Verifying SimpleGT Transformations Using an Intermediate Verification Language

Zheng Cheng, Rosemary Monahan, James F. Power

Computer Science Department, Maynooth University, Ireland

VOLT 2015

Model Driven Engineering (MDE) and Model Transformation

Model transformation is an important ingredient in MDE.

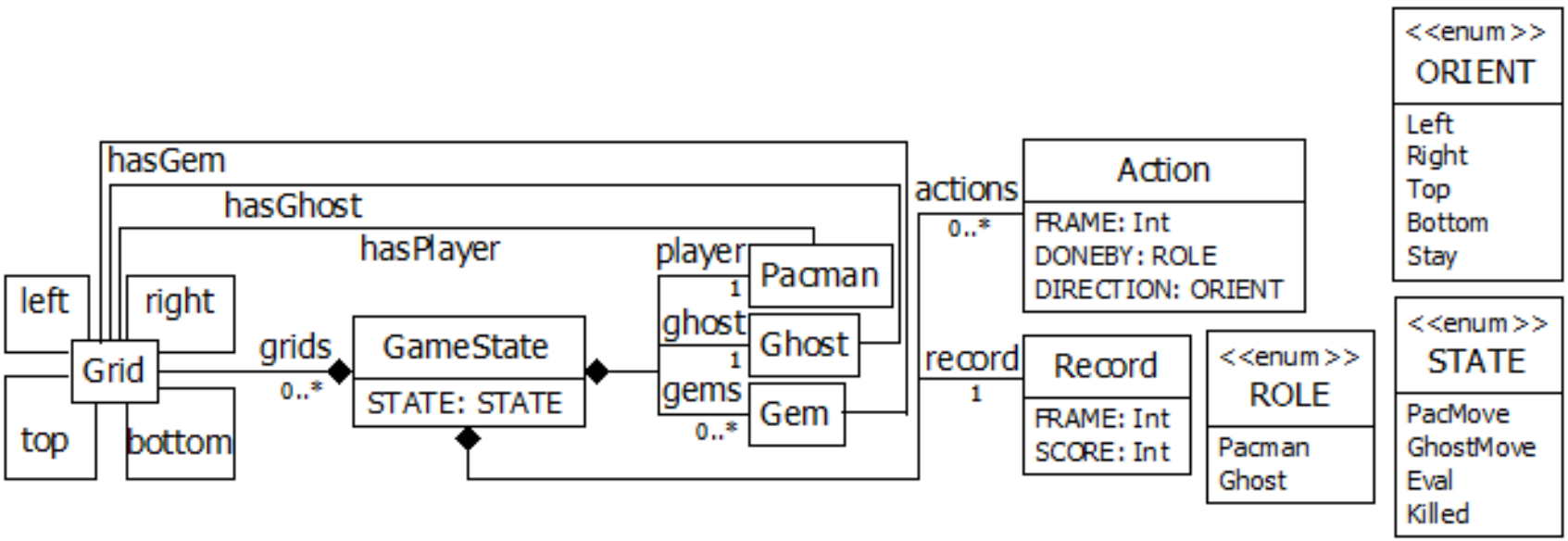
- Operational model transformation, e.g. Kermeta, QVTo.
- Relational model transformation, e.g. ATL, ETL.
- Graph model transformation (GT), e.g. SimpleGT, Henshin.

Model Driven Engineering (MDE) and Model Transformation

Proving the correctness of SimpleGT.

- "Correctness" means assumptions about the GT, given as **contracts**.
- GT is well suited to describe scenarios such as distributed systems or behaviours of structure-changing systems (e.g. mobile networks).

Pacman Metamodel (Syriani and Vangheluwe, 2013)



SimpleGT Transformation

```
rule PlayerMoveLeft{
   from
      s:P!GameState(STATE=~PacMove,record=~rec), rec:P!Record,
      pac:P!Pacman,
      grid2:P!Grid, grid1:P!Grid(hasPlayer=~pac,left=~grid2),
      act:P!Action(DONEBY=~Pacman,FRAME=~rec.FRAME,DIRECTION=~Left)
   not grid2: P!Grid(hasEnemy=~ghost), ghost: P!Ghost
   to
      s:P!GameState(STATE=~GhostMove,record=~rec), rec:P!Record,
      pac:P!Pacman,
      grid2:P!Grid(hasPlayer=~pac), grid1:P!Grid(left=~grid2) } ...
```

