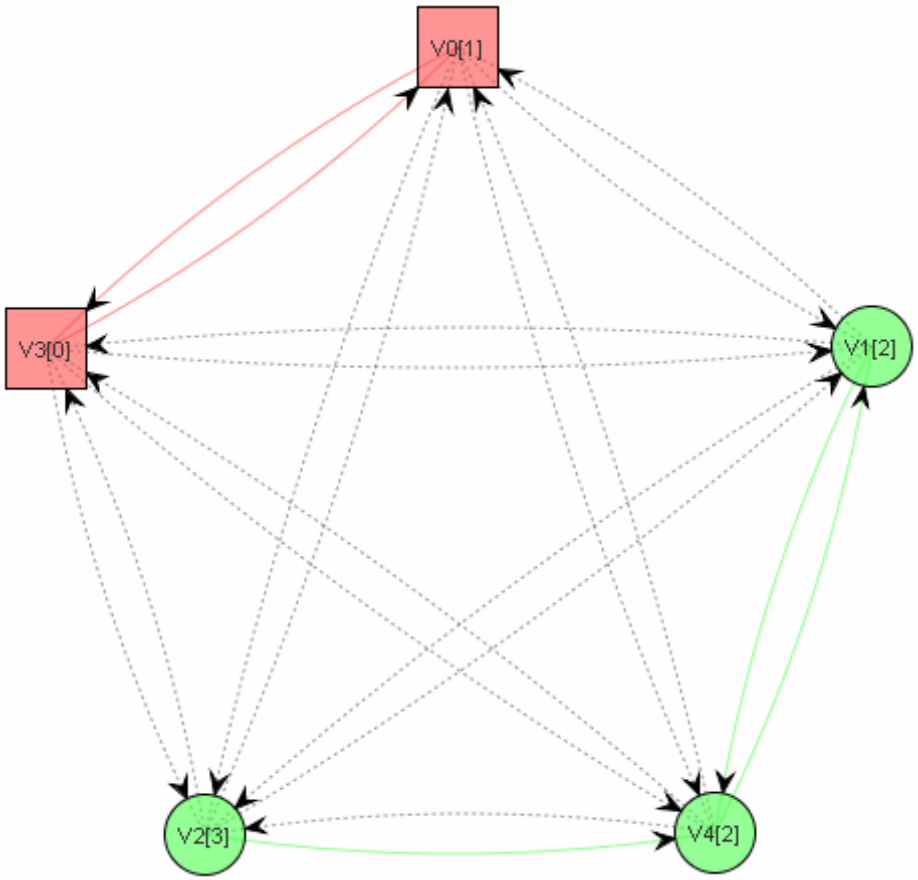
Imperial College London

Workbench for preprocessor design and evaluation with parity games

Michael Huth, Nir Piterman, Huaxin Wang

Parity Games

- Players: $p \in (0, 1)$
- Vertices: $v \in V_0 \cup V_1$
- Edges: $e \in V \times V$
- Priorities: $|v| \in (1 \ldots n)$
- Memoryless strategies



Parity Games

- Model checking
- Synthesis problem
- Automata determinization

- Memoryless result
- UP $\cap$ co-UP
- Sub-EXP
- Many other solvers
- No benchmark consensus
- Hardness of games not consistent across solvers

Our goals

- Maintain a collection of interesting games
- Find universally hard games
- Compare power of preprocessors
- Able to use algebra to quickly retrieve witness games by specifying computational features we want to see in the game
- And to ask questions like:
  - Which of preprocessors p and q is more powerful with respect to a partial solver S?
  - Can S reduce the residual computed by S any further? Etc.

Algebra

- $P ::= a \mid P;P \mid P^{+} \mid f(P)$
  - Atom preprocessor
  - Composition
  - Closure
  - Function lifting

- Requirements
  - $res(G,P) \subseteq G$
  - $v \notin res(G,P) \rightarrow P$ decides v correctly
  - res(G,P) consistent with G
  - $|v|_{res(G,P)} \leq |v|_{G}$

Algebra

- P ::= a | P;P | P^+ | f(P)
  - Atom preprocessor
  - Composition
  - Closure
  - Function lifting

- a1, remove priority gaps
- a2, mark lax vertex when its priority does not make any impact in any path through it
- a3, lossy but consistent compression of priority intervals into 3 distinct values, and solve index-3 game

Algebra

- P ::= a | P;P | P^+ | f(P)
  - Atom preprocessor
  - Composition
  - Closure
  - Function lifting

