NJIT New Jersey's Science & Technology University

COMPUTER SCIENCE

On Faults and Faulty Programs

Ali Jaoua, Marcelo Frias, Ali Mili

RAMICS 2014

Marienstatt im Westerwald, Apr/May 2014

Outline

- What's Wrong with Faults
- Correctness and Relative Correctness
- Faults and Monotonic Fault Removal
- Definite Faults
- Beyond Nice Definitions: Applications
- Conclusion

What's Wrong with Faults

2004: Avizienis, Laprie, Randell, Landwehr

- Terminology for dependability
  - Fault (attribute of a product that precludes its correct behavior).
  - Error (state of the program produced by sensitization of the fault).
  - Failure (violation of the system specification pursuant the sensitization of a fault).
- Failure
  - Well defined property, with respect to a well defined specification

What's Wrong with Faults

Many issues with defining faults:

- Characterization of a fault dependent on implicit design:
  - Has no official existence.
  - Is not documented/ validated/ vetted.
  - Gap between designer's intent, tester's understanding of the intent.
  - Contingent upon implicit assumptions about other parts of the product.

What's Wrong with Faults

The same failure may be blamed on many fault configurations:

- **Neither the location,**
- **Nor the number,**
- **Nor the nature of the fault is determined**
  - **Wrong operator,**
  - **Wrong operand,**
  - **Wrong condition,**
  - **Missing path.**
- **What does it mean to remove the fault?**
  - **It certainly does not mean that now the program is correct, since it may still have other faults.**
  - **We are lucky if we did not make it worst.**

What's Wrong with Faults

Specification: $R = \{(x, x')|x' = x^2\ mod\ 5\}$.

```
{read(x);  x=x*2;  x=x%5; write(x);}
```

```
{read(x);  x=x*2;  x=x%5; write(x);}
```

```
{read(x);  x=x*2;  x=((x/2)**2)%5; write(x);}
```

```
{read(x);  x=x*2;  x=((x/2)**2); x=x%5; write(x);}
```

```
{read(x);  x=x*2;  x=x*x; x=(x/4)%5; write(x);}
```

What's Wrong with Faults

This casts a shadow on such concepts as

- Fault density,
- Fault proneness,
- Estimates of the number of faults.

If the same failure can be remedied by changing one statement or two statements,

- Does that count as one fault or two faults,

If a missing path is remedied by adding a new path of 20 lines,

- how many faults is that?

Outline

- What's Wrong with Faults
- Correctness and Relative Correctness
- Faults and Monotonic Fault Removal
- Definite Faults
- Beyond Nice Definitions: Applications
- Conclusion

Correctness and Relative Correctness

Program functions

```
#include <iostream>  ...  ...  ...
void count (char q[]) {int let, dig, other, i, l; char c;
   i=0; let=0; dig=0; other=0; l=strlen(q);  // body init
   while (i<l) {                             // cond t
      c = q[i];                              // body b0
      if (’A’<=c && ’Z’>c) let+=2;           // cond c1, body b1
      else
      if (’a’<=c && ’z’>=c) let+=1;          // cond c2, body b2
      else
      if (’0’<=c && ’9’>=c) dig+=1;          // cond c3, body b3
      else
         other+=1;                           // body b4
      i++;}                                  // body inc
   printf ("%d %d %d\n", let, dig, other);}  //  body p
```

Correctness and Relative Correctness

Program functions

$$COUNT = INIT \circ ((T \cap B)^* \cap \widehat{\overline{T}}) \circ P.$$

$$B = B0 \circ NEST \circ INC,$$

$$NEST = (C1 \cap B1) \cup \overline{C1} \cap ((C2 \cap B2) \cup \overline{C2} \cap ((C3 \cap B3) \cup \overline{C3} \cap B4)).$$

Granularity determines precision of fault diagnosis.

Correctness and Relative Correctness

Refinement, Correctness

Definition 2.1. Refinement, due to [BEM92]. *Let R and R' be two relations on set S. We say that R* refines *relation R' (and we write: $R \sqsupseteq R'$) if and only if: $RL \cap R'L \cap (R \cup R') = R'$.*

Program g is correct with respect to R iff G refines R.

