On the Quest for an Acyclic Graph

Mikoláš Janota[1] Radu Grigore[2] Vasco Manquinho[1]

RCRA 2017, Bari

[1] INESC-ID/IST, University of Lisbon, Portugal
[2] School of Computing, University of Kent, UK

Introduction

- We wish to reason over directed graphs in SAT/QBF/SMT.

Introduction

- We wish to reason over directed graphs in SAT/QBF/SMT.
- Each graph corresponds to a binary relation.

Introduction

- We wish to reason over directed graphs in SAT/QBF/SMT.
- Each graph corresponds to a binary relation.
- Graphs are model with Boolean variables representing edges.

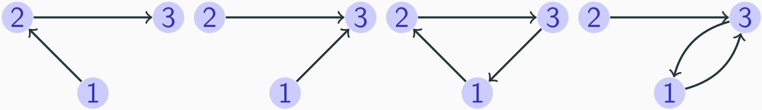
Introduction

- We wish to reason over directed graphs in SAT/QBF/SMT.
- Each graph corresponds to a binary relation.
- Graphs are model with Boolean variables representing edges.
- A variable e_{ij} is true iff there is an edge from i to j in the considered graph.

- We wish to reason over directed graphs in SAT/QBF/SMT.
- Each graph corresponds to a binary relation.
- Graphs are model with Boolean variables representing edges.
- A variable e_{ij} is true iff there is an edge from i to j in the considered graph.

Example

$$(e_{12} \vee e_{13}) \wedge (\neg e_{12} \vee \neg e_{13}) \wedge e_{23} \wedge \neg e_{32} \wedge \neg e_{21}$$



Objective: Avoid Cycles

- Given a formula φ determining a set of graphs generate a formula φ' that only considers those of in φ but with no cycles.

Objective: Avoid Cycles

- Given a formula φ determining a set of graphs generate a formula φ' that only considers those of in φ but with no cycles.
- Modular approach: devise a checker formula ψ s.t. $\varphi' = \varphi \wedge \psi$.

Example: Memory Models

```
start state: r1=r2=x=y=0
thread 1:
  x := 1
  r1 := y
thread 2:
  y := 1
  r0 := x
```

- Question: Is state r1=r2=0 possible at the end?

Example: Memory Models

```
start state: r1=r2=x=y=0
thread 1:
  x := 1
  r1 := y
thread 2:
  y := 1
  r0 := x
```

- Question: Is state r1=r2=0 possible at the end?
- Answer: NO with interleavings, but

  YES with many relaxed/weak memory models.

Example: Memory Models

```
start state: r1=r2=x=y=0
thread 1:
  x := 1
  r1 := y
thread 2:
  y := 1
  r0 := x
```

- Question: Is state r1=r2=0 possible at the end?
- Answer: NO with interleavings, but

  YES with many relaxed/weak memory models.
- We are using a QBF solver in this application, acyclic relations are required.

Example: No-Sink Family

- A well-ordered set must have a greatest element.

Example: No-Sink Family

- A well-ordered set must have a greatest element.
- If each node as at least one outgoing edge,
  there must be a cycle.

Example: No-Sink Family

- A well-ordered set must have a greatest element.
- If each node as at least one outgoing edge,
  there must be a cycle.
- Together with acyclicity, the following is UNSAT:

$$\bigwedge_{i \in [n]} \bigvee_{j \in [n]} e_{ij}$$

Example: The Supervisor Problem

- Consider n employees;

Example: The Supervisor Problem

- Consider n employees;
- $\ldots$ employee i can supervise at most u_i other employees;

Example: The Supervisor Problem

- Consider n employees;
- $\ldots$ employee i can supervise at most u_i other employees;
- $\ldots$ employee i is supervised by at least l_i other employees;

Example: The Supervisor Problem

- Consider n employees;
- ... employee i can supervise at most u_i other employees;
- ... employee i is supervised by at least l_i other employees;
- ... there may be no cycles in the supervision relation.

Example: The Supervisor Problem

- Consider n employees;
- ... employee i can supervise at most u_i other employees;
- ... employee i is supervised by at least l_i other employees;
- ... there may be no cycles in the supervision relation.
- NP-complete [Hartung and Nichterlein, 2015]

Example: The Supervisor Problem

- Consider n employees;
- ... employee i can supervise at most u_i other employees;
- ... employee i is supervised by at least l_i other employees;
- ... there may be no cycles in the supervision relation.
- NP-complete [Hartung and Nichterlein, 2015]
- Note: no-sink is special case. There is no solution if everyone is to have at least one supervisor.

Encoding: Transitive Closure

- Idea: Generate a transitive closure of the edge relation and disable self-loops.

