Debugging Inputs

[ICSE'20] Lukas Kirschner, et al.

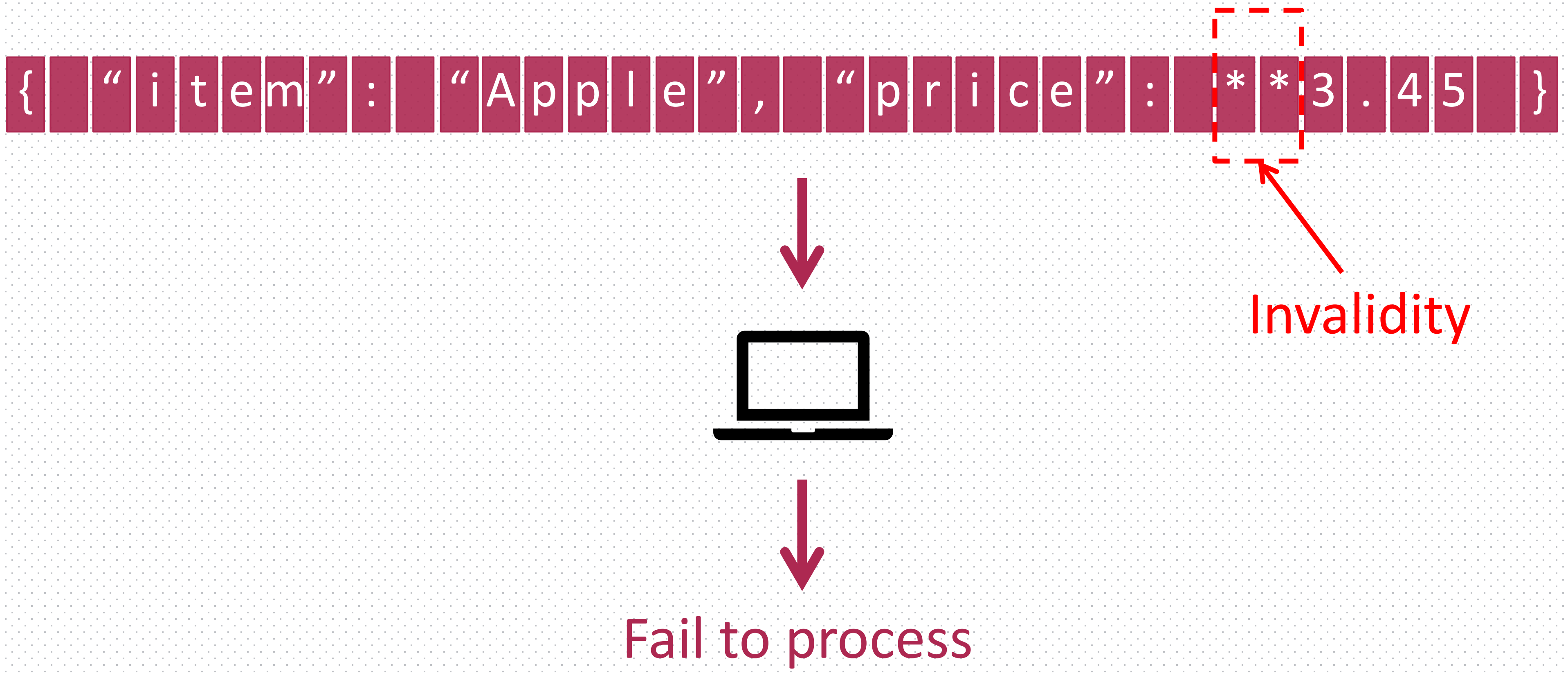
Group Paper Reading Lingyu Zhang 2022/10/18

Outline

- Motivation
- Maximizing Delta Debugging
  - Lexical repair
  - Syntactic repair
- Evaluation

Invalid Inputs also Cause Faulires of Programs

- Inputs of programs could get corrupted (e.g., hardware failures, hardware aging, transmission errors)



