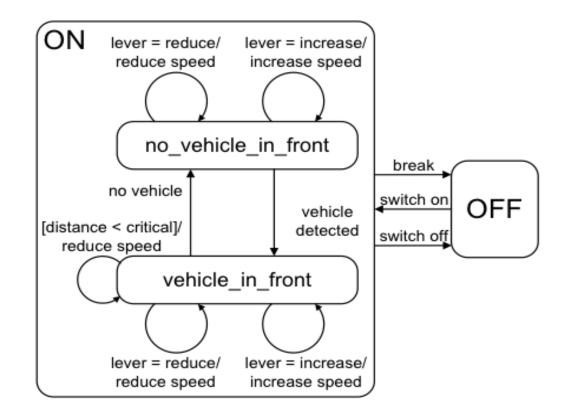
Semantic Mutation Testing

John A. Clark, Haitao Dan, Robert M Hierons

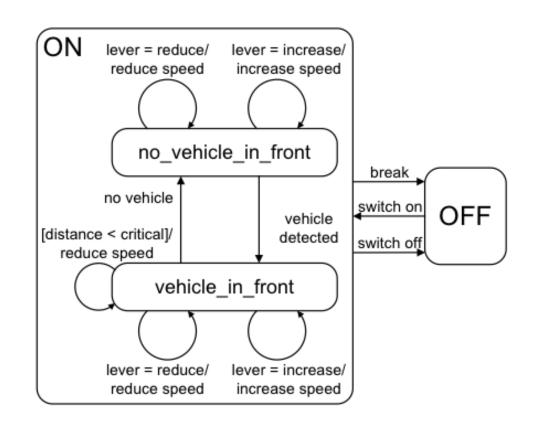
The 8th CREST Open Workshop, 27-10-2010

An example: cruise control



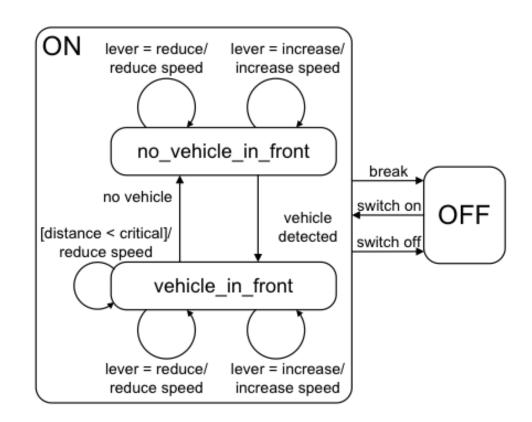
Question

 What happens in no_vehicle_in_front if brake and level=increase?



Another question

What happens in no_vehicle_in_front if a vehicle is detected and level=increase?



The problem

• Traditional mutation operators introduce changes similar to 'slips'.

- Sometimes a developer/user will make semantic mistakes:
 - They will misunderstand the semantics of part of the language they are using

Semantic Mutation

• A developer has been using language X with semantics L and moves to X with semantics L'.

 How do we find test data to find resultant faults?

An alternative: switching between programming languages

- Developer moves between two languages at the same level of abstraction that have different semantics for a common construct.
- Example:
 - Logical connectives in C and Ada.
 - C uses short-circuit evaluation;
 - Ada has alternatives (with and without short-circuit evaluation)

Scenario: refinement/retrenchment

- Similar constructs can have different semantics.
- Examples:
 - integer division in Z and Ada
 - retrenching infinite types (issues with precision, bounds on the types)

A simple framework

- We have a syntactic entity N in a language with semantics L.
- Traditional mutation operators transform (N,L) to some (N',L)
- Semantic mutation operators transform (N,L) to some (N,L') [or maybe even (N',L')]
- They aim to find a different type of mistake.

