Automated Error Localization and Correction for Imperative Programs

Robert Könighofer and Roderick Bloem

IAIK – Graz University of Technology

robert.koenighofer@iaik.tugraz.at

www.iaik.tugraz.at

This work was supported in part by the European Commission through project DIAMOND (FP7-2009-IST-4-248613).

Outline

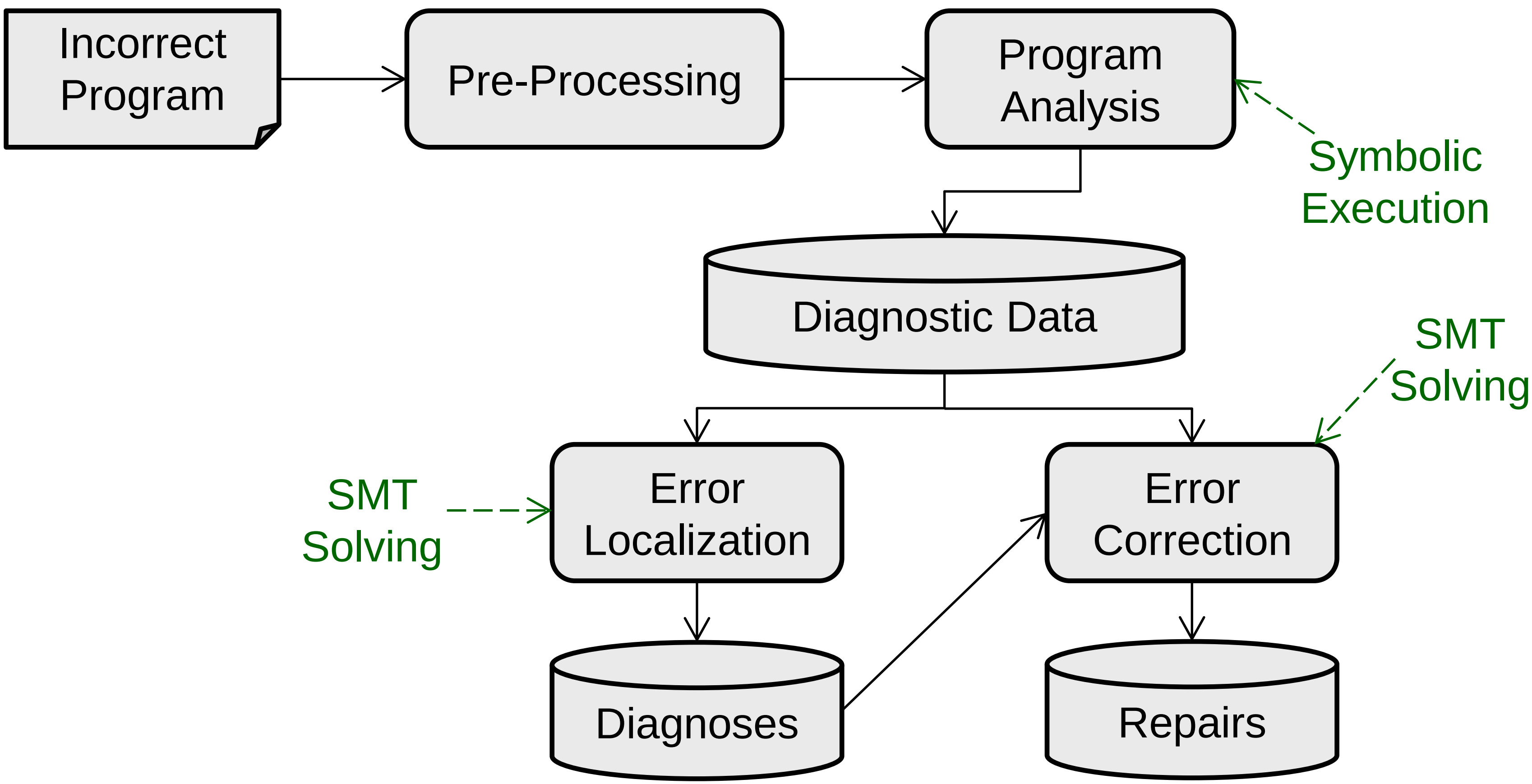
- Addressed problem
- Debugging approach
  - Program analysis
  - Error localization
  - Error correction
- Experimental results
- Summary and conclusions

