UNIVERSITÄT
DUISBURG
ESSEN

*Open-Minded*

# *Mutating Sample Solutions to Improve Prolog Exercise Tasks and Their Test Suites*

Ivan Khu Tanujaya, Janis Voigtländer, Oliver Westphal ■ 19.09.2022

# What this paper/talk is about



From a review on the submitted version:

> *It seems the article is in a limbo between a teaching experience report and the application and evaluation of the mutation testing technique.*

Yeah, that's true. Intended was mostly the former, actually.

# The teaching setting

UNIVERSITÄT DUISBURG ESSEN
*Open-Minded*

Courses:

- „Einführung in die Logik“,
  1st semester for students of an interdisciplinary (Computer Science, Psychology, Business Administration) Bachelor programme,
  Prolog makes up about 20% of the course and comes between propositional and predicate logic
- „Programmierparadigmen“,
  4th semester for students of a CS Bachelor programme,
  Prolog makes up about 35% of the course and comes after Haskell

In both courses, weekly exercises are offered via an e-learning system with immediate feedback.

# The Prolog task type (in Autotool)

UNIVERSITÄT DUISBURG ESSEN
*Open-Minded*

For example, for a fact base like

```
female(anna). female(juliet). ...
male(harry). male(luke). ...
child(lisa,anna). child(mary,juliet). ...
```

define a predicate brother/2.

On submission:

```
brother(X,Y) :- male(X),child(X,Y).
```

obtained feedback:

```
... incorrect. Your submission gives:
X = luke, Y = juliet;
X = luke, Y = harry; ...
```

# The Prolog task type (in Autotool)

UNIVERSITÄT DUISBURG ESSEN
*Open-Minded*

For example, for a fact base like

```
female(anna). female(juliet). ...
male(harry). male(luke). ...
child(lisa,anna). child(mary,juliet). ...
```

define a predicate brother/2.

On submission:

```
brother(X,Y) :- male(X),child(X,Z),child(Y,Z).
```

obtained feedback:

```
... incorrect. Your submission gives:
... ; X = luke, Y = luke; ...
```

# The Prolog task type (in Autotool)

UNIVERSITÄT DUISBURG ESSEN
*Open-Minded*

For example, for a fact base like

```
female(anna). female(juliet). ...
male(harry). male(luke). ...
child(lisa,anna). child(mary,juliet). ...
```

define a predicate brother/2.

On submission:

```
brother(X,Y) :- male(X),child(X,Z),child(Y,Z),x\=Y.
```

obtained feedback:

```
... incorrect. Your submission gives:
... ; X = luke, Y = luke; ...
```

# The Prolog task type (in Autotool)



For example, for a fact base like

```
female(anna). female(juliet). ...
male(harry). male(luke). ...
child(lisa,anna). child(mary,juliet). ...
```

define a predicate brother/2.

On submission:

```
brother(X,Y) :- male(X),child(X,Z),child(Y,Z),X\=Y.
```

obtained feedback:

