APT

A tool for analysing and synthesising Petri nets and transition systems

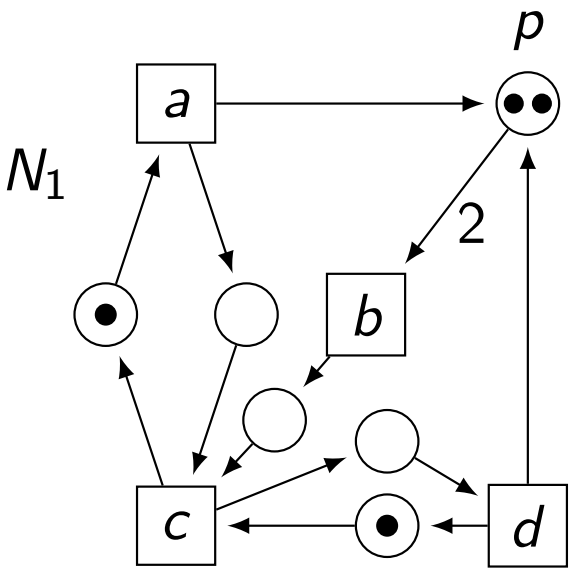
Eike Best and Uli Schlachter
{eike.best,uli.schlachter}@informatik.uni-oldenburg.de

Carl von Ossietzky Universität Oldenburg

ICE 2015

Outline

Petri nets and transition systems

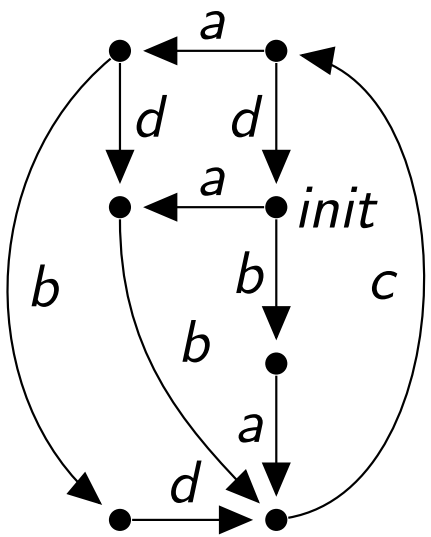


Petri net

analysis

synthesis

lts



state space
reachability graph
transition system

Motivation

- versatility

Motivation

- versatility
- transparency

Motivation

- versatility
- transparency
- extensibility

Motivation

- versatility
- transparency
- extensibility
- modularity

Motivation

- versatility
- transparency
- extensibility
- modularity
- bare-bonedness

Motivation

- versatility
- transparency
- extensibility
- modularity
- bare-bonedness
- portability

Motivation

- versatility
- transparency
- extensibility
- modularity
- bare-bonedness
- portability
- availability

APT

The team

12 students, 4 supervisors, one year project

APT

The tool

- Java 7
- Single JAR file
- Command line
- Non-interactive
- About 80 different modules
- Each module solves one specific problem
  e.g. “compute the reachability graph of a given Petri net”

Demonstration

Algorithms for Petri nets

- Structural properties
  - plain
  - pure
  - connectedness
  - marked graph/T-net
  - output-nonbranching
  - conflict-free
  - S- and T-invariants
  - traps and siphons

Algorithms for Petri nets

- Structural properties
  - plain
  - pure
  - connectedness
  - marked graph/T-net
  - output-nonbranching
  - conflict-free
  - S- and T-invariants
  - traps and siphons
- Behavioral properties
  - reachability graph
  - coverability graph
  - k-boundedness
  - (weak) liveness
  - word in language
  - behavioural conflict free
  - binary conflict free
  - properties of the reachability graph

Algorithms for lts

- Structural properties
  - determinism
  - total reachability
  - persistency
  - reversibility
  - connectedness
  - weak/strong separability

Algorithms for lts

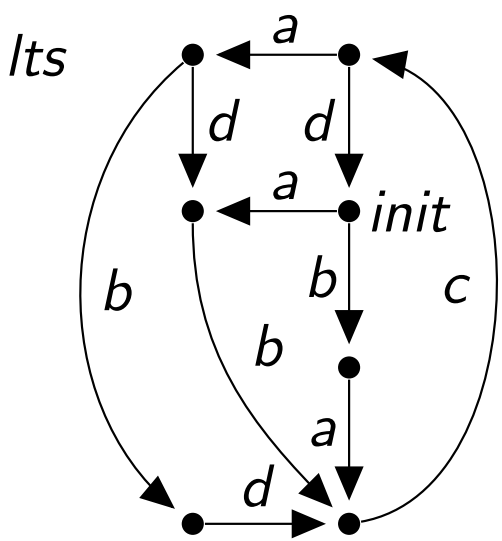
- Structural properties
  - determinism
  - total reachability
  - persistency
  - reversibility
  - connectedness
  - weak/strong separability
- Equivalences
  - language equivalence
  - isomorphism
  - bisimulation