Program g is correct with respect to R iff $dom(R \cap G) = dom(R)$.

Correctness and Relative Correctness

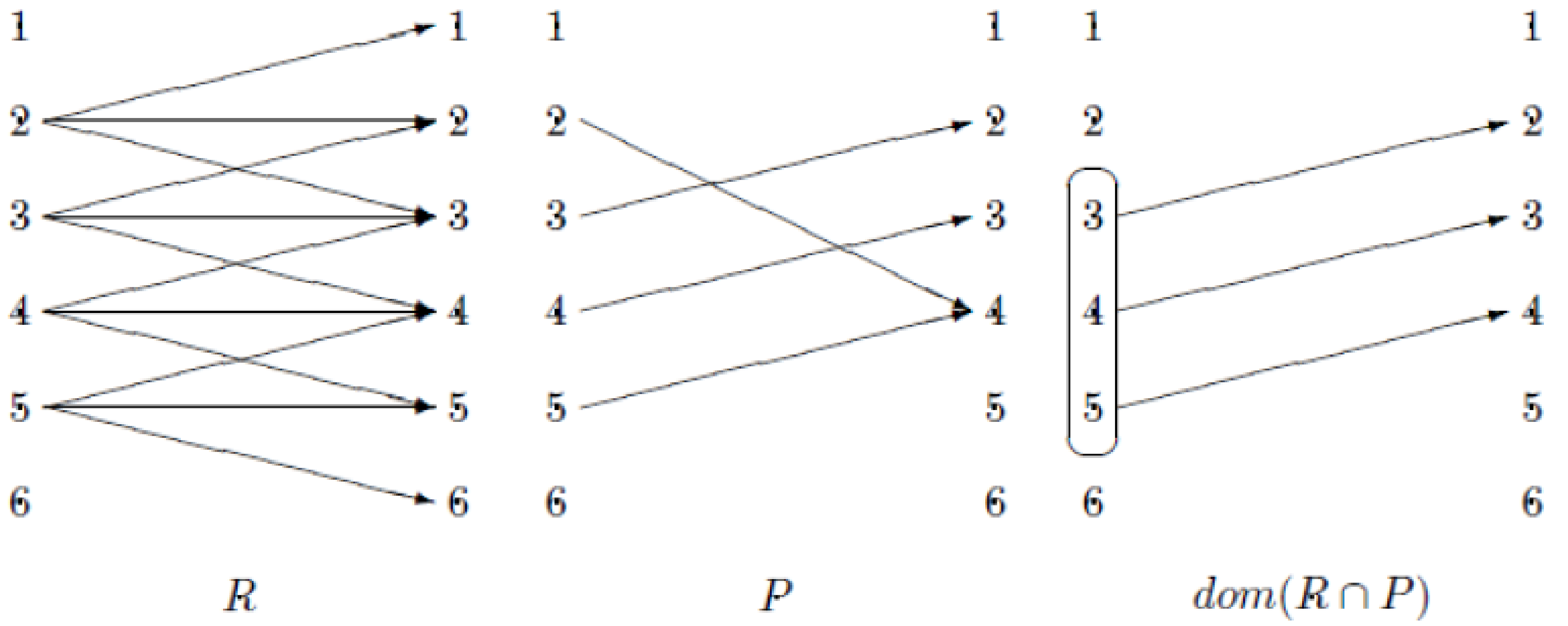


Figure 1: Interpretation of $dom(R \cap P)$

Correctness and Relative Correctness

Relative Correctness

Definition 2.4. Relative Correctness. *Given a relation R on space S and two programs* g *and* g' *on space S, we say that* g *is* more-correct *than* g' *with respect to R if and only if*

