# Lecture 12 — Automated Testing II

David J. Pearce

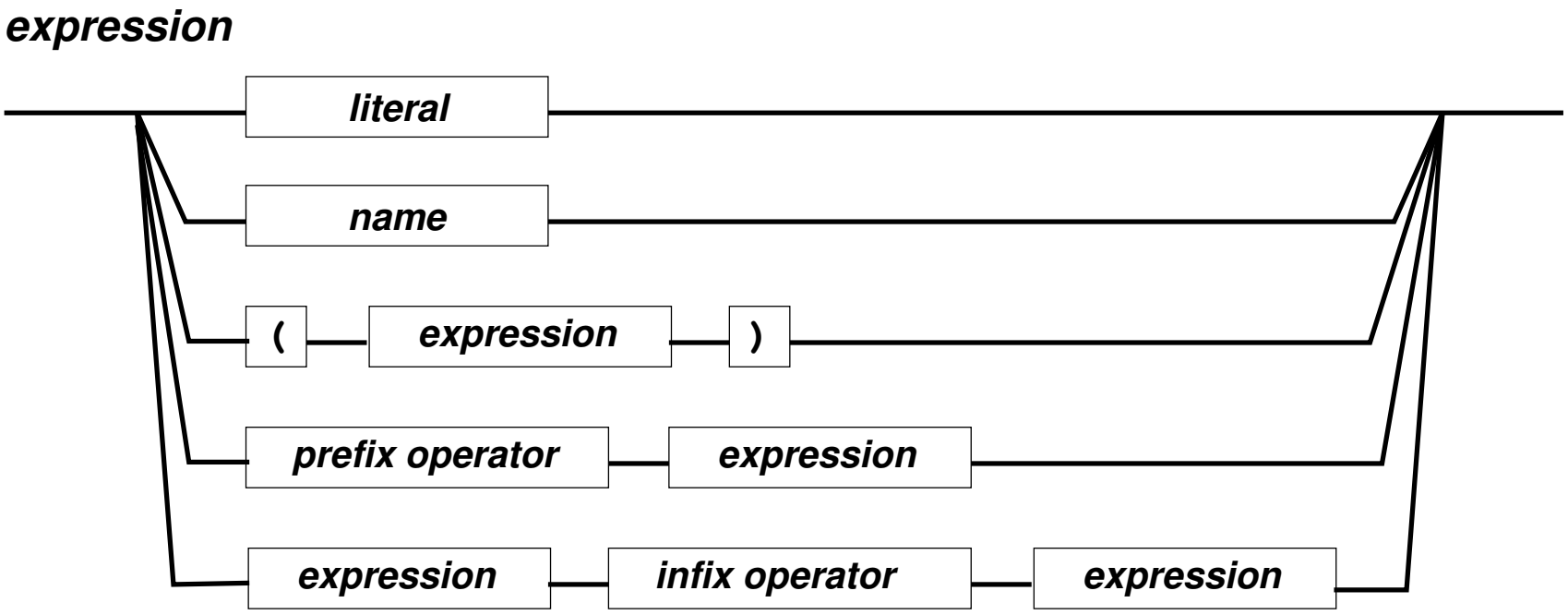
*School of Engineering and Computer Science*
*Victoria University of Wellington*

# Fuzz Testing

"*In fuzz testing, we can roughly distinguish between two techniques:* **Generative** *approaches try to create new random input, possibly using certain constraints or rules.* **Mutative** *approaches try to derive new testing inputs from existing data by randomly modifying it.*"