OCL Contracts

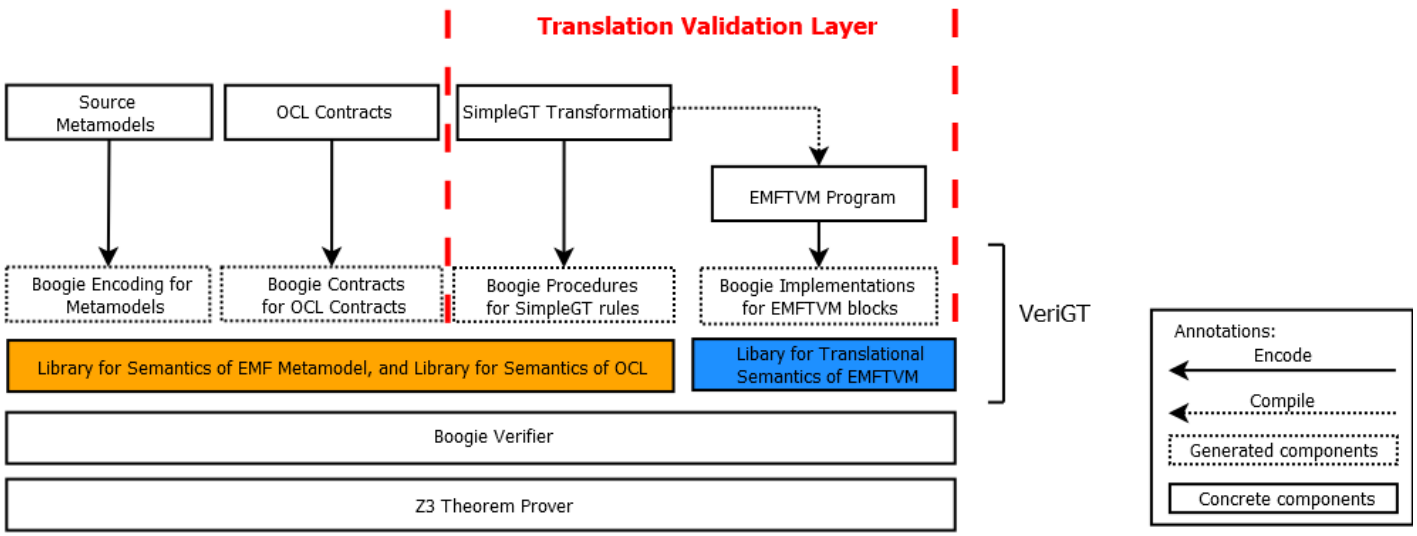
```
... -- well-formatness contracts of the Pacman game.

-- all grids containing a gem must be reachable by Pacman.
context Grid inv gemReachable:
  Grid.allInstances()->forall(g1,g2:Grid|not g1.hasPlayer.isOclUndefined()
    and not g2.hasGem.isOclUndefined() implies reachable(g1,g2))

-- exists a path where the ghost never kills Pacman.
context GameState inv PacmanSurvive:
  self.STATE==GhostMove implies self.grids->forall(
    g1:Grid|g1.hasEnemy.oclIsKindOf(Ghost) implies
      not g1.hasPlayer.oclIsKindOf(Pacman))

-- the Pacman must move within a time interval I.
context Action inv PacmanMoved:
  let col:Sequence=Action.allInstances()->select(
    a:Action|a.DONEBY=Pacman and not a.Direction=Stay)->asSequence() in
      col->forall(i:int|0<=i<col->size()-1 implies
        col->at(i+1).FRAME-col->at(i).FRAME<=I)
```

Objective of Our Research

Design a deductive verifier for the SimpleGT language, which enables its formal verification.

- Investigate whether the Boogie intermediate verification language (IVL) can be used to systematically designing modular and reusable verifiers.
- Investigate whether the translation validation approach from compiler verification can be automated to ensure the soundness of the verification [Cheng et al. 2015].

