Regular Games

General Game Description Language Based on Finite Automata

Radosław Miernik, radoslaw.miernik@cs.uni.wroc.pl
Computational Intelligence Research Group,
Institute of Computer Science, University of Wrocław

General Game Playing

General Game Playing (GGP) aims to design, implement, and analyze artificial intelligence programs in a general way, i.e., **not bound to a single game** (more generally, a single problem). It is considered a necessary step towards AGI.

While some GGP environments focus on **API-level generalism** (where the agent can play all games using a fixed API interface), most of them use a **formal game description language** of varying complexity levels (where the agent can play all games describable using this language).

Arguably, the most popular GGP language is Game Description Language (GDL), created at Stanford University in 2005 [1], describing all **finite**, **deterministic**, **turn-based** games with **perfect information**.

Regular Boardgames

The objective of Regular Boardgames (RBG) [2, 3] was to join the key GGP properties, such as **expressiveness**, **efficiency**, and **naturalness**, in one formalism, while compensating certain drawbacks of the existing languages.

This makes RBG more suitable for various research and practical developments in GGP. While dedicated mainly to describing human-playable board games, **RBG covers the same class of games as GDL**, often with significantly shorter and easier to understand definitions.

RBG was the first GGP language that allowed efficient encoding and playing games with complex rules and a large branching factor (e.g., Amazons, Arimaa, large Chess variants, Go, International Checkers, Paper Soccer).

Regular Games

Regular Games (RG) is meant to supersede RBG by being an even more **expressive** and **performant** GGP language. Most importantly, it is designed to describe games with **imperfect information**, e.g., Poker.

The main difference is that it no longer defines the game as a **regular expression** but as a **finite automaton** instead. Such a paradigm change results in new challenges but also a wide array of game-independent improvements.

There is also a "High Level RG" (HRG) language. However, contrary to "High Level RBG", it is an **entirely different language** – not a set of extensions. It is meant to look more like a general programming language, where the underlying automaton is more of an implementation detail.

Interpreter Performance

Variant	No optimizations	All optimizations
`.hrg`	48.60 ± 1.26	28.24 ± 0.34
`.hrg-rf`	47.22 ± 0.74	29.16 ± 0.44
`.rg`	52.60 ± 1.02	39.68 ± 0.56
`.rbg`	388.7 ± 0.38	54.07 ± 0.78
`.kif`	458.4 ± 16.9	192.8 ± 7.81

Average TicTacToe simulation time (in microseconds).

While the translations allow us to run GDL and RBG games using RG's interpreter or compiler, the paradigm change come at a price.

The `-rf` variant of the HRG game version uses the `--reuseFunctions` option to produce a smaller automaton at a cost of (potentially) more variables.

Automaton Size

Variant	No optimizations	All optimizations
`.hrg`	92 : 87	54 : 49
`.hrg-rf`	57 : 54	34 : 31
`.rg`	30 : 27	27 : 24
`.rbg`	333 : 322	139 : 126
`.kif`	1260 : 1132	626 : 487

