Knor: reactive synthesis using Oink

Tom van Dijk, Feije van Abbema, Naum Tomov
t.vandijk@utwente.nl

TACAS, 8 April 2024

OUTLINE

- Reactive synthesis from LTL to Boolean circuits
- Knor in the synthesis competition
- Five strategies to improve circuit size
- Opportunities for future work

REACTIVE SYNTHESIS

Synthesis

- Given some specification, construct a satisfying implementation

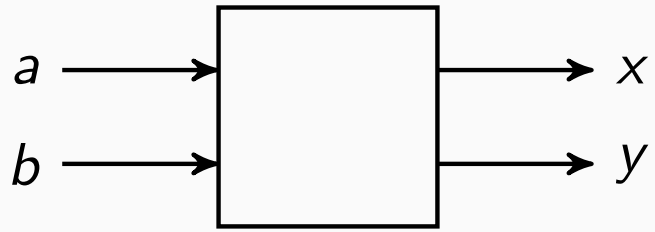
REACTIVE SYNTHESIS

Synthesis

- Given some specification, construct a satisfying implementation

Reactive synthesis

- Construct a system that interacts continuously
- The system "reacts" to input from environment



(black box with inputs a, b and outputs x, y)

- Specification over input/output signals in linear temporal logic (LTL)
- Mealy machine (alternative: Moore machine)

LTL SYNTHESIS

1. Preprocess LTL formula
   - Handle easy cases
   - Decompose / normalize

LTL SYNTHESIS

1. Preprocess LTL formula
   - Handle easy cases
   - Decompose / normalize
2. Construct deterministic parity (or Rabin) automaton
   - via DELA
   - via LDBA
   - via very weak alternating automata
   - via NBA/NCA
   - then determinize/parityize

LTL SYNTHESIS

1. Preprocess LTL formula
   - Handle easy cases
   - Decompose / normalize
2. Construct deterministic parity (or Rabin) automaton
   - via DELA
   - via LDBA
   - via very weak alternating automata
   - via NBA/NCA
   - then determinize/parityize
3. Construct parity game from parity automaton (split I/O)

LTL SYNTHESIS

1. Preprocess LTL formula
   - Handle easy cases
   - Decompose / normalize
2. Construct deterministic parity (or Rabin) automaton
   - via DELA
   - via LDBA
   - via very weak alternating automata
   - via NBA/NCA
   - then determinize/parityize
3. Construct parity game from parity automaton (split I/O)
4. Solve parity game

LTL SYNTHESIS

1. Preprocess LTL formula
   - Handle easy cases
   - Decompose / normalize
2. Construct deterministic parity (or Rabin) automaton
   - via DELA
   - via LDBA
   - via very weak alternating automata
   - via NBA/NCA
   - then determinize/parityize
3. Construct parity game from parity automaton (split I/O)
4. Solve parity game
5. Extract Mealy machine from winning strategy

LTL SYNTHESIS

1. Preprocess LTL formula
   - Handle easy cases
   - Decompose / normalize
2. Construct deterministic parity (or Rabin) automaton
   - via DELA
   - via LDBA
   - via very weak alternating automata
   - via NBA/NCA
   - then determinize/parityize
3. Construct parity game from parity automaton (split I/O)
4. Solve parity game
5. Extract Mealy machine from winning strategy
6. Encode Mealy machine as Boolean circuit

KNOR IN SYNTCOMP

Oink

- State-of-the-art explicit-state parity game solver (TACAS 2018)
- Implements $\pm$ 30 algorithms
- Can run some algorithms multi-core

