, 289, 291, 294-out	100%

Test cases sufficient for path coverage and minimal condition coverage
C_2 and C_{3b} met
C_2 subsumes C_1
C_1 subsumes C_0

5.6 getRAZthrFt

Test Case ID	altFt < 1000
TC #1	T
TC #2	F

Test Case ID	altFt < 5000
TC #3	T
TC #4	F

Test Case ID	altFt < 10,000
TC #5	T
TC #6	F

Test Case ID	altFt < 20,000
TC #7	T
TC #8	F

Test Case ID	altFt < 42,000
TC #9	T
TC #10	F

Test Case ID	Parameters	Coverage	
	altFt	path	%
TC#7	800	245-in, 246, 247-out	18%
TC#8	3,000	245-in, 246, 248, 249-out	36%
TC#9	7,000	245-in, 246, 248, 250, 251-out	55%
TC#10	14,000	245-in, 246, 248, 250, 252, 253-out	73%
TC#11	33,000	245-in, 246, 248, 250, 252, 254, 255-out	91%
TC#12	48,000	245-in, 246, 248, 250, 252, 254, 257-out	100%

Test cases sufficient for path coverage and minimal condition coverage
C_2 and C_{3b} met
C_2 subsumes C_1
C_1 subsumes C_0

5.7 getTAZthrFt

Test Case ID	Parameters	Coverage	
	altFt	path	%
TC#1	38,000	260-in, 261, 262-out	50%
TC#2	43,000	260-in, 261, 264-out	100%

Test cases sufficient for path coverage
C_2 met
C_2 subsumes C_1
C_1 subsumes C_0
C_{3b} not necessary due to single atomic

5.8 getModTauS

Test Case ID	Parameters			Coverage	
	rangeNmi	closureRateKnots	dModNmi	path	%
TC#1	5	100	0.34	297-in, 298-out	100%

Test cases sufficient for path coverage and statement coverage
C_0 and C_2 met
C_1 and C_{3b} not necessary due to lack of control flow

5.9 getVvelForAlim

Test Case ID	sense == UPWARD	sense == DOWNWARD
TC #1	T	F
TC #2	F	T
TC #3	F	F

Test Case ID	toReturn > kMaxGaugeVerticalVelocity_.toFeetPerMin() - 500
TC #4	T
TC #5	F

Test Case ID	toReturn < kMinGaugeVelocity_.toFeetPerMin() + 500
TC #7	T
TC #8	F

Test Case ID	Parameters					Coverage	
	sense	altFt	vSepAtCpaFt	intrProjAltFt	rangeTauS	path	%
TC #9	UP	45,000	300	N/A	48	366-in,371,372,373,374,375,383-out	40%
TC #10	UP	1,001	700	N/A	15	366-in,371,372,373,374,383-out	47%
TC #11	DOWN	45,000	300	N/A	48	366-in,371,372,376,377,378,379,383-out	80%
TC #12	DOWN	1,001	700	N/A	15	366-in,371,372,376,377,378,383-out	87%
TC #13	UNKNOWN	4,200	420	N/A	42	366-in,371,372,376,380,381,383-out	100%

*intrProjAlt is depricated, and no longer used in the method
Test cases sufficient for path coverage and minimal condition coverage.
C_2 and C_{3b} met
C_2 subsumes C_1
C_1 subsumes C_0

5.10 determineActionRequired

Test Case ID	threatClass = RESOLUTION_ADVISORY
TC #1	T
TC #2	F

Test Case ID	threatClass = NON_THREAT_CLASS
TC #3	T
TC #4	F

Test Case ID	connection->consensusAchieved	connection->currentSense=DOWNWARD	connection->currentSense=UPWARD
TC #5	T	T	F
TC #6	T	F	T
TC #7	F	F	F

connection->currentSense=DOWNWARD | connection->currentSense=UPWARD
T
T
F

connection->consensusAchieved & (connection->currentSense=DOWNWARD | connection->currentSense=UPWARD)
T
T
F

Test Case ID	tempSense_ = UNKOWN
TC #8	T
TC #9	F

*The control flow of this method depends on connection found in the activeConnections_ array, as well as tempSense_. These are global variables that depend on the state of the Decider. See 7.1.2.10 for test cases.

5.11 determineThreatClass

Test Case ID	slantRangeNmi<6	abs(altSepFt)<1,200	slantRangeNmi<6 & abs(altSepFt)<1,200
TC #1	T	T	T
TC #2	F	F	F

Test Case ID	closingSpeedKnots>0	prevThreatClass≥TRAFFIC_ADVISORY
TC #3	T	T
TC #4	F	F

tauPassesTaThreshold(userPosition.altitude.toFeet(), taModTauS, verticalTauS, altSepFt)
T
F

prevThreatClass≥TRAFFIC_ADVISORY | tauPassesTaThreshold(userPosition.altitude.toFeet(), taModTauS, verticalTauS, altSepFt)
T
F

closingSpeedKnots>0 & (prevThreatClass≥TRAFFIC_ADVISORY | tauPassesTaThreshold(userPosition.altitude.toFeet(), taModTauS, verticalTauS, altSepFt))
T
F

Test Case ID	prevThreatClass=RESOLUTION_ADVISORY
TC #5	T
TC #6	F

tauPassesRaThreshold(userPosition.altitude.toFeet(), raModTauS, verticalTauS, altSepFt)
T
F

prevThreatClass≥TRAFFIC_ADVISORY | tauPassesTaThreshold(userPosition.altitude.toFeet(), taModTauS, verticalTauS, altSepFt)
T
F