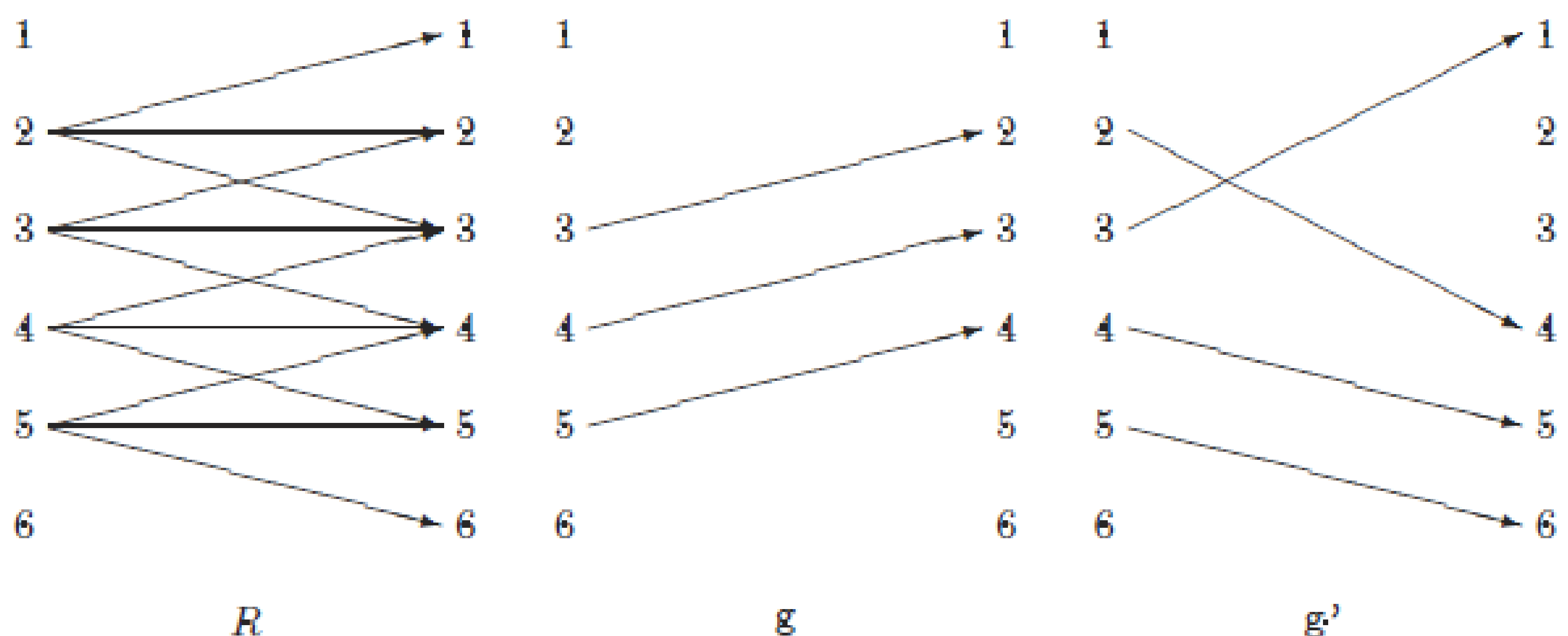
$$(G \cap R)L \supseteq (G' \cap R)L.$$

Also, we say that g *is* strictly-more-correct *than* g' *with respect to R if and only if*