- In general
  - P;P $\neq$ P
  - P;Q =,<,> Q
  - P;Q $\neq$ Q;P

Imperial College London

Algebra

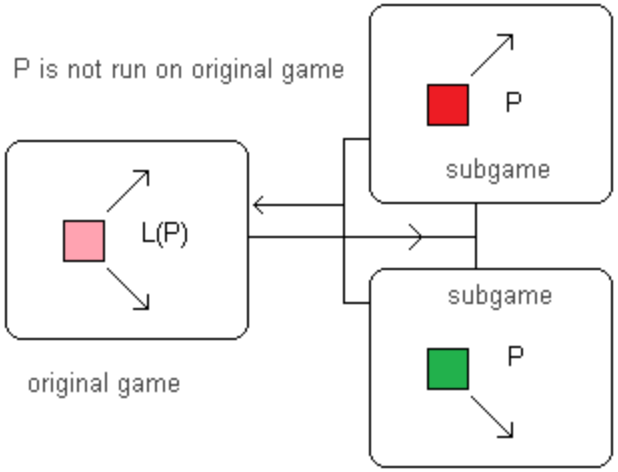
- $P ::= a \mid P;P \mid P^{+} \mid f(P)$
  - Atom preprocessor
  - Composition
  - Closure
  - Function lifting

- $P^{+} = P;P;P\ldots\ldots$

Algebra

- P ::= a | P;P | P^+ | f(P)
  - Atom preprocessor
  - Composition
  - Closure
  - Function lifting

- $f(P) \neq P$
- Many such functions can be defined
- We introduced one in the paper:



Query Language

```
QueryLanguage := QueryType GVar : Constraints
QueryType := ALL | SOME
Constraints := (Constraint) | (NOT Constraints) |
               (Constraints AND Constraints) | (Constraints OR Constraints)

Constraint := Fragment == Fragment | Number >= Number | Number <= Number |
              Number > Number | Number < Number | Nodes SUBSET Nodes |
              Edges SUBSET Edges | Colors SUBSET Colors
Fragment := Game | Nodes | Edges | Number | Colors
Game := GVar | RESIDUAL(Game, Prep)
Nodes := SOLUTION(Game, Prep, Player) | NODES(Game)
Edges := EDGES(Game)
Number := COUNT(Nodes) | COUNT(Edges) | COUNT(Colors)
Colors := COLORS(Game)
Player := X | Y

Prep := Atom | Prep; Prep | L(Prep) | P(Prep)
Atom := A1 | A2 | A3 | EXP
```

Imperial College London

Query Language

- Examples
  - ALL g : ((SOLUTION(g, P, X) == SOLUTION(g, EXP, X) AND (SOLUTION(g, P, Y) == SOLUTION(g, EXP, Y))), sanity check on solver P, whether it is correct (returning the same result as EXP)
  - ALL g : (NODES(RESIDUAL(g, A1;A2;A3)) SUBSET NODES(RESIDUAL(g, A2;A3))) , whether A1A2A3 is more powerful than A2A3, the answer is no, same when they are lifted
  - ALL g : (NODES(RESIDUAL(g, C(L(L(A2;A3))))) SUBSET NODES(RESIDUAL(g, EXP))), whether closure of A2A3 lifted twice can solve all games in the database completely

