HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Measuring Coverage of Prolog Programs Using Mutation Testing

Alexandros Efremidis Joshua Schmidt Sebastian Krings Philipp Körner

Institut für Informatik, Universität Düsseldorf
Universitätsstr. 1, D-40225 Düsseldorf

6th September 2018

Motivation

- avoid and identify bugs
- increase maintainability
- enhance development progress

Code Coverage Metrics

- Predicate, Clause, Sub-Goal Coverage
- Branch Coverage
- Path Coverage

```
is_list([]).
is_list([_|T]):
    is_list(T).
```

```
: begin_tests(suite).
test(test1):
    is_list([foo]).
: end_tests(suite).
```

Mutation Testing

- generation of mutants
- checking the test's outcome
- dead: at least one test fails
- alive: all tests pass

```
is_list([]).
is_list([_|T]):
    is_list(T).
```

```
:- begin_tests(suite).
test(test1):
   is_list([foo]).
:- end_tests(suite).
```

Mutation Score

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

$$\mu_{score} = \frac{\mu_{dead}}{\mu_{dead} + \mu_{alive}}$$

Mutation Operators

- *sensible* operators: it is expected to yield semantically different programs
- *foolish* operators: it is expected to retain the code's original semantics

Predicate Removal

- obviously sensible
- comparable to predicate coverage

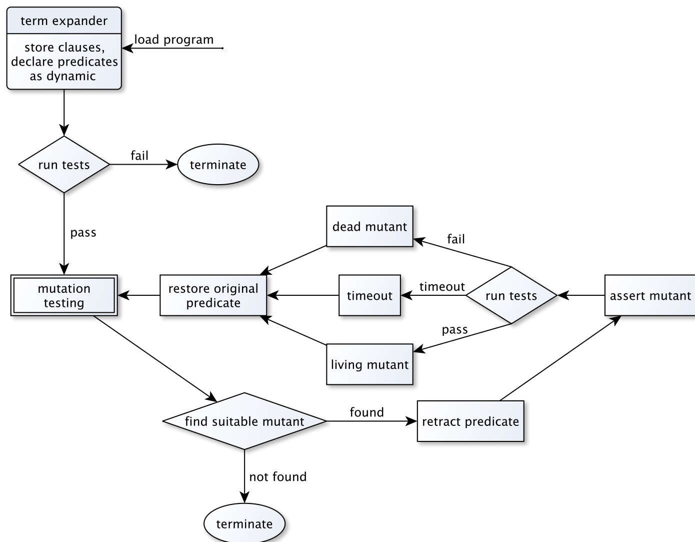
Interchanging Relational Operators

- sensible
- can easily become foolish when not negating the operator

```
min(A, B, A):
    A < B, !.
min(A, B, B).
```

```
min(A, B, A):
    A >= B, !.
min(A, B, B).
```

A Mutation Testing Framework



HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Empirical evaluation

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Table: Comparison of Prolog Coverage Tools

Prolog File	LoC	Predicates	Clauses	Clause Cover-age	Predicate Cover-age	Sub-Goal Cover-age	Mutation Cover-age
ast_interpreter	107	2	20	100.00%	100.00%	100.00%	88.10%
compiler	165	15	42	80.95%	62.50%	86.04%	52.60%
parser	80	38	16	100.00%	100.00%	100.00%	95.00%
rational_trees	39	4	10	100.00%	100.00%	100.00%	93.70%
rt_bytecode	105	4	33	93.93%	50.00%	95.24%	92.70%
alloy2b	725	74	198	84.41%	75.95%	87.68%	83.50%
eight_puzzle	161	21	44	77.27%	67.10%	82.28%	78.50%

Future Work

- detecting semantically equivalent mutations
- automated test generation

Conclusion

- a mutation testing framework has been developed
- sets of mutation operators regarded *sensible* or *foolish* have been determined
- empirical evaluation on larger code examples
  $\rightarrow$ reporting different results compared to other metrics
- mutation testing should be used for more insight into code coverage

Fin

Time for questions!

Empirical Evaluation - Extension

Mutation	ast_interpreter	compiler	parser	rational_trees	rt_bytecode	alloy2b	eight_puzzle
remove predicate	0/3	4/11	0/5	0/5	0/4	8/68	4/17
; to ,	0/2	0/2	0/3	0/2	0/1	1/15	1/0
, to ;	0/30	10/34	1/14	4/12	12/31	38/245	22/22
= to \=	0/1	0/2	0/0	0/1	0/0	4/51	1/0
\= to =	0/0	0/0	0/0	0/0	0/0	0/1	0/0
=:= to =\=	0/0	0/0	0/0	0/0	0/0	0/0	0/0
=\= to =:=	0/0	0/0	0/0	0/0	0/0	0/0	0/0
== to \==	0/2	0/0	0/0	0/3	0/1	1/2	0/0
\== to ==	0/0	0/0	0/0	0/0	0/0	0/0	0/0
> to =<	0/0	0/0	0/0	0/0	0/0	0/2	1/2
>= to <	0/0	0/4	0/7	0/0	0/0	0/0	0/0
< to >=	0/0	0/2	0/0	0/0	0/0	0/1	0/1
=< to >	0/0	0/0	2/5	0/0	0/0	0/0	0/1
+ to -	0/0	0/6	0/0	0/0	0/0	0/2	0/2
- to +	0/0	4/35	0/0	0/5	0/0	0/1	0/2
* to +	0/0	0/0	0/0	0/0	0/0	0/0	0/0
/ to -	0/0	0/0	0/0	0/0	0/0	0/0	0/0
increase number	0/0	10/34	11/3	0/0	0/15	9/151	2/30
decrease number	0/0	9/35	12/2	0/0	1/14	10/151	4/28
negate expression	0/19	6/24	0/7	0/9	2/29	15/102	6/22
true to false	0/3	0/2	0/0	0/2	0/0	1/2	0/0
false to true	0/3	1/2	0/0	0/0	0/0	8/0	0/0
var to _	33/184	122/245	0/69	5/103	133/252	244/1327	95/138
atom to _	0/0	1/2	0/0	0/4	1/3	48/185	3/0
[] to _	0/1	12/6	3/0	0/1	2/1	42/51	11/1
permute cut	1/0	4/1	0/1	2/0	16/0	18/21	1/3
reverse predicate	3/0	13/2	3/2	5/0	4/0	38/22	18/0

Cut Tranformations

- may be *foolish* very often
- *here:* green to green cut transformation

```
element(_, []):
 false, !.
element(Element, [Head|_]):
 Element == Head, !.
element(Element, [_|Tail]):
 element(Element, Tail).
```

```
element(_, []):
 !,
 false.
element(Element, [Head|_]):
 Element == Head, !.
element(Element, [_|Tail]):
 element(Element, Tail).
```

Cut Tranformations

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

- *sensible* when permuting to another position, resulting in semantically nonequivalent code

```
element(_, []):
 false, !.
element(Element, [Head|_]):
 Element == Head, !.
element(Element, [_|Tail]):
 element(Element, Tail).
```

```
element(_, []):
 false, !.
element(Element, [Head|_])  :
 !,
 Element == Head.
element(Element, [_|Tail])  :
 element(Element, Tail).
```