Prevalence of Invalid Inputs

- Eight programs with three input formats

Table 3: Mined Input Files

Input Format	#Crawled Files	#Unique Files	#Valid Files	#Invalid Files	*Cause of Invalidity (#files rejected by)* Grammar	≥ 1 subject	All subjects	Mean Valid Size (KiB)	Mean Invalid Size (KiB)
JSON	8654	7006	6948	222	164	58	52	12.84	0.78
Wave. OBJ	509	480	455	25	0	25	0	401.57	64.15
DOT	381	349	303	48	2	46	4	4.74	2.88

4%

Crash

Lexing/parsing error in ANTLR[1]

[1] https://en.wikipedia.org/wiki/ANTLR

Shortcomings of Existing Practice

- Application-specific methods or input minimization

`{ "item": "Apple", "price": **3.45 }`

Failing input

$$\Big\downarrow\; dd_{min}$$

`{`

Minimum input triggering failure

Neither helpful for diagnosis nor a basis for data recovery

Generic Input Debugging

- Identifying which parts of the input data prevent processing

- Recovering as much of the input data as possible

```
{ "item": "Apple", "price": 3.45 }
```

Recovered input

Maximizing Delta Debugging (Lexical)

- Two sets $C_{\surd}$ and $C_{\times}$

$C_{\times} =$ | { | | “ | i | t | e | m | ” | : | | “ | A | p | p | l | e | ” | , | | “ | p | r | i | c | e | ” | : | | * | * | 3 | . | 4 | 5 | | } |

$$test(C_{\times}) = \text{Fail}$$

$C_{\surd} =$

$$test(C_{\surd}) = \text{Pass}$$

Maximizing Delta Debugging (Lexical)

- Goal: finding $C'_{\surd}$ such that $C'_{\surd} \subset C_{\times}$, $test(C'_{\surd})$ = Pass, and $\Delta = C_{\times} - C'_{\surd}$ is 1-minimal

$C'_{\surd}$ = { "item": "Apple", "price": 3.45 }

Δ = **

Maximizing Delta Debugging (Lexical)

- Systematically maximizing the $C'_{\surd}$ by recursively invoking $dd_{max}(C'_{\surd}, n)$

- Initially, $C'_{\surd} = \emptyset$, n=2, $\Delta = C_{\times} - C'_{\surd} = C_{\times}$

- Key insight: splitting Δ into n parts and trying to move some of them into $C'_{\surd}$

Maximizing Delta Debugging (Lexical)

- $dd_{max}(\emptyset, 2)$

Δ_1= `{ "item": "Apple",`

Δ_2= ` "price": **3.45 }`

$\nabla_1 = C_{\times} - \Delta_1$ Fail

$C'_{\surd} \cup \Delta_1 = \Delta_1$ Fail

$\nabla_2 = C_{\times} - \Delta_2$ Fail

$C'_{\surd} \cup \Delta_2 = \Delta_2$ Fail

Case 1: $\forall\ i$, $test(\nabla_i)$=Fail, $test(C'_{\surd} \cup \Delta_i)$= Fail, and $n < |C_{\times} - C'_{\surd}|$

Action: $n = \min(|C_{\times}|, 2n)$ = 4

Maximizing Delta Debugging (Lexical)

- $dd_{max}(\emptyset, 4)$

Δ_1 = `{ "item":` Δ_2 = ` "Apple",`

Δ_3 = ` "price":` Δ_4 = ` **3.45 }`

$\nabla_1 = C_\times - \Delta_1 =$ ` "Apple", "price": **3.45 }` Fail $C'_{\surd} \cup \Delta_1 = \Delta_1$ Fail

$\nabla_2 = C_\times - \Delta_2 =$ `{ "item": "price": **3.45 }` Fail $C'_{\surd} \cup \Delta_2 = \Delta_2$ Fail