Encoding: Transitive Closure

- Idea: Generate a transitive closure of the edge relation and disable self-loops.
- Transitive closure I

$$\psi_n(\vec{e},\vec{y}) \;\coloneqq\; \bigwedge_{i\in[n]} \neg y_{ii} \wedge \bigwedge_{i,j,k\in[n]} (y_{ij} \wedge y_{jk} \Rightarrow y_{ik}) \wedge \bigwedge_{i,j\in[n]} (e_{ij} \Rightarrow y_{ij})$$

Encoding: Transitive Closure

- Idea: Generate a transitive closure of the edge relation and disable self-loops.
- Transitive closure I

$$\psi_n(\vec{e}, \vec{y}) \coloneqq \bigwedge_{i \in [n]} \neg y_{ii} \wedge \bigwedge_{i,j,k \in [n]} (y_{ij} \wedge y_{jk} \Rightarrow y_{ik}) \wedge \bigwedge_{i,j \in [n]} (e_{ij} \Rightarrow y_{ij})$$

- Transitive closure II

$$\psi_n(\vec{e}, \vec{y}) \coloneqq \bigwedge_{i \in [n]} \neg y_{ii} \wedge \bigwedge_{i,j,k \in [n]} (y_{ij} \wedge e_{jk} \Rightarrow y_{ik}) \wedge \bigwedge_{i,j \in [n]} (e_{ij} \Rightarrow y_{ij})$$

Encoding: Unary/Binary labeling

- Idea: Any DAG can be topologically sorted.

Encoding: Unary/Binary labeling

- Idea: Any DAG can be topologically sorted.
- Label each node with a number $l \in 1..|V|$ such that it is connected only to nodes with a greater label.

$$\psi_n(\vec{e}, \vec{y}_1, \ldots, \vec{y}_n) \;:=\; \bigwedge_{i,j \in [n]} \big(e_{ij} \Rightarrow \mathrm{less}(\vec{y}_i, \vec{y}_j)\big)$$

Encoding: Unary/Binary labeling (Cont.)

- Comparison for binary encoding ($n\lceil \log_2 n \rceil$ variables).

$$
\begin{aligned}
\mathrm{lex}_0() &:= 0 \\
\mathrm{lex}_b(\vec{y}y, \vec{z}z) &:= (\neg y \wedge z) \vee \big((\neg y \vee z) \wedge \mathrm{lex}_{b-1}(\vec{y}, \vec{z})\big)
\end{aligned}
$$

Encoding: Unary/Binary labeling (Cont.)

- Comparison for binary encoding ($n\lceil \log_2 n \rceil$ variables).

$$
\begin{aligned}
\mathrm{lex}_0() &:= 0 \\
\mathrm{lex}_b(\vec{y}y, \vec{z}z) &:= (\neg y \wedge z) \vee \big((\neg y \vee z) \wedge \mathrm{lex}_{b-1}(\vec{y}, \vec{z}\,)\big)
\end{aligned}
$$

- Comparison for unary encoding (n^2 variables):

$$
\begin{aligned}
\mathrm{lessunr}(\vec{y}, \vec{z}, \vec{u}\,) &:= \bigwedge_{i=1}^{n-1} \big((\neg y_i \vee \neg u_i) \wedge (z_i \vee \neg u_i)\big) \wedge \bigvee_{i=1}^{n-1} u_i \\
\mathrm{unary}(\vec{y}\,) &:= \bigwedge_{i=2}^{n-1} (y_{i-1} \Rightarrow y_i)
\end{aligned}
$$

Encoding: Warshall algorithm Based

Idea: Perform an "unrolling" of the Floyd-Warshall.

WARSHALL

```
for k ∈ [n]
    for i ∈ [n]
        for j ∈ [n]
            a_ij := Or(a_ij, And(a_ik, a_kj))
```

Encoding: Warshall algorithm Based

Idea: Perform an “unrolling” of the Floyd-Warshall.

WARSHALL

for $k \in [n]$
  for $i \in [n]$
    for $j \in [n]$
      $a_{ij} := \mathrm{Or}(a_{ij}, \mathrm{And}(a_{ik}, a_{kj}))$

$$
\begin{aligned}
\psi_n(\vec{x}, \vec{y}) \ := \ & \bigwedge_{i \in [n]} \neg y_{iin} \wedge \bigwedge_{i,j \in [n]} (x_{ij} \Rightarrow y_{ij0}) \wedge \bigwedge_{i,j,k \in [n]} (y_{ij(k-1)} \Rightarrow y_{ijk}) \\
& \wedge \bigwedge_{i,j,k \in [n]} (y_{ik(k-1)} \wedge y_{kj(k-1)} \Rightarrow y_{ijk})
\end{aligned}
$$