Test Case ID	Parameters		Coverage	
	interCopy	conn	path	%
TC #7	Aircraft(“craft”, “128.0.0.10”) { positionCurrent = LLA(80.6, 110.2, 1256.0, AngleUnits.DEGREE, DistanceUnits.FEET), positionTime = 12481697627	ResolutionConnection(“AB:CD:EF:12:34:56” “BA:DC:FE:21:43:65”, “198.0.10.16”, 1234, Aircraft(“guy”, “198.0.10.16”) { userPosition = LLA(80.0, 110.0, 1200.0, AngleUnits.DEGREE, DistanceUnits.FEET) threatClassification =	97-in, 98-130, 132, 134-136, 138-139, 140, 154-out	50%

	positionOldTime = 12481541267 }	TRAFFIC_ADVISORY }		

TC #8	Aircraft(
	“craft”,
	“128.0.0.10”
	) {
	positionCurrent = LLA(
	80.5, 110.4, 1256.0,
	AngleUnits.DEGREE,
	DistanceUnits.FEET
	)
	positionTime =
	12481697627
	positionOldTime =
	12481541267
	}
		ResolutionConnection(
		“AB:CD:EF:12:34:56”
		“BA:DC:FE:21:43:65”,
		“198.0.10.16”,
		1234,
		Aircraft(“guy”, “198.0.10.16”
		) {
		userPosition = LLA(80.0, 110.0, 812.0,
		AngleUnits.DEGREE,
		DistanceUnits.FEET)
		threatClassification =
		RESOLUTION_ADVISORY
		}
			97-in, 98-130, 132, 134-136, 138-139, 143, 154-out	67%

TC #9	Aircraft(
	“craft”,
	“128.0.0.10”
	) {
	positionCurrent = LLA(
	80.5, 110.4, 1300.0,
	AngleUnits.DEGREE,
	DistanceUnits.FEET
	)
	positionTime =
	12481697627
	positionOldTime =
	12481474833
	}
		ResolutionConnection(
		“AB:CD:EF:12:34:56”
		“BA:DC:FE:21:43:65”,
		“198.0.10.16”,
		1234,
		Aircraft(“guy”, “198.0.10.16”
		) {
		userPosition = LLA(80.0, 110.0, 812.0,
		AngleUnits.DEGREE,
		DistanceUnits.FEET)
		threatClassification =
		RESOLUTION_ADVISORY
		}
			97-in, 98-130, 132, 134-136, 147, 154-out	83%

TC #10	Aircraft(“craft”, “128.0.0.10”	ResolutionConnection(“AB:CD:EF:12:34:56” “BA:DC:FE:21:43:65”,	97-in, 98-130, 132, 151, 154-out	100%

	) {
	positionCurrent = LLA(
	80.5, 110.4, 2356.0,
	AngleUnits.DEGREE,
	DistanceUnits.FEET
	)
	positionTime =
	12481697627
	positionOldTime =
	12481474833
	}
		"198.0.10.16",
		1234,
		Aircraft("guy", "198.0.10.16"
		) {
		userPosition = LLA(82.0, 112.0, 812.0,
		AngleUnits.DEGREE,
		DistanceUnits.FEET)
		threatClassification =
		NON_THREAT_TRAFFIC
		}		

*Parameters show valid class constructors as well as internal variable values to achieve path coverage.
*All variables used in conditions are derived from the stated values in intrCopy and conn.
Test cases sufficient for path coverage and minimal condition coverage.
C_2 and C_{3b} met
C_2 subsumes C_1
C_1 subsumes C_0

5.12 TauPassesTAThreshold

Test Case ID	vSepFt > getTaZthrFt(altFt)
TC#1	T
TC#2	F

Test Case ID	altFt<1,000	modTauS<20	vertTauS<20	altFt<1,000 & modTaus<20 & vertTauS<20
TC#3	T	T	T	T
TC#4	F	F	F	F

Test Case ID	altFt<2,350	modTauS<25	vertTauS<25	altFt<2,350 & modTaus<25 & vertTauS<25
TC#5	T	T	T	T

TC#6	F	F	F	F

Test Case ID	altFt<5,000	modTauS<30	vertTauS<30	altFt<5,000 & modTaus<30 & vertTauS<30
TC#7	T	T	T	T
TC#8	F	F	F	F

Test Case ID	altFt<10,000	modTauS<40	vertTauS<40	altFt<10,000 & modTaus<40 & vertTauS<40
TC#9	T	T	T	T
TC#10	F	F	F	F

Test Case ID	altFt<20,000	modTauS<45	vertTauS<45	altFt<20,000 & modTaus<45 & vertTauS<45
TC#11	T	T	T	T
TC#12	F	F	F	F

Test Case ID	altFt≥20,000	modTauS<48	vertTauS<48	altFt≥20,000 & modTaus<48 & vertTauS<48
TC#13	T	T	T	T
TC#14	F	F	F	F

Test Case ID	altFt<1,000	modTauS<20	altFt<1,000 & modTaus<20
TC#15	T	T	T
TC#16	F	F	F

Test Case ID	altFt<2,350	modTauS<25	altFt<2,350 & modTaus<25
TC#17	T	T	T
TC#18	F	F	F

Test Case ID	altFt<5,000	modTauS<30	altFt<5,000 & modTaus<30
TC#19	T	T	T
TC#20	F	F	F

Test Case ID	altFt<10,000	modTauS<40	altFt<10,000 & modTaus<25
TC#21	T	T	T
TC#22	F	F	F

Test Case ID	altFt<20,000	modTauS<45	altFt<20,000 & modTaus<45
TC#23	T	T	T
TC#24	F	F	F

Test Case ID	altFt≥20,000	modTauS<48	altFt≥20,000 & modTaus<48
TC#25	T	T	T
TC#26	F	F	F

Test Case ID	Parameters				Coverage	
	alftFt	modTauS	vertTauS	vSepFt	path	%
TC#27	900	18	19	1,300	157-in, 159, 160, 161-out	11%
TC#28	2,000	23	24	1,300	157-in, 159, 160, 162, 163-out	19%

TC#29	4,000	28	29	1,300	157-in, 159, 160, 162, 164, 165-out	26%
TC#30	9,000	38	39	1,300	157-in, 159, 160, 162, 164, 166, 167-out	33%
TC#31	19,000	43	44	1,300	157-in, 159, 160, 162, 164, 166, 168, 169-out	41%
TC#32	20,000	46	47	1,300	157-in, 159, 160, 162, 164, 166, 168, 170, 171-out	48%
TC#33	20,000	50	40	1,300	157-in, 159, 160, 162, 164, 166, 168, 170, 173-out	52%
TC#34	900	18	19	800	157-in, 159, 175, 176-out	59%
TC#35	2,000	23	24	800	157-in, 159, 175, 177, 178-out	67%
TC#36	4,000	28	29	800	157-in, 159, 175, 177, 179, 180-out	74%
TC#37	9,000	38	39	800	157-in, 159, 175, 177, 179, 181, 182-out	81%
TC#38	19,000	43	44	800	157-in, 159, 175, 177, 179, 181, 183, 184-out	89%
TC#39	20,000	46	47	800	157-in, 159, 175, 177, 179, 181, 183, 185, 186-out	96%
TC#40	20,000	50	40	800	157-in, 159, 175, 177, 179, 181, 183, 185, 188-out	100%