Current status and future work

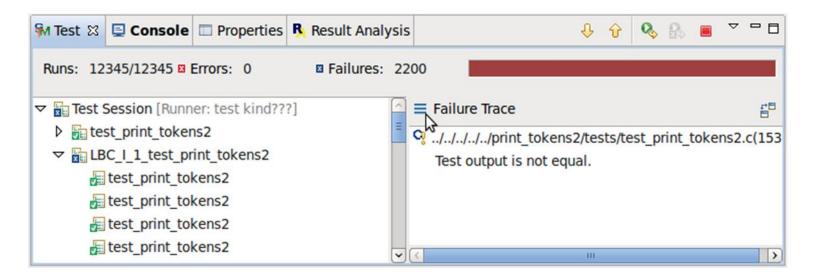
- Prototype tool being developed for C
- Some experiments being conducted to explore nature of semantic mutants:
 - How many are produced?
 - How do they relate to traditional syntactic mutants?
 - What are good operators?
 - Are there many trivial or equivalent mutants?
- More experiments

A Semantic Mutation Tool for C

GUI of SMT-C*

M	SMT - C			
<u>F</u> ile SMT Se <u>a</u> rch				
🗟 SMT Navigator 🛿 🗖 🗖	Image: Image	I Mu SMT Mutant 🛛 🗖 🗖		
□ 🔄 🌫 🏹	😪 C Compare	□ 🔄 🏹		
▶ • c print tokens	▽ 🖻 Translation Unit	▶ • c print tokens		
✓ ^c print_tokens2	 open_character_stream 	▶ [■] ^c print_tokens2		
👂 🐰 Binaries		▷ [■] ^c replace		
▷ III Archives		▷ ^{■ c} schedule		
autom4te.cache	😪 C Compare Viewer 🚽 🦻 🊱 🖓 🎪 🕸 🖓	▷ [•] c schedule2		
🕨 🔁 build-aux	print_tokens2/mutarc/print_tokens2.c print_tokens2/mutarc/print_tokens2.c	▷ ^{■ c} space		
	38 if (fname == NULL) fp = s 38 if (fname == NULL) fp	▷ [•] c space1		
▼ 🔁 LBC_I_1	39 else if ((fp = fopen (fna) 39 else { 40 fprintf (stdout, "The) 40 fprintf (stdout, 1 = 1	▼ [■] c tcas		
	41 exit (0); 41 exit (0);			
autom4te.cache	42 }	▽ 🗁 OAAN_1		
🕨 🗁 build-aux	43 else { 44 abort (); 44 43 return (fp); 44}	マ 🔁 tcas		
🕨 👝 smtlib	45 } 45	⊽ 🗁 src		
arc 🔁 🔁	46 return (fp); 46 char get_char (fp)	🗟 tcas.c		
D 👝 .deps	47 character_stream fp; 48 48	DAAN_2		
D 👝 .libs	49 <mark>char get char (fn)</mark> 49 char ch:	DAAN_3		
print_tokens2.c				
print_tokens2-r	₩ Test 📮 Console 🔲 Properties 🦻 Result Analysis 🖾			
🗋 Makefile	Project: Live mutants: Killed mutants:	Killed by:		
Makefile.am	print_tokens SBRC_4 IMB_1	test_print_tokens.c:152:33		
Makefile.in	SSDL 11	<pre>test_print_tokens.c:152:99</pre>		
i print_tokens2	SSDL_12 IMB_11	test_print_tokeps_c-152-10		
🕨 🗁 tests 🔍 👻	Mutant total: 484 Alive total: 38 Killed total: 446	Mutant score: 0.02		
		Mutant score: 0.92		
/print_tokens2/mutants/LBC_I_1/print_tokens2/src/print_tokens2.c				

GUI of Test Runner*



•Running results of test suites and testcases: statistics and the result for each test suite and testcase with graphical highlight;