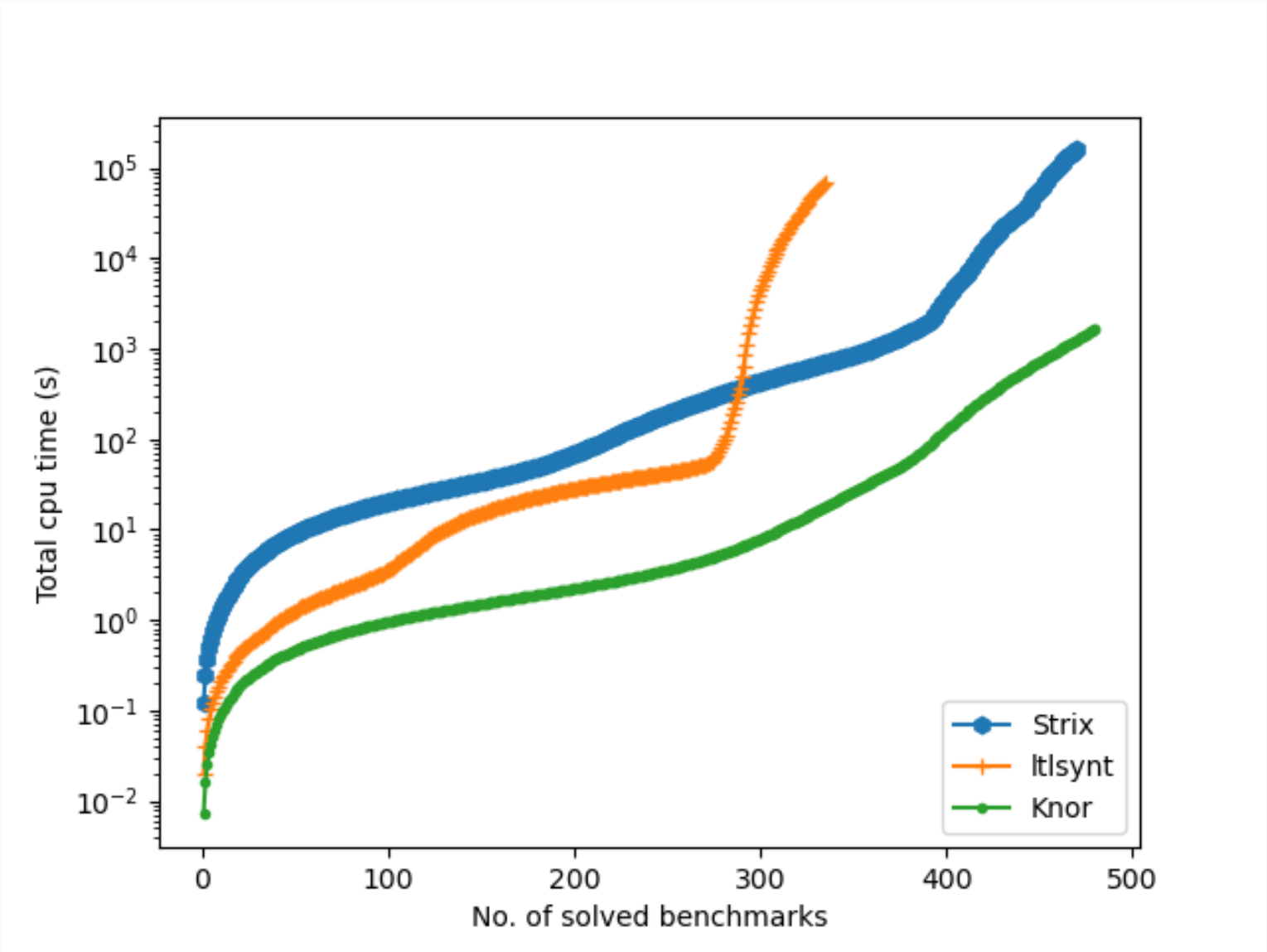
PGAME track in SYNTCOMP

- New track in SYNTCOMP 2020
- Input: DPA in the "extended eHOA" format
- Output: Boolean circuit in the "AIGER" format

Knor: Oink in SYNTCOMP

- From eHOA input file to parity game in Oink (2020)
- From parity game strategy in Oink to AIGER output file (2021)
- Winner in 2021, 2022 by runtime