$$(G \cap R)L \supset (G' \cap R)L.$$

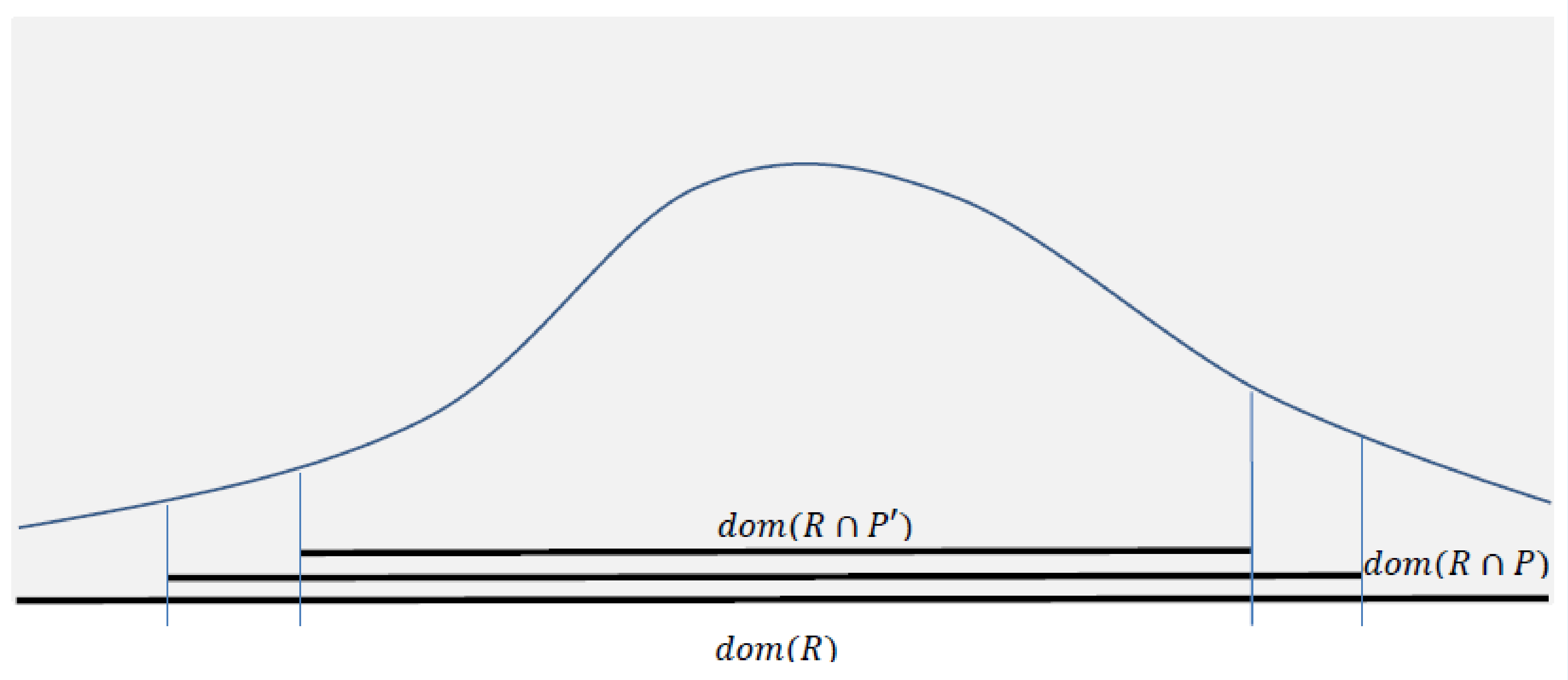
Correctness and Relative Correctness

Relative Correctness does not mean preserving correct behavior:



Correctness and Relative Correctness

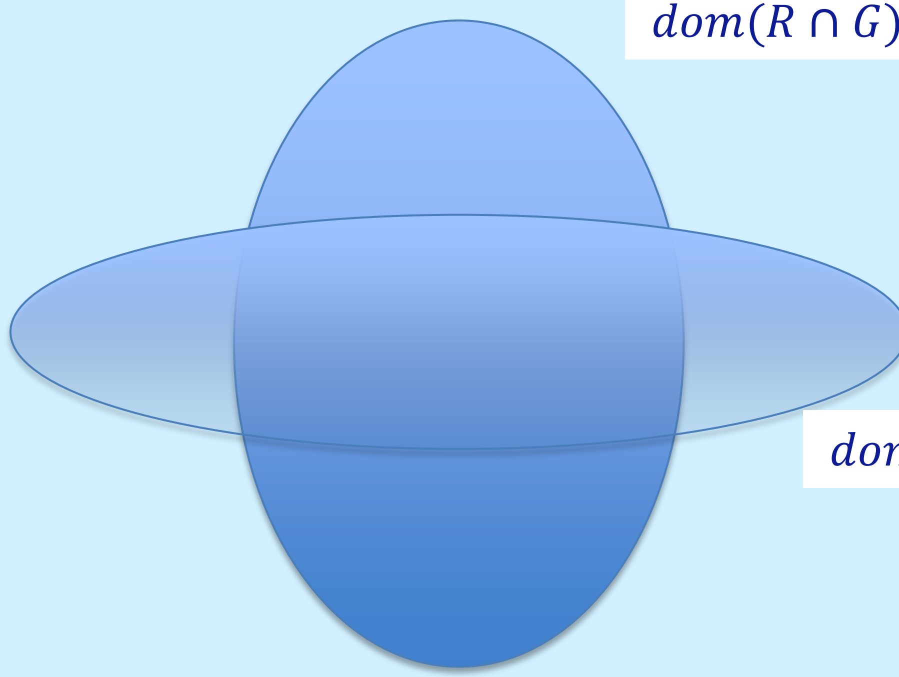
Relative Correctness and Reliability



Correctness and Relative Correctness

A program may be more reliable w/o being more-correct.