Test cases sufficient for path coverage and minimal condition coverage.
C_2 and C_{3b} met
C_2 subsumes C_1
C_1 subsumes C_0

5.13 TauPassesRAThreshold

Test Case ID	vSepFt > getTaZthrFt(altFt)
TC#1	T
TC#2	F

Test Case ID	altFt<1,000
TC#3	T
TC#4	F

Test Case ID	altFt<2,350	modTauS<15	vertTauS<15	altFt<2,350 & modTaus<15 & vertTauS<15
TC#5	T	T	T	T
TC#6	F	F	F	F

Test Case ID	altFt<5,000	modTauS<20	vertTauS<20	altFt<5,000 & modTaus<20 & vertTauS<20
TC#7	T	T	T	T
TC#8	F	F	F	F

Test Case ID	altFt<10,000	modTauS<25	vertTauS<25	altFt<10,000 & modTaus<25 & vertTauS<25
TC#9	T	T	T	T
TC#10	F	F	F	F

Test Case ID	altFt<20,000	modTauS<30	vertTauS<30	altFt<20,000 & modTaus<30 & vertTauS<30

TC#11	T	T	T	T
TC#12	F	F	F	F

Test Case ID	altFt≥20,000	modTauS<35	vertTauS<35	altFt≥20,000 & modTaus<35 & vertTauS<35
TC#13	T	T	T	T
TC#14	F	F	F	F

Test Case ID	altFt<1,000
TC#15	T
TC#16	F

Test Case ID	altFt<2,350	modTauS<15	altFt<2,350 & modTaus<15
TC#17	T	T	T
TC#18	F	F	F

Test Case ID	altFt<5,000	modTauS<20	altFt<5,000 & modTaus<20
TC#19	T	T	T
TC#20	F	F	F

Test Case ID	altFt<10,000	modTauS<25	altFt<10,000 & modTaus<25
TC#21	T	T	T
TC#22	F	F	F

Test Case ID	altFt<20,000	modTauS<30	altFt<20,000 & modTaus<30
TC#23	T	T	T
TC#24	F	F	F

Test Case ID	altFt≥20,000	modTauS<35	altFt≥20,000 & modTaus<35
TC#25	T	T	T
TC#26	F	F	F

Test Case ID	Parameters				Coverage	
	alftFt	modTauS	vertTauS	vSepFt	path	%
TC#27	1,000	13	14	900	192-in, 194, 195, 196-out	11%
TC#28	2,000	13	14	900	192-in, 194, 195, 197, 198-out	19%
TC#29	4,000	18	19	900	192-in, 194, 195, 197, 199, 200-out	26%
TC#30	9,000	23	24	900	192-in, 194, 195, 197, 199, 201, 202-out	33%
TC#31	19,000	28	29	900	192-in, 194, 195, 197, 199, 201, 203, 204-out	41%
TC#32	20,000	33	34	900	192-in, 194, 195, 197, 199, 201, 203, 205, 206-out	48%
TC#33	20,000	50	40	900	192-in, 194, 195, 197, 199, 201, 203, 205, 208-out	52%
TC#34	900	13	14	-2	192-in, 194, 209, 210, 211-out	59%
TC#35	2,000	13	14	800	192-in, 194, 209, 210, 212, 213-out	67%

TC#36	4,000	18	19	800	192-in, 194, 209, 210, 212, 214, 215-out	74%
TC#37	9,000	23	24	800	192-in, 194, 209, 210, 212, 214, 216, 217-out	81%
TC#38	19,000	28	29	800	192-in, 194, 209, 210, 212, 214, 216, 218, 219-out	89%
TC#39	20,000	33	34	800	192-in, 194, 209, 210, 212, 214, 216, 218, 220, 221-out	96%
TC#40	20,000	36	34	800	192-in, 194, 209, 210, 212, 214, 216, 218, 220, 223-out	100%

Test cases sufficient for path coverage and minimal condition coverage.
C_2 and C_{3b} met
C_2 subsumes C_1
C_1 subsumes C_0

5.14 determineResolutionSense

Test Case ID	Parameters		Coverage	
	userAltFt	intrAltFt	path	%
TC#1	1,400	1,300	227-in, 228, 229-out	67%
TC#2	1,200	1,100	227-in, 228, 231-out	100%

Test cases sufficient for path coverage
C_2 met
C_2 subsumes C_1
C_1 subsumes C_0
C_{3b} not needed due to single atomic

5.15 getRecRangePair

Test Case ID	sense != UNKOWN	rangeTauS > 0.0	sense != UNKOWN & rangeTauS > 0.0
TC#1	T	T	T

TC#2	F	F	F

Test Case ID	sense == UPWARD
TC#3	T
TC#4	F

Test Case ID	Parameters						Coverage	
	sense	userVvelFtPerM	intrVvelFtPerM	userAltFt	intrAltFt	rangeTauS	path	%
TC#4	UPWARD	N/A	N/A	N/A	N/A	30.4	301-302-in, 304, 306, 307-317, 319, 325-330, 352-353, 359-361, 363-out	64%
TC#5	DOWNWARD	N/A	N/A	N/A	N/A	24.6	301-302-in, 304, 306, 307-317, 319, 321-324, 352-353, 359-361, 363-out	82%
TC#6	UNKNOWN	N/A	N/A	N/A	N/A	0.0	301-302-in, 304, 306, 354-356, 359-361, 363-out	100%

*userVvelFtPerM, intrVvelIFt, userAltFt, intrAltFt are not necessary since they have no impact on control flow.