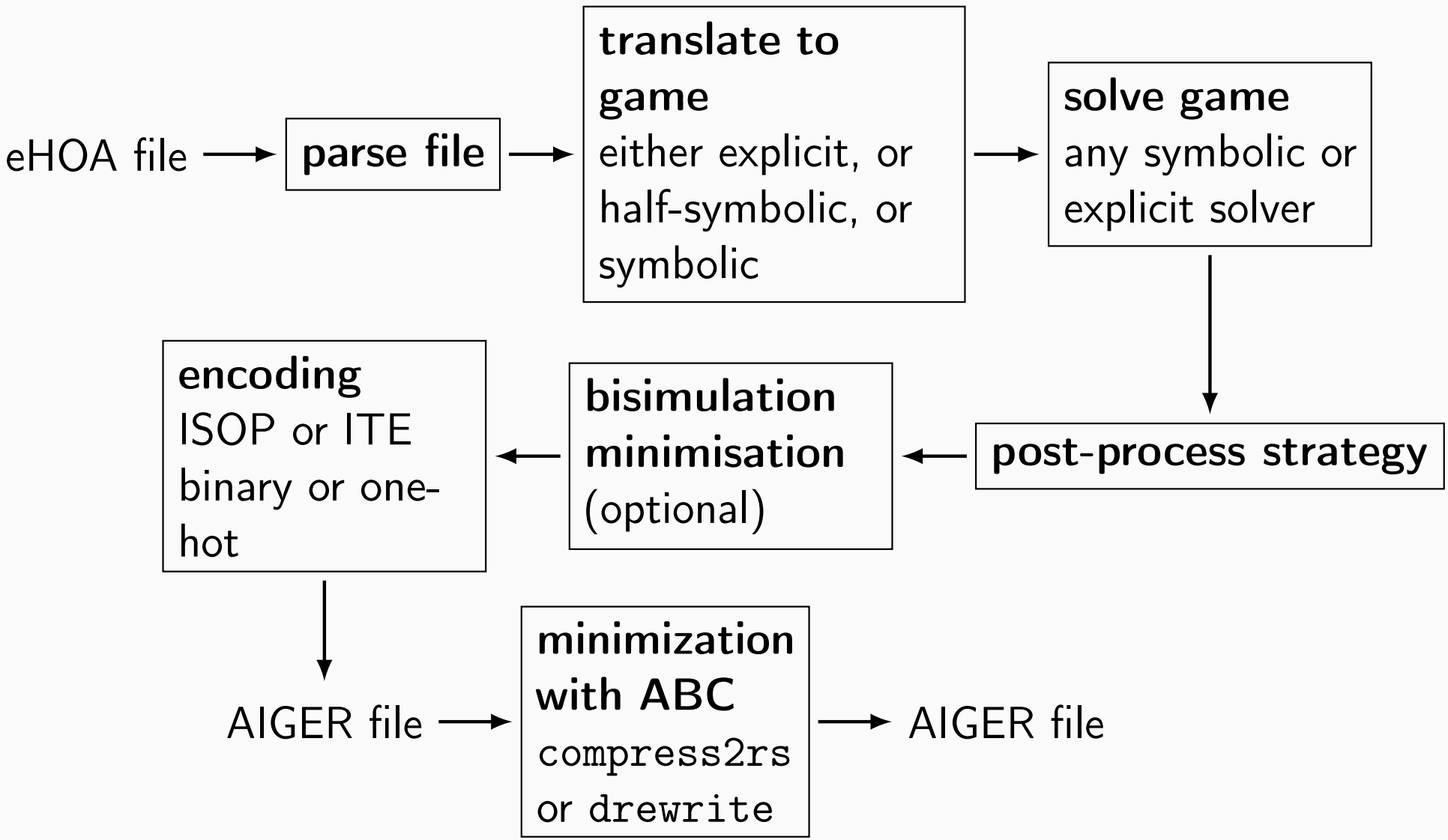
KNOR IN SYNTCOMP 2022



Controller quality score

Solver	Configuration	Score
Strix	hoa_synthesis_parallel	385
ltlsynt	pgsyntabc1	358
Knor	synt_sym_abc	340

KNOR OVERVIEW



KNOBS

- Game construction method
  - explicit, half-symbolic, fully symbolic
- Choice of parity game solver
  - symbolic, or any explicit solver in Oink
- Bisimulation minimisation of Mealy machine
  - yes or no
- Encoding to circuit (latches, AND-gates, inverters)
  - state encoding: binary / onehot
  - logic encoding: ISOP / ITE
- Minimization strategy
  - no minimization, or `compress2rs`, or `drw; b; drf`