Addressed Problem

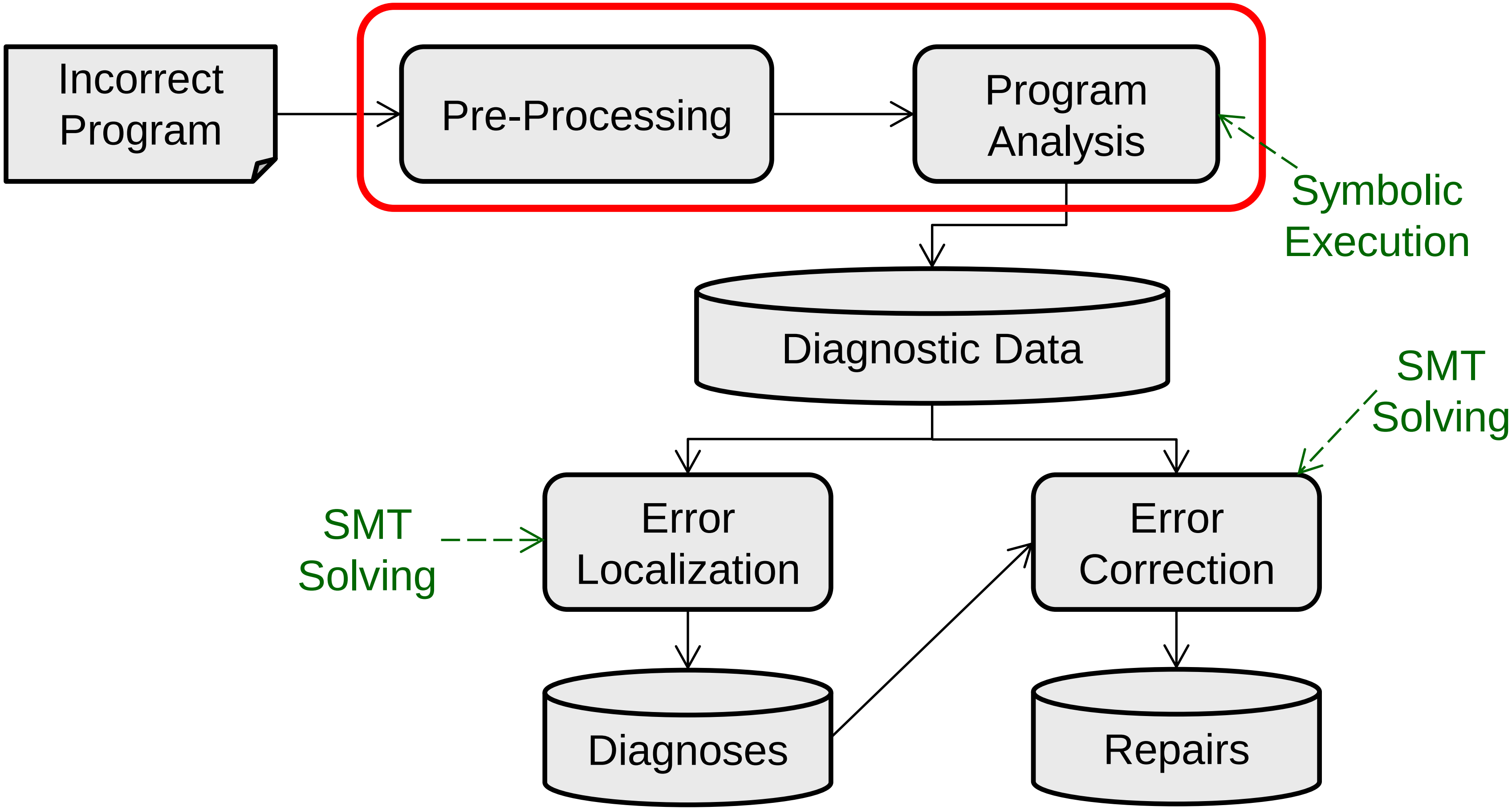
- Automate debugging of simple software
- Input:
  - Incorrect program
  - Specification:
    - Assertions
    - Reference Implementation
- Output:
  - Potential error locations
  - Potential repairs

```
int max(int x, int y) {
  int res = x;
  if(y > x)
    res = x;
  assert(res>=x && res>=y);
  return res;
}
```

Debugging Approach



Debugging Approach



Debugging Approach:

Pre-Processing

- Identification of Components
  - Right-hand side (RHS) of assignments
  - Efficient program analysis

```
int max(int x, int y) {
  int res = x;
  if(y > x)
    res = x;
  assert(res>=x && res>=y);
  return res;
}
```

Debugging Approach:

Pre-Processing

- Identification of Components
  - Right-hand side (RHS) of assignments
  - Efficient program analysis

```
int max(int x, int y) {
  int res = x;    c0
  if(y > x)
    res = x;    c1
  assert(res>=x && res>=y);
  return res;
}
```

Debugging Approach:

Pre-Processing

- Identification of Components
  - Right-hand side (RHS) of assignments
  - Efficient program analysis
  - Can handle all incorrect expressions

```
int max(int x, int y) {
  int res = x;      c0
  if(y > x)
    res = x;        c1
  assert(res>=x && res>=y);
  return res;
}
```

```
int max(int x, int y) {
  int res = x;      c0
  int tmp = y > x;  c2
  if(tmp)
    res = x;        c1
  assert(res>=x && res>=y);
  return res;
}
```

Debugging Approach:

Pre-Processing

- Components may be faulty
- Transformation:

```
int max(int x, int y) {
  int res = x;        c0
  if(y > x)
    res = x;          c1
  assert(res>=x && res>=y);
  return res;
}
```

⇒

```
bool AC_c0; // AC = 'Assume Correct'
bool AC_c1;
int max(int x, int y) {
  int res = AC_c0 ? x : repair_c0(x,y);
  if(y > x)
    res = AC_c1 ? x : repair_c1(x,y,res);
  assert(res>=x && res>=y);
  return res;
}
```

Debugging Approach:

Program Analysis

- When is my buggy program correct?
- We use Symbolic Execution
  - Allows incomplete analysis

```
int max(int x, int y) {
  int res = x;
  if(y > x)
    res = x;
  assert(res>=x && res>=y);
  return res;
}
```

Symbolic Execution $\Rightarrow$

$$Correct(X, Y) = Y \leq X$$

Debugging Approach:

Program Analysis

- When is my program correct?

```
bool AC_c0; // AC = 'Assume Correct'
bool AC_c1;
int max(int x, int y) {
  int res = AC_c0 ? x : repair_c0(x,y);
  if(y > x)
    res = AC_c1 ? x : repair_c1(x,y,res);
  assert(res>=x && res>=y);
  return res;
}
```

$$
\begin{aligned}
&Correct(X, Y, R_0, R_1, \mathrm{AC\_c}_0, \mathrm{AC\_c}_1) = \\
&\exists A_0, A_1: \\
&[Y > X \wedge A_1 \geq X \wedge A_1 \geq Y \vee \\
&Y \leq X \wedge A_0 \geq X \wedge A_0 \geq Y] \wedge \\
&A_0 = \mathrm{AC\_c}_0\ ?\ X : R_0 \wedge \\
&\mathrm{A}_1 = \mathrm{AC\_c}_1\ ?\ X : R_1
\end{aligned}
$$

$$
\begin{aligned}
R_0 &= repair\_c_0(X, Y) \\
R_1 &= repair\_c_1(X, Y, X)
\end{aligned}
$$

Debugging Approach:

Program Analysis

- When is my buggy program correct?

