Continuity Analysis of Floating Point Software

David Bushnell
TracLabs
david.h.bushnell@nasa.gov

Robust Software Engineering
NASA Ames Research Center

VVFCS element
SSAT Project
NASA Aviation Safety Program

7/14/11

Software Continuity: What is it? Why is it Important?

Continuity:

Small changes have small effects!

Software Continuity: What is it? Why is it Important?

Continuity:

Small changes have small effects!

"... unlike physical systems,
software is not continuous;
small changes can cause
large failures ..."

Peter Wegner, 1979

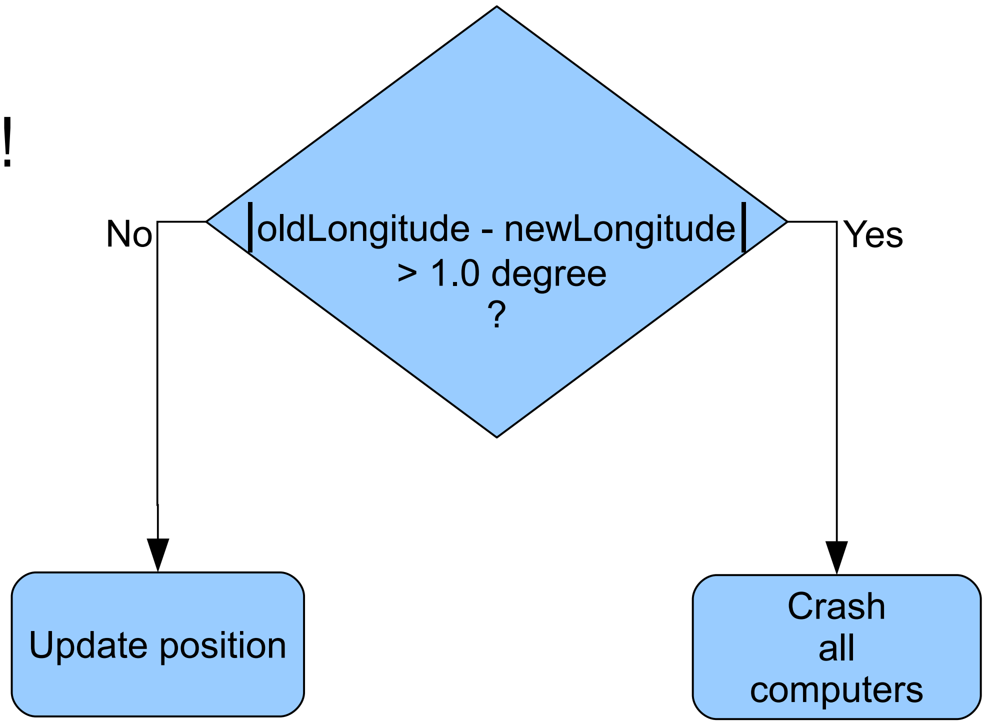
Software Continuity: What is it? Why is it Important?

Continuity:

Small changes have small effects!

"... unlike physical systems,
software is not continuous;
small changes can cause
large failures ..."