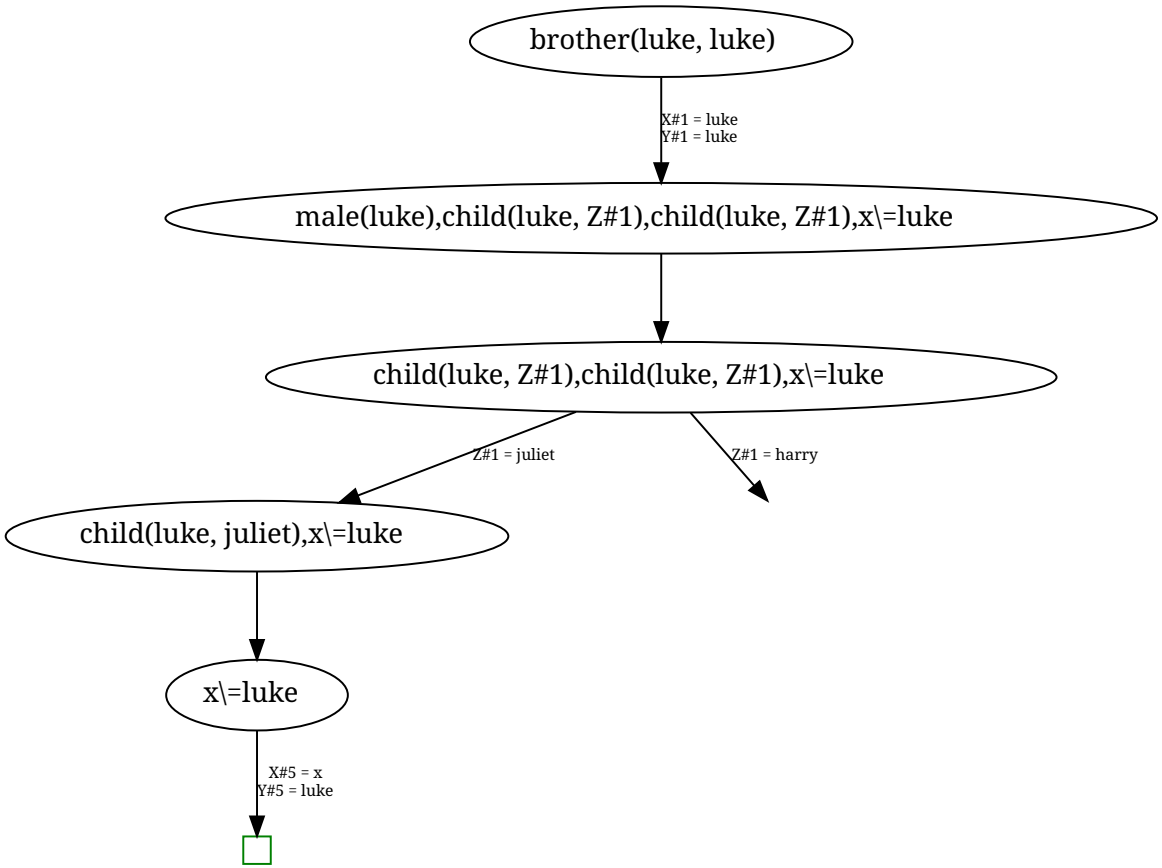
ok

# The Prolog task type (in Autotool)

UNIVERSITÄT DUISBURG ESSEN
*Open-Minded*

More helpful feedback on this erroneous submission:

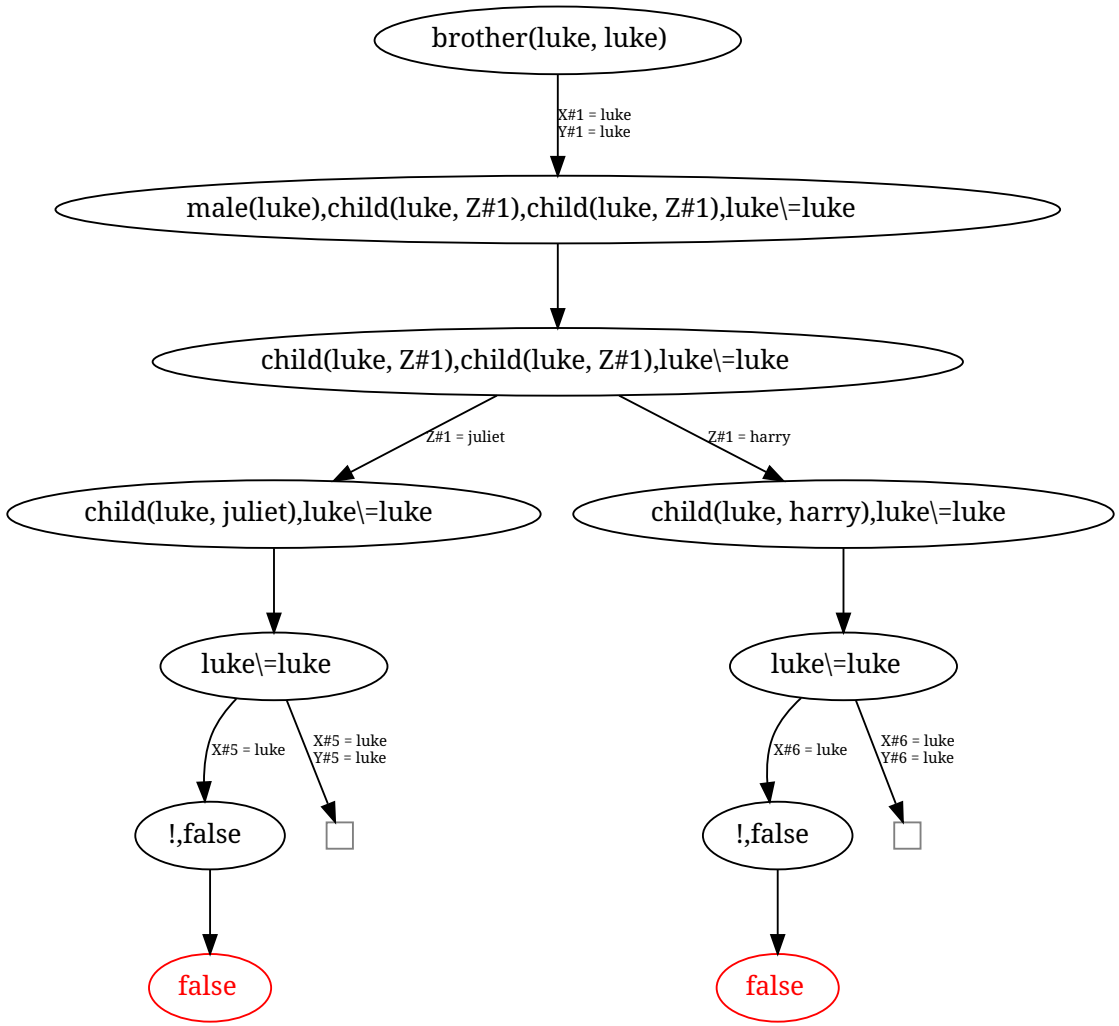
```
brother(X,Y) :- male(X),child(X,Z),child(Y,Z),x\=Y.
```



# The Prolog task type (in Autotool)

UNIVERSITÄT DUISBURG ESSEN
*Open-Minded*

. . . vs. what holds for the correct submission with X\=Y, namely:



# Features that can be used in tests

UNIVERSITÄT DUISBURG ESSEN
*Open-Minded*

```
Global timeout: 10000
% and/or prefix individual tests with [<timeout>]
predicate(X,Y): predicate(x1,y1),predicate(x2,y2),...
true_statement
!predicate(X,Y)[answers hidden]: predicate(x1,y1),...
!true_statement[hidden]
!(<description, shown on failure>)true_statement
@... % shows derivation tree if test fails
-... % negates test expectation
new named_by_student(X,Y): <predicate description>

... other features / hidden auxiliaries ...
```

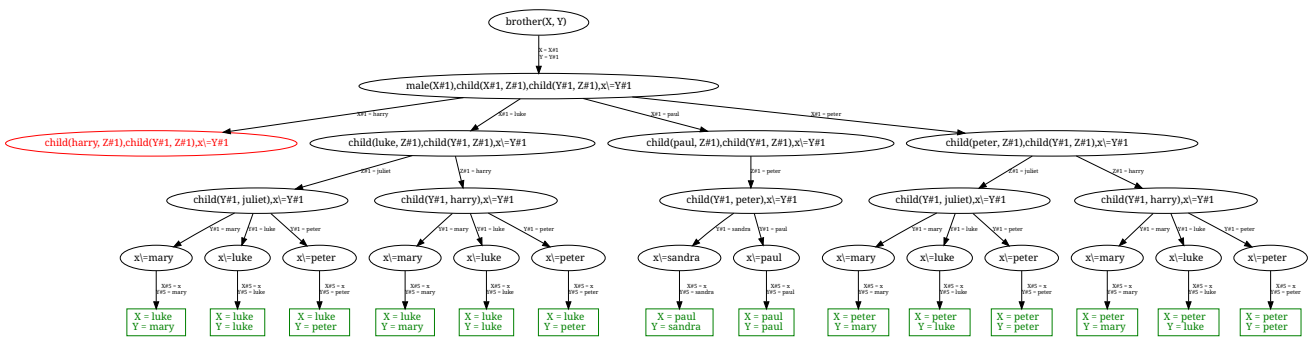
# Possible test suite for the example task

