Algorithms and optimisations for parity games applied to Boolean Equation Systems

Jeroen Keiren
j.j.a.keiren@student.tue.nl

Technische Universiteit Eindhoven
University of Technology

6th July 2009

Outline

Introduction

Complexity of software and hardware increases

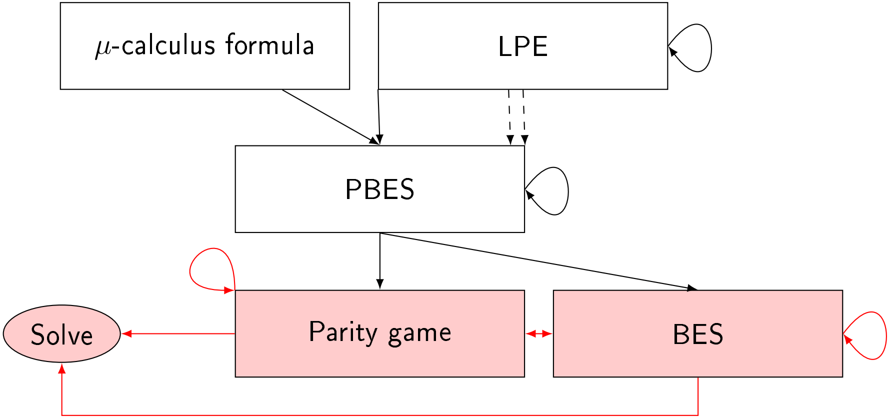
- Requires techniques for verifying correctness:
  - Testing (not exhaustive)
  - Formal proof techniques (require human intellect)
  - Model checking (automated, but time-consuming)

Model checking:

- Property (logic formula)
- Model
- *Does model satisfy property?*

Figure: Roertunnel (ANP)

Methodology



BESs & Parity games

Definition

A BES is a system of fixpoint equations of the form $\sigma X = f$, with $\sigma \in \{\mu, \nu\}$ a fixpoint symbol, and f a logic formula.

Intuition: solution satisfies equations (multiple solutions), fixpoints indicate chosen solution; equations higher in the system take priority over lower ones

Definition

A parity game is a graph game played by two players, $Even$ and Odd. Every vertex v has integer priority $p(v)$. Player $Even$ wins infinite play if least priority that occurs infinitely often is even.

BESs and parity games are equivalent

Algorithms & Optimisations

Practice shows BES algorithms not efficient, except for restricted BES forms

What algorithm should we implement to speed up solving of BESs?

Parity game algorithms	
Small progress measures (2000)	Recursive (1993)
Strategy improvement (2000)	Recursive with preservation (2008)
Optimal strategy improvement (2008)	Dominion decomposition (2006)
Bigstep (2007)	3 satisfiability encodings
Model checker (1998)	

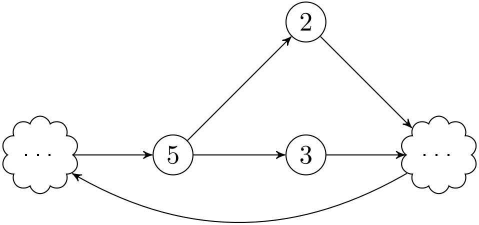
- Complexity depends on number of edges, vertices, priorities
- Optimisation techniques known that can be applied to all algorithms
  - Remove edges
  - Decrease priorities
  - Solve special cases (with more efficient algorithms)
  - SCC decomposition

Priority propagation

- Most optimisations described in both worlds
- Priority propagation new for BES

Intuition: Play that visits vertex v infinitely often must visit one of v's successors infinitely often; reduce v's priority to largest priority of v's successors. (Same can be done for predecessors)

Example



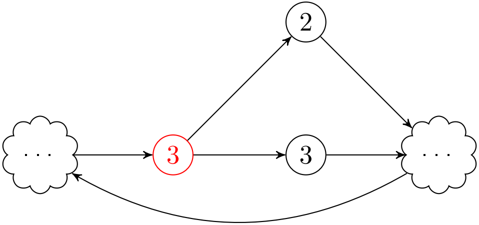
TU/e Technische Universiteit Eindhoven University of Technology

Priority propagation

- Most optimisations described in both worlds
- Priority propagation new for BES

Intuition: Play that visits vertex v infinitely often must visit one of v's successors infinitely often; reduce v's priority to largest priority of v's successors. (Same can be done for predecessors)

Example



TU/e Technische Universiteit Eindhoven University of Technology

Priority propagation (2)

Concept translates to BES :

Theorem

Let $\mathcal{E}_0, \mathcal{E}_1, \mathcal{E}_2$ be BESs, then solution of $\mathcal{E}_0\mathcal{E}_1(\sigma X = f)\mathcal{E}_2 = \mathcal{E}_0(\sigma' X = f)\mathcal{E}_1\mathcal{E}_2$, provided that $\mathsf{occ}(f) \cap \mathsf{bnd}(\mathcal{E}_1\mathcal{E}_2) = \emptyset$ and $X \in \mathsf{occ}(f) \Rightarrow \sigma = \sigma'$

- Theorem inspired upon priority propagation
- Also holds if $X \notin \mathsf{occ}(\mathcal{E}_1) \cup \mathsf{occ}(\mathcal{E}_2)$ and $X \in \mathsf{occ}(f) \Rightarrow \sigma = \sigma'$
- Proofs in semantics of BES, with induction to length of $\mathcal{E}_1$