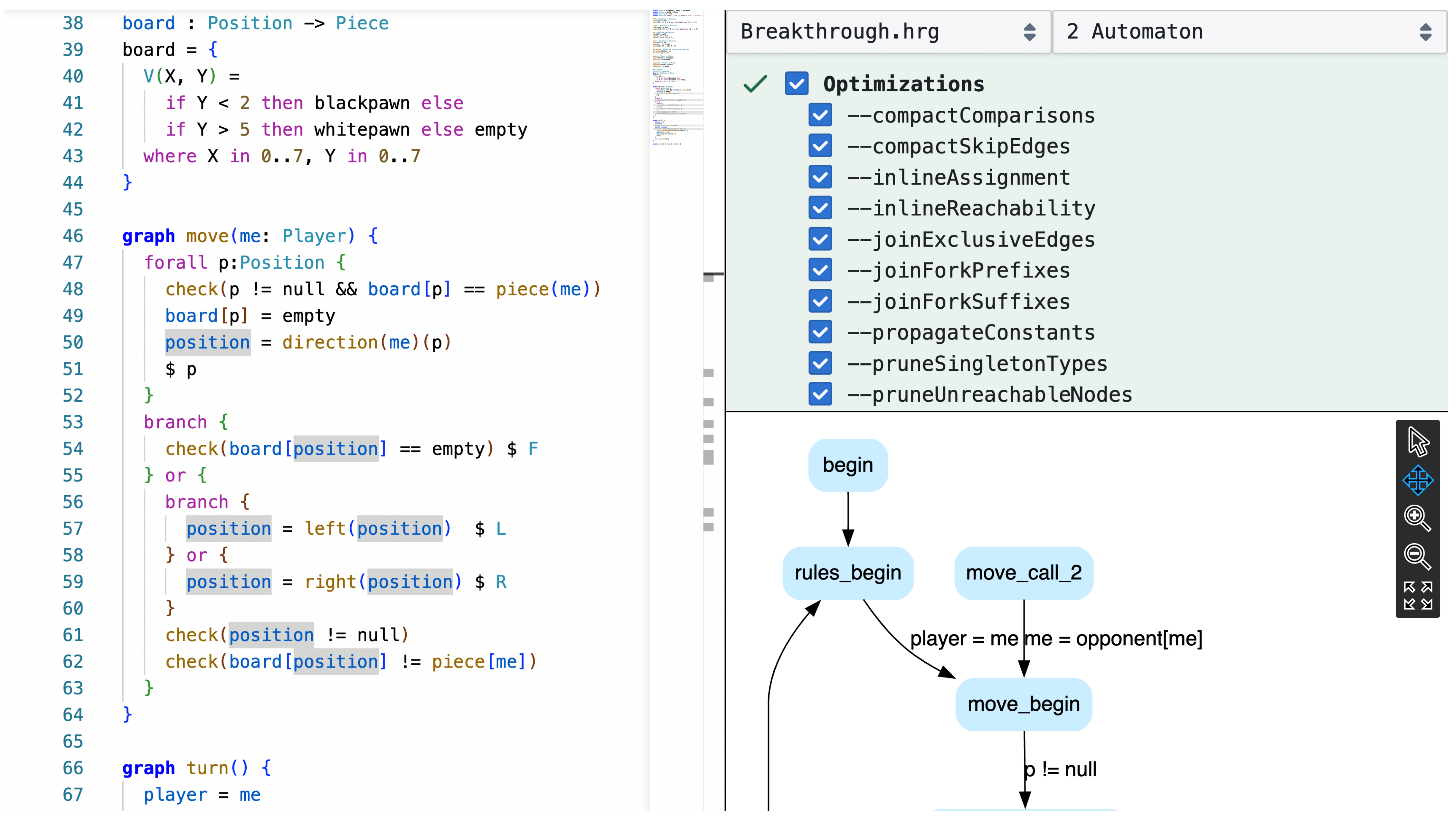
TicTacToe automaton size – number of edges and nodes.

Every translation may be approached differently, resulting in different automaton characteristics, like their size, number of used variables, or expressions complexity.

The RBG translation is based on Thompson's algorithm, i.e., a non-deterministic automaton accepting legal moves.

The GDL translation builds an automaton that performs depth-first search of the legal moves. The construction itself is trivial, but requires *grounding* of all terms in the game definition, which often results in an exponential explosion.

Integrated Development Environment



```
board : Position -> Piece
board = {
  V(X, Y) =
    if Y < 2 then blackpawn else
    if Y > 5 then whitepawn else empty
  where X in 0..7, Y in 0..7
}

graph move(me: Player) {
  forall p:Position {
    check(p != null && board[p] == piece(me))
    board[p] = empty
    position = direction(me)(p)
    $ p
  }
  branch {
    check(board[position] == empty) $ F
  } or {
    branch {
      position = left(position)  $ L
    } or {
      position = right(position) $ R
    }
    check(position != null)
    check(board[position] != piece[me])
  }
}

graph turn() {
  player = me
```

Breakthrough.hrg
2 Automaton

- [x] Optimizations
  - [x] --compactComparisons
  - [x] --compactSkipEdges
  - [x] --inlineAssignment
  - [x] --inlineReachability
  - [x] --joinExclusiveEdges
  - [x] --joinForkPrefixes
  - [x] --joinForkSuffixes
  - [x] --propagateConstants
  - [x] --pruneSingletonTypes
  - [x] --pruneUnreachableNodes

A fully-fledged Integrated Development Environment (IDE) is **available at radekmie.dev/rg**. The code editor on the left implements most features available in commercial IDEs, like syntax highlighting, type hints, and symbol renaming.