Test cases sufficient for path coverage and minimal condition coverage.

C_2 and C_{3b} met

C_2 subsumes C_1

C_1 subsumes C_0

6. Data Flow Based Test

6.1 Analyze

Node	Variable	DCU	DPU
21-in	intruder	{22}	∅

*Test cases for path coverage also sufficient for all uses criterion
* See 5.1, TC #1

6.2 getThreatClassStr

Node	Variable	DCU	DPU
25-in	threatClass	∅	{(27,28-out), (27,29), (29,30-out), (29,31), (31,32-out), (31,33), (33,34-out), (33,36-out)}

*Node 36 unable to be reached due enumerator values
*Se 5.2, TC #9, TC #10, TC #11, TC #12

6.3 getAlimFt

Node	Variable	DCU	DPU
234-in	altFt	∅	{(235,236-out), (235,237), (237,238-out), (237-239), (239,240-out), (239,242-out)}

*Test cases for path coverage also sufficient for all uses criterion
* See 5.3, TC #7, TC #8, TC #9, TC #10

6.4 getRADmodNmi

Node	Variable	DCU	DPU
267-in	altFt	∅	{(268,269-out), (268,270), (270,271-out), (270,272), (272, 273-out), (272,274), (274,275-out), (274,276), (276,277-out), (276, 279-out)}

*Test cases for path coverage also sufficient for all uses criterion
* See 5.4, TC #7, TC #8, TC #9, TC #10, TC #11, TC #12

6.5 getTADmodNmi

Node	Variable	DCU	DPU
282-in	altFt	∅	{(283,284-out), (283,285), (285,286-out), (285,287), (287,288-out), (287,289), (289,290-out), (289,291), (291,292-out), (291,294-out)}

*Test cases for path coverage also sufficient for all uses criterion
* See 5.5, TC #7, TC #8, TC #9, TC #10, TC #11, TC #12

6.6 getRAZthrFt

Node	Variable	DCU	DPU
245-in	altFt	∅	{(246,247-out), (246,248), (248,249-out), (248,250), (250,251-out), (250,252), (252,253-out), (252,254), (254,255-out), (254,257-out)}

*Test cases for path coverage also sufficient for all uses criterion
* See 5.6, TC #7, TC #8, TC #9, TC #10, TC #11, TC #12

6.7 getTAZthrFt

Node	Variable	DCU	DPU
260-in	altFt	∅	{(261, 262-out), (261, 264-out)}

*Test cases for path coverage also sufficient for all uses criterion
* See 5.7, TC #1, TC #2

6.8 getModTauS

Node	Variable	DCU	DPU
297-in	rangeNmi	{298-out}	∅
297-in	closureRateKnots	{298-out}	∅
297-in	dmodNmi	{298-out}	∅

*Test cases for path coverage also sufficient for all uses criterion
* See 5.8, TC #1

6.9 getVvelForAlim

Node	Variable	DCU	DPU
366-in	sense	∅	{(372, 373), (372, 376), (376, 377), (376, 381)}
366-in	altFt	{373, 377}	∅
366-in	vsepAtCpaFt	{373,377}	∅
366-in	intrProjAtFt	∅	∅
366-in	rangeTauS	{373,377}	∅
371	toReturn	{383-out}	{(374,375), (374,383-out), (378,379),(378,383-out)}
373	toReturn	{383-out}	{(374,375), (374,383-out)}
375	toReturn	{383-out}	∅
377	toReturn	{383-out}	{(378,379),(378,383-out)}
379	toReturn	{383-out}	∅
381	toReturn	{383-out}	∅

*Test cases for path coverage also sufficient for all uses criterion
* See 5.9, TC #9, TC #10, TC #11, TC #12, TC #13

6.10 determineActionRequired

Node	Variable	DCU	DPU
40-in	intruder	{41-49, 87-94}	∅
41-49	intrCopy	{41-49, 56-58, 60-75}	∅
41-49	connection	{52, 54, 58, 60-75, 78-83}	{(53,55), (53, 54)}
41-49	threatClass	{87-94}	{(51, 52), (51, 77), (77, 78-83), (77, 87-94)}
41-49	mySense	{56-58, 60-75}	∅
41-49	recRange	{78-83}	∅
54	mySense	{56-58, 60-75}	∅
56-58	mySense	{56-58, 60-75}	∅
60-75	recRange	∅	∅
78-83	connection-> currentSense	∅	∅
78-83	recRange.negative.valid	∅	∅
78-83	recRange.position.valid	∅	∅
87-94	intruder-> threatClassification	∅	∅

*See 5.10 for test case omission justification

6.11 determineThreatClass

Node	Variable	DCU	DPU
97-in	intrCopy	{98-130}	∅
97-in	conn	{98-130}	∅
98-130	userPosition	∅	{(134-136, 138-139), (134-136, 147), (138-139, 140), (138-139, 143)}
98-130	closingSpeedKnots	{98-130}	{(134-136,138-139), (134-136,147), (138-139, 140), (138-139, 143)
98-130	altSepFt	{(98-130)}	{(132, 134-136), (132, 151), (134-136, 138-139), (134-136, 147), (138-139, 140), (138-139, 143)}
98-130	verticalTauS	{134-136,138-139}	∅
98-130	taModTauS	∅	{(134-136, 138-139), (134-136, 147)}
98-130	raModTauS	∅	{(138-139, 140), (138-139, 143)}
98-130	newThreatClass	{154-out}	∅
151	newThreatClass	{154-out}	∅
147	newThreatClass	{154-out}	∅
143	newThreatClass	{154-out}	∅
140	newThreatClass	{154-out}	∅
98-130	slantRangeNmi	{98-130}	{(132, 134-136), (132, 151)}
98-130	prevThreatClass	∅	{(134-136,138-139), (134-136, 147), (138-139, 140), (138-139, 143)}

*Test cases for path coverage also sufficient for all uses criterion
* See 5.11, TC #7, TC #8, TC #9, TC #10