–Holler *et al.*

# **JavaScript Interpreter**: Generating Inputs



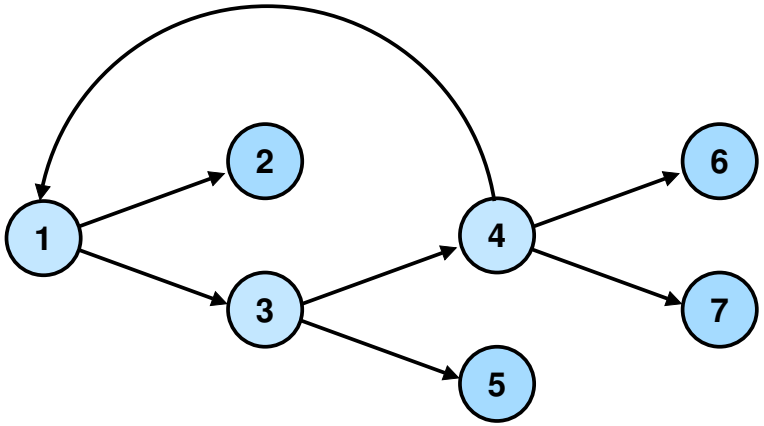

- **Q)** *How to generate programs from JavaScript grammar?*

- **A)** Use a **random walk**!

# Random Walks

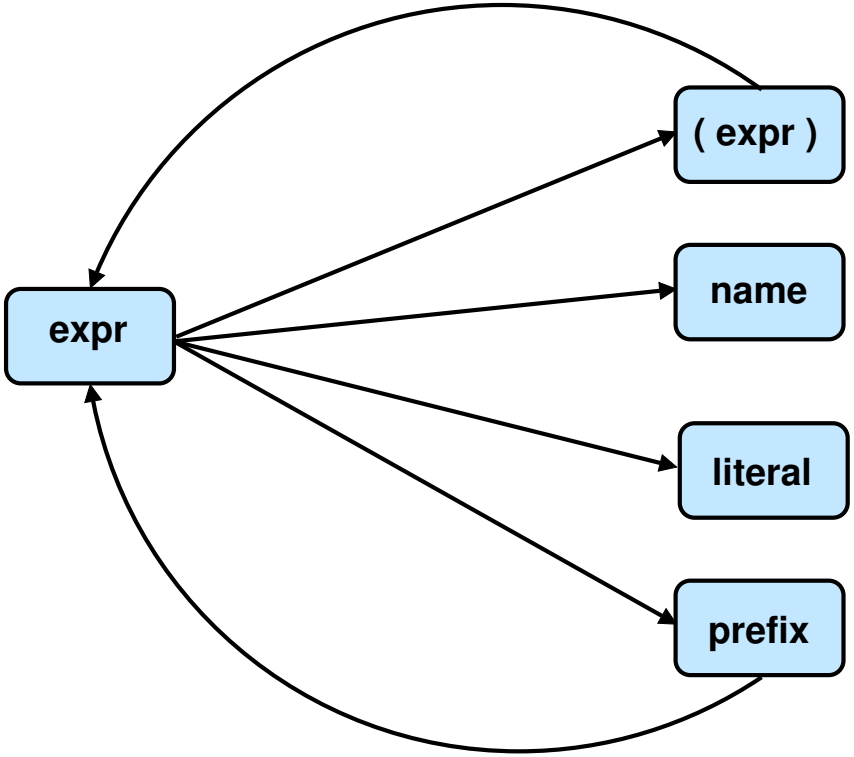
"*A* **random walk** *is a mathematical object, known as a stochastic or random process, that describes a* **path** *that consists of a succession of* **random steps** *on some mathematical space such as the integers.*"

–Wikipedia




- Assume some **probability** for each edge (e.g. evens)
- Starting from root, randomly **choose edge** until **terminal reached**

# **Grammar-Based** Generation




- *How long does random walk take to* **terminate?**
- *What can we do to* **speed** *this up?*

# **Limitations** of Grammar-Based Generation

```
function f(a) {
  var s = 0;
  y = e + 1;
  return sum;
}
```

- Easy to generate random **literals**, but what about **names**?
- Why is this a **problem** for generating valid programs?
- Could generate programs and **discard** invalid ones. **Bad idea?**
- What **else** could we do?