$dom(R \cap G)$



$dom(R \cap G')$

Correctness and Relative Correctness

Quantifying Relative Correctness

- $\forall \boldsymbol{G}'$: $(\boldsymbol{R} \cap \boldsymbol{G})\boldsymbol{L} \supseteq (\boldsymbol{R} \cap \boldsymbol{G}')\boldsymbol{L}$.
  –
- $\forall \boldsymbol{R}$: $(\boldsymbol{R} \cap \boldsymbol{G})\boldsymbol{L} \supseteq (\boldsymbol{R} \cap \boldsymbol{G}')\boldsymbol{L}$.
  –

Correctness and Relative Correctness

Quantifying Relative Correctness

- $\forall G'$: $(R \cap G)L \sqsupseteq (R \cap G')L$.
  - G is correct with respect to R.
- $\forall R$: $(R \cap G)L \sqsupseteq (R \cap G')L$.
  - G refines G'.

Outline

- What's Wrong with Faults
- Correctness and Relative Correctness
- Faults and Monotonic Fault Removal
- Definite Faults
- Beyond Nice Definitions: Applications
- Conclusion

Faults and Monotonic Fault Removal

Definition 3.1. Contingent Faults. *Let g be a program on space S, and let $\theta(G_1, G_2, G_3, ...G_n)$ be a relational representation of program g at a given level of granularity. We say that G_i is a* fault *of program g with respect to specification R if and only if there exists a relation G'_i on S such that $\theta(G_1, G_2, G_3, ..., G'_i, ...G_n)$ is strictly-more-correct with respect to R than $\theta(G_1, G_2, G_3, ..., G_i, ...G_n)$.*

***Contingent fault*: contingent upon the hypothesis that we are focusing the blame on Gi.**

We may have to broaden it to include any number of fault loci.

Faults and Monotonic Fault Removal

Definition 3.2. Monotonic Fault Removal. *Let g be a program on space S, whose expression is $\theta(G_1, G_2, G_3, ..., G_i, ...G_n)$ and let G_i be a contingent fault in g. We say that the substitution of G_i by G'_i is a* monotonic fault removal *if and only if program g' defined by $\theta(G_1, G_2, G_3, ..., G'_i, ...G_n)$ is strictly-more-correct than g.*

To be a fault: Unary property.

To be a monotonic fault removal: binary property (faulty statement and its replacement).

Faults and Monotonic Fault Removal

In the same way that program construction proceeds, ideally, by stepwise refinement,

$$R \overline{\lesssim} R_1 \overline{\lesssim} R_2 \overline{\lesssim} R_3 \overline{\lesssim} R_4 \overline{\lesssim} \ldots g.$$

Program testing ought to proceed, ideally, by stepwise monotonic fault removal.

$$g \overline{\sqsubseteq} g_1 \overline{\sqsubseteq} g_2 \overline{\sqsubseteq} g_3 \overline{\sqsubseteq} g_4 \overline{\sqsubseteq} \cdots \underline{g}.$$

Faults and Monotonic Fault Removal

Illustration:

g_{01}	The program obtained from g when we replace (`let+=2`) by (`let+=1`).
g_{10}	The program obtained from g when we replace (`'Z'>c`) by (`'Z'>=c`).
g_{11}	The program obtained from g when we replace (`let+=2`) by (`let+=1`) (`'Z'>c`) by (`'Z'>=c`).

- $(R_0 \cap G)L = \{(s,s')|q \in list\langle \alpha_a \cup \nu \cup \sigma\rangle\}$.
- $(R_0 \cap G_{01})L = \{(s,s')|q \in list\langle(\alpha_A \setminus \{'Z'\}) \cup \alpha_a \cup \nu \cup \sigma\rangle\}$.