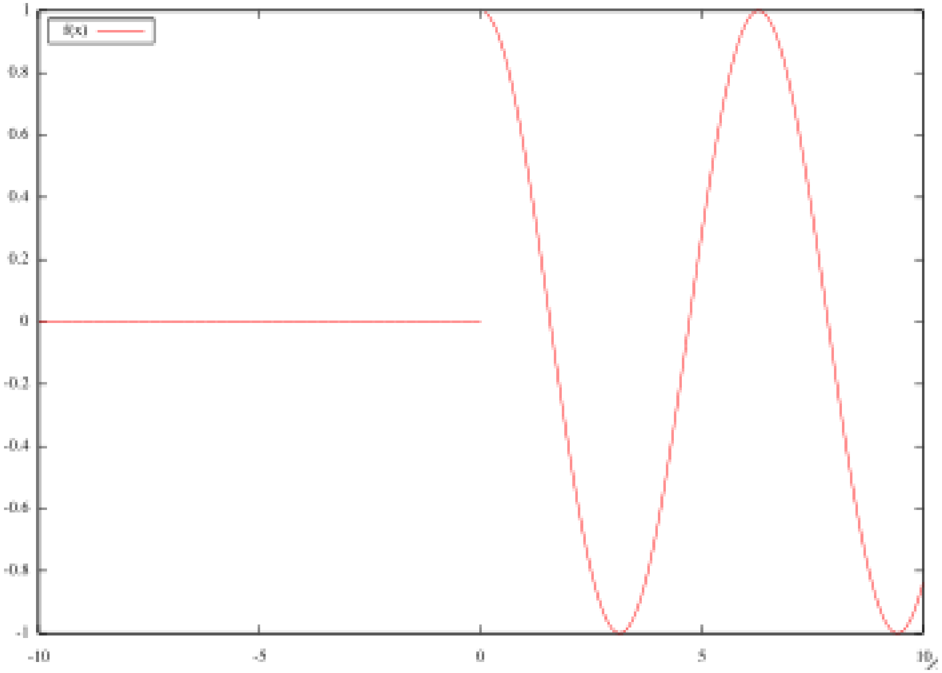
Peter Wegner, 1979



F22 Raptor
Crossing International Date Line
February 11, 2007

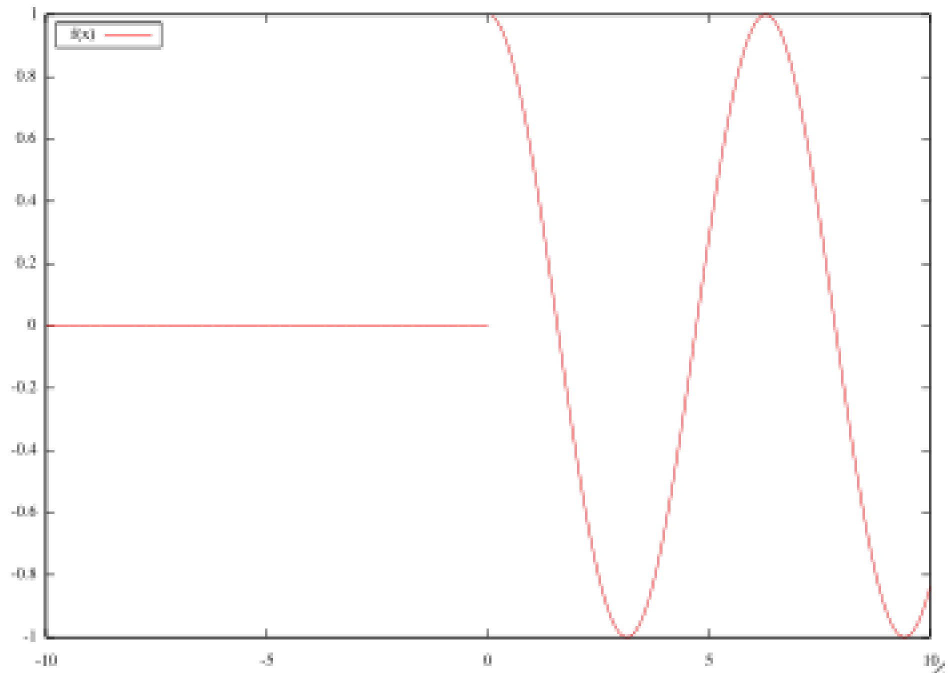
Problem Statement

```
double f(double x) {
  if (x < 0.0)
    return 0.0;
  else
    return cos(x);
}
```



Problem Statement

```
double f(double x) {
  if (x < 0.0)
    return 0.0;
  else
    return cos(x);
}
```



Where does the software's control structure cause `f(x)` to be discontinuous?

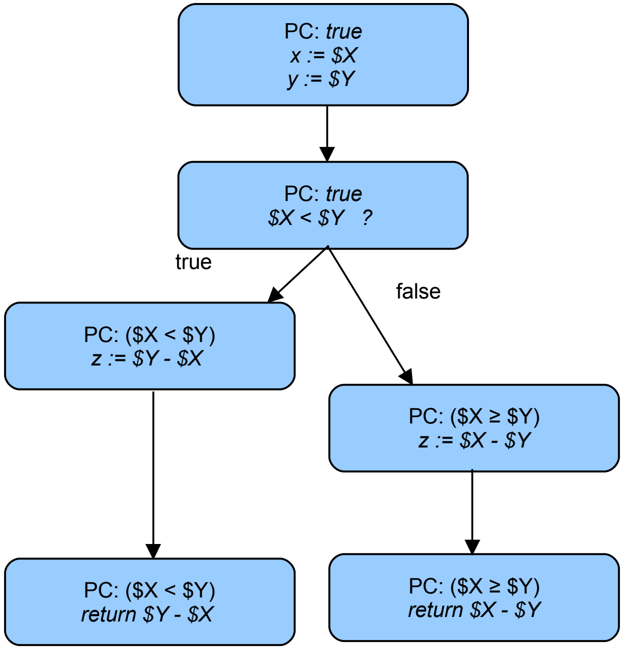
How to generate test cases near the discontinuities?

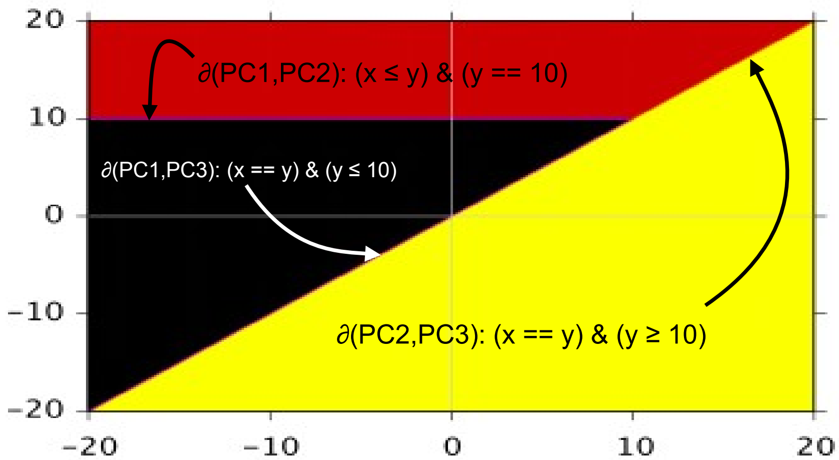
Approach

Symbolic Execution

+

Boundary Analysis





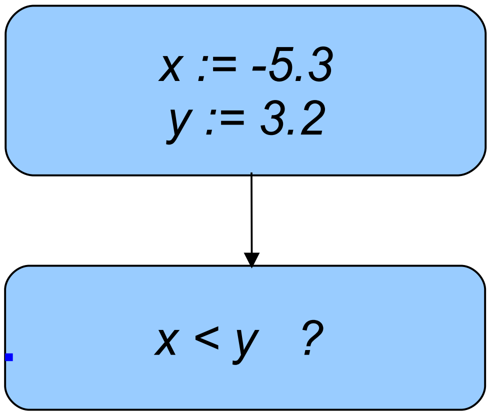
Concrete Execution for Testing

```
double g(double x, y) {

  double z;
  if (x < y)
    z = y-x;

  else
    z = x-y;

  return z;
}
```

x := -5.3
y := 3.2

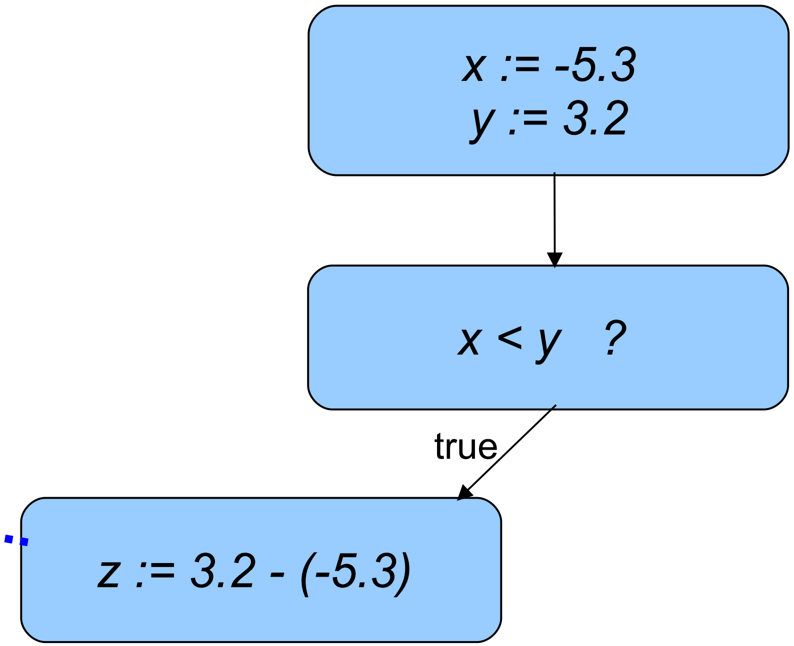
Concrete Execution for Testing

```
double g(double x, y) {

   double z;
   if (x < y)
      z = y-x;

   else
      z = x-y;

   return z;
}
```