UNIVERSITÄT DUISBURG ESSEN
*Open-Minded*

```
brother(X,Y): brother(peter,mary),
              brother(peter,luke),
              brother(luke,mary),
              brother(luke,peter),
              brother(paul,sandra)

-@brother(luke,luke)
```

On the other hand, `@brother(X,Y):...` not so useful here, due to:



# How to come up with good test suites?

UNIVERSITÄT DUISBURG ESSEN
*Open-Minded*

Trying our best (within the confines of our setup):

- practice and experience
- looking at student submissions
- improving from year to year
- anticipating typical mistakes
- catering for possible course specific misconceptions

On the last point, a few “false friends” when doing Prolog after Haskell . . .

# On “false friends” (Prolog after Haskell)



List syntax:

| | | |
|---|---|---|
| x:xs | [X\|Xs] | |
| x:y:zs | [X\|Y\|Zs]? | [X\|[Y\|Zs]] or [X,Y\|Zs] |

Typing:

| | |
|---|---|
| p :: [a] → Bool; p as = . . . | p([As]) :– ... ? |

Accidental singleton lists:

```
reverse([],[Ys],[Ys]).
reverse([X|Xs],[Ys],[Zs]) :- reverse([Xs],[X|Ys],[Zs]).
```

# On “false friends” (Prolog after Haskell)



Non-exclusionary patterns:

```
intersperse(_,[],[]).
intersperse(_,[Y],[Y]).
intersperse(X,[Y|Ys],[Y,X|Zs]) :- intersperse(X,Ys,Zs).
```

With only positive tests (e.g., mimicking the test suite used for a corresponding Haskell task), the problem is not even detected!

So, how can we systematically improve and grow our test suites in light of such issues?

# Mutating sample solutions



Mutation testing:

- Change a code base by applying small changes.
- On the obtained mutants, run an existing test suite.
- Check how many mutants survive (by passing all tests).
- Draw (quantitative) conclusions about the test suite.

Typically applied to “programming in the large”, also in context of Prolog.