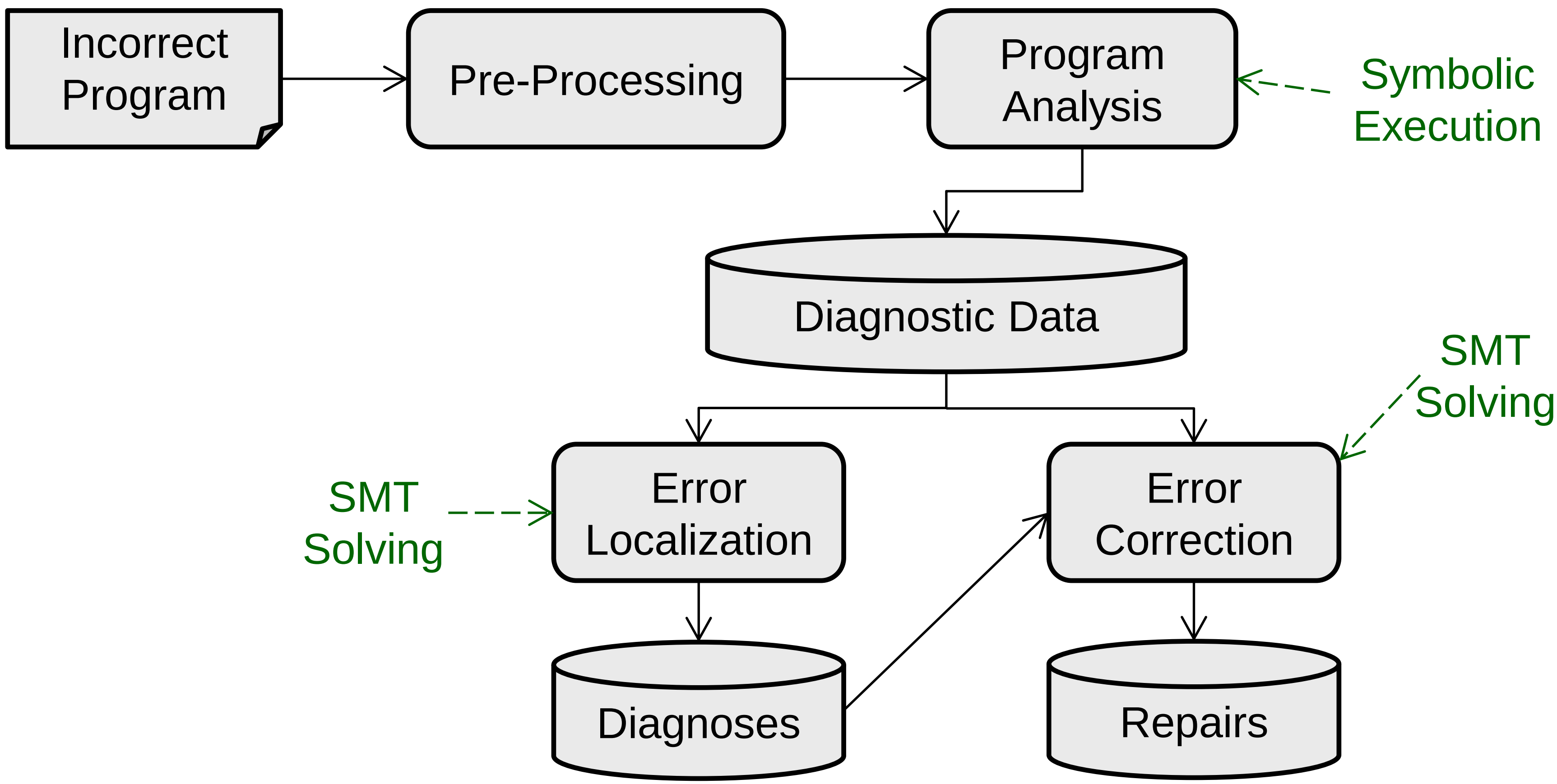
```
bool AC_c0; // AC = 'Assume Correct'
bool AC_c1;
int max(int x, int y) {
  int res = AC_c0 ? x : repair_c0(x,y);
  if(y > x)
    res = AC_c1 ? x : repair_c1(x,y,res);
  assert(res>=x && res>=y);
  return res;
}
```

$$Correct(X,Y,R_0,R_1,\mathrm{AC\_c}_0,\mathrm{AC\_c}_1) =$$
$$\exists A_0, A_1:$$
$$[Y > X \wedge A_1 \geq X \wedge A_1 \geq Y \vee$$
$$Y \leq X \wedge A_0 \geq X \wedge A_0 \geq Y] \wedge$$
$$A_0 = \mathrm{AC\_c}_0\ ?\ X : R_0 \wedge$$
$$\mathrm{A}_1 = \mathrm{AC\_c}_1\ ?\ X : R_1$$

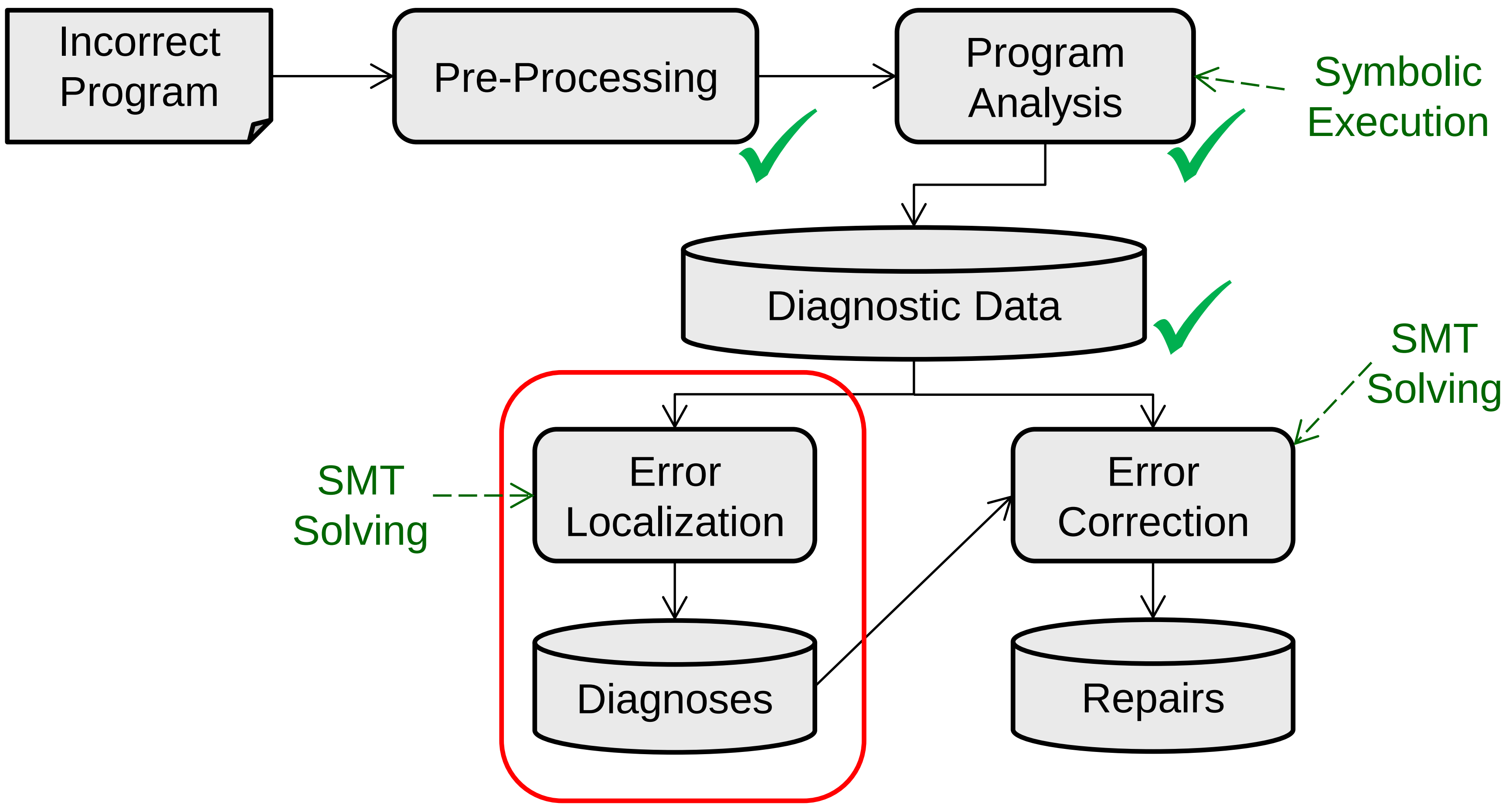
$$R_0 = repair\_c_0(X,Y)$$
$$R_1 = repair\_c_1(X,Y,X)$$