$\nabla_3 = C_\times - \Delta_3 =$ `{ "item": "Apple", **3.45 }` Fail $C'_{\surd} \cup \Delta_3 = \Delta_3$ Fail

$\nabla_4 = C_\times - \Delta_4 =$ `{ "item": "Apple", "price":` Fail $C'_{\surd} \cup \Delta_4 = \Delta_4$ Fail

Case 1: All Fail

Action: $n = \min(|C_\times|, 2n) = 8$

Maximizing Delta Debugging (Lexical)

- $dd_{max}(\emptyset, 8)$

Δ_6= ` **3.`

$\nabla_6 = C_\times - \Delta_6$= `{ “item”: “Apple”, “price”:45 }` Pass

Case 2: $\exists j,\ \forall i < j, test(\nabla_i)$ = Fail, $test(C'_\surd \cup \Delta_i)$ = Fail, $test(\nabla_j)$ = Pass

Action: $C'_\surd = \nabla_j = \nabla_6$, $n = 2$

Maximizing Delta Debugging (Lexical)

- $dd_{max}(\nabla_6, 2)$

Δ_{6_1} = [ ][*][*]

$\nabla_{6_1} = C_\times - \Delta_{6_1}$ = [{][ ][“][i][t][e][m][”][:][ ][“][A][p][p][l][e][”][,][ ][“][p][r][i][c][e][”][:][3][.][4][5][ ][}] Pass

Case 2: $\exists j,\ \forall\, i < j, test(\nabla_i)$ = Fail, $test(C'_{\surd} \cup \Delta_i)$ = Fail, $test(\nabla_j)$ = Pass

Action: $C'_{\surd} = \nabla_j = \nabla_{6_1}, n = 2$

Maximizing Delta Debugging (Lexical)

- $dd_{max}(\nabla_{6_1}, 2)$

$\Delta_{6_{1_1}} =$ ␣* $\Delta_{6_{1_2}} =$ *

$\nabla_{6_{1_1}} =$ { “item”: “Apple”, “price”:*3.45 } Fail

$C'_{\surd} \cup \Delta_{6_{1_1}} = \nabla_{6_{1_2}}$ Fail

$\nabla_{6_{1_2}} =$ { “item”: “Apple”, “price”: *3.45 } Fail

$C'_{\surd} \cup \Delta_{6_{1_2}} = \nabla_{6_{1_1}}$ Fail

Case 1: All Fail

Action: $n = \min(|C_{\times}|, 2n) = 3$

Maximizing Delta Debugging (Lexical)

- $dd_{max}(\nabla_{6_1}, 3)$

$\Delta_{6_{1_1}}$ = [ ] $\Delta_{6_{1_2}}$ = [*] $\Delta_{6_{1_3}}$ = [*]

$\nabla_{6_{1_1}}$ = { “item”: “Apple”, “price”:**3.45 } Fail

$C'_{\surd} \cup \Delta_{6_{1_1}}$ = { “item”: “Apple”, “price”: 3.45 } Pass

Case 3: $\exists j,\ \forall i < j, test(\nabla_i)$ = Fail, $\forall k \leq j, test(C'_{\surd} \cup \Delta_k)$ = Fail, $test(C'_{\surd} \cup \Delta_j)$ = Pass

Action: $C'_{\surd} = C'_{\surd} \cup \Delta_i = \nabla_{6_1} \cup \Delta_{6_{1_1}}, n = \max(n\text{-}1, 2) = 2$

Maximizing Delta Debugging (Lexical)

- $dd_{max}(\nabla_{6_1} \cup \Delta_{6_{1_1}}, 2)$

$\Delta_a =$ `*` $\Delta_b =$ `*`

$\nabla_a =$ `{ “item”: “Apple”, “price”: *3.45 }` Fail

$C'_{\surd} \cup \Delta_a = \nabla_a$ Fail

$\nabla_b =$ `{ “item”: “Apple”, “price”: *3.45 }` Fail

$C'_{\surd} \cup \Delta_b = \nabla_b$ Fail

Case 4: $\forall\ i$, $test(\nabla_i)$=Fail, $test(C'_{\surd} \cup \Delta_i)$= Fail, and $n \geq |C_{\times} - C'_{\surd}|$