6.12 TauPassesTAThreshold

Node	Variable	DCU	DPU
157-in	altFt	∅	{(159,160), (159, 174-175), (160,161-out), (160,162), (162,163-out), (162,164), (164,165-out), (164,166), (166,167-out), (166,168), (168,169-out), (168,170), (170,171-out), (170,172-173-out), (174-175,176-out), (174-175,177), (177,178-out), (177,179), (179,180-out), (179,181), (181,182-out), (181,183), (183,184-out), (183,185), (185, 186-out), (185,186-out), (185,187-188-out)}
157-in	modTauS	∅	{(160,161-out), (160,162), (162,163-out), (162,164), (164,165-out), (164,166), (166,167-out), (166,168), (168,169-out), (168,170), (170,171-out), (170,172-173-out), (174-175,176-out), (174-175,177), (177,178-out), (177,179), (179,180-out), (179,181), (181,182-out), (181,183), (183,184-out), (183,185), (185, 186-out), (185,186-out), (185,187-188-out)}
157-in	vertTauS	∅	{(160,161-out), (160,162), (162,163-out), (162,164), (164,165-out), (164,166), (166,167-out), (166,168), (168,169-out), (168,170), (170,171-out), (170,172-173-out)}
157-in	vSepFt	∅	{(159,160), (159,174-175)}

*Test cases for path coverage also sufficient for all uses criterion
* See 5.12, TC #27, TC #28, TC #29, TC #30, TC #31, TC #32, TC #33, TC #34, TC #35, TC #37, TC #38, TC #39, TC #40

6.13 TauPassesRAThreshold

Node	Variable	DCU	DPU
192-in	altFt	⌀	{(194,195), (194,209-210), (195,196-out), (195,197), (197,198-out), (197,199), (199,200-out), (199, 201), (201, 202-out), (201,203), (203,204-out), (203,205), (205,206-out), (205, 207-208-out), (209-210,211-out), (209-210,212), (212,213-out), (212,214), (214,215-out), (214,216), (216,217-out), (216,218), (218,219-out), (218,220), (220,221-out), (220,222-223-out)}
192-in	modTauS	⌀	{(195,196-out), (195,197), (197,198-out), (197,199), (199,200-out), (199, 201), (201, 202-out), (201,203), (203,204-out), (203,205), (205,206-out), (205, 207-208-out), (209-210,211-out), (209-210,212), (212,213-out), (212,214), (214,215-out), (214,216), (216,217-out), (216,218), (218,219-out), (218,220), (220,221-out), (220,222-223-out)}
192-in	vertTauS	⌀	{(195,196-out), (195,197), (197,198-out), (197,199), (199,200-out), (199, 201), (201, 202-out), (201,203), (203,204-out), (203,205), (205,206-out), (205, 207-208-out)}
192-in	vSepFt	⌀	{(194,195), (194,209-210)}

*Test cases for path coverage also sufficient for all uses criterion
* See 5.13, TC #27, TC #28, TC #29, TC #30, TC #31, TC #32, TC #33, TC #34, TC #35, TC #37, TC #38, TC #39, TC #40

6.14 determineResolutionSense

Node	Variable	DCU	DPU
227-in	userAltFt	⌀	{(228,229-out), (228,231-out)}
227-in	intrAltFt	⌀	{(228,229-out), (228,231-out)}

*Test cases for path coverage also sufficient for all uses criterion
* See 5.14, TC #1, TC #2

6.15 getRecRangePair

Node	Variable	DCU	DPU
301-302-in	sense	{307-317}	{(306, 307-356), (306, 354-356), (319, 325-330), (319, 321-324),
301-302-in	userVvelFtPerM	{307-317}	∅
301-302-in	intrVvelFtPerM	{307-317}	∅
301-302-in	userAltFt	{307-317}	∅
301-302-in	intrAltFt	{307-317}	∅
301-302-in	rangeTauS	{307-317}	{(306, 307-317), (306, 354-356)}
304	positive	{359-361}	∅
304	negative	{359-361}	∅
307-317	absoluteMinVvelToAchieveAlim	{307-317, 325-330, 321-324}	∅
307-317	toPrint	{363-out}	∅
325-330	negative.maxVerticalSpeed	{359-361}	∅
325-330	negative.minVerticalSpeed	{359-361}	∅
325-330	positive.maxVerticalSpeed	{359-361}	∅
325-330	positive.minVerticalSpeed	{359-361}	∅
321-324	negative.maxVerticalSpeed	{359-361}	∅
321-324	negative.minVerticalSpeed	{359-361}	∅
321-324	positive.maxVerticalSpeed	{359-361}	∅
321-324	positive.minVerticalSpeed	{359-361}	∅
352-353	positive.valid	∅	∅

352-353	negative.valid	∅	∅
354-356	positive.valid	∅	∅
354-356	negative.valid	∅	∅
359-361	toPrint	{363-out}	∅

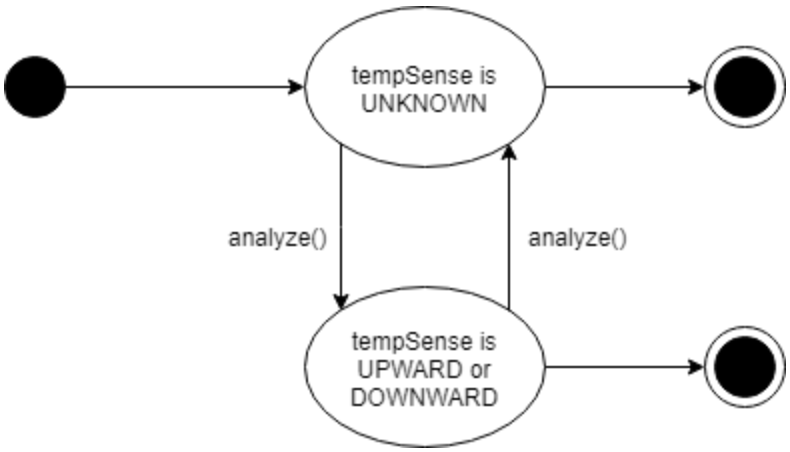
*Test cases for path coverage also sufficient for all uses criterion
* See 5.15, TC #4, TC #5, TC #6

7. Class Test Strategy

Modality of the Class

It was determined that the class Decider is non-modal. This was determined by

The class's state is determined by the variable tempSense_. This variable is used in the control flow of the determineActionRequired method. We have determined that this makes the variable quasi-modal, however, this is an out-of-scope value that is determined by external data. Also, we have determined that the only method being called externally is the analyze method, meaning there is no order of method calls. Because of this, we have determined that the Decider class is a non-modal class as the state does not affect the order of method calls.