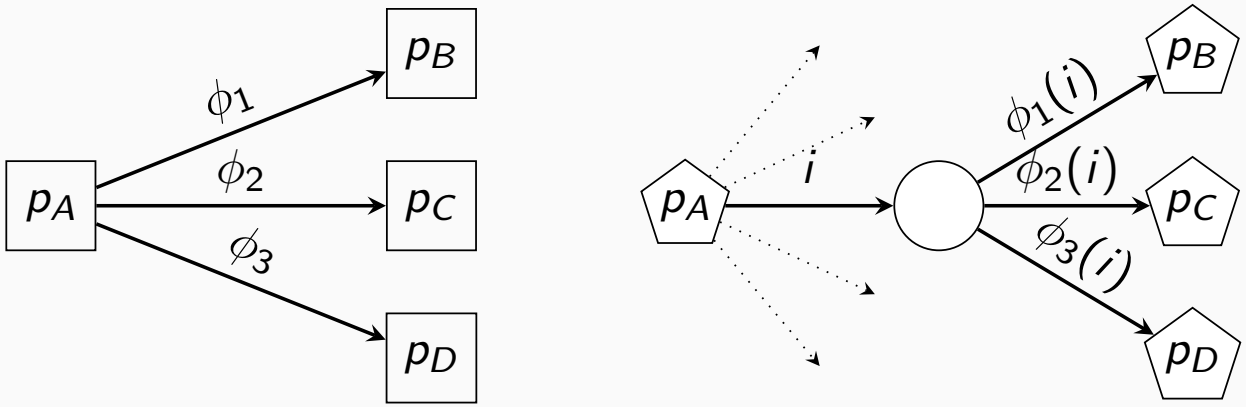
INPUT FORMAT

- Hanoi Omega-Automata (HOA) format
  - Explicit states
  - Transitions have Boolean formulas over signals
- Extension: which signals are controllable (output signals)

CONSTRUCTING THE GAME

Goal: split the transitions

ϕ_1, ϕ_2, ϕ_3 are Boolean formulas over I and O



- Explicit: enumerate every valuation of I/O signals
- Half-symbolic: construct a BDD for each state
- Fully symbolic: represent entire automaton in one BDD
  - Variable order: state, input, output, priority, next state
  - Can construct explicit parity game on the BDD nodes (!!)

CONSTRUCTING THE GAME

From eHOA to explicit parity game

Technique	**Sum of Vertices**	**Time** (sec)
explicit	622,987,565	1,177.91
half-symbolic	8,491,540	18.28
symbolic	620,510	11.76

CONSTRUCTING THE GAME

Model	explicit	half-symbolic	symbolic
amba_decomposed_lock_15	T.O.	46	24
amba_decomposed_lock_14	T.O.	46	24
amba_decomposed_lock_13	T.O.	46	24
TwoCountersDisButA9	T.O.	668,065	7,249
amba_decomposed_lock_12	402,997,254	46	24
amba_decomposed_lock_11	100,820,998	46	24
amba_decomposed_lock_10	25,237,510	46	24
TwoCountersGui	21,022,475	256	155
TwoCountersDisButA8	15,254,863	497,310	4,721
full_arbiter_8	11,287,306	1,669,066	177,690
amba_decomposed_lock_9	6,323,718	46	24
amba_decomposed_encode_16	4,981,507	876	330
TwoCountersDisButA7	3,939,305	98,947	2,365
TwoCountersDisButA6	3,806,249	101,175	1,733

PARITY GAME SOLVERS

- Symbolic solver (fpi)
- Explicit solvers in Oink

PARITY GAME SOLVERS

- Symbolic solver (fpi)
- Explicit solvers in Oink

Solver	**Circuit size**		**Time** (sec)
	binary	onehot	
symbolic fpi (`--sym`)	317,403	122,514	18.45
fixpoint with justifications (`--fpj`)	350,035	139,900	0.16
fixpoint with freezing (`--fpi`)	353,120	140,297	0.22
strategy iteration (`--psi`)	334,149	140,916	0.57
priority promotion (`--npp`)	427,048	161,244	0.17
Zielonka (`--zlk`)	480,472	175,427	0.18
tangle learning (`--tl`)	604,044	213,632	0.17

- Symbolic solver best for circuit size, but slower than all others
  - (only slow for a few inputs)
- Future work: design a solver for small strategies?