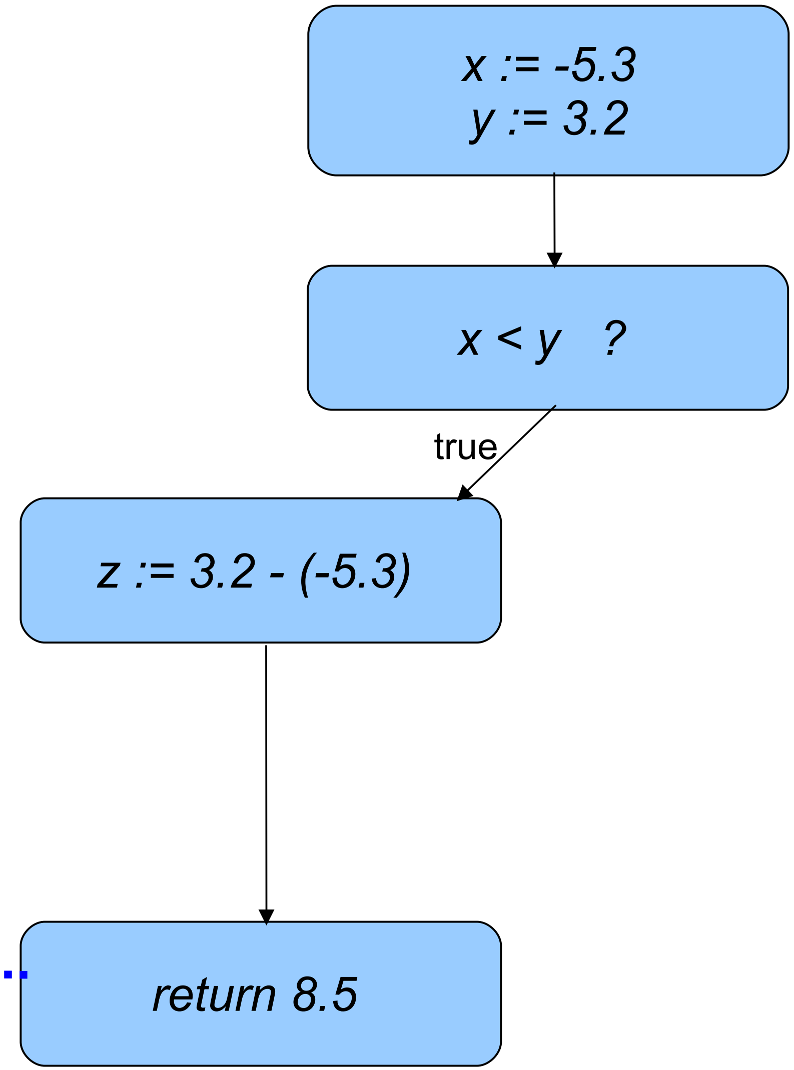
Concrete Execution for Testing

```
double g(double x, y) {

  double z;
  if (x < y)
    z = y-x;

  else
    z = x-y;


  return z;
}
```



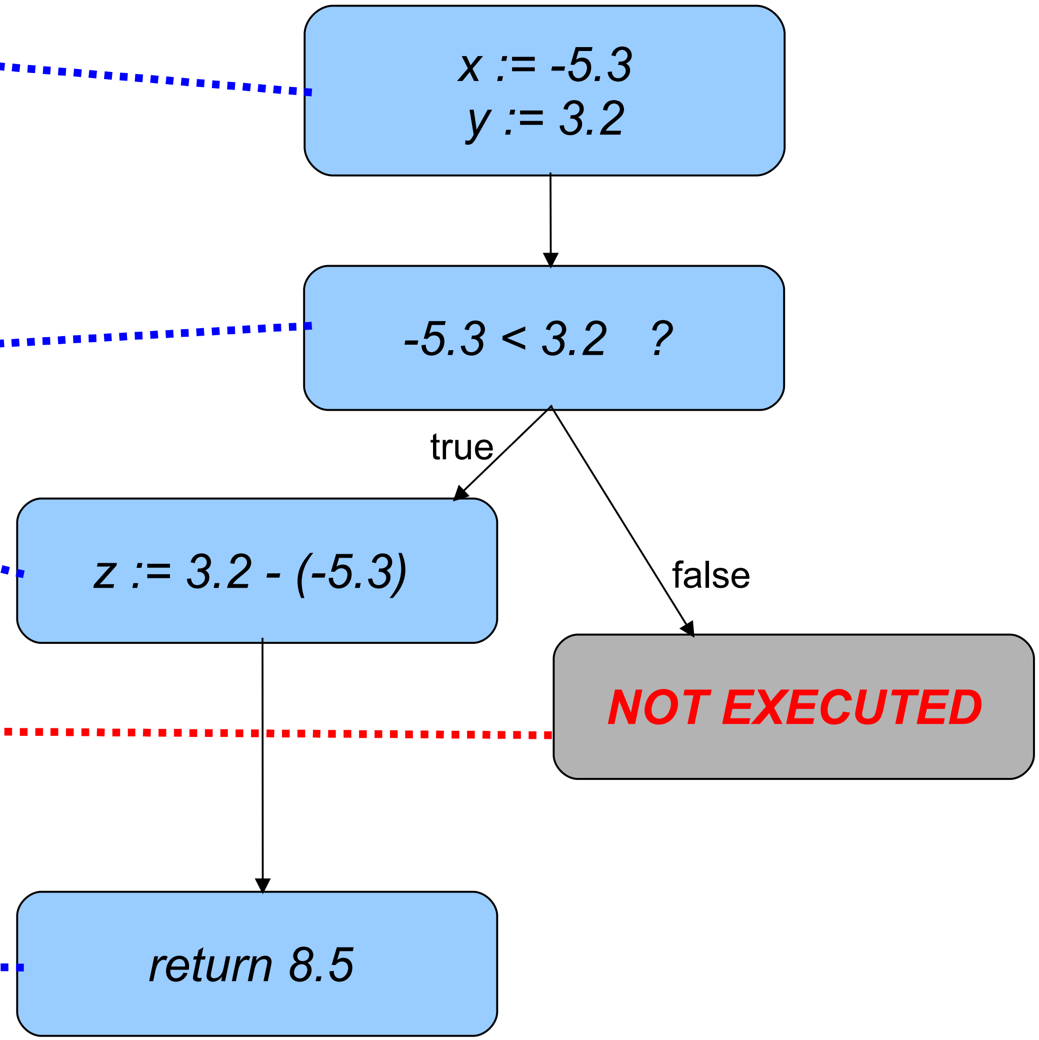
Concrete Execution for Testing

```
double g(double x, y) {

  double z;
  if (x < y)
    z = y-x;

  else
    z = x-y;

  return z;
}
```