- $(R_0 \cap G_{10})L = \{(s,s')|q \in list\langle \alpha_a \cup \nu \cup \sigma\rangle\}$.
- $(R_0 \cap G_{11})L = \{(s,s')|q \in list\langle \alpha_A \cup \alpha_a \cup \nu \cup \sigma\rangle\}$.

Faults and Monotonic Fault Removal

Illustration:

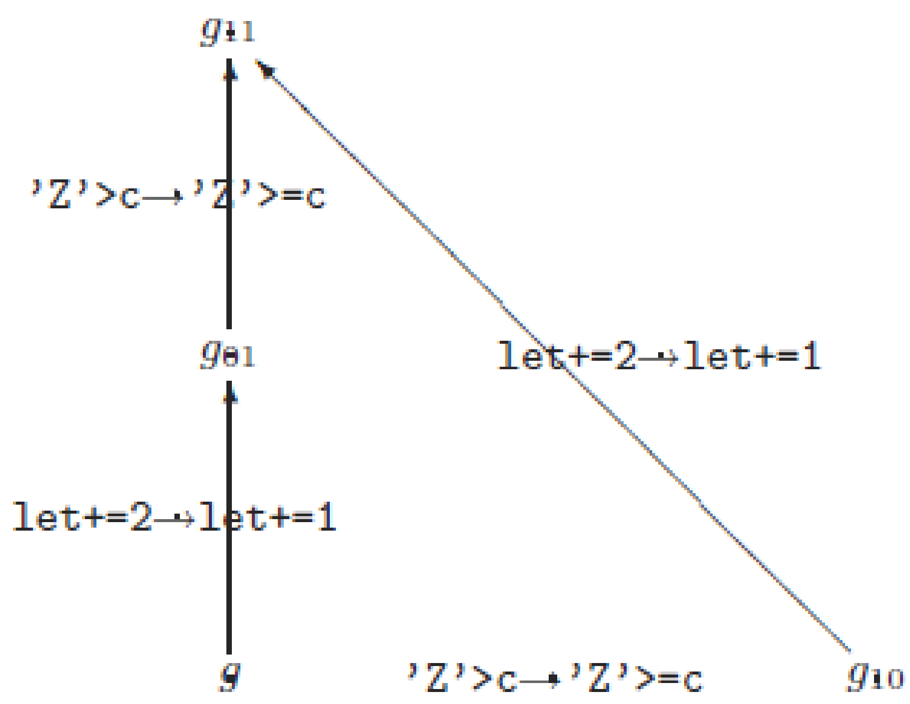


Fig. 2. Monotonic and Non Monotonic Fault Removals

Faults and Monotonic Fault Removal

Does every fault removal have to be monotonic (produce a more-correct program?)

- Yes.

What about the transformation of g into g10?

- We broaden the definition of fault to include more than one location (other reasons to do so, anyway) and we view the transition (g,g10,g11) as a single fault removal.

Outline

- What's Wrong with Faults
- Correctness and Relative Correctness
- Faults and Monotonic Fault Removal
- Definite Faults
- Beyond Nice Definitions: Applications
- Conclusion

Definite Faults

Not all faults are contingent.

- Some faults are so damaging that no amount of mitigation can salvage them.
- Examples:
  - Loss of injectivity in preprocessing.
  - Loss of surjectivity in postprocessing.

Definite Faults

Loss of Injectivity.

Lemma 4.2. ***Right Divisibility.*** *The relational equation in X: $QX \sqsupseteq R$, admits a solution in X if and only if R and Q satisfy the following condition:*

$$RL \subseteq QL \wedge \overline{\widehat{Q}(\overline{R} \cap RL)}L = L \ .$$

Proposition 4.3. ***Definite Fault, for loss of injectivity.*** *We consider a relation R on space S and a program g on S of the form $g = \{g_1;\ g_2\}$. If R and G_1 do not satisfy the right divisibility condition (with G_1 as Q), then g_1 is definitely faulty with respect to R.*

Definite Faults

Loss of Injectivity.

Specification:

- Sorting an array:
  - Preprocessing: destroy one cell.
  - Nothing that post-processing can do recover from the loss.