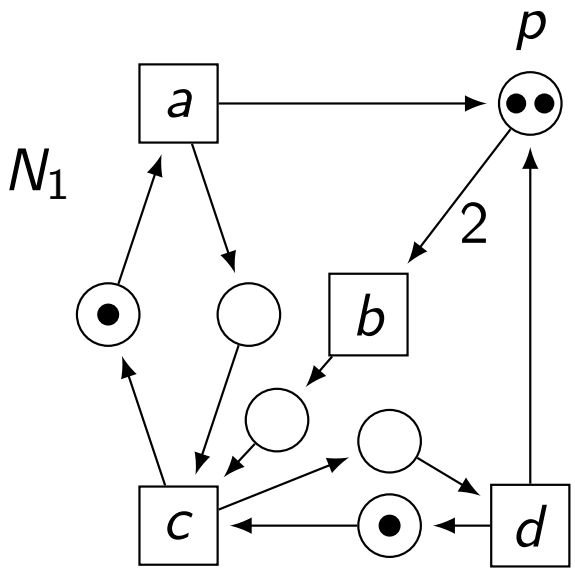
Algorithms for lts

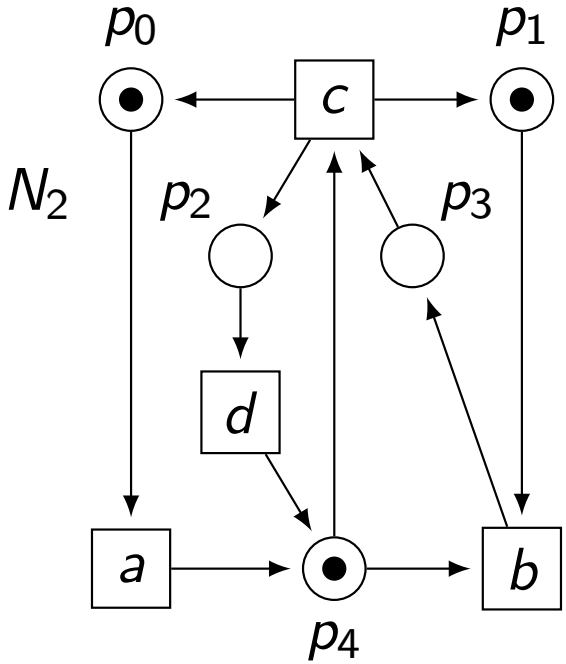
- Structural properties
  - determinism
  - total reachability
  - persistency
  - reversibility
  - connectedness
  - weak/strong separability
- Equivalences
  - language equivalence
  - isomorphism
  - bisimulation
- PN-Synthesis
  - Find Petri net with isomorphic reachability graph
  - Find Petri net with language-equivalent reachability graph

What is Petri net synthesis?

An lts and two different Petri nets solving it:







Petri net synthesis

- Solve Its with Petri net (isomorphic / language equivalent)
- Free Petri nets: No two transitions generate same letter

Petri net synthesis

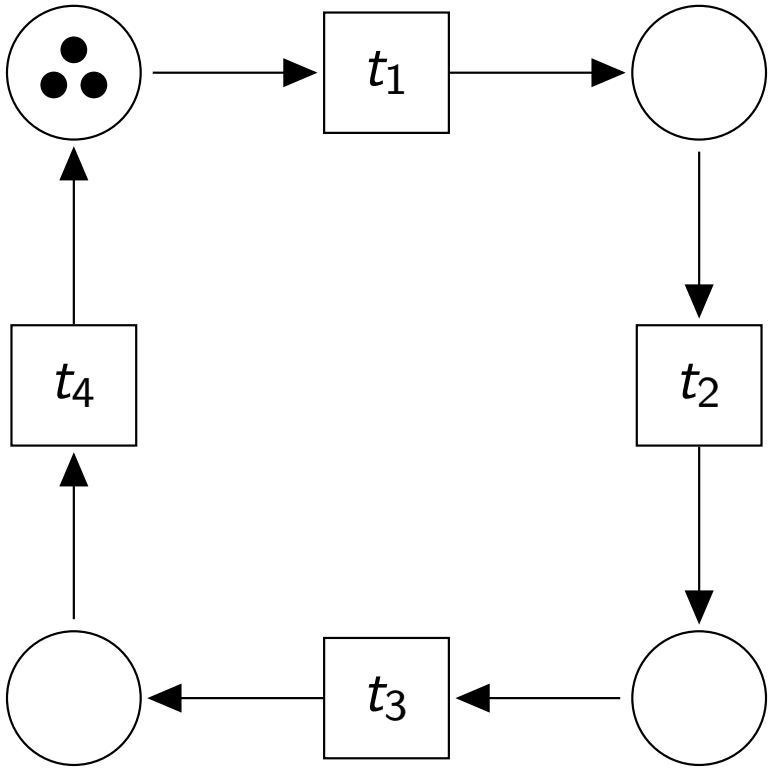
- Solve Its with Petri net (isomorphic / language equivalent)
- Free Petri nets: No two transitions generate same letter
- Restrict the net to combination of:
  - pure
  - plain
  - output-nonbranching
  - t-net
  - conflict-free
  - k-bounded
  - safe
- Distributed Petri nets:
  - Assign locations to letters
  - Letters with different location do not have "shared state": disjoint presets