Diagnostic Data

Debugging Approach



Debugging Approach



Debugging Approach:

Error Localization

- **Goal**: compute diagnoses

  Sets of components that

  - … if assumed to be incorrect, make the program repairable
  - … can be modified to make the program correct
  - … may be responsible for the incorrectness

Debugging Approach:

Error Localization

- **Goal**: compute diagnoses

  Sets of components that
  - … if assumed to be incorrect, make the program repairable
  - … can be modified to make the program correct
  - … may be responsible for the incorrectness

```
int max(int x, int y) {
  int res = x;        ← c0
  if(y > x)
    res = x;          ← c1
  assert(res>=x && res>=y);
  return res;
}
```

Assumption c_0	Assumption c_1	Repairable	Diagnosis
correct	correct	no	
correct	incorrect	yes	$\{c_1\}$
incorrect	correct	no	
incorrect	incorrect	yes	$\{c_0, c_1\}$

Debugging Approach:

Error Localization

- **Goal**: compute diagnoses

  Sets of components that

  - … if assumed to be incorrect, make the program repairable
  - … can be modified to make the program correct
  - … may be responsible for the incorrectness

```
int max(int x, int y) {
  int res = x;          <- c0
  if(y > x)
    res = x;            <- c1
  assert(res>=x && res>=y);
  return res;
}
```

Assumption c_0	Assumption c_1	Repairable	Diagnosis
correct	correct	no	
correct	incorrect	yes	$\{c_1\}$
incorrect	correct	no	
incorrect	incorrect	yes	$\{c_0, c_1\}$

Debugging Approach:

Error Localization

- **Goal**: compute diagnoses

  Sets of components that
  - … if assumed to be incorrect, make the program repairable
  - … can be modified to make the program correct
  - … may be responsible for the incorrectness

```
int max(int x, int y) {
  int res = x;          <- c0
  if(y > x)
    res = repair(…);    <- c1
  assert(res>=x && res>=y);
  return res;
}
```

Assumption c_0	Assumption c_1	Repairable	Diagnosis
correct	correct	no	
correct	incorrect	yes	$\{c_1\}$
incorrect	correct	no	
incorrect	incorrect	yes	$\{c_0, c_1\}$

Debugging Approach:

Error Localization

- **Goal**: compute diagnoses

  Sets of components that
  - … if assumed to be incorrect, make the program repairable
  - … can be modified to make the program correct
  - … may be responsible for the incorrectness

```
int max(int x, int y) {
  int res = repair(…);   <- c0
  if(y > x)
    res = x;             <- c1
  assert(res>=x && res>=y);
  return res;
}
```

Assumption c_0	Assumption c_1	Repairable	Diagnosis
correct	correct	no	
correct	incorrect	yes	$\{c_1\}$
incorrect	correct	no	
incorrect	incorrect	yes	$\{c_0, c_1\}$

Debugging Approach:

Error Localization

- **Goal**: compute diagnoses

  Sets of components that

  - … if assumed to be incorrect, make the program repairable
  - … can be modified to make the program correct
  - … may be responsible for the incorrectness

```
int max(int x, int y) {
  int res = repair(…);      <- c0
  if(y > x)
    res = repair(…);        <- c1
  assert(res>=x && res>=y);
  return res;
}
```

Assumption c_0	Assumption c_1	Repairable	Diagnosis
correct	correct	no	
correct	incorrect	yes	$\{c_1\}$
incorrect	correct	no	
incorrect	incorrect	yes	$\{c_0, c_1\}$

Debugging Approach:

Error Localization

- Checking for Repairability:

```
bool AC_c0; // AC = 'Assume Correct'
bool AC_c1;
int max(int x, int y) {
  int res = AC_c0 ? x : repair_c0(x,y);
  if(y > x)
    res = AC_c1 ? x : repair_c1(x,y,res);
  assert(res>=x && res>=y);
  return res;
}
```

Program → Analysis

Diagnostic Data:

$$Correct(X, Y, R_0, R_1, \mathrm{AC\_c}_0, \mathrm{AC\_c}_1) =$$
$$\exists A_0, A_1:$$
$$[Y > X \land A_1 \geq X \land A_1 \geq Y \lor$$
$$Y \leq X \land A_0 \geq X \land A_0 \geq Y] \land$$
$$A_0 = \mathrm{AC\_c}_0\ ?\ X : R_0 \land$$
$$\mathrm{A}_1 = \mathrm{AC\_c}_1\ ?\ X : R_1$$

Debugging Approach:

Error Localization

- Checking for Repairability:

```
bool AC_c0; // AC = 'Assume Correct'
bool AC_c1;
int max(int x, int y) {
  int res = AC_c0 ? x : repair_c0(x,y);
  if(y > x)
    res = AC_c1 ? x : repair_c1(x,y,res);
  assert(res>=x && res>=y);
  return res;
}
```