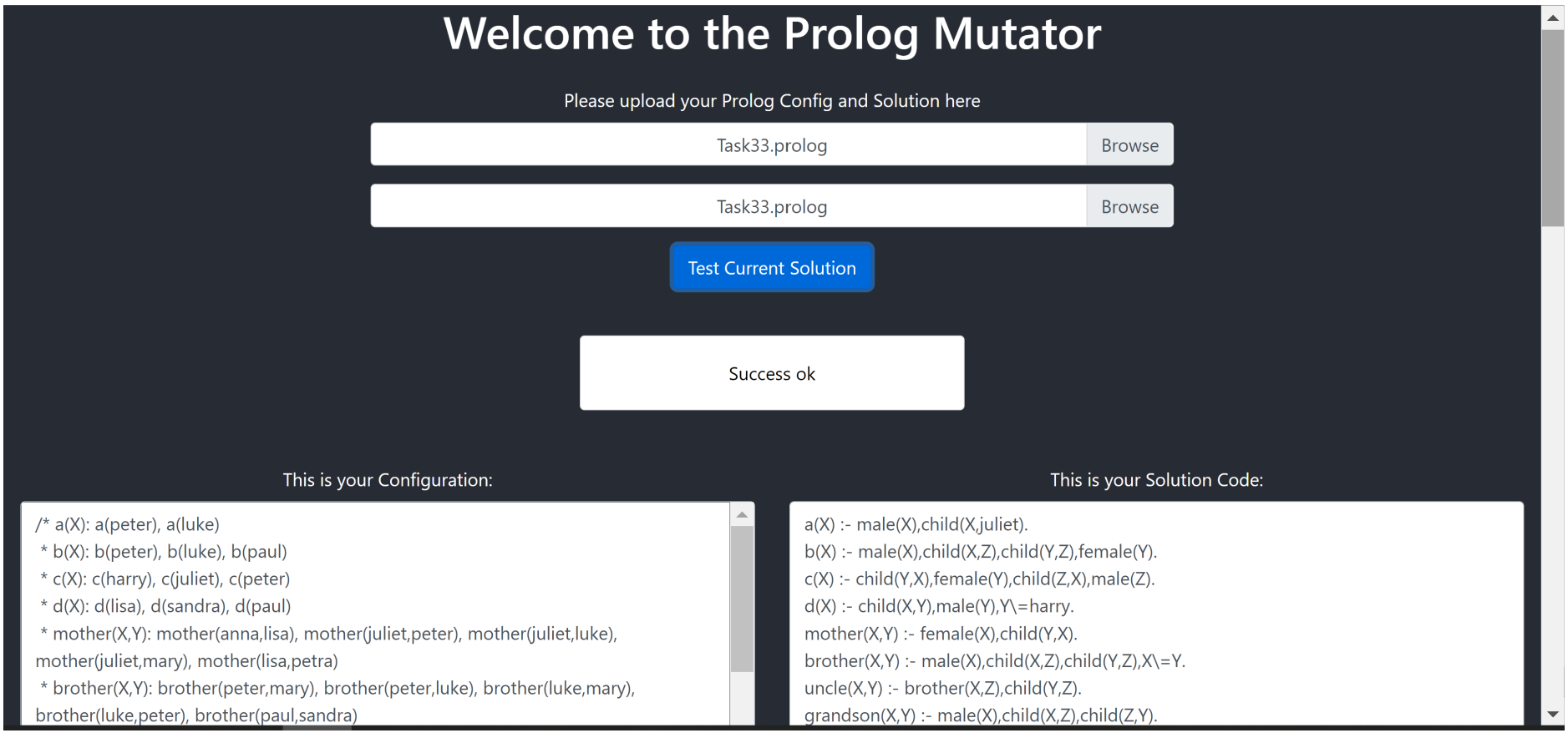
Particularities in our situation:

- All artifacts are rather small.
- We have a sample solution that is known to be correct.
- We have more specific quality questions about the test suite.
- We can try to imitate typical mistakes made by students.