Class Test Strategy

The techniques that will be used are the class testing strategies are following the Binder strategy of testing.

7.1 Method Scope Testing

7.1.1 Category Partition Test

7.1.1.1 Analyze

Primary	Secondary	Parameters	Variables
Calls the determineActionRequired method and passes the intruder object	N/A	see 3.1 for parameters, equivalence classes, and boundary value analysis	N/A

7.1.1.2 getThreatClassStr

Primary	Secondary	Parameters	Variables
Return string form of threatClass	N/A	see 3.2 for parameters, equivalence classes, and boundary value analysis	N/A

7.1.1.3 getAlimFT

Primary	Secondary	Parameters	Variables
Returns ALIM for given altitude, returns -1 if altitude is less than 1000	N/A	see 3.3 for parameters, equivalence classes, and boundary value analysis	N/A

7.1.1.4 getRADmodNmi

Primary	Secondary	Parameters	Variables
Returns the RA DMOD for the given altitude	N/A	see 3.4 for parameters, equivalence classes, and boundary value analysis	N/A

7.1.1.5 getTADmodNmi

Primary	Secondary	Parameters	Variables
Returns the TA DMOD for the given altitude	N/A	see 3.5 for parameters, equivalence classes, and boundary value analysis	N/A

7.1.1.6 getRAZthrFt

Primary	Secondary	Parameters	Variables

Returns the RA ZTHR for the given altitude, returns -1 if less than 1000	N/A	see 3.6 for parameters, equivalence classes, and boundary value analysis	N/A

7.1.1.7 getTAZthrFt

Primary	Secondary	Parameters	Variables
Returns the TA ZTHR for the given altitude, returns -1 if less than 1000	N/A	see 3.7 for parameters, equivalence classes, and boundary value analysis	N/A

7.1.1.8 getModTaus

Primary	Secondary	Parameters	Variables
Returns the modified tau given the range, clojure rate, and dmod	N/A	see 3.8 for parameters, equivalence classes, and boundary value analysis	N/A

7.1.1.9 getVvelForALIM

Primary	Secondary	Parameters	Variables
Returns the vertical velocity required to achieve alim	N/A	see 3.9 for parameters, equivalence classes, and boundary value analysis	N/A

7.1.1.10 determineActionRequired

Primary	Secondary	Parameters	Variables
Using the intruder, analyzes if the intruder is a threat, and begins the process of calculating actions required to avoid collision	Communicates to the intruder required information, and saves the calculated Sense of the event, as well as the positive and negative recommendation range pairs	see 3.10 for parameters, equivalence classes, and boundary value analysis	tempSense_, activeConnections_

Variables	Equivalence Class ID | Description	Representative
1. tempSense_	**1.1:** tempSense_ is UPWARD	**1.1:** UPWARD
	1.2: tempSense_ is DOWNWARD	**1.2:** DOWNWARD
	1.3: tempSense_ is UNKNOWN	**1.3:** UNKNOWN
	1.a: tempSense_ ∈ { }	**1.a:** ∅
	1.b: tempSense_ is a char	**1.b:** 's'

	1.c: tempSense_ is a String **1.d:** tempSense_ is an int **1.e:** tempSense_ is a boolean	**1.c:** “UP” **1.d:** 2 **1.e:** false

2. activeConnections_	**2.1:** activeConnections_ is a concurrency::concurrent_unordered_map with a valid string key and connection pointer
	
	2.a: active connections is an empty concurrency::concurrent_unordered_map
	2.b: activeConnections_ ∈ { }
	2.c: activeConnections_ is a char
	2.d: activeConnections_ is a string
	2e: activeConnections_ is a boolean
		2.1: concurrent_unordered_map<std::string, ResolutionConnection*>() {
		“Intr”,
		ResolutionConnection(
		“AB:CD:EF:12:34:56”
		“BA:DC:FE:21:43:65”,
		“198.0.10.16”,
		1234,
		Aircraft(“Intr”, “198.0.10.16”)
		)*
		}
		2.a: concurrent_unordered_map<std::string, ResolutionConnection*>()
		2.b: ∅
		2.c: ‘i’
		2.d: “Intruder”
		2.e: true

**** activeConnections_ repeats in TC#2 and TC#3**

TC ID	tempSense_	activeConnections_	Covered EC	Exp. Results

TC#1	UPWARD	concurrent_unordered_map<std::string, ResolutionConnection*>() {
		“Intr”,
		ResolutionConnection(
		“AB:CD:EF:12:34:56”
		“BA:DC:FE:21:43:65”,
		“198.0.10.16”,
		1234,
		Aircraft(“Intr”, “198.0.10.16”)
		)*
		}
			1.1, 2.1	PASS

TC#2	DOWNWARD	**	1.2, 2.1	PASS
TC#3	UNKNOWN	**	1.3, 2.1	PASS
TC#4	∅	concurrent_unordered_map<std::string, ResolutionConnection*>()	1.a, 2.a	FAIL
TC#5	's'	∅	1.b, 2.b	FAIL
TC#6	"UP"	'i'	1.c, 2.c	FAIL
TC#7	2	"Intruder"	1.d, 2.d	FAIL
TC#8	false	true	1.e, 2.e	FAIL

7.1.1.11 determineThreatClass

Primary	Secondary	Parameters	Variables
Determines and returns the appropriate ThreatClass based on the intruder and connection provided	N/A	see 3.11 for parameters, equivalence classes, and boundary value analysis	N/A

7.1.1.12 tauPassesTAThreshold

Primary	Secondary	Parameters	Variables
Returns whether the supplied taus triggers a traffic advisory at the supplied altitude	N/A	see 3.12 for parameters, equivalence classes, and boundary value analysis	N/A

7.1.1.13 tauPassesRAThreshold

Primary	Secondary	Parameters	Variables
Returns whether the supplied taus triggers a resolution advisory at the supplied altitude	N/A	see 3.13 for parameters, equivalence classes, and boundary value analysis	N/A

