LERO Software for a better world

A World Leading SFI Research Centre

Science Foundation Ireland sfi For what's next

The case for Grammatical Evolution in test generation

Aidan Murphy, Thomas Laurent & Anthony Ventresque

11th International Workshop on Genetic Improvement
@ GECCO 2022, Boston, July 9

University College Dublin
Ireland's Global University

Motivation

Generating tests is a critical task.

Time consuming for human experts.

Many techniques aim to reduce this load.

Search-based techniques are among the most widely used.

Search Based Test Generation

Shown very promising results in many domains.

Many examples:

- EvoSuite,
- Randoop,
- UtBot,
- Many more....

Search Based Test Generation

Still shows many limitations:

- Incorporate Human Expertise.
- Create Complex Objects.
- Domain Flexibility.

Grammatical Evolution

Evolutionary computation technique.

Create syntactically correct structures in any language.

Uses a grammar to define the search space.

Offers a possible solution to SBST’s limitations.

Grammatical Evolution

Incorporate Human Expertise

```
<Test>  ::= [<var1>,<var2>]
<var1>    ::= Random();
<var2>   ::= Random();
```

```
<Test>  ::= [<var1>,<var2>]
<var1>    ::= Random();
<var2>   ::= <var1> + Random();
```

Grammatical Evolution

Create Complex Objects

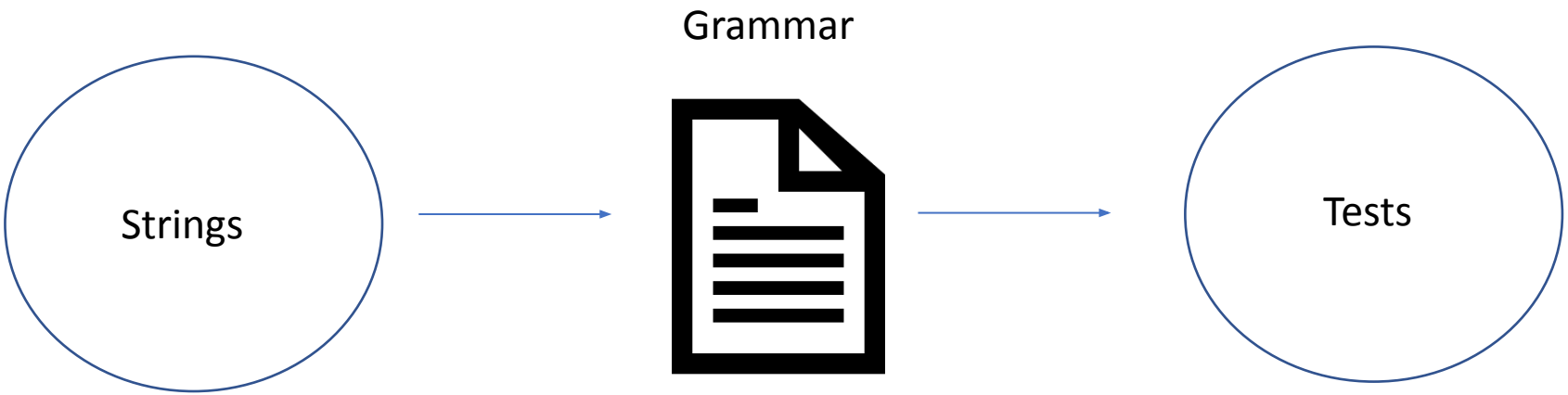
```
src = "X:while(1){try{while(2){try{var a;break X;}" +
 "finally{}}}finally{}}";
```

Example grammar

```
<start>          ::= "X:<cfg>"
<cfg>            ::= <try> | <try_catch>
<try>            ::= <try_statement> | <try_recursion>
<try_catch>      ::= <try_catch> | <try_catch_finally>
<try_statement>  ::= try{<cond>
                         } finally {}
<try_recursion>  ::= try{<cond>
                         <cfg>
                         } finally {}
<try_catch>      ::=  try{<cond>
                         } catch(<exception>){
                         }
<try_catch_finally> ::=  try{<cond>
                         } catch(<exception>){
                         } finally {}
<cond>           ::= <condition> | <condition>; break X;
<condition>      ::= while(<num>) | if(<num>) | for(<num>) | var <variable>
<num>            ::= 1 | 2 | 3 | 4
<variable>       ::= a | b | c
```

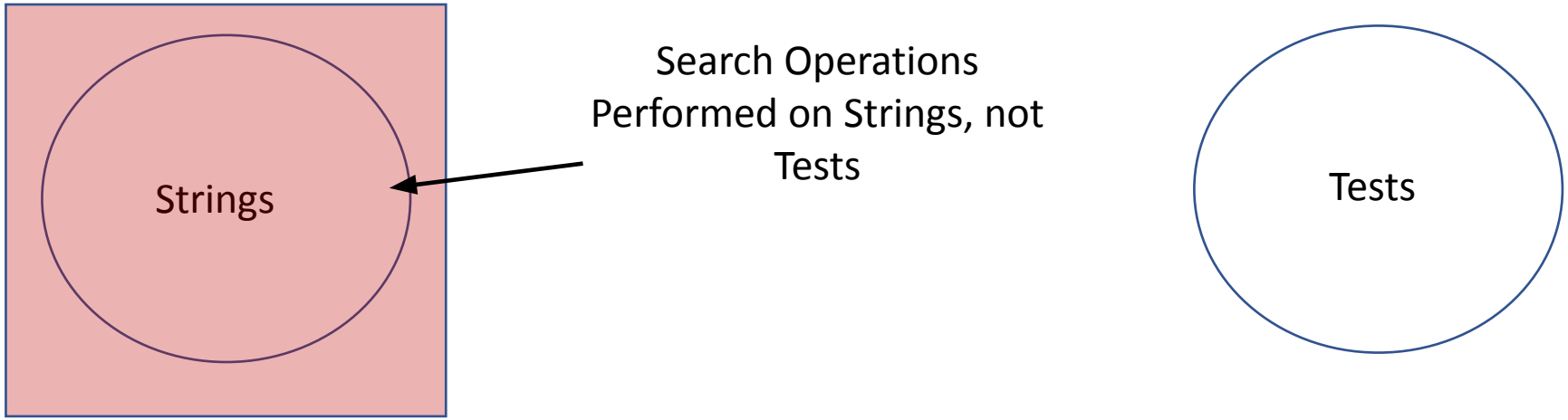
Grammatical Evolution

Domain Flexibility



Grammatical Evolution

Domain Flexibility



Summary

Grammatical Evolution can help SBTG overcome some of it deficiencies.

Incorporate domain knowledge in the grammar.

Allow the user to easily specify the structure of complex objects.

Flexibility of changing domain as the search is conducted on an individuals strings, not the tests themselves.

Questions?