# **Mutation-Based** Testing

"*Mutation testing ... is used to design new software tests and evaluate the quality of existing software tests. Mutation testing involves* **modifying a program** *in small ways*"

- Start with a **valid input** ...

... then **mutate** that input in some way

... (ideally in way that **retains** validity)

... and **repeat**!

# **JavaScript Interpreter:** Example Mutations

```
function abs(x){
  if(x >= 0) {
     return x;
   } else {
     return -x;
} }
```

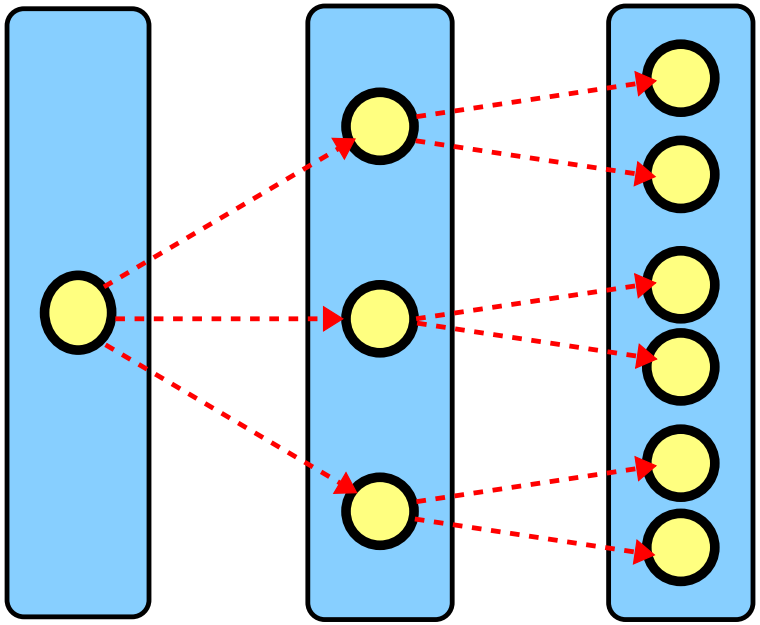
- **Mutation 1:** swap operator (e.g. `>=` ⟶ `<`)
- **Mutation 2:** swap identifier (e.g. `x` ⟶ `y`)
- **Mutation 3:** swap statement (e.g. `if` ⟶ `while`)

# **JavaScript Interpreter:** Operator-Swap Mutation

```
function max(arr){
  var max = arr[0];
  //
  for(var i = 0; i < arr.length; ++i) {
    if(max < arr[i]) {
      max = arr[i];
    }
  }
  return max;
}
```

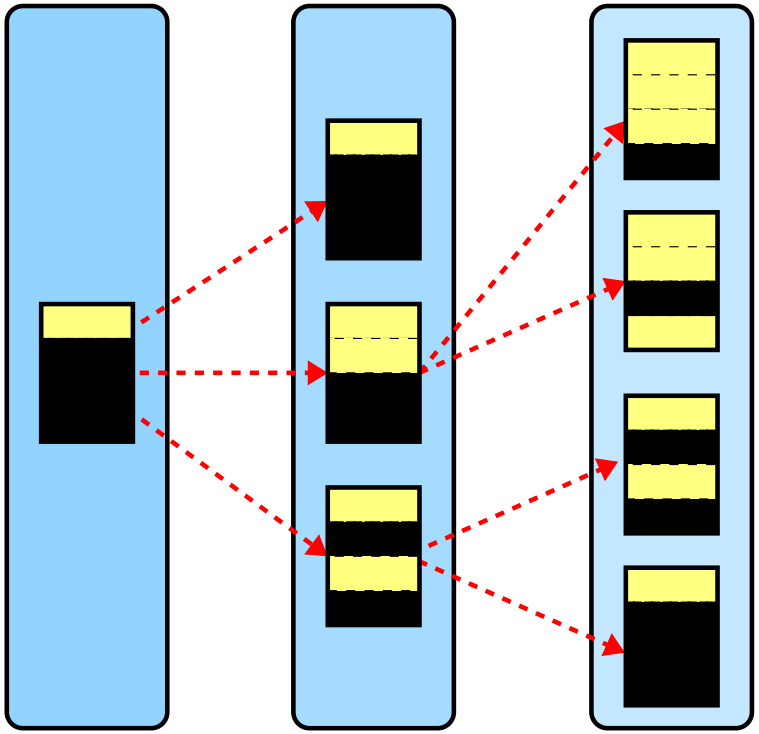
- *Are operator swaps always* **validity preserving?**
- *Are name swaps always* **validity preserving?**
- *Are statement swaps always* **validity preserving?**