# The tool created

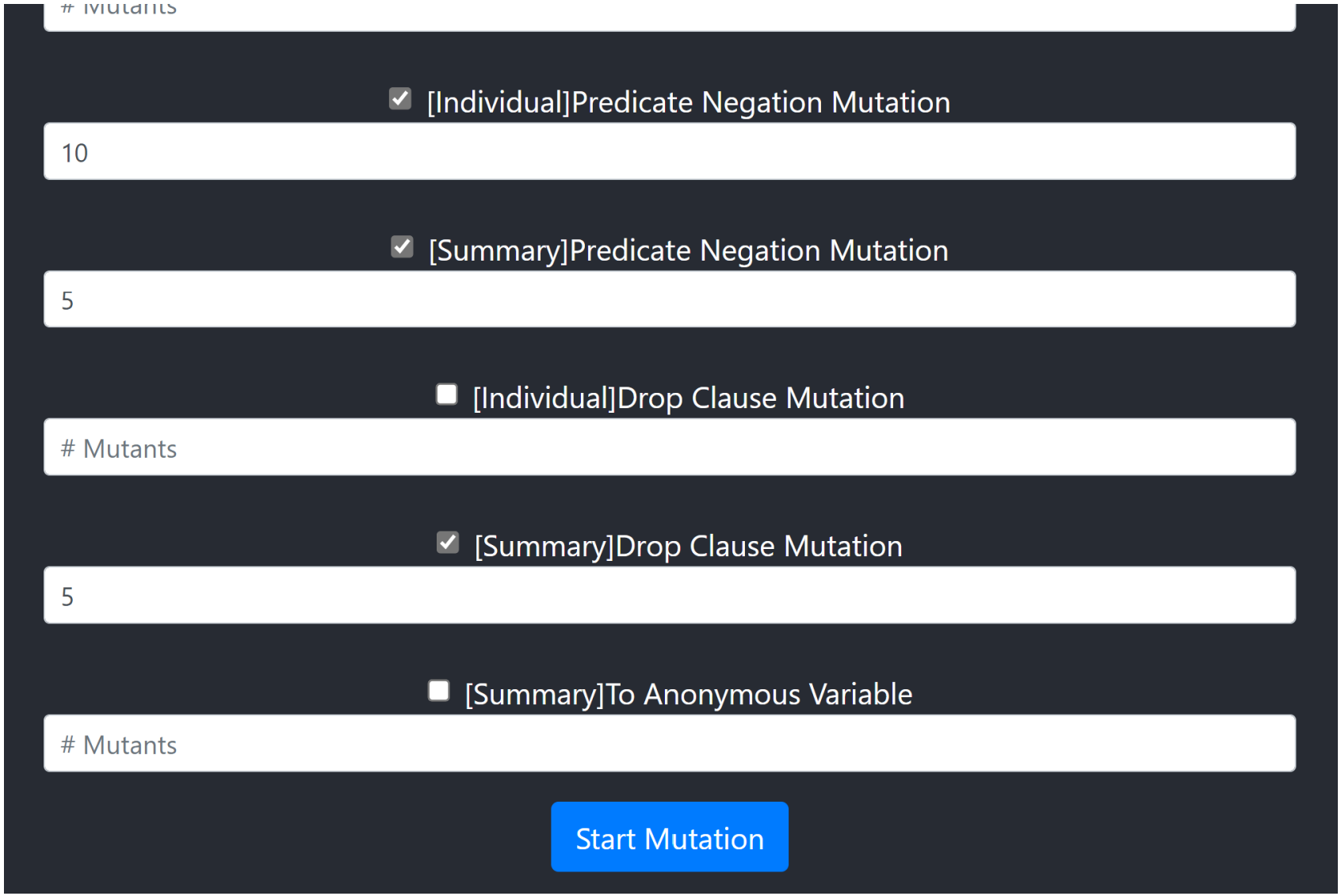
UNIVERSITÄT DUISBURG ESSEN

*Open-Minded*



# The tool created

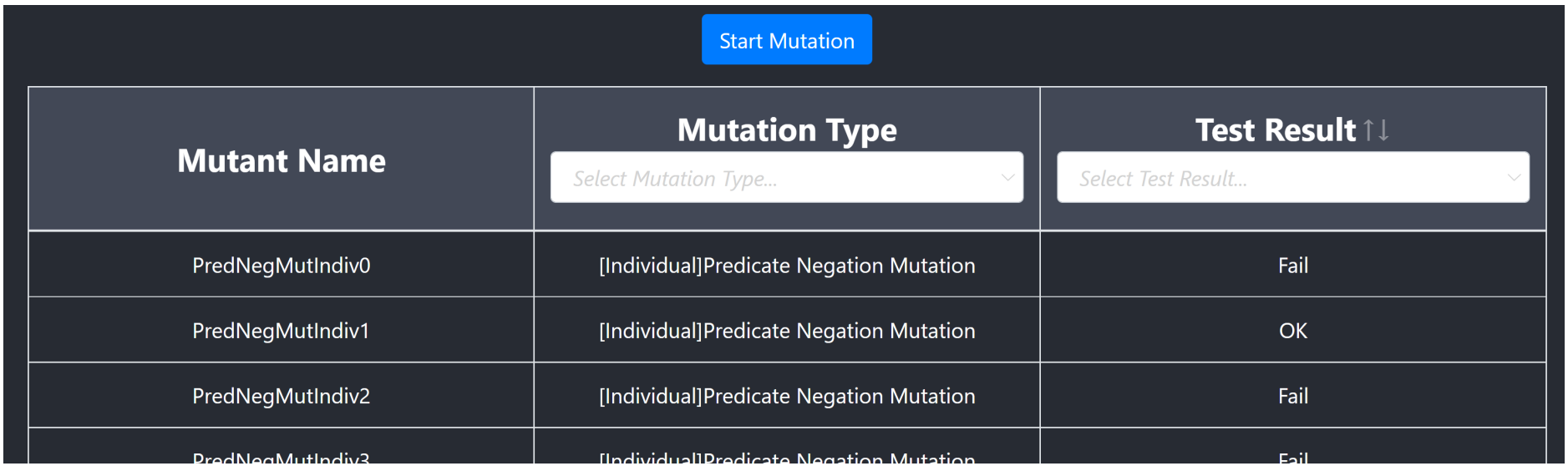
UNIVERSITÄT DUISBURG ESSEN
*Open-Minded*



# The tool created

UNIVERSITÄT DUISBURG ESSEN

*Open-Minded*



# The tool created

UNIVERSITÄT DUISBURG ESSEN

*Open-Minded*

| Mutant Name | Mutation Type | Test Result |
|---|---|---|
| | Select Mutation Type... | OK |
| PredNegMutIndiv1 | [Individual]Predicate Negation Mutation | OK |

**Code Diff**

```
1   1   a(X) :- male(X),child(X,juliet).
2   2   b(X) :- male(X),child(X,Z),child(Y,Z),female(Y).
3       c(X) :- child(Y,X),female(Y),child(Z,X),male(Z).
    3   c(X) :- child(Y,X), \+ female(Y),child(Z,X),male(Z).
4   4   d(X) :- child(X,Y),male(Y),Y\=harry.
5   5   mother(X,Y) :- female(X),child(Y,X).
6   6   brother(X,Y) :- male(X),child(X,Z),child(Y,Z),X\=Y.
```

```
a(X) :- male(X),child(X,juliet).
b(X) :- male(X),child(X,Z),child(Y,Z),female(Y).
c(X) :- child(Y,X), \+ female(Y),child(Z,X),male(Z).
d(X) :- child(X,Y),male(Y),Y\=harry.
mother(X,Y) :- female(X),child(Y,X).
brother(X,Y) :- male(X),child(X,Z),child(Y,Z),X\=Y.
uncle(X,Y) :- brother(X,Z),child(Y,Z).
grandson(X,Y) :- male(X),child(X,Z),child(Z,Y).
```

```
Success

ok
```

Re-Test Mutant

# Coming back to the examples



For the sample solution

```
brother(X,Y) :- male(X),child(X,Z),child(Y,Z),X\=Y.
```

the previously displayed defects motivating `-@brother(luke,luke)` would be created by mutation operators

- Drop Literal
- Variable to Atom

respectively.

Similarly for the intersperse-example (depending on the specific sample solution).

For the “accidental singleton lists” cases, want a mutation operator

- Wrap Variable into List

# Conclusion



Experiences:

- The tool has already proved its worth (for us) in practice.
- We found issues in several tasks we had been using (sometimes with slight variation) over the years. Some issues were in the test suite, some in the task material.
- A particular vulnerability in test suites was neglect of negative tests.
- Sometimes unexpectedly correct mutants appear. We learned something about our exercise task then.

Possible future work:

- extending the mutation catalog
- exploring test suite creation from scratch
- using information about surviving mutants of student programs