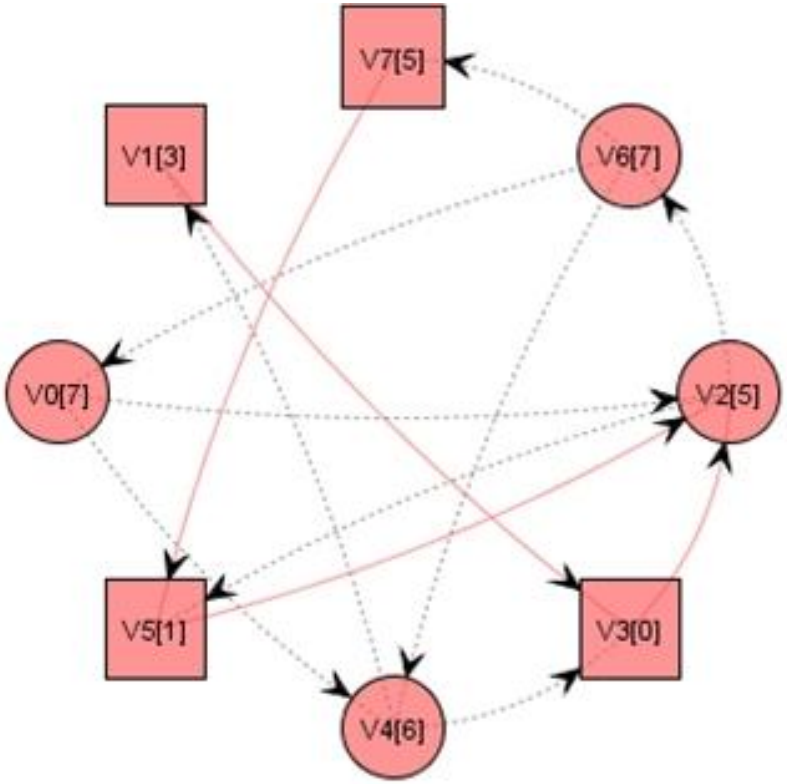
Query Language

- Interpretations
  - Variable range over available games defined in the database
  - Definitive results:
    - Positive witness from existential queries
    - Negative witness from universal queries
  - Other results returned without witness are only correct with respect to the scope of the database

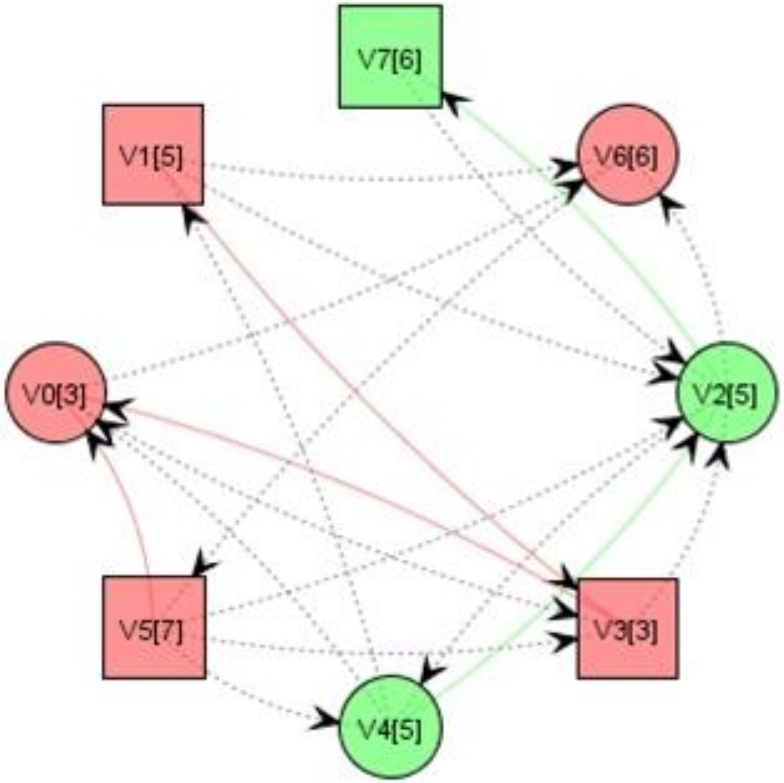
Imperial College London

Example Witness Games

- A2;A3 > L(L(A2;A3))