Concrete Execution for Testing

```
double g(double x, y) {

  double z;
  if (x < y)
    z = y-x;

  else
    z = x-y;

  return z;
}
```



Symbolic Execution

```
double g(double x, y) {

  double z;
  if (x < y)
    z = y-x;

  else
    z = x-y;

  return z;
}
```

PC (Path Condition): *true*
x := $X
y := $Y

Symbolic Execution

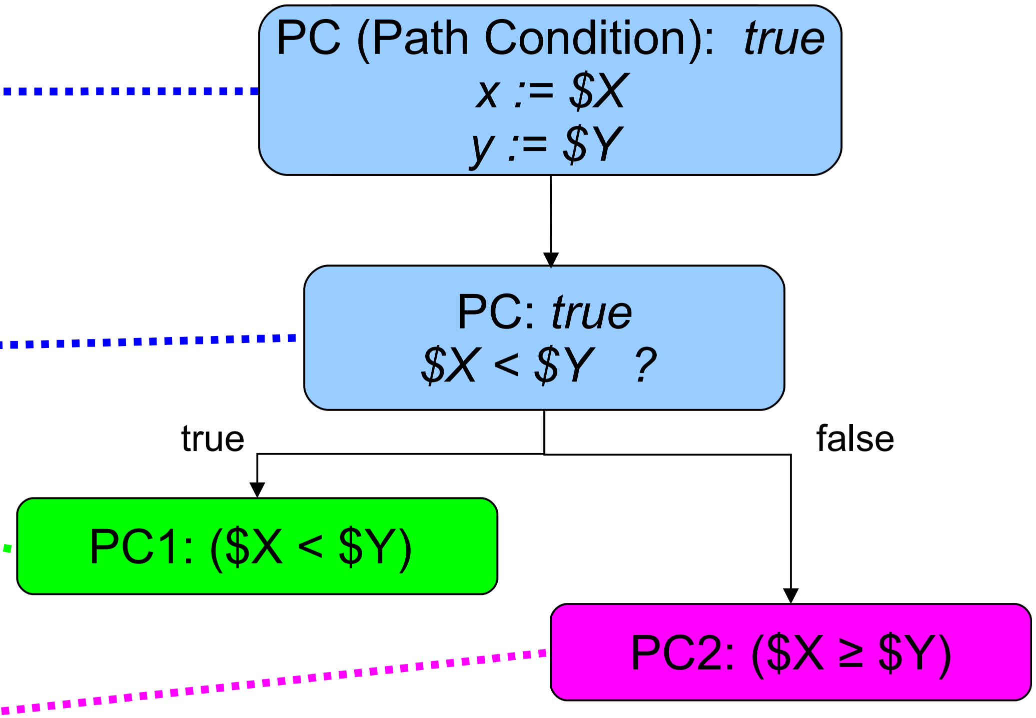
```
double g(double x, y) {

  double z;
  if (x < y)
    z = y-x;

  else
    z = x-y;

  return z;
}
```

PC (Path Condition): *true*
x := $X
y := $Y

PC: *true*
$X < $Y   ?

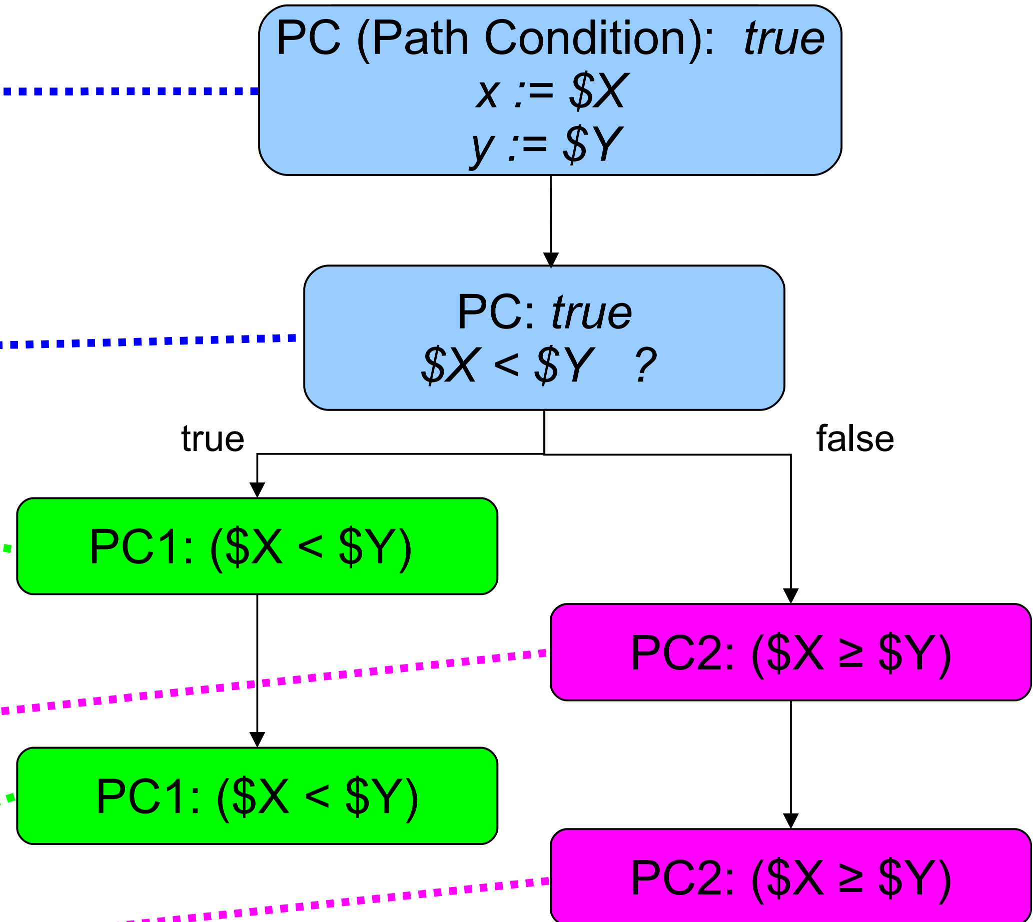
Symbolic Execution

```
double g(double x, y) {

   double z;
   if (x < y)
     z = y-x;

   else
     z = x-y;


   return z;
}
```