# Feedback-Directed Random Testing



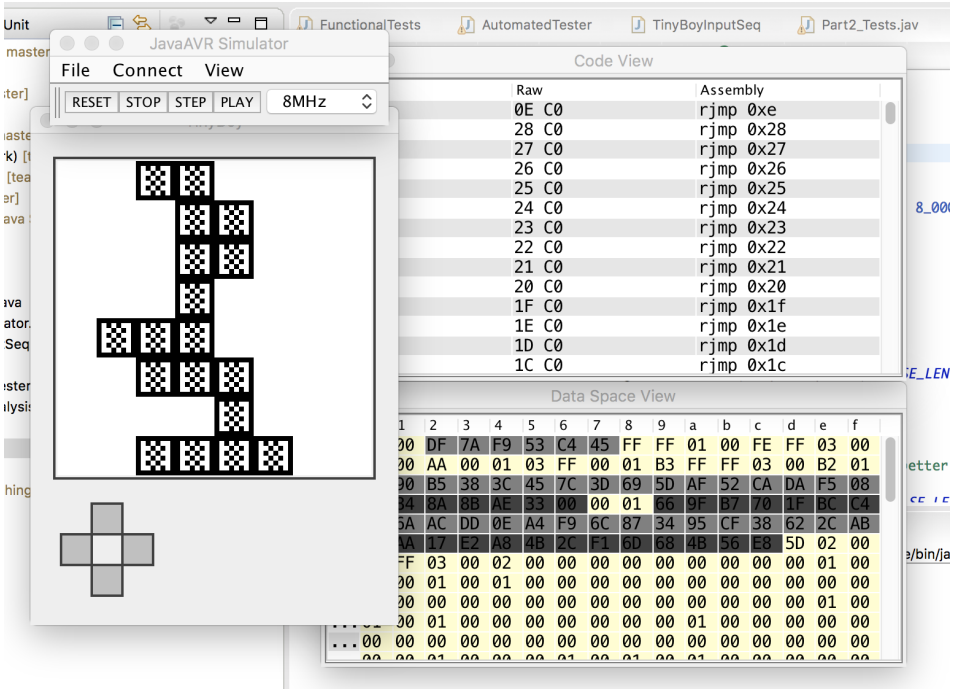
- Begin with one (or more) **seed inputs**
- **Randomly mutate** inputs to produce **more inputs**
- **Minimise** inputs using some criteria (**why?**)

# Feedback-Directed Random Testing (Example)



- Assume **instruction coverage** as feedback criteria
- **Discard** generated inputs which do not **increase coverage**
- This helps progress towards **greater coverage**