The right panel begins with translation settings (e.g., which optimizations should be applied), a game benchmarking tool, and an automaton visualization. All options are applied immediately and work on-the-fly, **right in the user's browser**.

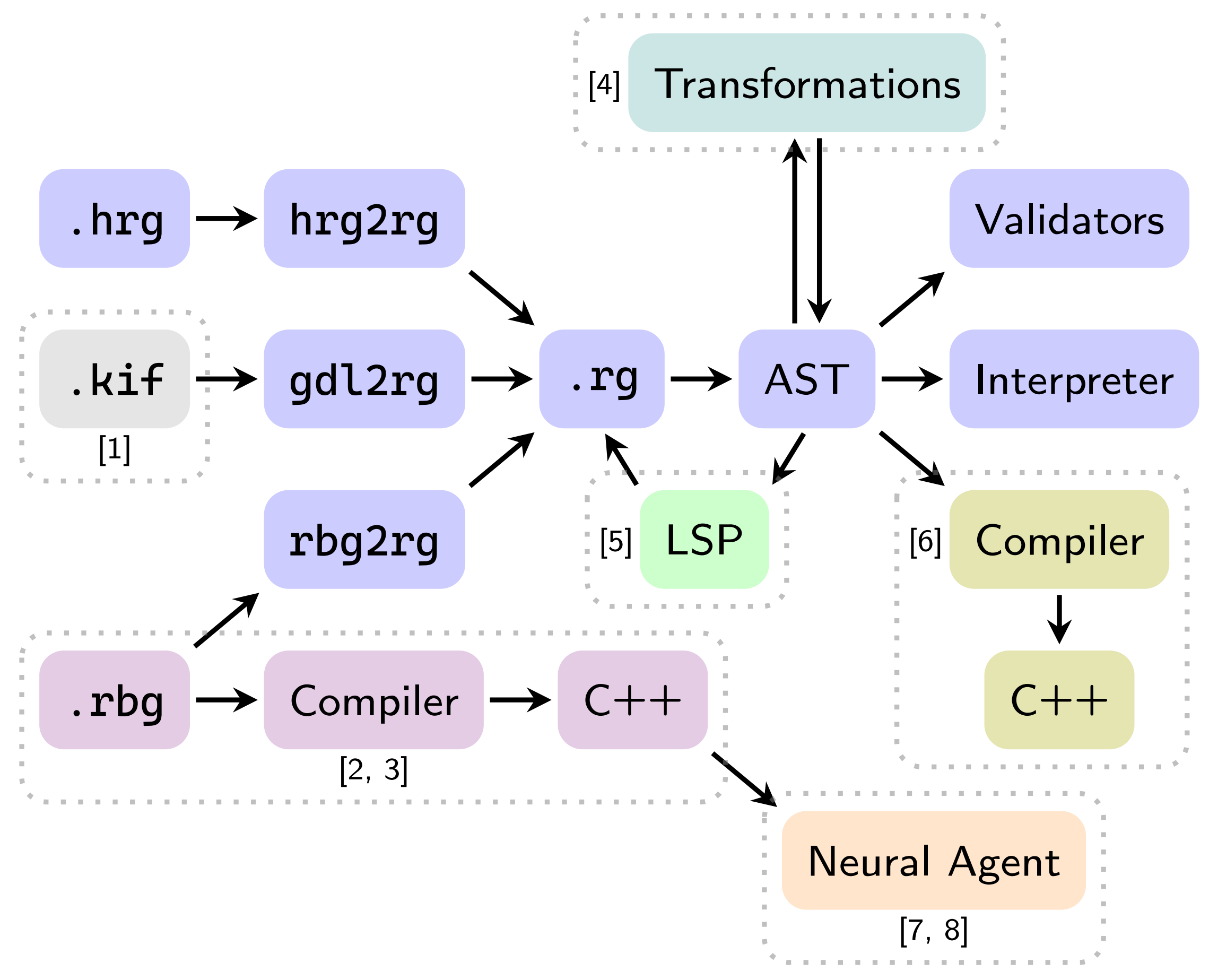
Breathrough.hrg

```
domain Piece = blackpawn | empty | whitepawn
domain Player = white | black
domain Score = 0 | 100
domain Position = null | V(X,Y) where X in 0..7, Y in 0..7

// Other directions look similar...
left : Position -> Position
left(null) = null
left(V(X, Y)) = if X == 0 then null else V(X - 1, Y)

direction : Player -> Position -> Position
direction(white) = up
direction(_) = down

piece : Player -> Piece
piece(white) = whitepawn
piece(_) = blackpawn

opponent : Player -> Player
opponent(white) = black
opponent(_) = white

me : Player
position : Position
board : Position -> Piece
board = {
  V(X, Y) =
    if Y < 2 then blackpawn else
    if Y > 5 then whitepawn else empty
  where X in 0..7, Y in 0..7
}

graph move(me: Player) {
  forall p:Position {
    check(p != null && board[p] == piece(me))
    board[p] = empty
    position = direction(me)(p)
    $ p
  }
  branch {
    check(board[position] == empty) $ F
  } or {
    branch {
      position = left(position)  $ L
    } or {
      position = right(position) $ R
    }
    check(position != null)
    check(board[position] != piece[me])
  }
}

graph turn() {
  player = me
  move(me)
  board[position] = piece(me)
  player = keeper
  if direction(me)(position) == null ||
     not(reachable(move(opponent(me)))) {
    goals[me] = 100
    goals[opponent(me)] = 0
    end()
  }
  me = opponent(me)
}

graph rules() {
  loop {
    turn()
  }
}
```