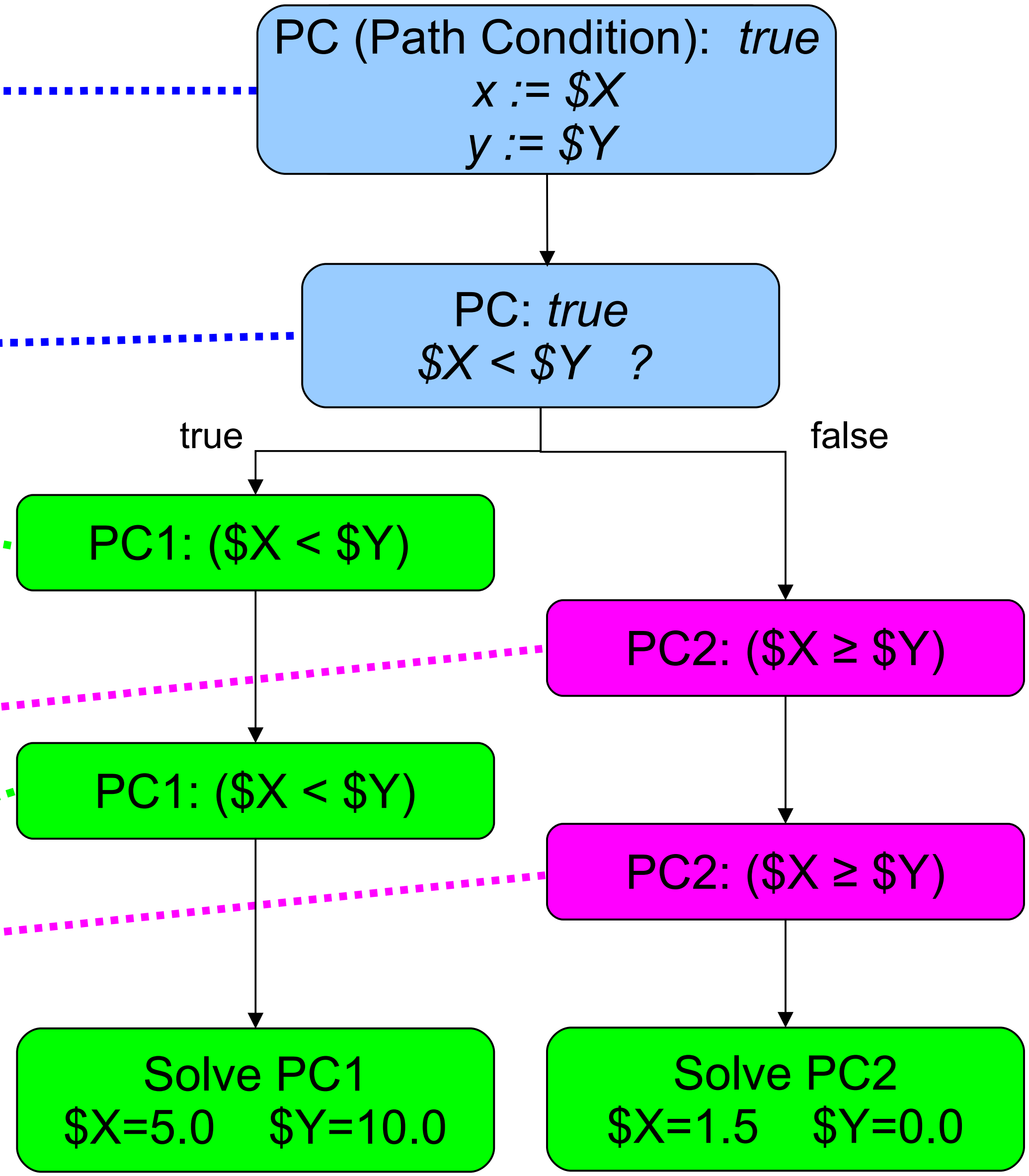
Symbolic Execution

```
double g(double x, y) {

  double z;

  if (x < y)

    z = y-x;

  else

    z = x-y;

  return z;

}
```



Symbolic Execution

```
double g(double x, y) {

  double z;

  if (x < y)

     z = y-x;

  else

     z = x-y;

  return z;
}
```



Java Pathfinder: JPF

- General purpose Java verification tool
  - Software model checking (concurrency bugs: deadlock, race conditions)
  - UML statechart verification
  - Floating point overflow/underflow/catastrophic cancelation
  - Symbolic execution (test case generation)
  - Many more ...
- Developed at NASA/Ames
  - Started in 1999, work continues to this day
  - Recognized by numerous awards from both NASA and the verification community
  - Open source since 2005

Symbolic Pathfinder: SPF

Symbolic Pathfinder:

Extends JPF for symbolic execution

Supports booleans, integers, floating point, complex data structures, strings

Supports many constraint solvers:

- Coral

  High performance non-linear floating point constraint solver

- Choco

  Solver for linear/non-linear, integer/floating point constraints, mixed constraints

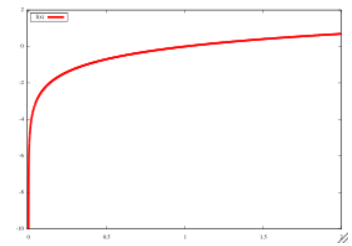
- CVC3

  Solver for integer/real linear constraints

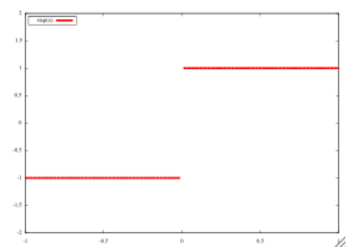
Discontinuities

Discontinuities come in many forms, including:

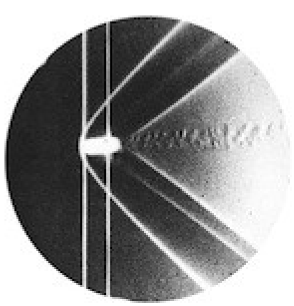
- Non-convergence: $y = -\sum_{1}^{\infty} \frac{(1-x)^n}{n}$



- Blackbox: `y = step(x)`



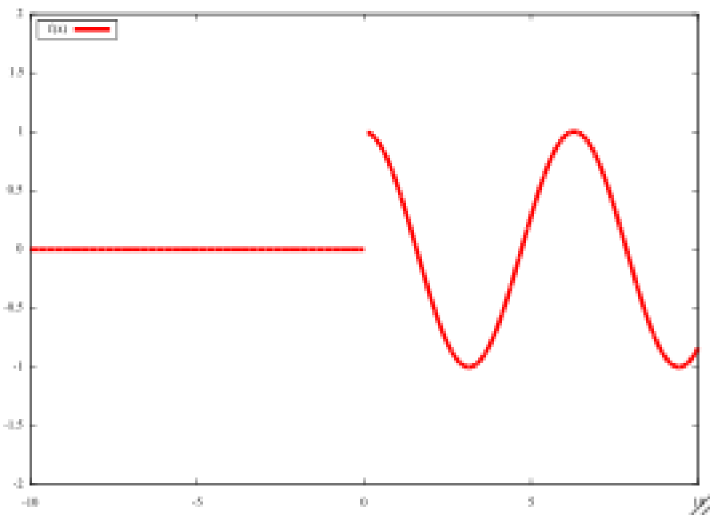
- Physical discontinuities in the model e.g. shockwaves:



Discontinuities

We analyze discontinuities of another type:
Those that arise from the structure of the software

```
if (x < 0)
  return 0.0
else
  return cos(x);
```



Terminology

Path Condition: Boolean expression in symbolic inputs specifying a single execution path through the software.

Written as `PC, PC1, PC2, …`

All inputs satisfying a given path condition force the same execution path.

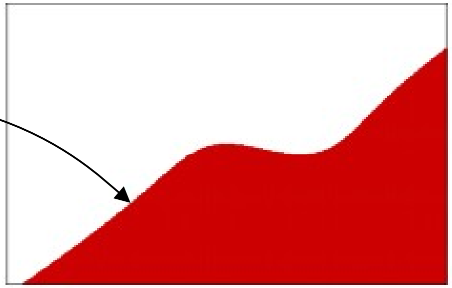
Example: `PC1: (x>0.0)∧(x+y<z)`

Terminology

***Boundary** between path conditions PC1 and PC2:*

The set of input values at which the software changes from the execution path for PC1 to the path for PC2.

Written as ∂(PC1,PC2)



Example:

if PC1:(x > y) and PC2:(x $\leq$ y)

then ∂(PC1,PC2): (x = y)

Terminology

Path Function: The function computed by all inputs satisfying a single path condition.

Example:

```
if (x < 0)
 return 0.0
else
 return cos(x);
```

For `PC1: (x < 0)`

`PF1: 0.0`

For `PC2: (x ≥ 0)`

`PF2: cos(x)`

Terminology

Discontinuity Condition: Boolean expression in symbolic inputs specifying a region where the function computed by the software is discontinuous.

Written as `DC`, `DC1`, ...

We are interested in discontinuity conditions associated with the software's control structures.

Calculating Boundaries of Path Conditions

First:

Calculate the *closure* of a relation.

For the floating point relations

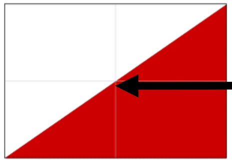
$\{<, \leq, =, \neq, \geq, >\}$

define the *closure* of the relation as in the table:

Relation	Closure
F < G	F ≤ G
F ≤ G	F ≤ G
F = G	F = G
F ≠ G	*true*
F ≥ G	F ≥ G
F > G	F ≥ G

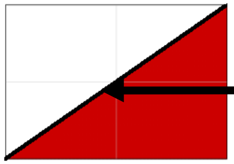
Example:

The region defined by (x < y) is:



Border missing

Its closure is (x ≤ y):



Border included

Calculating Boundaries of Path Conditions

Second:

Define the *closure of a path condition:*

Given a path condition `PC`

$$PC = R1 \wedge R2 \wedge \ldots$$

where $R1$, $R2$, ... are primitive relations.

Its closure is written `PC` or `cl(PC)`
and is given by

$$\overline{PC} = \overline{R1} \wedge \overline{R2} \wedge \ldots$$

Calculating Boundaries of Path Conditions

Finally:

Define the *boundary between two path conditions* `PC1` and `PC2` as:

$$\partial\left(PC1\,,PC2\right)=\overline{PC1}\wedge\overline{PC2}$$

Example: Boundaries

```
double step2(double x, double y)
   if (x < y)
     return 0.0;
   else
     return 1.0;
```

Path Conditions: $PC1 : (x < y)$ $PC2 : (x \geq y)$

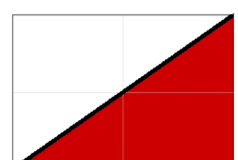
Boundary:

$$\partial(PC1, PC2) \equiv \overline{PC1} \wedge \overline{PC2}$$
$$\equiv (x \leqslant y) \wedge (x \geqslant y)$$
$$\equiv (x = y)$$

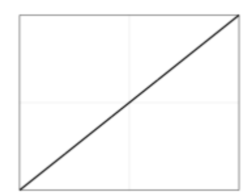
PC1:

$\wedge$

PC2:



$\equiv$ (x=y)



Discontinuity Conditions From Control Structures

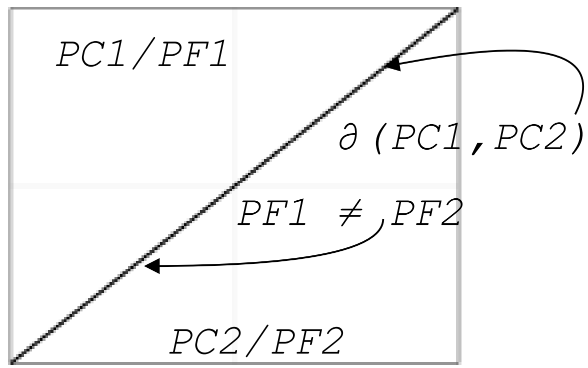
How can the output be discontinuous when control switches from one execution path to another?

