Can you write TicTacToe better than reinforcement learning engineers?

Massimo Fioravanti - Politecnico di Milano - github.com/rl-language/rlc

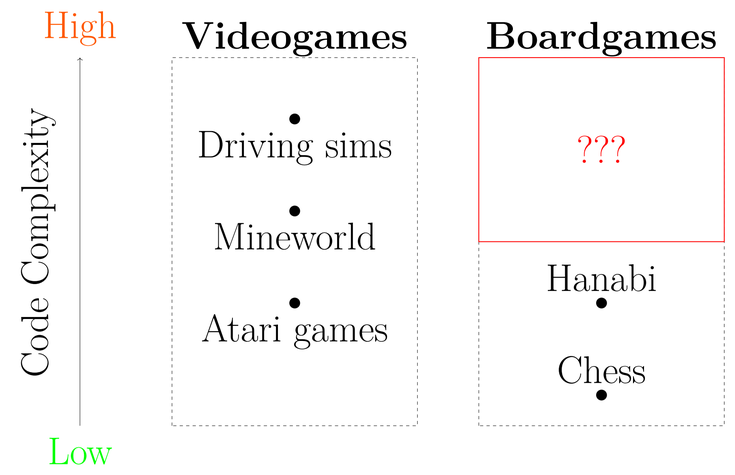
Where are the sequential decision making datasets?

Reinforcement learning environments (games, simulations, ...) requires:

- **State serializable to tensor** to be able to train networks
- **Efficient implementations** to speed up training
- **Suspendable execution** to wait for actor actions
- **Precondition checkable** to reject invalid actions

If you cannot use coroutines or different processes, you must write a class. The code complexity explodes quadratically with respect to the number of suspension points. General purpose languages **cannot** combine coroutines and metaprogramming.

```
class TicTacToe:
  def __init__(self):
    self.board = TicTacToeBoard()
    self.current_player = 0
    self.next_resumption_point = NormalTurn
    self.winner = None

  def is_done(self) -> Bool:
    return self.next_resumption_point == Ended

  def can_mark(self, x: int, y, int) -> Bool:
    return self.next_resumpion_point == NormalTurn and
           x >= 0 and x < 3 and y >= 3 and y <= 3 and
           not self.board.is_set(x, y)

  def mark(self, x: int, y: int):
    assert self.can_mark(x, y)
    self.board.set_marked_by_current_player(x, y)

    if self.board.three_in_a_line():
      self.next_resumption_point == Ended
      self.winner = self.current_player
    else:
      self.current_player = (self.current_player + 1) % 2
```



Example: while multiple reinforcement learning datasets offer videogames of various complexity, real world high code complexity board games datasets are almost non existant. This is surprising since board games for machine learning require no graphical engine and thus seems easier to implement, but living in another process and sending the video stream trivializes interoperability at a performance cost.

Boardgames require to manually explicitly state the entire each possible suspension point of the game.

Solution: Rulebook, a interoperable MLIR DSL to encode rules only

Requirement	Lang Feature	c	cpp	python	Rulebook(our)
Suspendability	Coroutines	x	✓	✓	✓
Efficiency	Compiled	✓	✓	x	✓
Autoserialization	Reflection	x	Not on coro	Not on coro	✓
Checkability	Preconditions	x	Not on coro	dynamic	✓

```
act play() -> TicTacToe:
 frm board : Board
 while !board.full():
  act mark(BInt<0, 3> x, BInt<0, 3> y) {
      board.get(x.value, y.value) == 0
  }

  board.set(x.value, y.value, board.
   current_player())

  if board.three_in_a_line_player():
      return
  board.next_turn()
```

Using a unique coroutine typechecking scheme, **Rulebook** turns a imperative program into classes you can use in other languages with no overhead. Using **Rulebook** TicTacToe from `C++` introduces **0 mallocs**. **RLC architecture on page 2.**

Results

Rulebook VS Google OpenSpiel	TicTacToe	Hanabi	Checkers	Battleship	Catch	Connect Four
Speedup VS CPP	0.81×*	1.9×	3.49×	188×*	1.2×*	1.05×
Relative Loc VS CPP (ignoring headers)	0.8×	0.11×	0.4×	0.14×	0.55×	0.8×

* in these benchmarks, differences between OpenSpiel and our (optional) undo mechanism dominates other timings.