Program Analysis

Diagnostic Data:

$$Correct(X, Y, R_0, R_1, \mathrm{AC\_c}_0, \mathrm{AC\_c}_1) =$$
$$\exists A_0, A_1:$$
$$[Y > X \wedge A_1 \geq X \wedge A_1 \geq Y \vee$$
$$Y \leq X \wedge A_0 \geq X \wedge A_0 \geq Y] \wedge$$
$$A_0 = \mathrm{AC\_c}_0\ ?\ X : R_0 \wedge$$
$$\mathrm{A}_1 = \mathrm{AC\_c}_1\ ?\ X : R_1$$

$$Repairable(AC\_c_0, AC\_c_1) =$$
$$\forall X, Y: \exists R_0, R_1:$$
$$Correct(X, Y, R_0, R_1, \mathrm{AC\_c}_0, \mathrm{AC\_c}_1)$$

Debugging Approach:

Error Localization

$$Repairable(AC\_c_0, AC\_c_1) = \forall X, Y: \exists R_0, R_1: \ Correct(X, Y, R_0, R_1, \mathrm{AC\_c}_0, \mathrm{AC\_c}_1)$$

- Use SMT-Solver to check repairability
  - Problem: Quantifier alternation
  - Check for a finite set of input values only
  - False positives

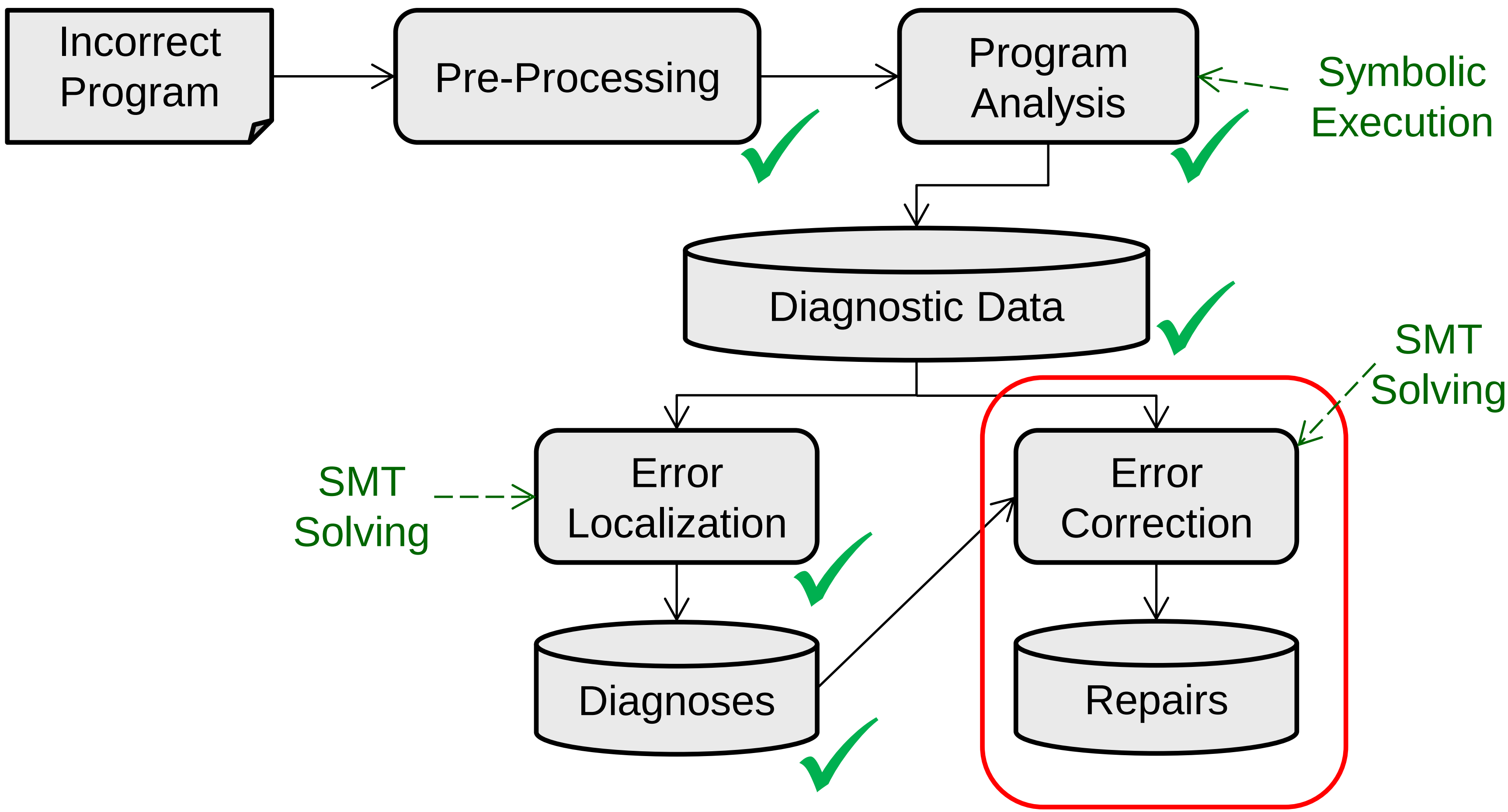
Debugging Approach:

Error Localization

$$Repairable(AC\_c_0, AC\_c_1) = \forall X, Y : \exists R_0, R_1 :\ Correct(X, Y, R_0, R_1, \mathrm{AC\_c}_0, \mathrm{AC\_c}_1)$$

- Use SMT-Solver to check repairability
  - Problem: Quantifier alternation
  - Check for a finite set of input values only
  - False positives
- A set D is a diagnosis iff
  Repairable(AC_c_0, AC_c_1, … ,AC_c_n) = true
  with AC_c_k = false iff $c_k \in D$
- Computing diagnoses naively: check every set
- Our algorithm: Unsatisfiable cores + hitting sets

Debugging Approach



Debugging Approach:

Error Correction

- We know which components are faulty
  - E.g.: $D_1=\{c_8\}$, $D_2=\{c_2, c_4\}$
- Error Correction
  - Compute replacements for the faulty components
  - E.g.: $c_8 \rightarrow x+9$

    $c_2 \rightarrow a <= b$ and $c_4 \rightarrow 1$

Debugging Approach:

Error Correction

- Use information about faulty components:

$$Correct(X, Y, R_0, R_1, \mathrm{AC\_c_0}, \mathrm{AC\_c_1}) = \exists A_0, A_1 : [Y > X \wedge A_1 \geq X \wedge A_1 \geq Y \vee Y \leq X \wedge A_0 \geq X \wedge A_0 \geq Y] \wedge A_0 = \mathrm{AC\_c_0}\ ?\ X : R_0 \wedge \mathrm{A}_1 = \mathrm{AC\_c_1}\ ?\ X : R_1$$

$$R_0 = repair\_c_0(X, Y)$$
$$R_1 = repair\_c_1(X, Y, X)$$

Diagnostic Data

$\mathrm{c_1}$=incorrect

AC_$\mathrm{c_0}$=true
AC_$\mathrm{c_1}$=false

$$Correct'(X, Y) = \exists A_0, A_1 : [Y > X \wedge A_1 \geq X \wedge A_1 \geq Y \vee Y \leq X \wedge A_0 \geq X \wedge A_0 \geq Y] \wedge A_0 = X \wedge \mathrm{A}_1 = repair\_c_1(X, Y, X)$$

Simplified

$$Correct'(X, Y) = Y \leq X \vee repair\_c_1(X, Y, X) \geq X \wedge repair\_c_1(X, Y, X) \geq Y$$

Debugging Approach:

Error Correction

$$Correct'(X,Y) = Y \leq X \vee repair\_c_1(X,Y,X) \geq X \wedge repair\_c_1(X,Y,X) \geq Y$$

- **Goal**: make Correct'(…) equal to true

Debugging Approach:

Error Correction

$$Correct'(X,Y) = Y \leq X \vee repair\_c_1(X,Y,X) \geq X \wedge repair\_c_1(X,Y,X) \geq Y$$

- **Goal**: make Correct'(…) equal to true
- Simpler problem:
  - Checking a given expression is easy
  - Example: repair_c_1(x,y,res) = x + 4

$$Correct'(X,Y) = Y \leq X \vee repair\_c_1(X,Y,X) \geq X \wedge repair\_c_1(X,Y,X) \geq Y \;\Rightarrow\; Correct''(X,Y) = Y \leq X \vee X+4 \geq X \wedge X+4 \geq Y$$

  - Correct''(0,5) = false → 'x + 4' is not a valid repair
  - Solver can also return a counterexample

Debugging Approach:

Error Correction

- Finding a substitution for repair() is more difficult
- **Idea**: Replace repair(…) with a template for expressions
- E.g.: $repair(x,y,res) = k_0 + k_1{*}x + k_2{*}y + k_3{*}res$

Debugging Approach:

Error Correction

- Finding a substitution for repair() is more difficult
- **Idea**: Replace repair(…) with a template for expressions
- E.g.: repair(x,y,res) = $k_0 + k_1{*}x + k_2{*}y + k_3{*}res$

$$Correct'(X,Y) = Y \le X \lor repair\_c_1(X,Y,X) \ge X \land repair\_c_1(X,Y,X) \ge Y$$

$$\Rightarrow$$

$$Correct''(X,Y,k_0,k_1,k_2,k_3) = Y \le X \lor k_0 + k_1 \cdot X + k_2 \cdot Y + k_3 \cdot X \ge X \land k_0 + k_1 \cdot X + k_2 \cdot Y + k_3 \cdot X \ge Y$$

- Find k_0, k_1, k_2, k_3 such that Correct''(X, Y, k_0, k_1, k_2, k_3) holds for all X,Y
- Now: find parameter values instead of expressions

Debugging Approach:

Error Correction

- Solvers cannot handle conditions like

$$\boxed{\exists k_0, k_1, k_2, k_3\colon \forall X, Y\colon\ Correct''(X, Y, k_0, k_1, k_2, k_3)}$$

  efficiently

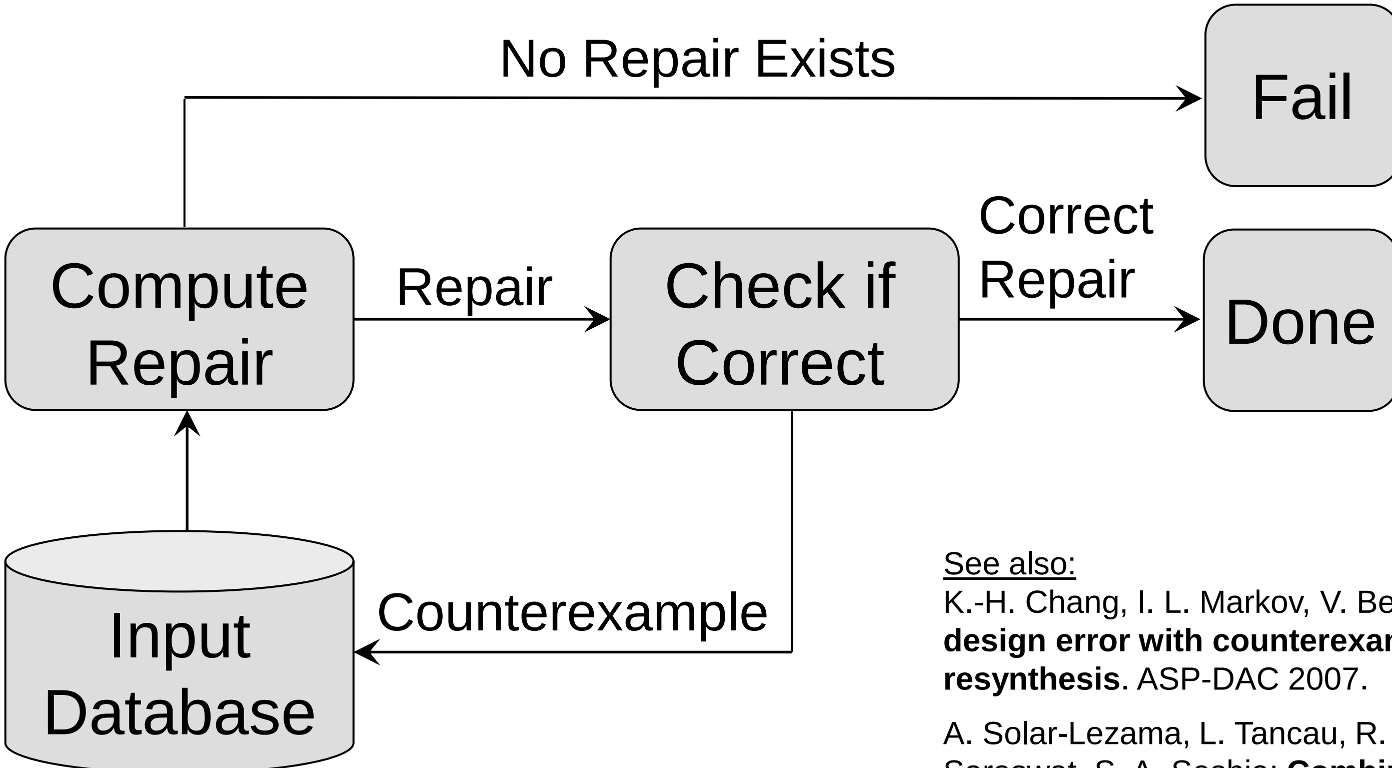
- Can find k_0, k_1, k_2, k_3 such that Correct(X, Y, k_0, k_1, k_2, k_3) is true for **some** inputs X, Y.