Action: done

`{ “item”: “Apple”, “price”: 3.45 }`

Lexical dd_{max} is slow

- $test$ runs 36 times in preceding example

Table 5: *ddmax* Efficiency on All Invalid Inputs for each technique (A) Baseline, (B) ANTLR, (C) Lexical *ddmax*

Invalid. Type	Inp. Form	Runtime (sec.) A	B	C	#Runs C
Real World	JSON	2	2	1227	341525
	OBJ	44	47	2065	6253
	DOT	48	166	3828	2783
Single Mut.	JSON	4	4	1584	45651
	OBJ	491	672	6151	3809
	DOT	58	60	1239	6077
Multiple Mut.	JSON	10	10	5903	1194577
	OBJ	624	728	9938	8577
	DOT	60	60	3365	34876
Mean		153	200	3981	72296

Speeding dd_{max} up

- Key insight: execute dd_{max} on Parse tree of the input

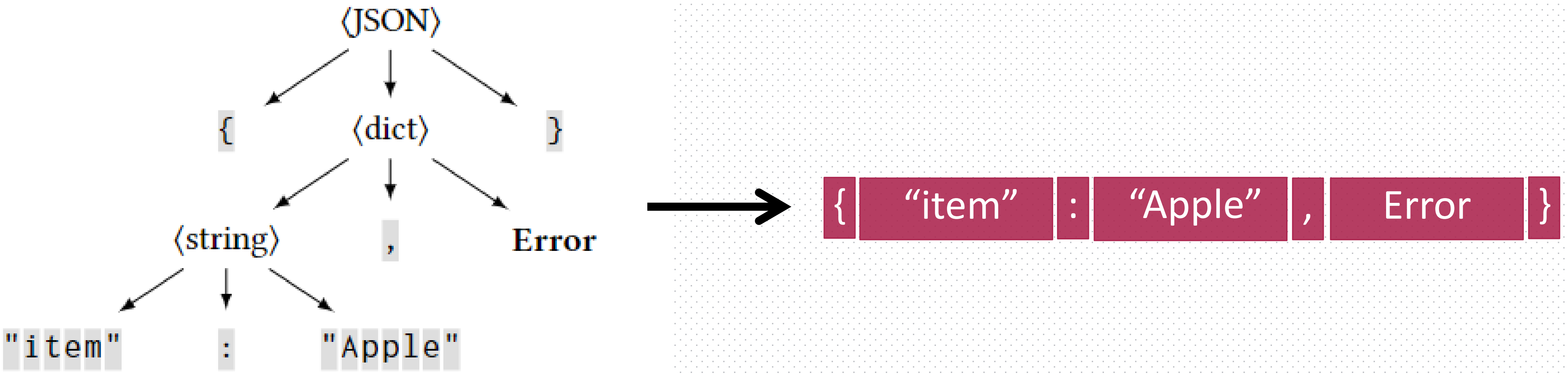


Figure 11: Parse tree of Figure 9

Maximizing Delta Debugging (Syntactic)

- $dd_{max}(\emptyset, 2)$

$\Delta_1 =$ { "item" : "Apple"

$\Delta_2 =$, Error }

$\nabla_1 = C_\times - \Delta_1$ Fail

$C'_\surd \cup \Delta_1 = \Delta_1$ Fail

$\nabla_2 = C_\times - \Delta_2$ Fail

$C'_\surd \cup \Delta_2 = \Delta_2$ Fail

Case 1: All Fail

Action: $n = \min(|C_\times|, 2n) = 4$

Maximizing Delta Debugging (Syntactic)