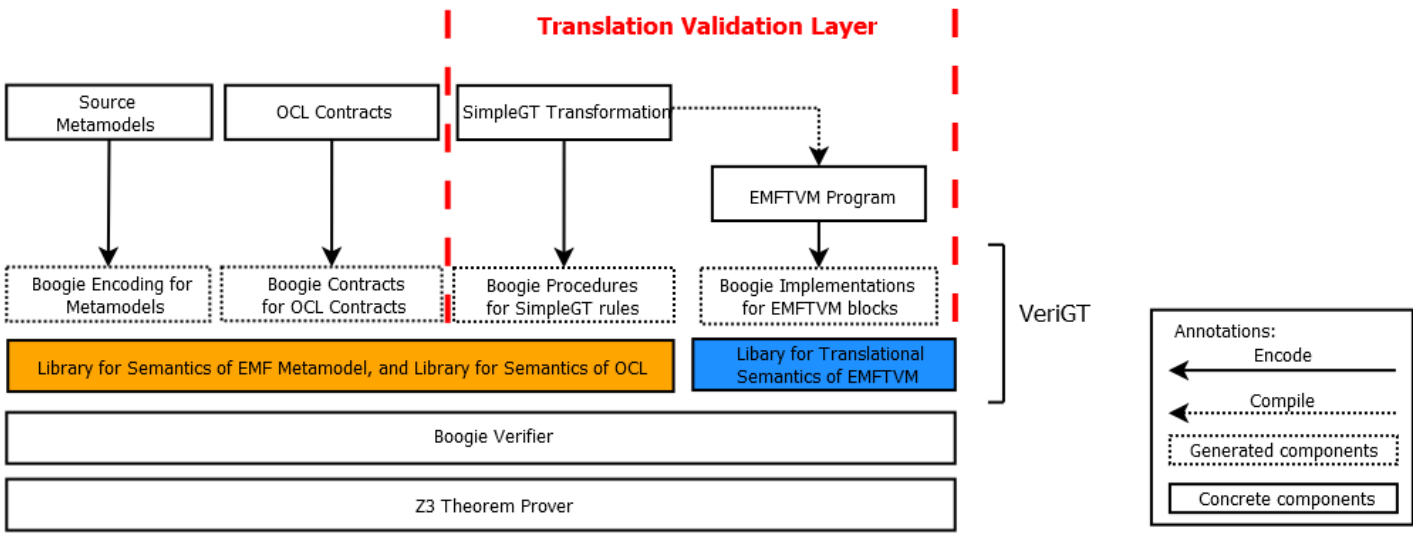
Bird's Eye View of VeriGT



Introduction to Boogie

```
1   procedure McCarthy(n: int);
2     returns (r: int)
3     ensures 100 < n ⟹ r = n − 10;
4     ensures n ≤ 100 ⟹ r = 91;
5
6   implementation McCarthy(n: int);
7   {
8     if (100 < n) {
9       r := n − 10;
10    } else {
11      call r := F(n + 11);
12      call r := F(r);
13    }
14  }
```

