POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Tesi di Laurea Magistrale in
Computer Science and Engineering
Ingegneria Informatica

Title of the thesis

Author:
Name Surname

Student ID:
xxxxxx

Advisor:
Prof. Name Surname

Academic Year:
20xx-xx

Dedicated to my family.

Abstract

Here goes the abstract.

Keywords: key, words, go, here

Abstract in lingua italiana

Qui va inserito l'abstract in italiano.

Parole chiave: qui, vanno, le, parole, chiave

Contents

1 | Introduction

2 State of the art

3 | Background

4 | Solution design

5 | Evaluation

5.1. Goals

5.2. Conditions

5.3. Baseline

In this section, we describe the baseline we use to evaluate the efficiency of fuzzers generated by RLC. As a baseline, we need a simpler method of automatically generating fuzzers for game descriptions. Any such method needs to establish an abstraction for how a game is described. We have chosen to use the abstraction of OpenSpiel, Google DeepMind's framework for applying reinforcement learning methods to games (TODO: I probably need to describe OpenSpiel in greater detail.).

The OpenSpiel repository includes some example games. We describe a simple method of generating fuzz targets for these games. These games, modified minimally to introduce bugs the fuzzer can find, form the baseline of our evaluation.

Generating fuzzers for OpenSpiel games

(TODO: I adapted this from the OpenSpiel paper, should figure out how to cite that.) Games in OpenSpiel are described as producedural extensive-form games. Where a game has:

- A finite set of players. Including a special player representing chance.
- A finite set of all possible actions players can take in all game states.
- A finite set of histories. Each history is a sequence of actions that were taken from the start of the game.
- A finite set of terminal histories that represent a finished game.
- A utility for each player for each terminal history.

- A player assigned to take the next action for each non-terminal history. Including a special player representing simultaneous states, where players act simultaneously choosing a joint action.
- A set of states, where each state is a set of histories such that histories in the same state can not be distinguished by the acting player.

Complete game descriptions are objects that implement some interface methods exposing the elements described above, along with some utility methods to simplify moving from a history to its successors. In particular, OpenSpiel game descriptions implement the following methods that are useful for generating a fuzzer:

- `Game::NewInitialState`
- `State::isTerminal`
- `State::isChanceNode`
- `State::isSimultaneousNode`
- `State::LegalChanceOutcomes`
- `State::LegalActions`
- `State::ApplyAction`

Depending only on these functions, we can generate a fuzz target analogous to the ones we generate for RL descriptions as shown in Algorithm 5.1, where *pickOne* is a function that picks an element of the given set consuming the next $log_2(n)$ bits of the fuzz input. After generating the fuzz target, we plug the fuzz target into LLVM's libFuzzer to generate the complete fuzzer.

Algorithm 5.1 Fuzzing and OpenSpiel game

```
 1: $state \leftarrow game.NewInitialState()$
 2: while $state.isTerminal()$ do
 3:     if $state.isChanceNode()$ then
 4:         $nextAction \leftarrow$ pickOne($state.LegalChanceOutcomes()$)
 5:     else if $state.isSimultaneousNode()$ then
 6:         $actions \leftarrow []$
 7:         for $player \in players$ do
 8:             $actions.append($pickOne$(state.LegalActions(player)))$
 9:         end for
10:         $nextAction \leftarrow ApplyAction(actions)$
11:     else
12:         $nextAction \leftarrow$ pickOne($state.LegalActions()$)
13:     end if
14:     $state.ApplyAction(nextAction)$
15: end while
```

It should be noted that OpenSpiel games may have two kinds of chance nodes. Explicit stochastic chance nodes expose multiple legal actions, as well as a probability distribution over those actions. On the other hand, sampled stochastic chance nodes expose a single action with non-deterministic behaviour. This fuzz target is only suitable for games with no samples stochastic chance nodes since the fuzz target has to be deterministic with respect to the fuzz input.

Tic tac toe

5.4. Results

6 Conclusion

Bibliography

List of Figures

List of Tables

Acknowledgements

Here you may want to acknowledge someone.