Encoding: Matrix Multiplication

- Idea: Simulate matrix multiplication

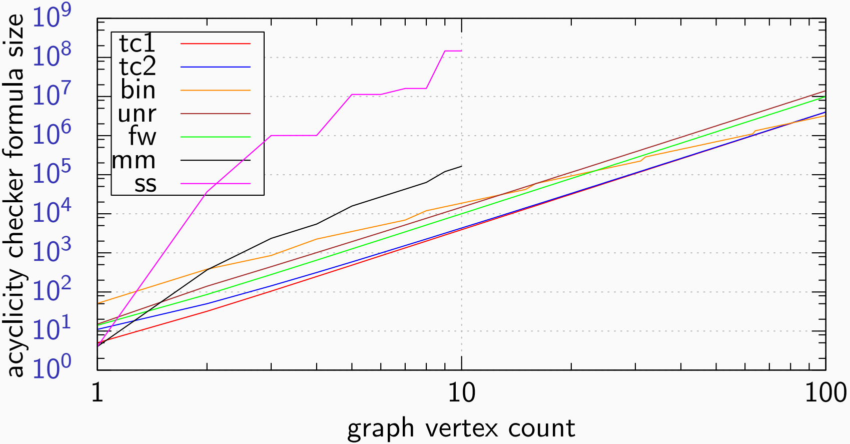
Encoding: Matrix Multiplication

- Idea: Simulate matrix multiplication
- Strassen algorithm permits less than cubic multiplication

Encoding: Matrix Multiplication

- Idea: Simulate matrix multiplication
- Strassen algorithm permits less than cubic multiplication
- Hard to efficiently encode into circuits.

Sizes of Encodings



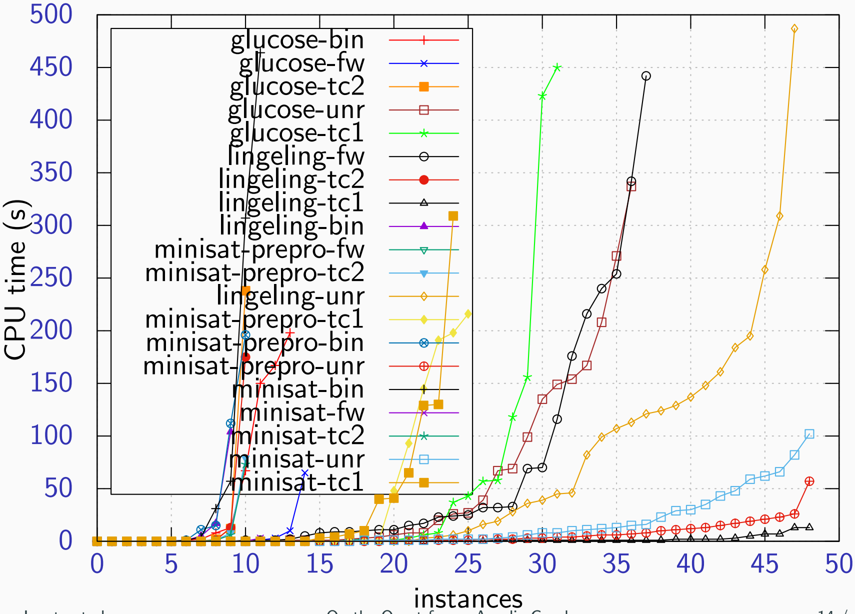
No sink (49):

solver/checker	tc I	unr	bin	fw	tc II
lingeling	**49**	48	10	38	11
glucose	32	37	14	15	11
minisat	25	**49**	12	12	11
minisat-prepro	26	**49**	11	19	11

Supervisor (441):

solver/checker	tc I	unr	bin	fw	tc II
lingeling	436	429	426	435	435
glucose	**437**	434	427	**437**	**437**
minisat	435	425	424	435	435
minisat-prepro	435	426	422	435	435

Experiments (Cont.)



Summary

- Studied encoding for graph acyclicity.

Summary

- Studied encoding for graph acyclicity.
- In our work eager, advantage that SAT/QBF/SMT is agnostic of acyclicity.

Summary

- Studied encoding for graph acyclicity.
- In our work eager, advantage that SAT/QBF/SMT is agnostic of acyclicity.
- Number of encodings developed and evaluated.

Summary

- Studied encoding for graph acyclicity.
- In our work eager, advantage that SAT/QBF/SMT is agnostic of acyclicity.
- Number of encodings developed and evaluated.
- Performance varies across encodings and solvers.

Summary

- Studied encoding for graph acyclicity.
- In our work eager, advantage that SAT/QBF/SMT is agnostic of acyclicity.
- Number of encodings developed and evaluated.
- Performance varies across encodings and solvers.
- More experiments.

Summary

- Studied encoding for graph acyclicity.
- In our work eager, advantage that SAT/QBF/SMT is agnostic of acyclicity.
- Number of encodings developed and evaluated.
- Performance varies across encodings and solvers.
- More experiments.
- When is eager better than lazy and the other way around?

Summary

- Studied encoding for graph acyclicity.
- In our work eager, advantage that SAT/QBF/SMT is agnostic of acyclicity.
- Number of encodings developed and evaluated.
- Performance varies across encodings and solvers.
- More experiments.
- When is eager better than lazy and the other way around?
- Can we get less-than cubic but practical?

Thank You for Your Attention!

Questions?

Hartung, S. and Nichterlein, A. (2015).
NP-hardness and fixed-parameter tractability of realizing degree sequences with directed acyclic graphs.
SIAM Journal on Discrete Mathematics, 29(4):1931–1960.