Bird's Eye View of VeriGT

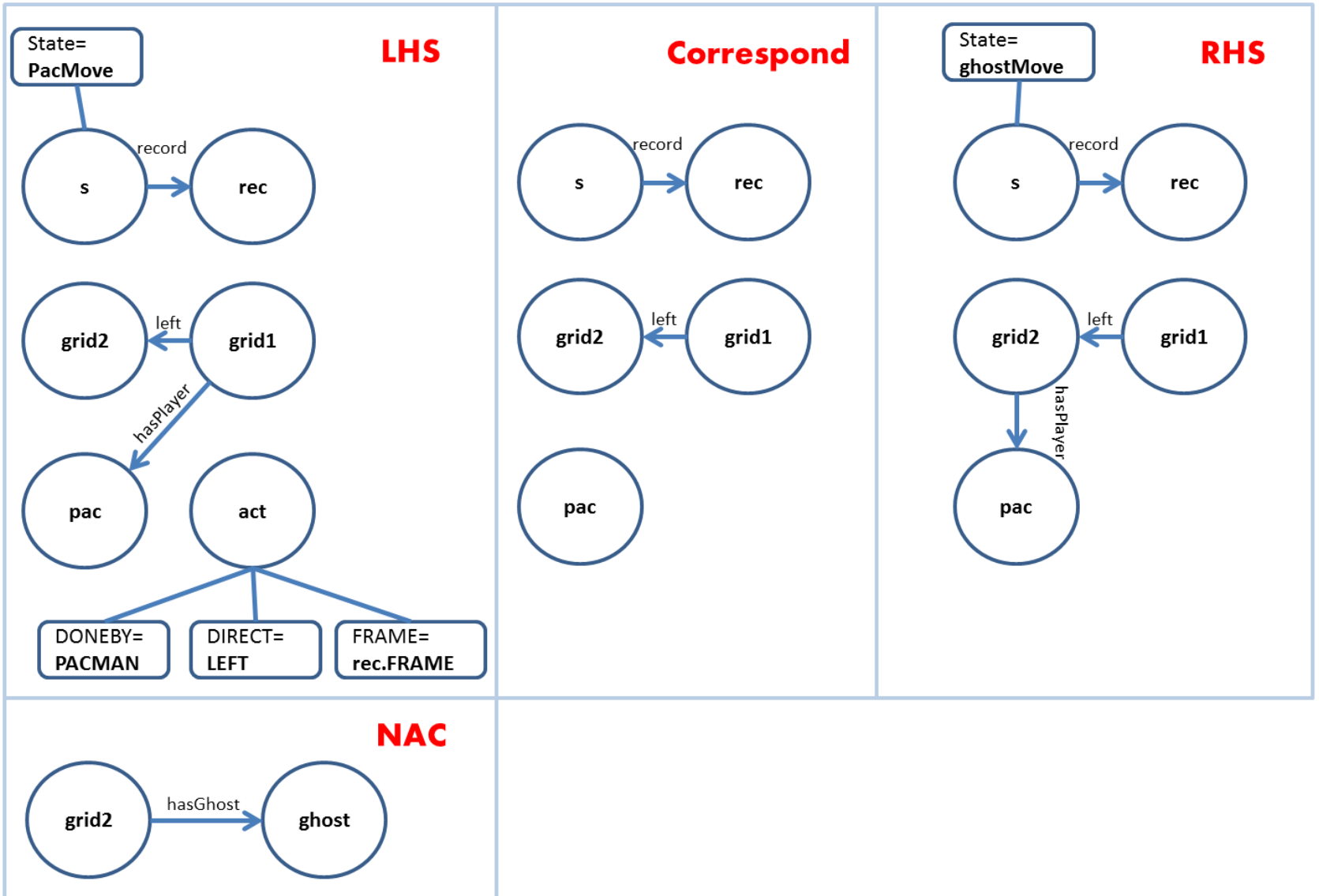


Example: SimpleGT Rule

```
rule PlayerMoveLeft{
   from
      s:P!GameState(STATE=~PacMove,record=~rec), rec:P!Record,
      pac:P!Pacman,
      grid2:P!Grid, grid1:P!Grid(hasPlayer=~pac,left=~grid2),
      act:P!Action(DONEBY=~Pacman,FRAME=~rec.FRAME,DIRECTION=~Left)
   not grid2: P!Grid(hasEnemy=~ghost), ghost: P!Ghost
   to
      s:P!GameState(STATE=~GhostMove,record=~rec), rec:P!Record,
      pac:P!Pacman,
      grid2:P!Grid(hasPlayer=~pac), grid1:P!Grid(left=~grid2) } ...
```