BISIMULATION MINIMISATION

- Apply bisimulation minimisation on the Mealy machine

BISIMULATION MINIMISATION

- Apply bisimulation minimisation on the Mealy machine

Solver	**Circuit size**		**Time** (sec)
	binary	onehot	
symbolic fpi (`--sym`) + minimisation	166,839	106,500	0.19
fixpoint with justifications (`--fpj`) + min.	205,937	124,489	0.15
symbolic fpi (`--sym`)	317,403	122,514	–
fixpoint with justifications (`--fpj`)	350,035	139,900	–

- Takeaway: always apply
- Future work: minimise parity game before solving?
- Future work: other Mealy machine minimisation methods?

ENCODING MEALY MACHINE AS BOOLEAN CIRCUIT

- How to encode states of the Mealy machine as latches
- How to encode logic as circuit (AND-gates and inverters)

ENCODING MEALY MACHINE AS BOOLEAN CIRCUIT

- How to encode states of the Mealy machine as latches
- How to encode logic as circuit (AND-gates and inverters)

Solver	**Encoding**	**Circuit size**	**Time**
symbolic fpi (`--sym`)	ISOP, onehot	102,294	0.69
symbolic fpi (`--sym`)	ITE, onehot	106,500	0.61
fixpoint with justifications (`--fpj`)	ISOP, onehot	113,134	0.72
fixpoint with justifications (`--fpj`)	ITE, onehot	124,489	0.64
symbolic fpi (`--sym`)	ITE, binary	166,839	0.09
fixpoint with justifications (`--fpj`)	ITE, binary	205,937	0.12
symbolic fpi (`--sym`)	ISOP, binary	431,316	1.39
fixpoint with justifications (`--fpj`)	ISOP, binary	476,502	1.61

- Takeaway: onehot encoding is best, ISOP slightly better
- Future work: other latch encoding strategies
- Future work: play with order of encoding from BDD to gates

POSTPROCESSING WITH ABC

- "Everyone uses post-processing with ABC" e.g. `compress2rs` (Strix)
- What about `drw; b; drf`?

POSTPROCESSING WITH ABC

- "Everyone uses post-processing with ABC" e.g. `compress2rs` (Strix)
- What about `drw; b; drf`?

Solver	Encoding	Method	Circuit size	Time
symbolic fpi (`--sym`)	ISOP	compress	61,434	149.26
symbolic fpi (`--sym`)	ITE	compress	62,506	121.27
fixpoint with justifications (`--fpj`)	ISOP	compress	71,240	125.29
fixpoint with justifications (`--fpj`)	ITE	compress	72,897	108.10
symbolic fpi (`--sym`)	ISOP	drewrite	80,077	58.72
symbolic fpi (`--sym`)	ITE	drewrite	80,425	53.21
fixpoint with justifications (`--fpj`)	ISOP	drewrite	80,454	60.88
fixpoint with justifications (`--fpj`)	ITE	drewrite	80,903	58.58
symbolic fpi (`--sym`)	ISOP		102,294	44.88
symbolic fpi (`--sym`)	ITE		106,500	39.81
fixpoint with justifications (`--fpj`)	ISOP		113,134	31.66
fixpoint with justifications (`--fpj`)	ITE		124,489	25.77