Related Work

Regular Games started with an informal language specification, basic type checker, and a naive interpreter. Over time, the project attracted multiple students to extend it in their theses, building a compiler, a Language Server Protocol server, and numerous optimization passes.

Thanks to **gdl2rg** and **rbg2rg**, we are no longer limited to native games – all games written in GDL and RBG can be automatically translated (with some overhead).



References

[1] Michael Thielscher. A General Game Description Language for Incomplete Information Games. *Proceedings of the AAAI Conference on Artificial Intelligence*, 24(1):994–999, 2010.

[2] Jakub Kowalski, Maksymilian Mika, Jakub Sutowicz, and Marek Szykuła. Regular Boardgames. *AAAI Conference on Artificial Intelligence*, 33(1):1699–1706, 2019.

[3] Jakub Kowalski, Radosław Miernik, Maksymilian Mika, Wojciech Pawlik, Jakub Sutowicz, Marek Szykuła, and Andrzej Tkaczyk. Efficient Reasoning in Regular Boardgames. In *IEEE Conference on Games*, pages 455–462, 2020.

[4] Jakub Cieśluk. Optimizations in Regular Games. Master's thesis, University of Wrocław, (in progress).

[5] Jakub Cieśluk. IDE for Regular Games. Engineer's thesis, University of Wrocław, 2024.

[6] Łukasz Galas and Wojciech Pawlik. Optimizations in Regular Games. Master's thesis, University of Wrocław, (in progress).

[7] Michał. Maras, Michał Kępa, Jakub Kowalski, and Marek Szykuła. Fast and Knowledge-Free Deep Learning for General Game Playing (Student Abstract). *AAAI Conference on Artificial Intelligence*, 38(21):23576–23578, 2024.

[8] Michał. Maras and Michał Kępa. Enhancing self-play learning of a general agent for Regular Boardgames. Master's thesis, University of Wrocław, (in progress).