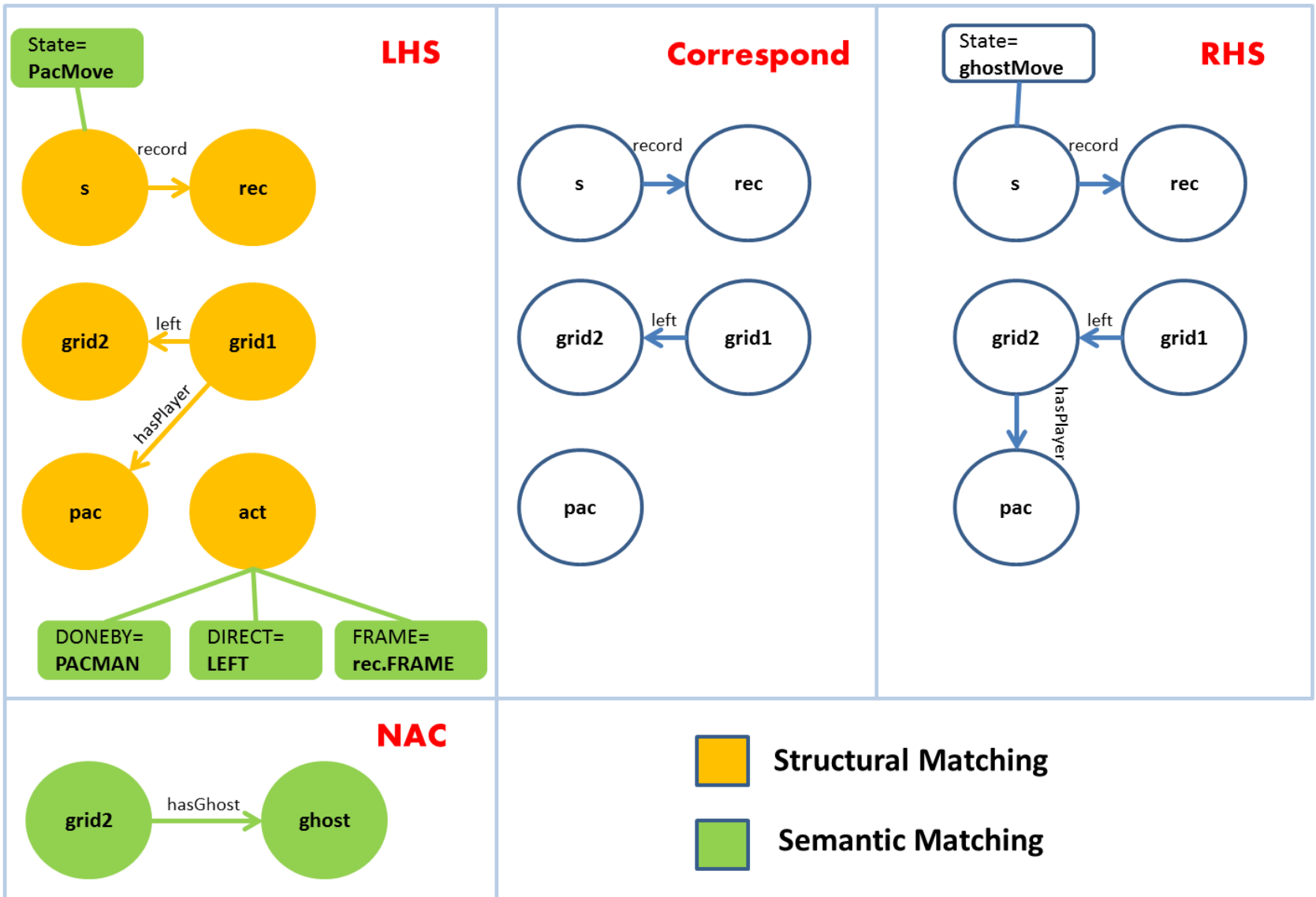
Example: SimpleGT Rule (a Graphical Representation)

PacmanMoveLeft:



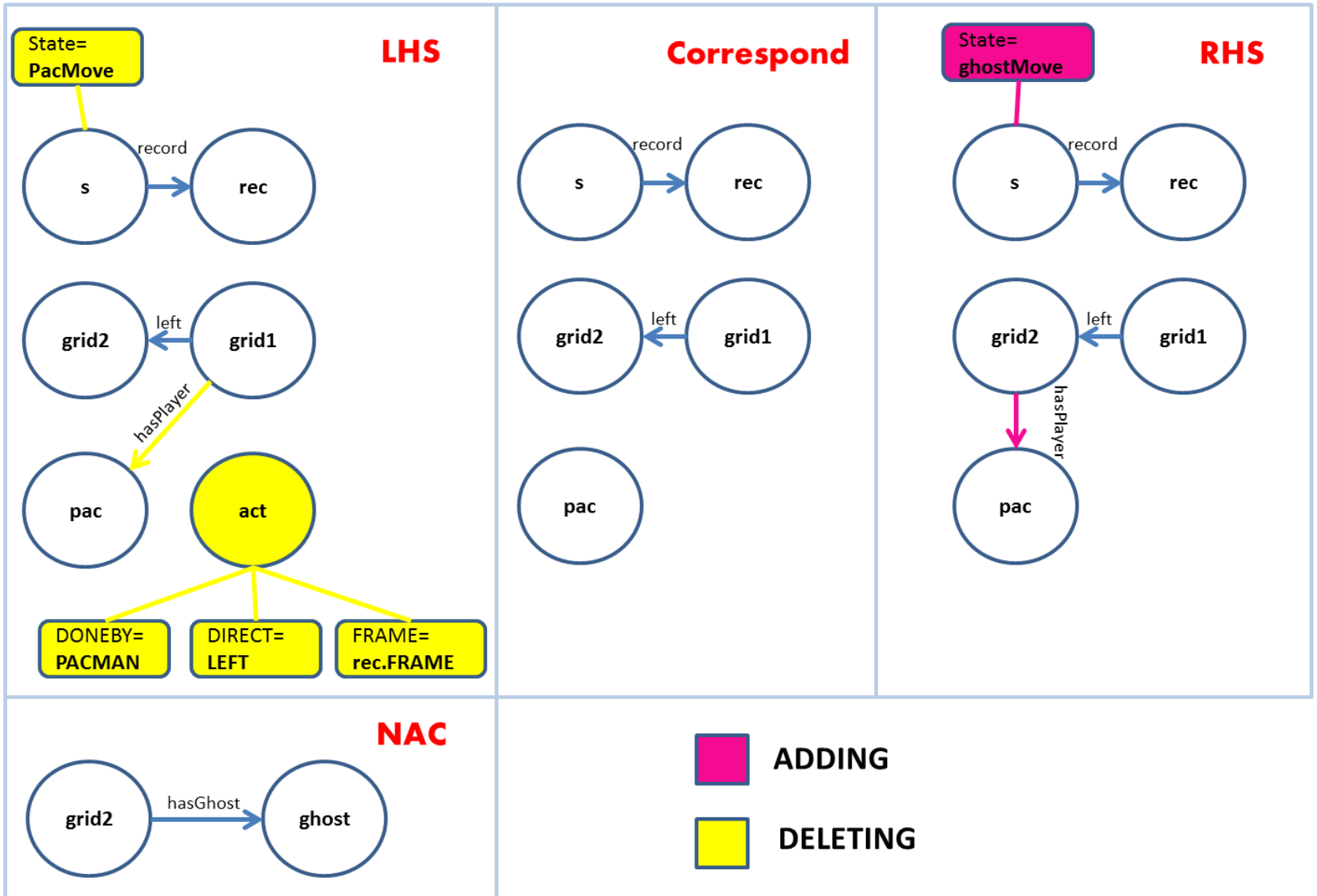
Example: SimpleGT Rule (Match)

PacmanMoveLeft (match step):

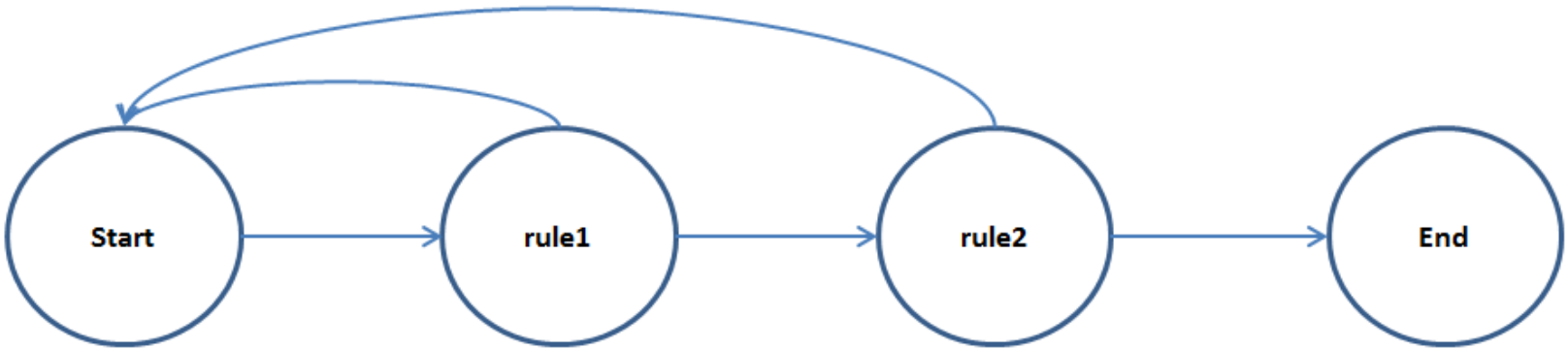


Example: SimpleGT Rule (Apply)