- $dd_{max}(\emptyset, 4)$

$\nabla_2 = C_\times - \Delta_2$ Fail

$\Delta_1 =$ { "item" $\Delta_2 =$: "Apple" $\Delta_3 =$, Error $\Delta_4 =$ }

$\nabla_1 = C_\times - \Delta_1 =$: "Apple" , Error } Fail $C'_{\surd} \cup \Delta_1 = \Delta_1$ Fail

$\nabla_2 = C_\times - \Delta_2 =$ { "item" , Error } Fail $C'_{\surd} \cup \Delta_2 = \Delta_2$ Fail

$\nabla_3 = C_\times - \Delta_3 =$ { "item" : "Apple" } Pass

Case 2: $\exists j,\ \forall i < j, test(\nabla_i)$ = Fail, $test(C'_{\surd} \cup \Delta_i)$ = Fail, $test(\nabla_j)$ = Pass

Action: $C'_{\surd} = \nabla_i = \nabla_3$, $n = 2$

Maximizing Delta Debugging (Syntactic)

- $dd_{max}(\nabla_3, 2)$

Δ_{3_1} = , Δ_{3_2} = Error

$\nabla_{3_1} = C_\times - \Delta_{3_1}$ = { "item" : "Apple" Error } Fail $C'_\surd \cup \Delta_{3_1} = \nabla_{3_2}$ Fail

$\nabla_{3_2} = C_\times - \Delta_{3_2}$ = { "item" : "Apple" , } Fail $C'_\surd \cup \Delta_{3_2} = \nabla_{3_1}$ Fail

Case 4: $\forall\ i$, $test(\nabla_i)$=Fail, $test(C'_\surd \cup \Delta_i)$= Fail, and $n \geq |C_\times - C'_\surd|$