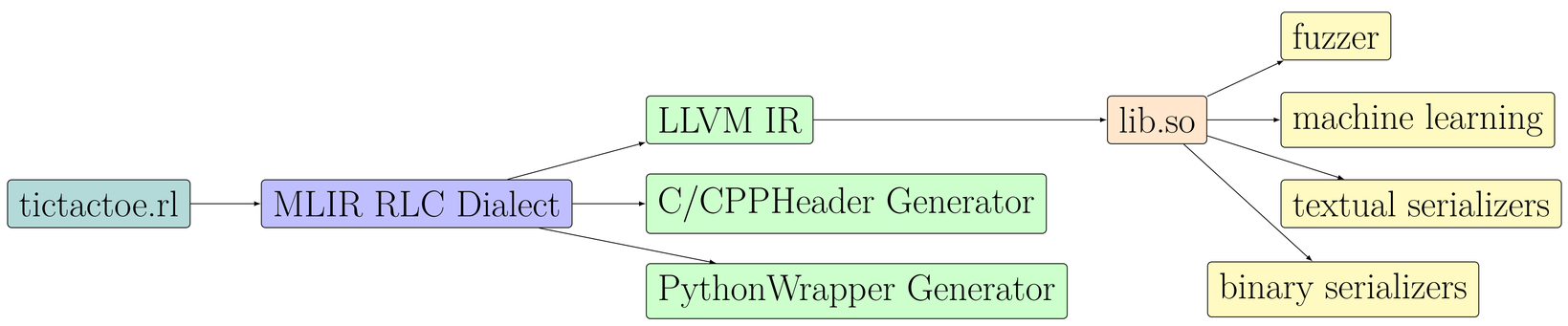
bidirectional interop CPP	✓
bidirectional interop Python	✓
UnrealEngine	✓
Godot Engine	✓
LSP and autocomplete support	✓
syntax highlighting	✓
windows	✓
linux	✓
macOS	✓
WebAssembly	✓
PIP installable package	✓
Fuzzer	✓

We have the first and only digital implementation of **Warhammer 40.000** core rules, a **95 percentile** boardgame in **rule complexity** on bbg.com, which rules (no graphic or data code) only took **~2200 lines of Rulebook Code**.
Can run on dektop, in the browser, with or without the engine. Can be fuzzed, delivered to the GPU, serialized textually.

Can you write TicTacToe better than reinforcement learning engineers?

Massimo Fioravanti - Politecnico di Milano - github.com/rl-language/rlc

Pipeline



Coroutine to classes

```
act play() -> TicTacToe:
 frm board : Board
 while !board.full():
  act mark(BInt<0, 3> x, BInt<0, 3> y) {
      board.get(x.value, y.value) == 0
  }

  board.set(x.value, y.value, board.current_player())

  if board.three_in_a_line_player():
      return
  board.next_turn()
```