Benchmarks

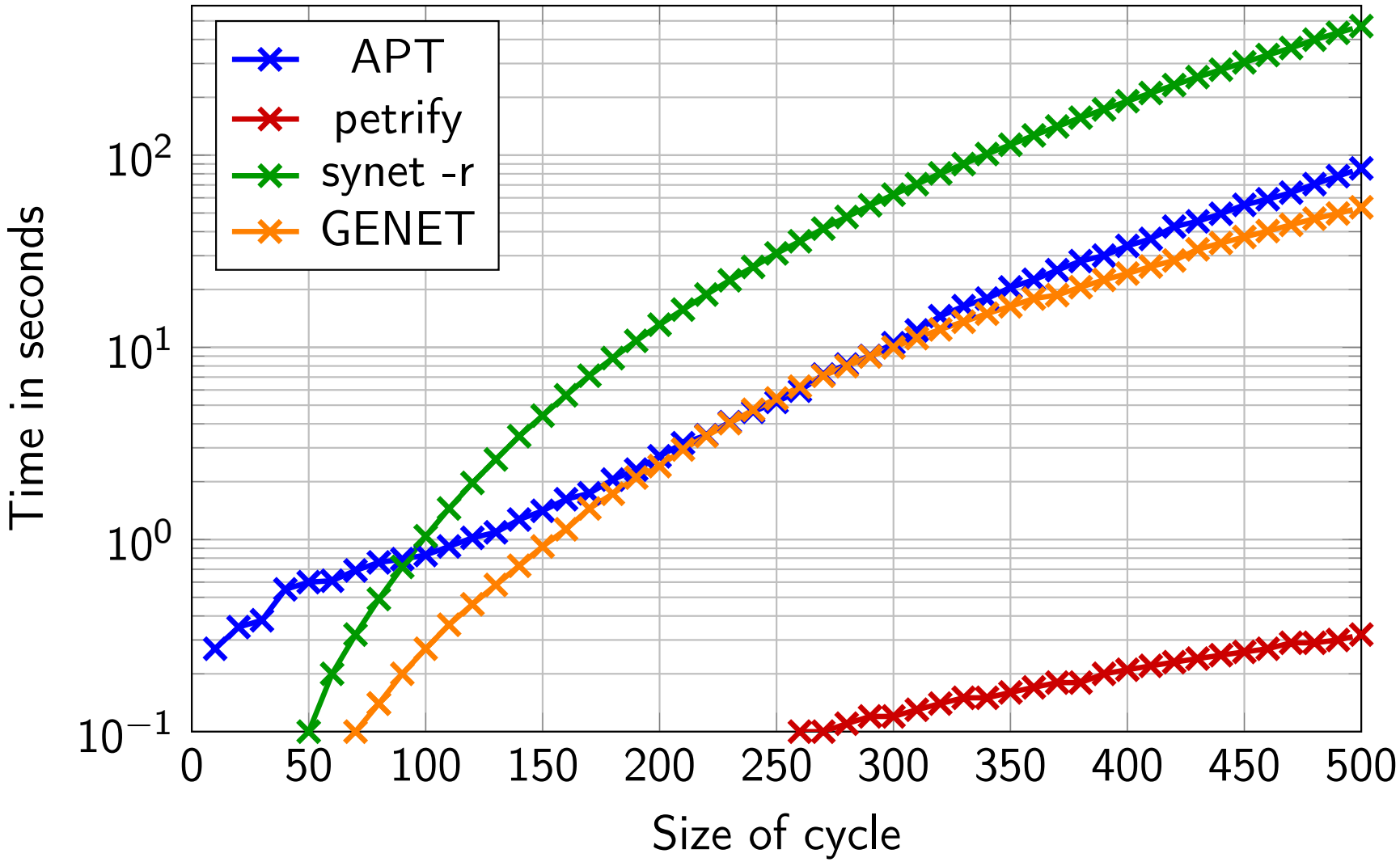
- Compare APT with petrify[1], synet[2] and GENET[3]
- Approach:
  - Generate Petri net of size n
  - Calculate reachability graph with APT
  - Measure: Synthesize pn
- Specs: Intel Core i7-4790 with 3.6GHz, 32 GiB of RAM