Action: done   { "item" : "Apple" }

Evaluation Workflow

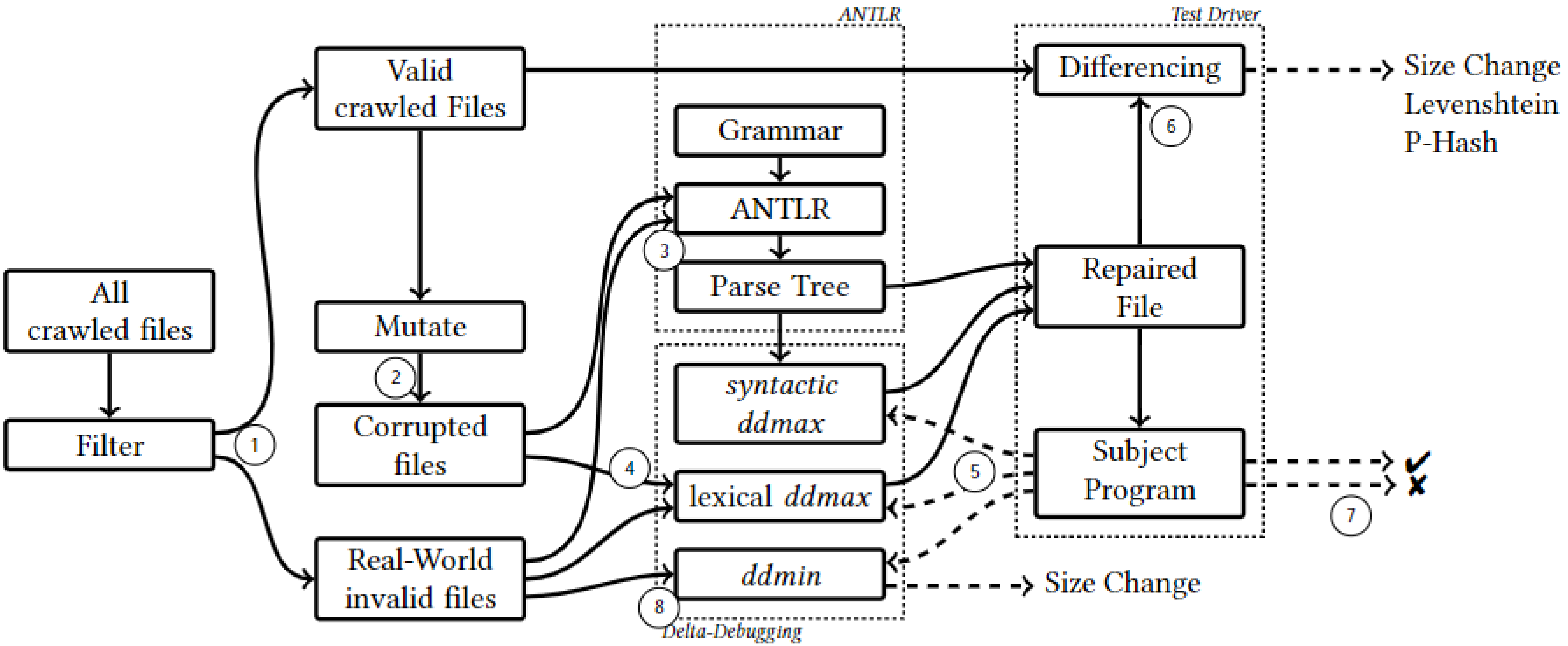


Figure 6: Workflow of the *ddmax* evaluation

Effectiveness

Table 4: *ddmax* Effectiveness on All Invalid Inputs

Invalid. Type	Format (#subjects)	#Possible Repairs	*# repaired input files* Base.	ANTLR	Lex.	Syn.
Real World	JSON (3)	167	0	40	38	62
	OBJ (3)	33	1	8	24	25
	DOT (2)	64	24	25	30	31
Single Mut.	JSON (3)	150	4	80	115	127
	OBJ (3)	150	34	82	146	144
	DOT (2)	100	43	66	92	82
Multiple Mut.	JSON (3)	150	4	45	79	112
	OBJ (3)	150	3	29	127	126
	DOT (2)	100	40	47	51	63
Total (3)		1064	153	422	702	772