# **Exercise:** Fuzz Testing AVR Emulator

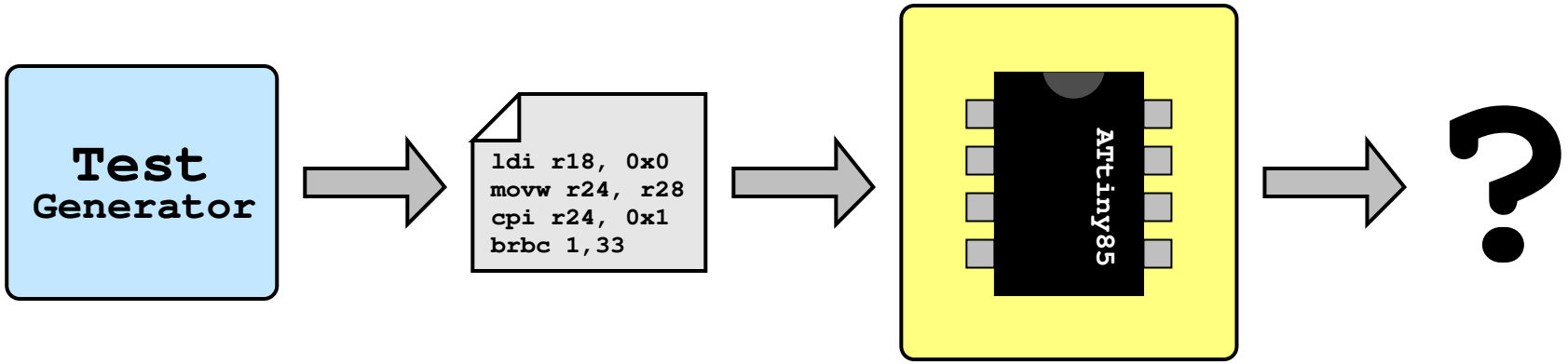



- Suppose want to fuzz test our AVR emulator

  ... *How would we go about this?*

  ... *What things might we look out for?*

# **Exercise:** Fuzz Testing JavaAVR Emulator




- JavaAVR written in **Java**
- What should our **test oracle** be?
- What Java exceptions are **definite errors**?

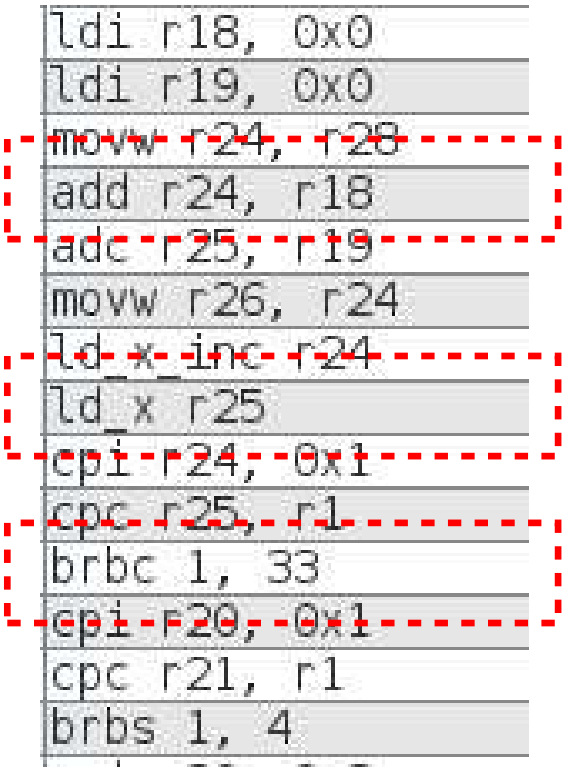
# **Exercise:** Fuzz Testing AVR Emulator

- **Valid inputs** should have ...

  ... **registers** which are defined and

  ... **memory Accesses** only on valid and defined addresses

  ... **branches** which go somewhere sensible



# Further Reading

- “Fuzzing with Code Fragments”, Holler, Herzig, Zeller. USENIX’12
- “JCrasher: an automatic robustness tester for Java”, Csallner and Smaragdakis. *Software Practice and Experience*, 2004.
- “Model-Based Testing of Automotive Systems”
- “Feedback-directed Random Test Generation”, Pacheco *et al.*, ICSE, 2007.
- “jFuzz: concolic whitebox testing for Java”, NFM’09.