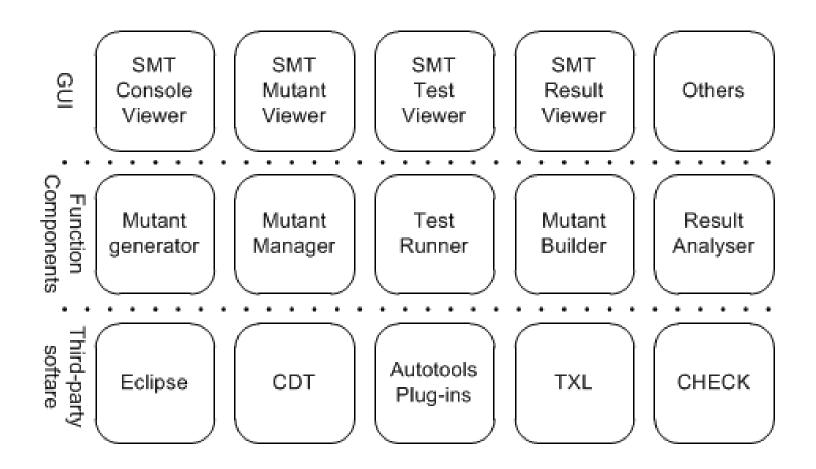
- Progress bar;
- •Test error traces.

Mutant generation*

M Mutant generation				
Select mutation elements and operators:				
▼ ✓ ^{■ c} print_tokens	OP Name	OP Description		
🗢 🗆 🗁 src	ASD	Additional semicolon delet		
✓ c print_tokens.c	LBC_I	Logic branch completion f		
¬ □ □ ^c print_tokens2	LBC_C	Logic branch completion f		
🗢 🗌 🗁 src	LBM_I	Logic branch modification		
Image: Print_tokens2.c	LBM_C	Logic branch modification		
▷ □ [•] c replace	□ MFC_E	Modification of float-point		
▷ □ ^{• c} schedule	□ MFC_R	Modification of float-point		
▷ □ ^{• c} schedule2	FTA_F	Float-point truncating adju		
▷ □ [•] c space	FTA_T	Float-point truncating adju		
▷ □ [•] space1	DIA_F	Division/modulus of interg		
▷ □ [•] c tcas	DIA_T	Division/modulus of interg		
▷ □ [•] c tot_info	IMB	Insert missing break		
	🗢 🔲 Traditional MOP			
	SBRC	Break to continue		
	SCRB	Continue to break		
Select All Deselect All Deselect All		Select <u>A</u> ll <u>D</u> eselect All		
Please choose the scope for generate mutants:				
Generate mutants for selected projects.				
Generate mutants for selected files.				
✓ Generate mutants for each match expression of selected files.				
		Cancel OK		

Mutant generation -- support three different scopes

Tool Architecture



Implementation Overview

- The tool is developed using Java and as Eclipse plug-ins.
- It also can be published as an independent testing tool based on Rich Client Platform (RCP) of Eclipse.
- For current version, TXL is used to drive the semantic mutation and Check is used to support mutant compilation and running tests.

TXL – as a prototyping mutation engine

- It is a generalized source-to-source translation system.
- It takes as input an piece of source code, and a set of transformation.
- It produces as output the transformed source code.
- Example:
 - txl source1.c tranform_rule.txl

Semantic Mutation Operators

• Thirteen semantic mutation operators have been implemented.

– ASD, MFC_R, FTA_F...

 6 traditional mutation operators were also implemented for conducting experiments to compare traditional and semantic mutation operators.

– SCRB, SSWM, SSDL ...

CHECK

- A unit testing framework for C.
- Check is based on Autotools.
- Many advanced features: run in fork mode (allow signal and early exit), test fixture, multiple suites in one runner, looping tests, test timeouts, determining test coverage, xml logging etc.

Future work of SMT-C

- Implement more semantic mutation operators.
- Improve the GUI, better integration with C development process.
- Enhance mutant generation function: mutant management, function scope mutation and efficiency.
- Accelerate the mutation generation and testing processes.

?