PacmanMoveLeft (apply step):

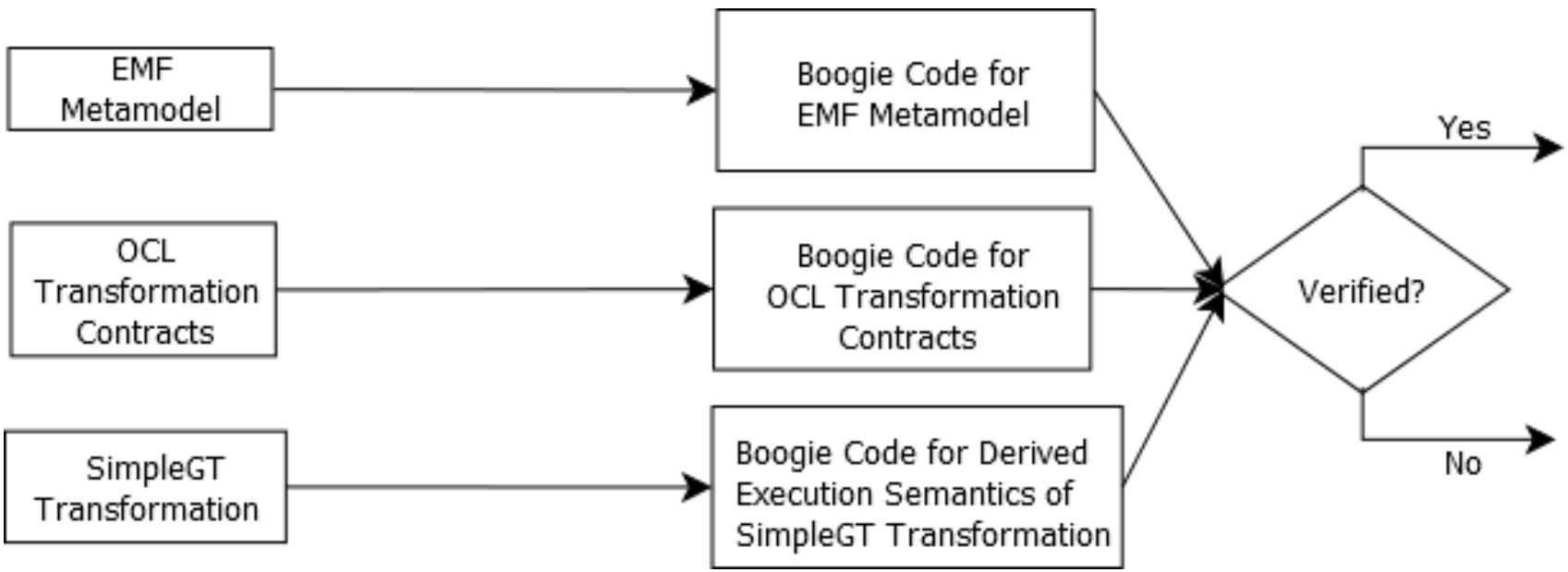


Rule Scheduling



- After each application, the rule scheduling restarts immediately.
- If there is no match for the first rule, it falls-off to match the second rule.

VeriGT in Action



Evaluation

	Boogie (LoC)	Veri. Time (s)
gemReachable	421	0.998
PacmanSurvive	467	1.747
PacmanMoved	439	0.109
Total	1327	2.854

Related Work

- Theorem proving (Unbounded) vs. Simulation/Model Checking/Model Finding (Bounded).
- The soundness of GT verification approaches based on theorem proving.
- Intermediate verification languages, e.g. Why3, Boogie.

Future Work

- Mature implementation of VeriGT integrated with Eclipse.
- Termination verification with VeriGT.

Q & A

Thanks to Maynooth University and reviewers of VOLT.

{zcheng,rosemary,jpower}@cs.nuim.ie

https://github.com/VeriATL/VeriGT