7.1.1.14 determineResoultionSense

Primary	Secondary	Parameters	Variables
Determines and returns the Sense based on the altitude of the aircraft and the intruder	N/A	see 3.14 for parameters, equivalence classes, and boundary value analysis	N/A

7.1.1.15 getRecRangePair

Primary	Secondary	Parameters	Variables

Returns a pair of recommendation ranges appropriate to the resolution advisory	N/A	see 3.15 for parameters, equivalence classes, and boundary value analysis	N/A

7.1.2 Source-Code Test

7.1.2.1 Analyze

***See section 5.1**

7.1.2.2 getThreatClassStr

***See section 5.2**

7.1.2.3 getAlimFt

***See section 5.3**

7.1.2.4 getRADmodNmi

***See section 5.4**

7.1.2.5 getTADmodNmi

***See section 5.5**

7.1.2.6 getRAZthrFt

***See section 5.6**

7.1.2.7 getTAZthrFt

***See section 5.7**

7.1.2.8 getModTauS

***See section 5.8**

7.1.2.9 getVvelForALIM

***See section 5.9**

7.1.2.10 determineActionRequired

***See section 4.10 for control flow graph of determineActionRequired**

Test Case ID	Parameters	Variables		Coverage nm	fghjdf
	intruder	tempSense_	activeConnections_	path	%
TC#1	Aircraft(“craft” “128.0.0.10”) { positionCurrent = LLA(80.6, 110.2, 1256.0, AngleUnits.DEGREE, DistanceUnits.FEET),	UPWARD	concurrent_unordered_map<std::string, ResolutionConnection*>() { “craft”, ResolutionConnection(“AB:CD:EF:12:34:56” “BA:DC:FE:21:43:65”,	40-in, 41-49, 51, 52, 53, 60-75, 87-94, 95-out	44%

	positionTime =
	12481697627
	positionOldTime =
	12481541267
	}
			“198.0.10.16”,
			1234,
			Aircraft(“craft”, “128.0.0.10”)
			) {
			consensus = true
			currentSense = UPWARD
			userPosition = LLA(80.0, 110.0, 1200.0,
			AngleUnits.DEGREE,
			DistanceUnits.FEET)
			threatClassification =
			TRAFFIC_ADVISORY
			}*
			}		

TC#2	Aircraft(“craft” “128.0.0.10”) {
	positionCurrent = LLA(
	80.6, 110.2, 1256.0,
	AngleUnits.DEGREE,
	DistanceUnits.FEET
	),
	positionTime =
	12481697627
	positionOldTime =
	12481541267
	}
		UNKNOWN	concurrent_unordered_map<std::string, ResolutionConnection*>() {
			“craft”,
			ResolutionConnection(
			“AB:CD:EF:12:34:56”
			“BA:DC:FE:21:43:65”,
			“198.0.10.16”,
			1234,
			Aircraft(“craft”, “128.0.0.10”)
			) {
			consensus = false
			currentSense = UNKOWN
			userPosition = LLA(80.0, 110.0, 1200.0,
			AngleUnits.DEGREE,
			DistanceUnits.FEET)
			threatClassification =
			TRAFFIC_ADVISORY
			}*
			}
				40-in, 41-49, 51, 52, 53, 55, 56-58, 60-75, 87-94, 95-out	56%

TC#3	Aircraft(“craft” “128.0.0.10”) { positionCurrent = LLA(80.6, 110.2, 1256.0,	UPWARD	concurrent_unordered_map<std::string, ResolutionConnection*>() { “craft”,	40-int, 41-49, 51, 52, 53, 55, 60-75,	63%

	AngleUnits.DEGREE,
	DistanceUnits.FEET
	),
	positionTime =
	12481697627
	positionOldTime =
	12481541267
	}
			ResolutionConnection(
			“AB:CD:EF:12:34:56”
			“BA:DC:FE:21:43:65”,
			“198.0.10.16”,
			1234,
			Aircraft(“craft” “128.0.0.10”)
			) {
			consensus = false
			currentSense = UNKOWN
			userPosition = LLA(80.0, 110.0, 1200.0,
			AngleUnits.DEGREE,
			DistanceUnits.FEET)
			threatClassification =
			TRAFFIC_ADVISORY
			}*
			}
				87-94,
				95-out	

TC#4	Aircraft(“craft”, “128.0.0.10”){
	positionCurrent = LLA(
	80.5, 110.4, 2356.0,
	AngleUnits.DEGREE,
	DistanceUnits.FEET
	)
	positionTime =
	12481697627
	positionOldTime =
	12481474833
	}
		UPWARD	concurrent_unordered_map<std::string, ResolutionConnection*>() {
			“craft”,
			ResolutionConnection(
			“AB:CD:EF:12:34:56”
			“BA:DC:FE:21:43:65”,
			“198.0.10.16”,
			1234,
			Aircraft(“craft” “128.0.0.10”)
			) {
			userPosition = LLA(82.0, 112.0, 812.0,
			AngleUnits.DEGREE,
			DistanceUnits.FEET)
			threatClassification =
			NON_THREAT_TRAFFIC
			}*
			}
				40-in, 41-49, 51, 77, 78-83, 87-94, 95-out	81%

TC#5	Aircraft(“craft”, “128.0.0.10”){	DOWNWARD	concurrent_unordered_map<std::string, ResolutionConnection*>() {	40-in, 41-49, 51, 77,	100 %

	positionCurrent = LLA(
	80.5, 110.4, 1256.0,
	AngleUnits.DEGREE,
	DistanceUnits.FEET
	)
	positionTime =
	12481697627
	positionOldTime =
	12481541267
	}
			“craft”,
			ResolutionConnection(
			“AB:CD:EF:12:34:56”
			“BA:DC:FE:21:43:65”,
			“198.0.10.16”,
			1234,
			Aircraft(“craft” “128.0.0.10”)
			) {
			userPosition = LLA(80.0, 110.0, 812.0,
			AngleUnits.DEGREE,
			DistanceUnits.FEET)
			threatClassification =
			RESOLUTION_ADVISORY
			}*
			}
				87-94,
				95-out	

*Parameters show valid class constructors as well as internal variable values to achieve path coverage.
*All variables used in conditions are derived from the stated values in intrCopy and the instance variables of tempSense_, and activeConnections_.
Test cases sufficient for path coverage and minimal condition coverage.
C_2 and C_{3b} met
C_2 subsumes C_1
C_1 subsumes C_0

7.1.2.11 determineThreatClass

***See section 5.11**

7.1.2.12 TauPassesTAThreshold

***See section 5.12**

7.1.2.13 TauPassesRAThreshold

***See section 5.13**

7.1.2.14 determineResolutionSense

***See section 5.14**

7.1.2.15 getRecRangePair

***See section 5.15**

7.1.3 Polymorphism Test

The polymorphism testing can not be done as there are no known sub or super classes within this class.