- **Rewrite play as a coroutine**

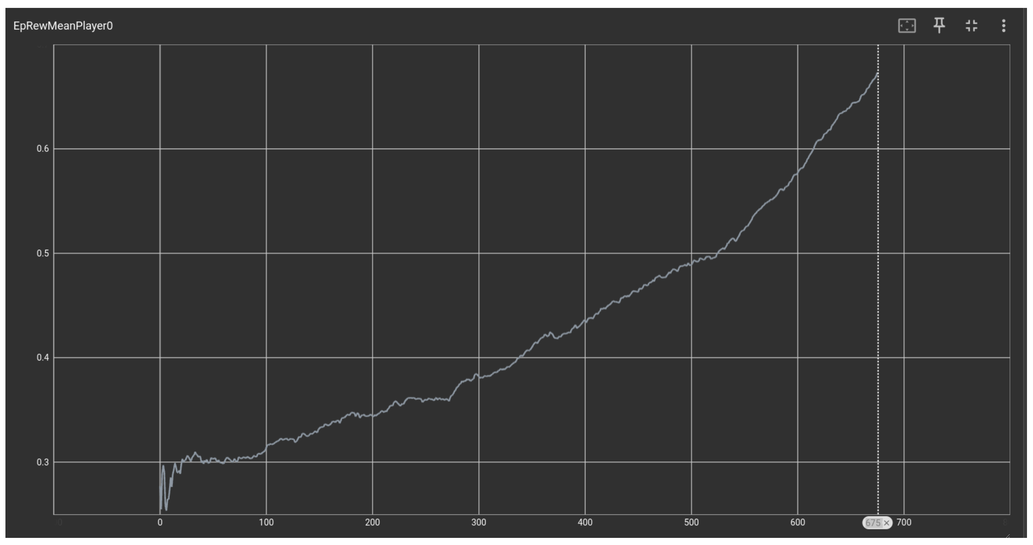
```
[always_inline]
fun _play_impl(TicTacToe coro_state, Args args):
 switch coro_state.resume_index [0, begin] [1, after_mark]
 begin:
  cbr !coro_state.board.full() loop, end

 loop:
  board.coro_state.resume_index = 1
  ret

 after_mark:
  coro_state.board.set(args.x, args.y, coro_state.board.
    current_player())
  cbr coro_state.board.three_in_a_line_player() end
  coro_state.board.next_turn()
  br loop

 end:
  board.coro_state.resume_index = -1
  ret
```

- **Emit class, inline coroutine, optimize**

```
class TicTacToe:
 Int resume_index = 0
 Board board = {}

 fun can_mark(Int x, Int y) -> Bool:
  return self.resume_index == 0 and self.board.get(x, y) == 0

 fun mark(Int x, Int y):
  begin:
   self.board.set(x, y, self.board.current_player())
   cbr self.board.three_in_a_line_player() end
   self.board.next_turn()
   board.self.resume_index = 1
   ret

  end:
   board.self.resume_index = -1
   ret

fun play() -> TicTacToe:
 let board : TicTacToe
 begin:
  cbr !coro_state.board.full() loop
  board.coro_state.resume_index = -1
  ret board

 loop:
  board.coro_state.resume_index = 1
  ret board
```

- **What is the catch?** Strong typechecking limitations. Mutually recursive coroutines impossible.

rlc-learn tic_tac_toe.rl -o /tmp/net



Using coroutines

```
fun main() -> Int:
    let game = play()
    game.mark(0, 0)
    game.mark(1, 0)
    game.mark(0, 1)
    game.mark(1, 1)
    game.mark(0, 2)
    return game.is_done()
```

Composition

```
fun alternate_tic_tac_toe() -> TicTacToeTwice:
    let g1 = play()
    let g2 = play()
    while not g1.is_done() and not g2.is_done():
        subaction g1
        subaction g2
```

rlc-play tic_tac_toe.rl /tmp/net

```
# game: 0
mark {x: 0, y: 2}
mark {x: 1, y: 0}
mark {x: 2, y: 2}
mark {x: 1, y: 2}
mark {x: 2, y: 0}
mark {x: 0, y: 1}
mark {x: 0, y: 0}
mark {x: 2, y: 1}
mark {x: 1, y: 1}
mark {x: 1, y: 1}
```

Acknowledgments and Disclosure of Funding

This work is partially supported by the Italian Ministry of Enterprises and Made in Italy (MIMIT) under the program “Accordi per l’innovazione nella filiera del settore automotive”, through the grant "Piattaforma ed ecosistema cooperativo, C- ITS ETSI standard per la mobilità digitale integrata", numero F/340043/01-04/X59, CUP B49J24001210005, finanziato a valere del Bando MISE – ACCORDI PER L’INNOVAZIONE NEL SETTORE AUTOMOTIVE D.M. 31/12/2021 e DD 10/10/2022