Definite Faults

Loss of Surjectivity

Lemma 4.4. ***Left Divisibility.*** *The relational equation in* X *:* $XQ \sqsupseteq R$, $\widehat{X}L \subseteq QL$, *admits a solution in* X *if and only if* R *and* Q *satisfy the following condition:*

$$RL \subseteq (\overline{\overline{R}\widehat{Q}} \cap L\widehat{Q})L \ .$$

Proposition 4.5. ***Definite Fault, for loss of surjectivity.*** *We consider a relation* R *on space* S *and a program* g *on* S *of the form* $g = \{g_1;\ g_2;\}$. *If* R *and* G_2 *do not satisfy the right divisibility condition (with* G_2 *as* Q*), then* g_2 *is definitely faulty with respect to* R.

Definite Faults

Loss of Surjectivity

- Specification:

$$R = \{(s, s')|s' = s^2 \ mod \ 6\} \ .$$

- Post processing:

$$g_2 = \{s = s \bmod 3\}$$

- No preprocessor can make up for this fault.

Outline

- What's Wrong with Faults
- Correctness and Relative Correctness
- Faults and Monotonic Fault Removal
- Definite Faults
- Beyond Nice Definitions: Applications
- Conclusion

Beyond Nice Definitions: Applications

We have lived happily for several decades without a definition of faults.

- We can live happily everafter…
- Why do we need a definition?

Applications:

- Streamline fault repair

Beyond Nice Definitions: Applications

Mutation Testing for Fault Repair

- Faults are within the range of mutation operators.
- Fault bound to one location.
- Realistic faults can be removed efficiently.
- The structure of the program is not in question.
- If a program passes the test, it is correct (fault removal confirmed).
- If a program fails the test, it is incorrect (fault removal should be rolled back).

Beyond Nice Definitions: Applications

All hypotheses highly questionable:

- Faults are within the range of mutation operators.
  - Good luck.
- Fault bound to one location. The structure of the program is not in question.
  - Limited scope.
- Realistic faults can be removed efficiently.
  - Painful dilemmas: realistic faults vs efficient fault removal.
- If a program passes the test, it is correct (fault removal confirmed).
  - May work on T but fail outside.
- If a program fails the test, it is incorrect (fault removal should be rolled back).
  - Does not have to be correct; only more-correct than original; not the last fault.

Beyond Nice Definitions: Applications

Specification R, faulty program g, candidate mutant g'.

- Is g' a legitimate improvement over g?
  - Compare $dom(R \cap G)$ and $dom(R \cap G')$.
- If modification buried inside a loop, it is difficult to compute G and G'.

Beyond Nice Definitions: Applications

Possible approach:

- Using invariant relations.
- Invariant relation of while t {b}:
  - Reflexive transitive superset of $(T \cap B)$
- Can be used to prove
  - Correctness,
  - Incorrectness

  of while loop with respect to specification V.

Beyond Nice Definitions: Applications

```
//  input:  specification V
//   output:  correctness diagnosis; incompatible InvRel.
cumulR=L; diagnosis=undecided;
While (diagnosis=undecided &&  moreInvRel)
   {R = nextInvRel();
     CumulR = CumulR ∩ R.
     if   subsume(CumulR, V)  {diagnosis = correct;}
     else
            if incompatible(R, V)  {diagnosis = incorrect;  return R;}
    }
// if  (diagnosis=undecided) we ran out of invariant relations.
```

Beyond Nice Definitions: Applications

Three outcomes

- Diagnosis = correct:
  - No fault to remove.
- Diagnosis = incorrect:
  - Invariant Relation culprit. Used to calculate monotonic correction (statements, variables,).
- Diagnosis = undecided:
  - Grow the database of Recognizers.

Outline

- What's Wrong with Faults
- Correctness and Relative Correctness
- Faults and Monotonic Fault Removal
- Definite Faults
- Beyond Nice Definitions: Applications
- Conclusion

Conclusion

Defined relative correctness, tripartite relation between a specification and two programs:

- Quantified over specifications: refinement.
  - Relative correctness: point-wise refinement.
- Quantified over programs: correctness.

Used relative correctness to define

- Contingent fault.
- Monotonic fault removal.
- Definite fault.

Explored some possible applications behind

- Nice looking definitions.

Infancy; envision to continue exploration.