- Can compute repairs that are correct for **some** inputs

Debugging Approach:

Error Correction

- Iterative repair refinement:

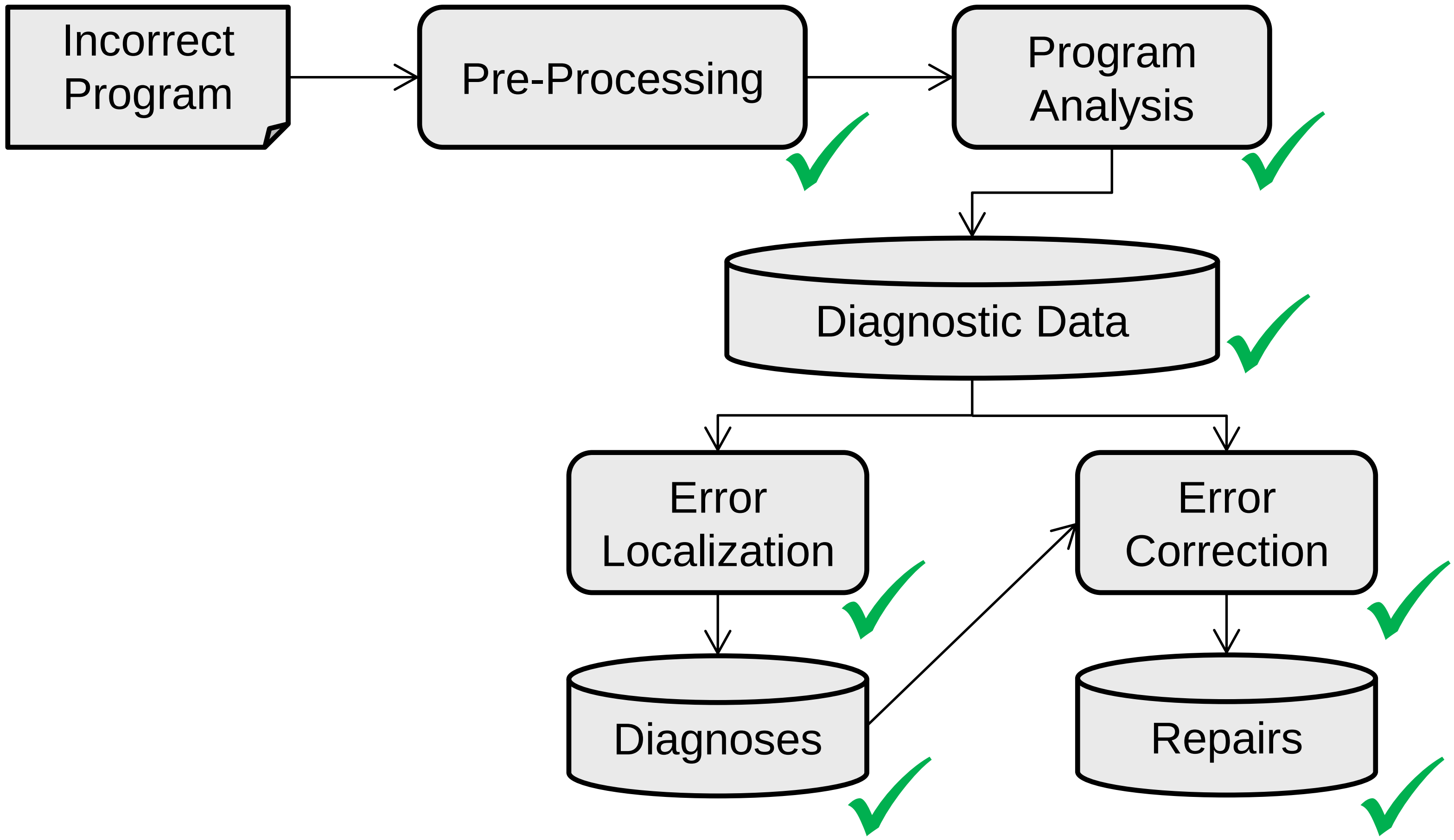


See also:

K.-H. Chang, I. L. Markov, V. Bertacco: **Fixing design error with counterexamples and resynthesis**. ASP-DAC 2007.

A. Solar-Lezama, L. Tancau, R. Bodik, V. Saraswat, S. A. Seshia: **Combinatorial sketching for finite programs.** ASPLOS 2006.

Debugging Approach



Implementation

- For simple C programs
- Tool: FoREnSiC
  - Various debugging methods
- Program analysis:
  - Concolic execution
  - Extension of CREST
- SMT-Solver:
  - Yices and Z3
  - Linear Integer Arithmetic
  - Bitvector Arithmetic under Development

Experimental Results:

Comparison with Sketch [1]

- TCAS example [2] (180 LOC, 12 inputs, 44 components):

	Diagnosis		Repair				Sketch (8 bit)		
	Found	Time [sec]	Found	Time [sec]	Template Variables	Memory [MB]	Time [sec]	Template Variables	Memory [MB]
tcas2	2	70	2	271	11	75	29	1	297
tcas7	2	123	5	42	10	70	20	4	1459
tcas8	2	122	5	37	10	79	6.3	1	42
tcas16	2	123	5	43	10	72	22	2	764
tcas17	2	125	5	41	10	70	11	2	666
tcas18	2	124	5	38	10	75	9.3	2	744
tcas19	2	123	5	40	10	70	9.6	1	42
tcas36	0	3.0	-	-	-	44	7.0	4	796

[1] A. Solar-Lezama, L. Tancau, R. Bodik, V. Saraswat, S. A: Seshia. **Combinatorial sketching for finite programs**. ASPLOS 2006.
[2] http://pleuma.cc.gatech.edu/aristotle/Tools/subjects/

Experimental Results:

Comparison with Sketch [1]

- TCAS example [2] (180 LOC, 12 inputs, 44 components):

	Diagnosis		Repair				Sketch (8 bit)		
	Found	Time [sec]	Found	Time [sec]	Template Variables	Memory [MB]	Time [sec]	Template Variables	Memory [MB]
tcas2	2	70	2	271	11	75	29	1	297
tcas7	2	123	5	42	10	70	20	4	1459
tcas8	2	122	5	37	10	79	6.3	1	42
tcas16	2	123	5	43	10	72	22	2	764
tcas17	2	125	5	41	10	70	11	2	666
tcas18	2	124	5	38	10	75	9.3	2	744
tcas19	2	123	5	40	10	70	9.6	1	42
tcas36	0	3.0	-	-	-	44	7.0	4	796

- Do not take too seriously; different solvers, objectives, …

Experimental Results:

Comparison with Sketch [1]

- TCAS example [2] (180 LOC, 12 inputs, 44 components):

	Diagnosis		Repair				Sketch (8 bit)		
	Found	Time [sec]	Found	Time [sec]	Template Variables	Memory [MB]	Time [sec]	Template Variables	Memory [MB]
tcas2	2	70	2	271	11	75	29	1	297
tcas7	2	123	5	42	10	70	20	4	1459
tcas8	2	122	5	37	10	79	6.3	1	42
tcas16	2	123	5	43	10	72	22	2	764
tcas17	2	125	5	41	10	70	11	2	666
tcas18	2	124	5	38	10	75	9.3	2	744
tcas19	2	123	5	40	10	70	9.6	1	42
tcas36	0	3.0	-	-	-	44	7.0	4	796

- Accurate diagnoses and correct repairs in reasonable time!

Experimental Results:

Example: tcas35

- Correct:

```
ClimbIn ? UpSep+100 : UpSep
```

Buggy:

```
ClimbIn ? UpSep : UpSep+100
```

c_4 c_5 c_6

- Diagnosis: $\{c_5,c_6\}$

Experimental Results:

Example: tcas35

- Correct:

```
ClimbIn ? UpSep+100 : UpSep
```

Buggy:

$\underbrace{\texttt{ClimbIn}}_{c_4}$? $\underbrace{\texttt{UpSep}}_{c_5}$: $\underbrace{\texttt{UpSep+100}}_{c_6}$

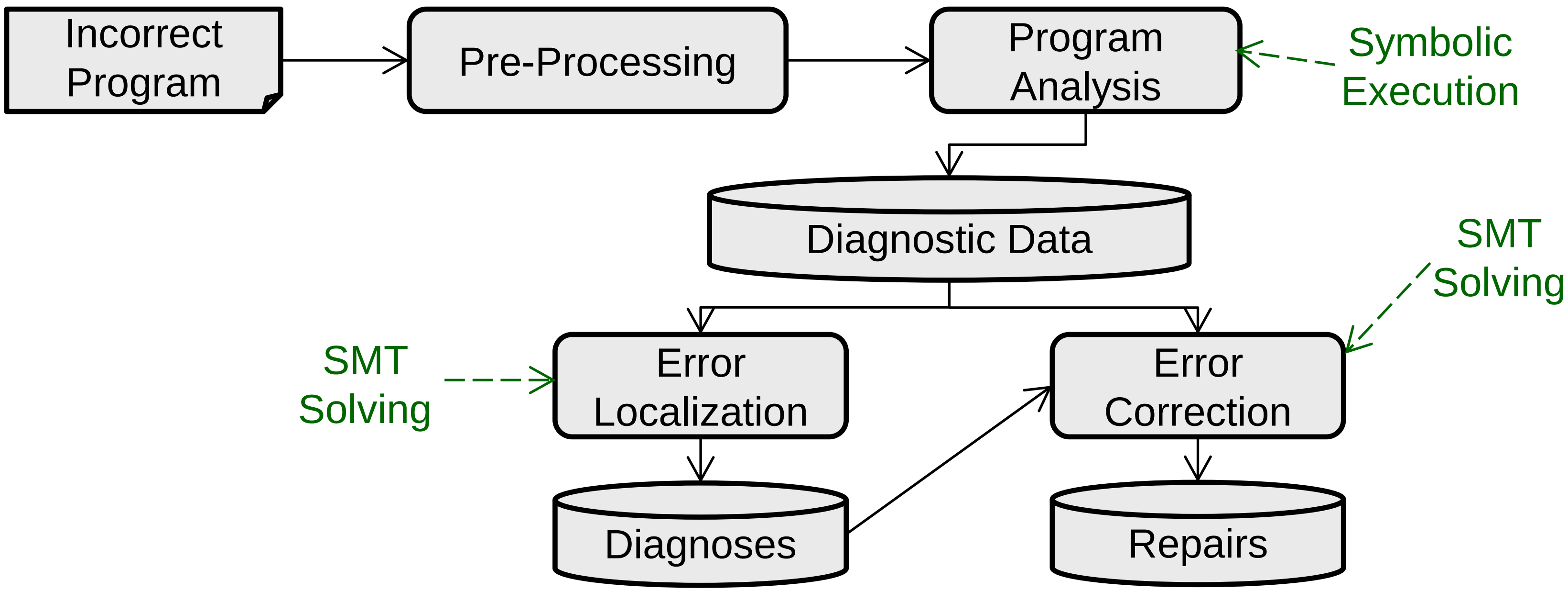
- Diagnosis: $\{c_5, c_6\}$
- Repair:

It.	c_5	c_6	
1	0	0	✗
2	728	0	✗
3	-ClimbIn+1000	0	✗
4	-ClimbIn+58	0	✗
5	UpSep+352	0	✗
6	UpSep-1000	0	✗
7	UpSep+100	0	✗
8	UpSep	-OtherTrAlt+11	✗
9	UpSep+1	UpSep-1000	✗

It.	c_5	c_6	
10	UpSep+2	CurVerSep-795	✗
11	UpSep+3	3114	✗
12	UpSep+100	UpSep-1000	✗
13	UpSep+100	-OtherTrAlt+11	✗
14	UpSep+4	CurVerSep	✗
15	UpSep+100	3114	✗
16	UpSep+5	UpSep+1000	✗
17	UpSep+100	OwnTrAlt	✗
18	UpSep+100	UpSep	✓

Summary and Conclusion

- **Problem**: debug incorrect programs
- **Debugging method**:



- Works nicely, but a lot of room for improvement!