7.2 Class Scope Test

Due to the modality of the class being non-modal, not having a set execution order, we will be applying the “Alpha-Omega” execution order for testing. Following this execution order we will test the methods of the class in the following order; Constructor, Access, Boolean, Modifier, Iterator, All remaining methods, and finally the Destructor.

7.2.1 Constructor

7.2.1.1 Decider(Aircraft* thisAircraft, concurrency::concurrent_unordered_map<std::string, ResolutuionConnection*>*)

7.2.2 Access

No Access methods found in class.

7.2.3 Boolean

7.2.3.1 tauPassesTAThreshhold

See section 5.12 test cases TC#31 - 40

7.2.3.2 tauPassesRAThreshhold

See section 5.13 test cases TC#27 - 40

7.2.4 Modifier

7.2.4.1 determineActionRequired

See section 7.2.1.10 test cases - TC#1, TC#2, TC#3, TC#4

7.2.4.2 analyze

See section 5.1 test cases - TC#1

7.2.5 Iterator

No Iterator methods found in class.

7.2.6 All other Methods

7.2.6.1 determineThreatClass

See section 5.11 test cases - TC#7, TC#8, TC#9, TC#10

7.2.6.2 determineResoltutionSense

See section 5.14 test cases - TC#1, TC#2

7.2.6.3 getRecRangePair

See section 5.15 test cases - TC#4, TC#5, TC#6

7.2.6.4 getVvelForAlim

See section 5.9 test cases - TC#9, TC#10, TC#11, TC#12, TC#13

7.2.6.5 getThreatClassStr

See section 5.2 test cases - TC#9, TC#10, TC#11, TC#12

7.2.6.6 getAlimFt

See section 5.3 test cases - TC#7, TC#8, TC#9, TC#10

7.2.6.7 getRADmodNmi

See section 5.4 test cases - TC#7, TC#8, TC#9, TC#10, TC#11, TC#12

7.2.6.8 getTADmodNmi

See section 5.5 test cases - TC#7, TC#8, TC#9, TC#10, TC#11, TC#12

7.2.6.9 getRAZthrFt

See section 5.6 test cases - TC#7, TC#8, TC#9, TC#10, TC#11, TC#12

7.2.6.10 getTAZthrFt

See section 5. test cases - TC#1, TC#2

7.2.6.11 getModTauS

See section 5. test cases - TC#1

7.2.7 Destructor

No Destructor methods found in class.

7.3 Flattened Class Scope Test and Class Interaction Test

The flatted class scope test can not be achieved as there are no methods in the program that make calls to other classes or methods.

8. QA Results

The method in the Decider class in which we chose to perform unit testing on is getVvelForAlim. We chose this method for testing because it determines a key part of the Decider class. It provides information to the aircraft as to what speed is needed in order to avoid other planes, making it a critical part of the Decider class.

8.1 Test cases for getVvelForAlim

Test Cases	Expected result	Actual result	Irregularity found	Defect found
TC#1	PASS	PASS	No	No
TC#2	PASS	PASS	No	No
TC#3	PASS	PASS	No	No
TC#4	PASS	PASS	No	No
TC#5	PASS	PASS	No	No
TC#6	PASS	FAIL	No	Yes
TC#7	PASS	FAIL	No	Yes

*Where an irregularity is defined here as any block of code which can cause unexpected results, while a defect has the potential to cause fault and/or failure

8.2 Defects found in method

After testing in we found defects in test cases 6 and 7. The defect that was found in both test case 6 and 7 is an division by zero error. There is no try-catch or any other way to handle any type of exception.

Error	Fault	Failure
The error found in getVvelForAlim was a division by zero error. When a value of 0 was passed in the rangeTauS parameter, either an inf or -inf value was returned from the method.	The determined action isn't processable since a vertical velocity is of inf or -inf. This could potentially cause further errors in the decider, as well as other components that call the decider and utilize the action it determines.	This could potentially cause a cascade of failures in the TCAS system, which would cause a the system to fail to produce a resolution in the event of a collision course.

9. Conclusions & Recommendations

9.1 Possible Fixes for Defect Found

There are three possible ways to hand an error like this.

- When calling getVvelForAlim, a check could be performed on the value of rangeTauS, and handled prior to being passed into the method.
- When getVvelForAlim is called, a check could be made on the value of rangeTauS, and a return could reflect a devision by zero error.
- After a call is made, a check of getVvelForAlim's result could be performed to ensure it is not inf or -inf. This way you don't continue to pass a bad value to the decision of the decider.

10. Test Suitability

10.1 Steps taken to achieve accurate specification based testing

- Header file for the Decider component was used to derive equivalence classes.
- All limits and values for parameters were derived from specification.
- In conjunction, object and enumerator values were derived from header files.
- All values and limits derived from the specification correspond to the intended role of the function.
- Parts 1 & 2 done by Anthony and Alec, double checked by Mike.
- Part 3 & 4 done by all members of the group and double checked by all members.
- Proofreading and final formatting done by Mike and Alec.

10.2 Steps taken to achieve accurate control flow and data based testing

- The control flows were split up between the members and checked by other members for correctness when done.
- The data flow annotations were split up between the members and checked by other members for correctness when done.
- Alec added method descriptions for each method, checked by Anthony.
- Alex fixed object inputs and valid outputs for section 3 where needed.