Data Recovery and Loss

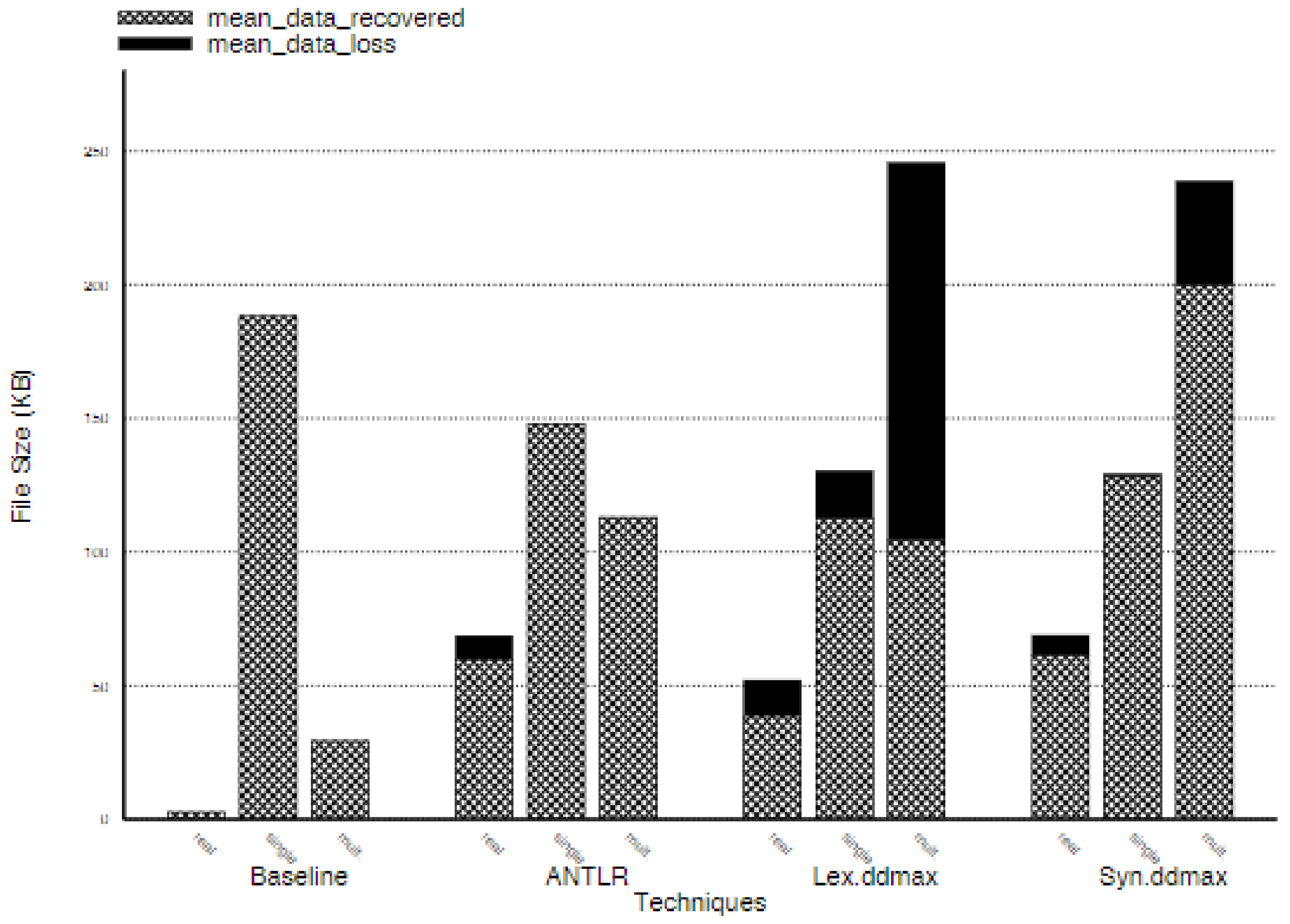


Figure 8: Data Recovered and Data Loss for all Inputs

Data Recovery and Loss

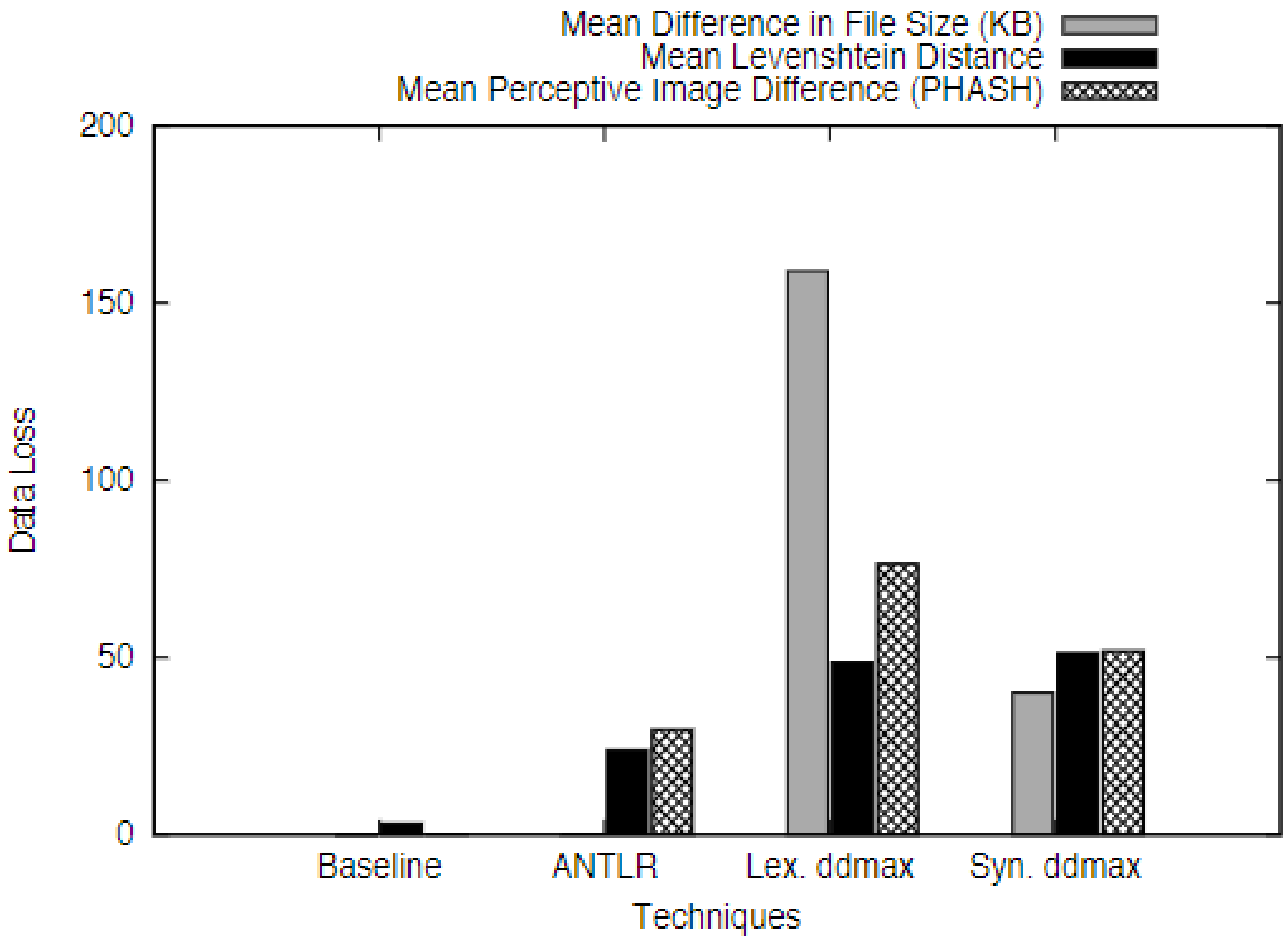


Figure 12: Data Loss Incurred for All Mutations

Efficiency

Table 5: *ddmax* Efficiency on All Invalid Inputs for each technique (A) Baseline, (B) ANTLR, (C) Lexical *ddmax*, (D) Syntactic *ddmax*.

Invalid. Type	Inp. Form	*Runtime (sec.)*				*#Runs*	
		A	B	C	D	C	D
Real World	JSON	2	2	1227	153	341525	6029
	OBJ	44	47	2065	1279	6253	3164
	DOT	48	166	3828	3018	2783	1162
Single Mut.	JSON	4	4	1584	1065	45651	129659
	OBJ	491	672	6151	4083	3809	1352
	DOT	58	60	1239	1244	6077	4565
Multiple Mut.	JSON	10	10	5903	2153	1194577	448801
	OBJ	624	728	9938	8132	8577	5043
	DOT	60	60	3365	2241	34876	11956
Mean		153	200	3981	2624	72296	70049

Diagnostic Quality

Table 6: Diagnostic Quality on Real-World Invalid Inputs for Ⓐ *ddmin* and Ⓑ *ddmax* diagnoses, and Ⓒ *ddmax* repair.

Format (#inputs)	Diagnosis (B) Ⓐ	Diagnosis (B) Ⓑ	Repair (B) Ⓒ	*Intersection (%)* Ⓐ ∩ Ⓑ	*Intersection (%)* Ⓐ ∩ Ⓒ
JSON (21)	2.909	19.095	103.476	13.88	23.18
OBJ (18)	2.722	1.000	189.000	18.03	11.46
DOT (27)	376.654	1.115	675.346	5.76	54.64
Mean	155.754	6.804	360.747	11.69	32.85

Thank you!

Comments are welcome!