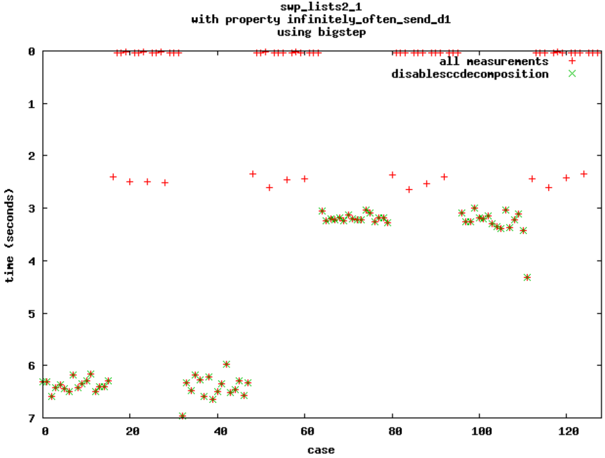
Experiments

$> 20,000$ runs, about three months CPU time

- What is the effect of optimisations?
  - 1 model, 4 properties, 2^7 combinations of optimisations
- How do algorithms depend on priorities?
  - 1 model, 4 properties, 2^6 combinations of optimisations, artificially increase number of priorities
- What is the most efficient algorithm?
  - Large number of models, properties inducing varying number of priorities (between 1 and 3), optimisations enabled

All experiments done for 11 algorithms

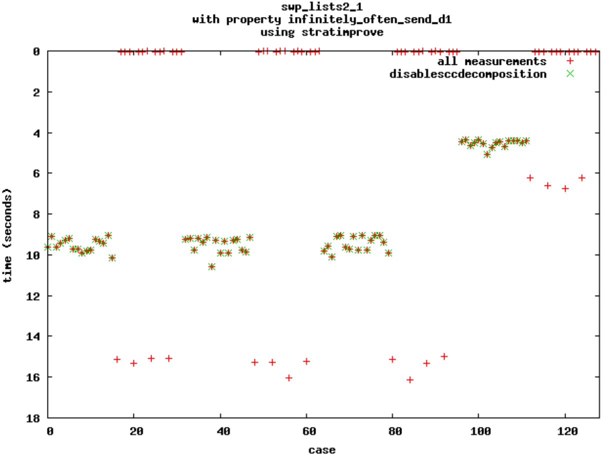
What is the effect of optimisations? (1)



- SCC decomposition dramatically speeds up compution

TU/e Technische Universiteit Eindhoven University of Technology

What is the effect of optimisations? (2)



- SCC decomposition is not the holy grail

TU/e Technische Universiteit Eindhoven University of Technology

What is the effect of optimisations? (3)

- SCC decomposition is beneficial
- Solving special games helps, but not a lot
- No other significant improvements observed with other known optimisations
- Can we do better?

Oblivious bisimulation

Yes we can!

Strong bisimulation minimation:

- Similar to well-known bisimulation minimisation of state spaces
- Equations related that have same rank and Boolean operand, and transfer conditions hold

Observation:

- Equations $\sigma X = Y$ and $\sigma X' = Y \wedge Y$ not related

Oblivious bisimulation minimisation:

- Similar to strong bisimulation minimisation, but restriction on Boolean operands dropped if all variables in right hand sides related
- $\sigma X = Y$ and $\sigma X' = Y \wedge Y$ related

Strong bisimulation generalised to arbitrary BESs by Reniers and Willemse

What is the most efficient algorithm? (1)

Case: SWP$_2$, infinitely often receive all d

$\lvert M \rvert$	2	3	4
size	157,165	872,008	3,097,875
Viasat	0.80	4.83	19.04
Strategy improvement SAT	0.79	4.83	19.13
Small progress measures SAT	0.80	4.83	19.01
Small progress measures	*1.31*	*9.58*	*31.21*
Strategy improvement	0.80	4.98	18.87
Optimal strategy improvement	0.81	4.88	19.13
Recursive	0.81	4.83	18.97
Recursive w. preservation	0.81	4.72	19.09
Dominion decomposition	0.81	5.01	18.98
Bigstep	0.81	4.90	19.96
Model checker	0.81	4.83	19.01

What is the most efficient algorithm? (2)

Accumulated performance indicators:

Strategy	# top 10%	# worst 10%
Viasat	278	216
Strategy improvement SAT	285	216
Small progress measures SAT	282	215
Small progress measures	*198*	*296*
Strategy improvement	286	214
Optimal strategy improvement	283	218
Recursive	292	212
Recursive w. preservation	285	219
Dominion decomposition	291	215
Bigstep	277	220
Model checker	286	215

What is the most efficient algorithm? (2)

- Small progress measures stands out negatively
- Only small differences between other algorithms
- Differences small because of optimisations
- Desire: algorithms that hardly show extremely bad performance
- Advice: choose bigstep and recursive algorithms

Technische Universiteit Eindhoven University of Technology

Conclusions

What algorithms should we implement?

- SCC decomposition and detecting and solving special cases
- Bigstep and recursive algorithms
- Oblivious bisimulation minimisation beats solvers and optimisations

Future work:

- Investigate further reductions for BESs (*e.g.* stuttering equivalence)
- Investigate effectiveness of generalised parity game algorithms in BES framework
- Develop reduction techniques in symbolic framework of PBES