Answer: when the values computed along Path 1 `(PC1)` do not match the values computed along Path 2 `(PC2)` at the crossover

Discontinuity Conditions From Control Structures

Crossover happens at the boundary between Path 1 and Path 2

The values computed are the path functions `PF1` and `PF2`



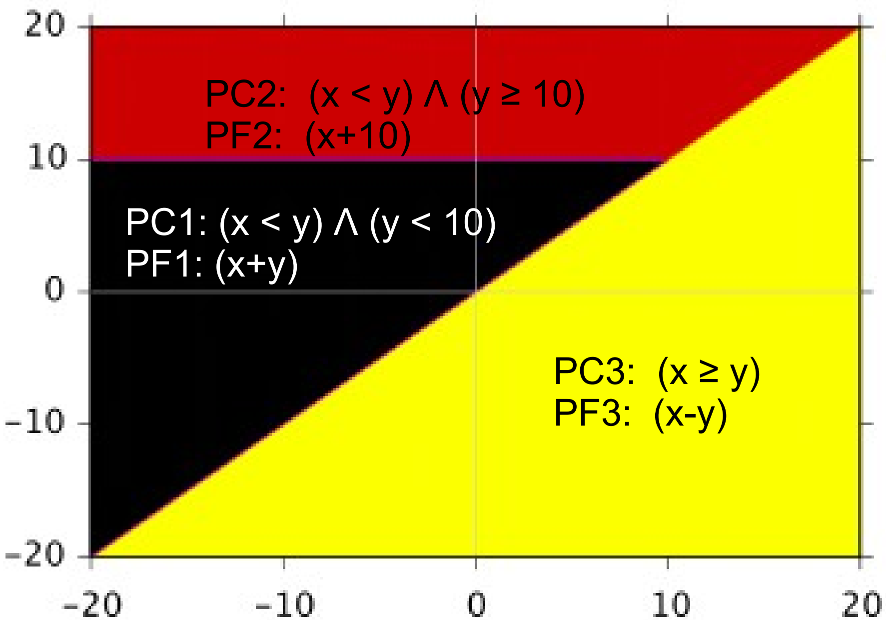
So when control switches from Path 1 to Path 2, the software will be discontinuous at all points satisfying:

$$\partial(PC1, PC2) \wedge (PF1 \neq PF2)$$

Example: Discontinuity Conditions

```
double sa(double x, double y)
    if (x < y)
     if (y < 10)
      return x+y;
     else
      return x+10;
    else
      return x-y;
```

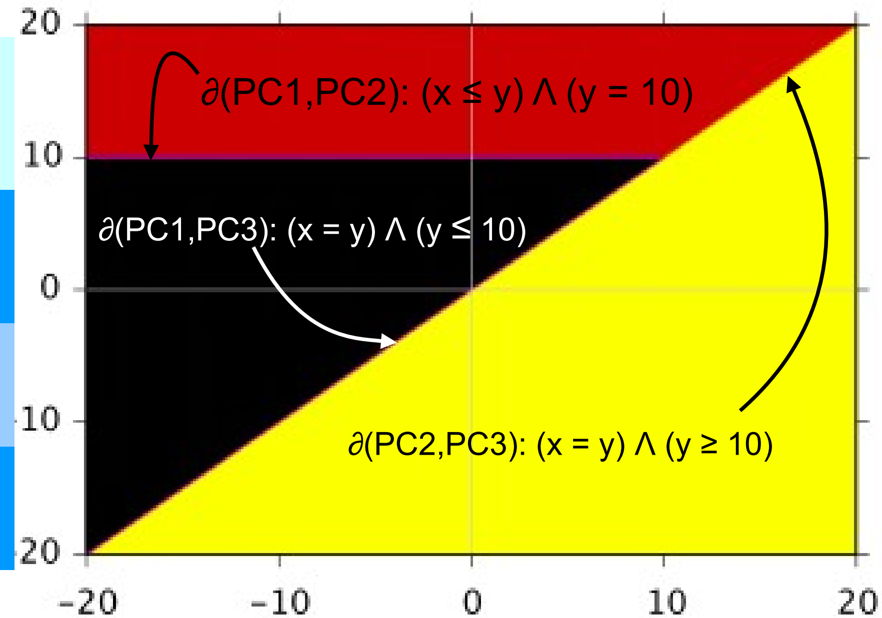
Path Condition	Path Function
PC1: $(x<y) \land (y<10)$	$x + y$
PC2: $(x<y) \land (y\geq 10)$	$x + 10$
PC3: $(x\geq y)$	$x - y$



Example: Discontinuity Conditions

```
double sa(double x, double y)
    if (x < y)
     if (y < 10)
      return x+y;
     else
      return x+10;
    else
      return x-y;
```

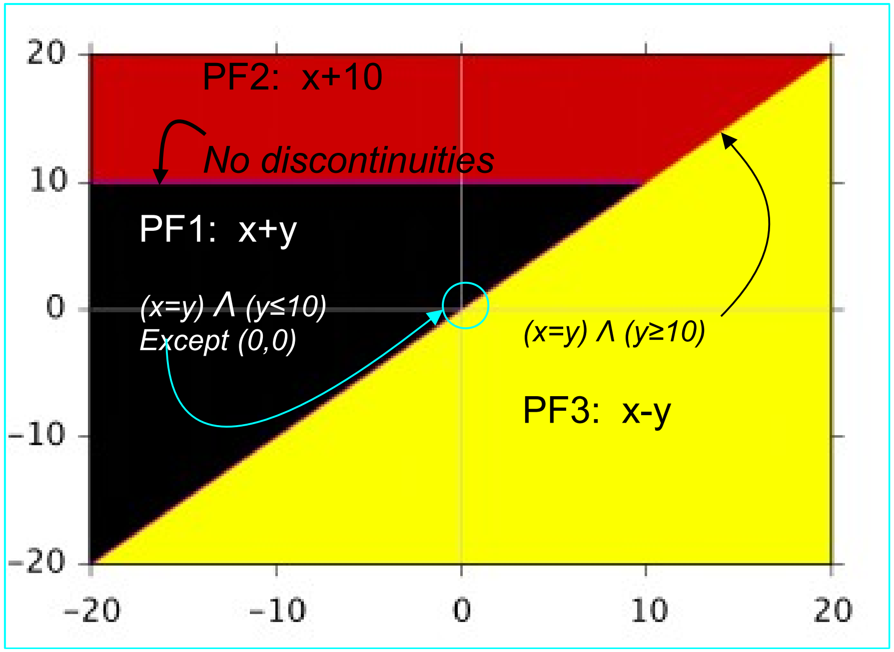
PCa PCb	∂(PCa,PCb) = cl(PCa) Λ cl(PCb)
PC1: (x<y) Λ (y<10) PC2: (x<y) Λ (y≥10)	(x≤y) Λ (y=10)
PC1: (z<y) Λ (y<10) PC3: (x≥y)	(x=y) Λ (y≤10)
PC2: (x<y) Λ (y≥10) PC3: (x≥y)	(x=y) Λ (y≥10)