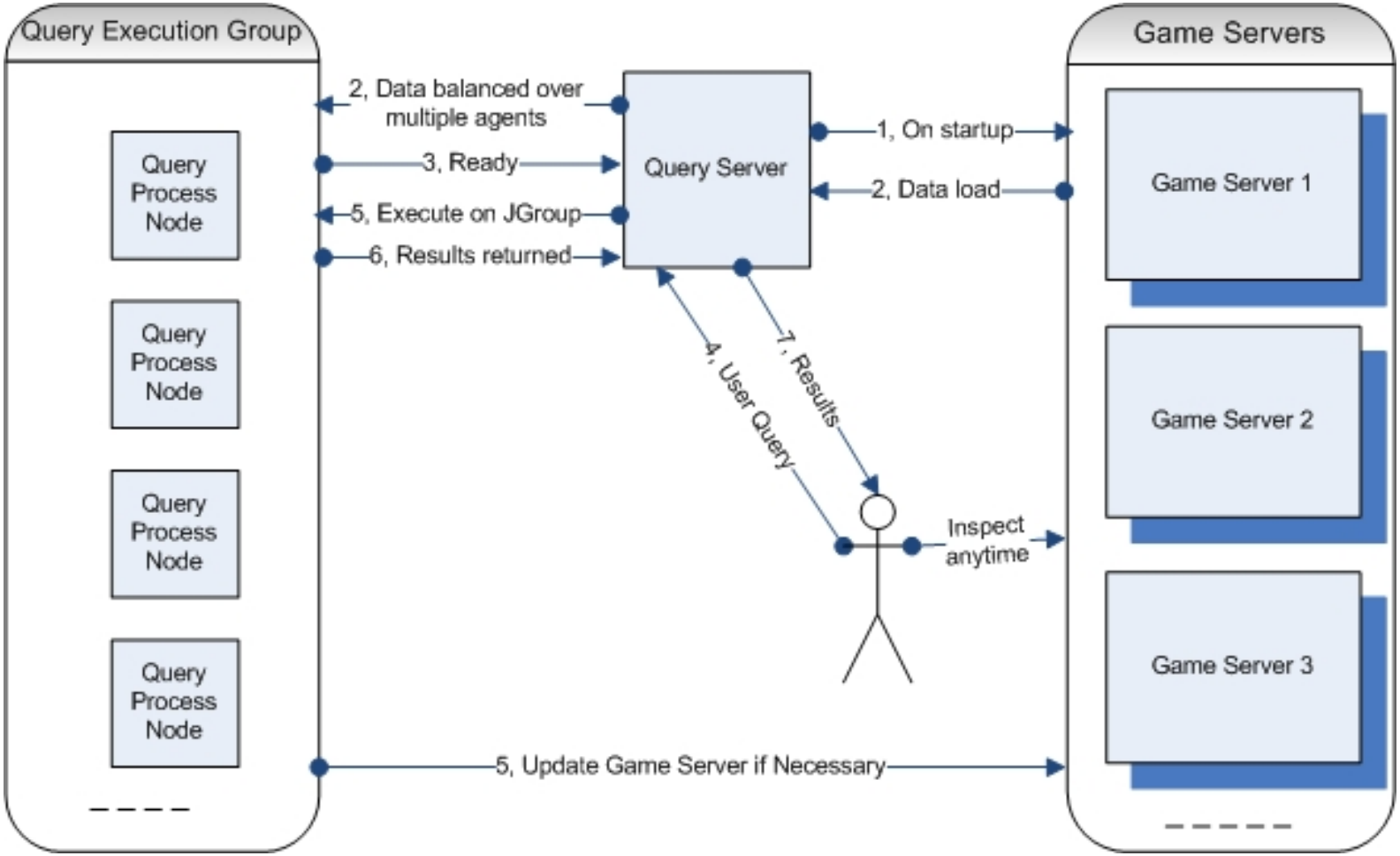


- A2;A3 > A1;A2;A3))



Imperial College London

Architecture



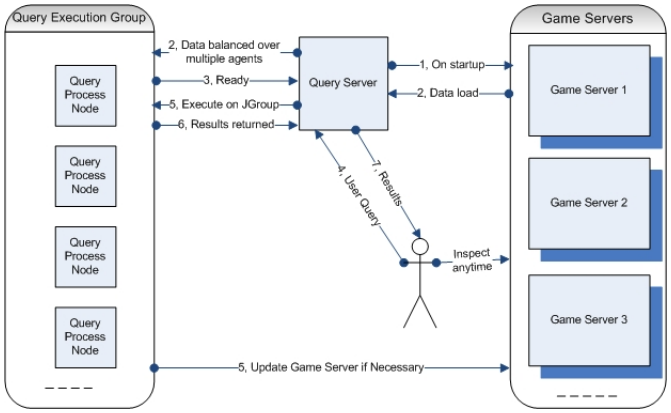
Imperial College London

The database

- Complete permutations of 4-vertices and 5-vertices non-isomorphic unique games $\cong$ 70 million instances
- Thousands of 8, 16, 32, 256 and 1024-vertices randomly generated games
- Collection of games mentioned from various papers on parity games
- Agent generating games on the fly for hypothesis testing
- Small scope hypothesis

Imperial College London

Extension



- Agents are deployed on different machines
- Evaluation is postponed to the agent
- Code drop for agent can happen at runtime
- Instrumentation can be applied to preprocessors as aspects (Aspect Oriented Programming, server is oblivious of these instrumentation code)
- Aspects are installed as predicate, and invoked using reflection by agents
- Usage:
  - Collect performance statistics, runtime, memory foot print
  - Checking of invariant/desired properties with respect to execution of preprocessors
  - Fast and pipelined regression test

Imperial College London

Demo + Q & A

Thank you!

Brought to you by: Michael Huth, Nir Piterman, Huaxin Wang