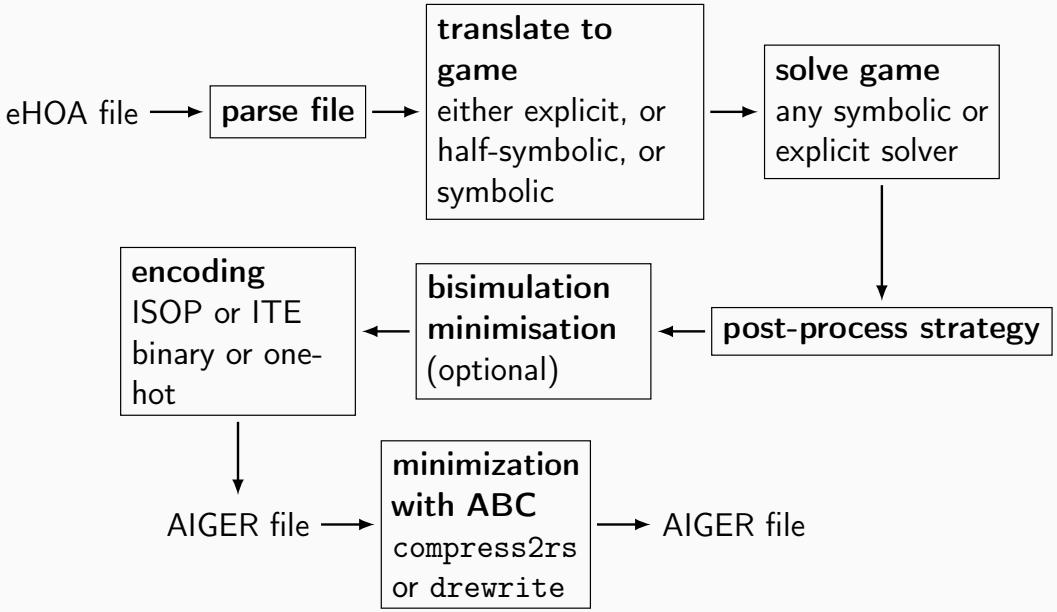
- Takeaway: trading time for space; good initial size: good final size
- Future work: try other ABC scripts; try other libraries e.g. Mockturtle
- Future work: can we find postprocessing-friendly encodings?

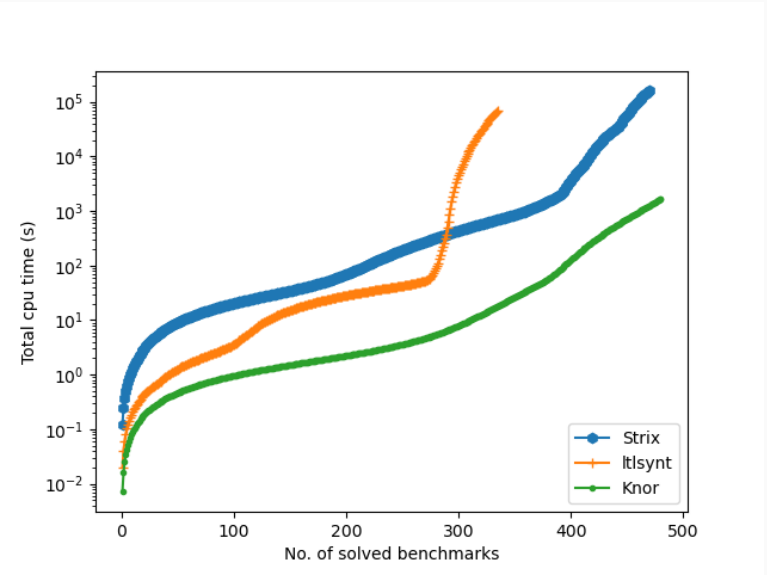
COMPARISON WITH OTHER TOOLS

Tool	**Circuit size**	
	no post-processing	with post-processing
strix	68,550	41,314
sym-bisim-isop-onehot	87,823	50,624
ltlsynt	544,804	98,996

- Knor is better than Strix for many benchmarks, except the `full_arbiter` benchmarks.

FINAL REMARKS





- Best result with fully symbolic methods + bisimulation minimisation + onehot encoding with ISOP + `compress2rs` post-processing
- Still many opportunities to improve
- Some SYNTCOMP-PGAME benchmarks are not derived from LTL
- Relying on ABC is not fun, but may be a necessary evil

This slide intentionally left empty.