Example: Discontinuity Conditions

```
double sa(double x, double y)
    if (x < y)
     if (y < 10)
      return x+y;
     else
      return x+10;
    else
      return x-y;
```

PCa PCb	∂(PCa,PCb) ∧ (PFa ≠ PFb)
PC1: $(x < y) \land (y < 10)$ PC2: $(x < y) \land (y \geq 10)$	$(x \leq y) \land (y = 10) \land (x + y \neq x + 10)$ $\equiv (x \leq y) \land (y = 10) \land (y \neq 10)$ $\equiv$ false
PC1: $(x < y) \land (y < 10)$ PC3: $(x \geq y)$	$(x = y) \land (y \leq 10) \land (x + y \neq x - y)$ $\equiv (x = y) \land (y \leq 10) \land (y \neq 0)$
PC2: $(x < y) \land (y \geq 10)$ PC3: $(x \geq y)$	$(x = y) \land (y \geq 10) \land (x + 10 \neq x - y)$ $\equiv (x = y) \land (y \geq 10)$



Status

Implemented on top of Symbolic Path Finder:

- Open source JPF extension: jpf-continuity
- Computes boundaries between path conditions
- Computes constraints along borders:
  - Discontinuity
  - Continuity
- Integrated with Coral constraint solver
  - Can solve highly non-linear floating point constraints
- Computes symbolic derivatives of path functions

Status

Demonstrated automatic generation of test cases at border discontinuities

- Applied to selected code from TSAFE
  (Tactical Separation Assisted Flight Environment)
- Automatically generates test cases demonstrating discontinuities along path condition borders

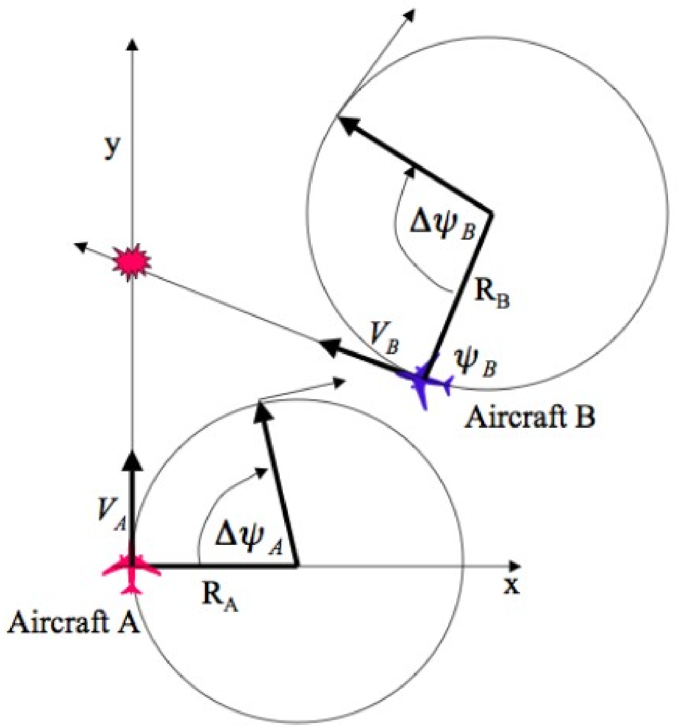


Diagram from: H. Erzberger, 2009

Beyond Discontinuity Analysis

Robustness:

- Where along the border are the path functions "extremely discontinuous"?
  Discontinuous: `PF1≠PF2`
  Non-Robust: `|PF1-PF2|>ε`

Global border behavior:

- How bad do the discontinuities get?
- How badly to the continuous points behave?

Beyond Discontinuity Analysis

- Compositional Analysis
  - If $f(x1,\ldots,xn) = h(g(x1,\ldots,xn))$ and we know the discontinuities of $h()$ and $g()$, what are the discontinuities of $f()$?
- Guided Test Generation
  - How to verify regions for safe execution of code?
- Polynomial Approximation
  - Useful when finding min, max, zeros:
    - Approximate with low-degree polynomials
    - Need to know discontinuities

Availability

JPF (Java Path Finder), SPF (Symbolic Path Finder), and jpf-continuity are all open source software

They are available for download at

- JPF: http://babelfish.arc.nasa.gov/trac/jpf/
- SPF: http://babelfish.arc.nasa.gov/trac/jpf/wiki/projects/jpf-symbc
- jpf-continuity: https://jpfcontinuity@bitbucket.org/jpfcontinuity/jpf-continuity

Acknowledgments

Thanks to Saswat Anand (Georgia Tech) for discussions and clarifications

Thanks to Misty Davies, Dimitra Giannakopoulou, Michael Lowry, Corina Pasareanu, and Neha Rungta (NASA Ames Robust Software Engineering group) for critiques and suggestions.

References

"Continuity analysis of programs", Chaudhuri, S., Gulwani, S., Lublinerman, R., POPL, 2010

"Symbolic robustness analysis", R. Majumdar and I. Saha. Real-Time Systems Symposium, IEEE International, 0:355–363, 2009

"Continuity in Software Systems", Hamlet, D., ISSTA, 2002

"Generalized Symbolic Execution for Model Checking and Testing", Khushid, S., Pasareanu, C., Visser, W., Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, 2003.

"Symbolic Execution and Program Testing", King, J.C., Commu- nications of the ACM, vol. 19(7), pp. 385–394, 1976.

"Separation Assurance in the Future Air Traffic System", Erzberger, H., ENRI International Workshop on ATM/CNS, 2009

Questions?