[1] `http://www.cs.upc.edu/~jordicf/petrify/`
[2] `https://www.irisa.fr/s4/tools/synet/`
[3] `http://www.cs.upc.edu/~jcarmona/genet.html`

Cycle of size $n = 4$ with $k = 3$ token

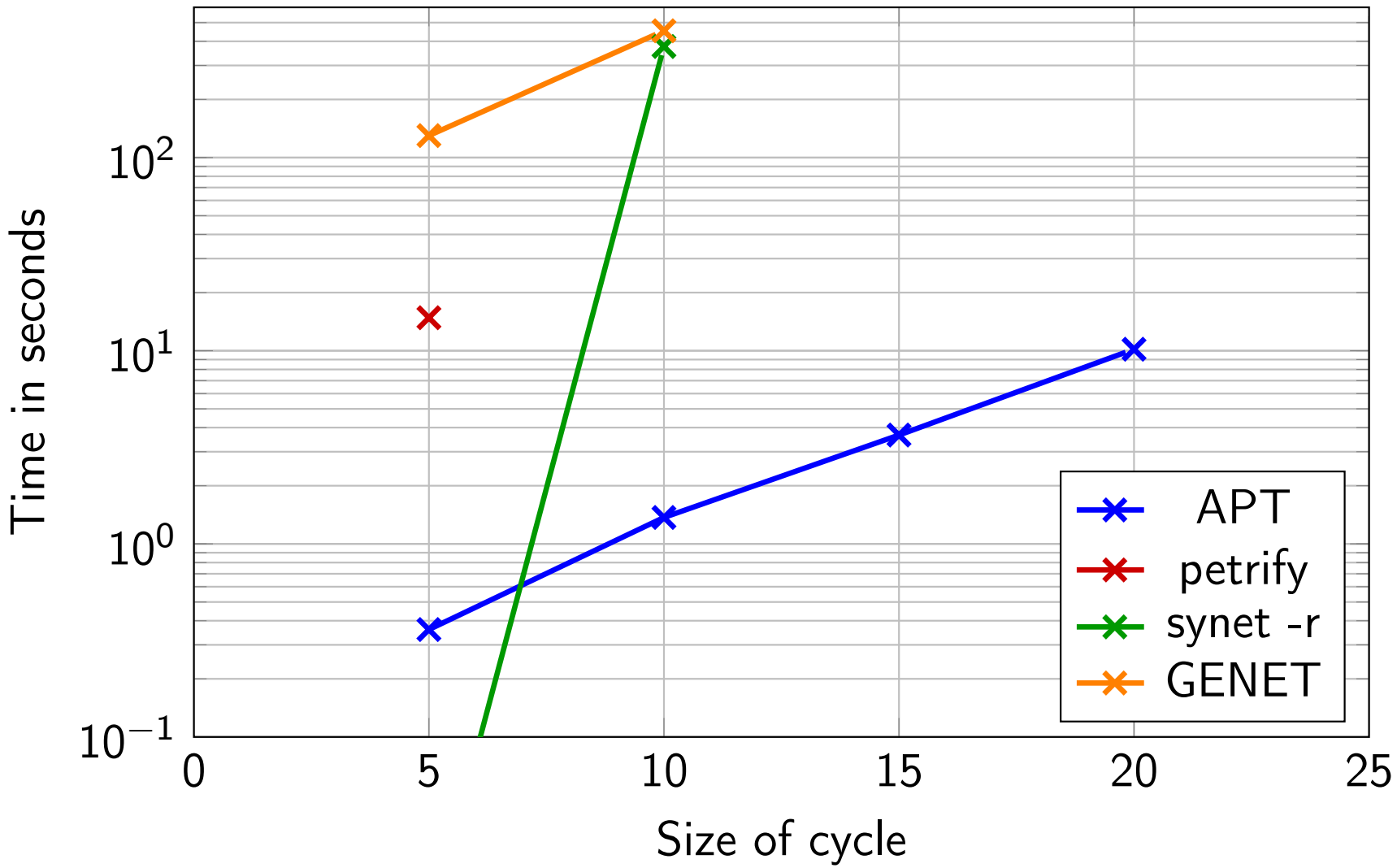


Synthesizing cycles with $k = 1$ tokens[4]: Run time



[4]Upper time limit: 600 sec

Synthesizing cycles with $k = 5$ tokens[5]: Run time



[5] Upper time limit: 600 sec

What's next?

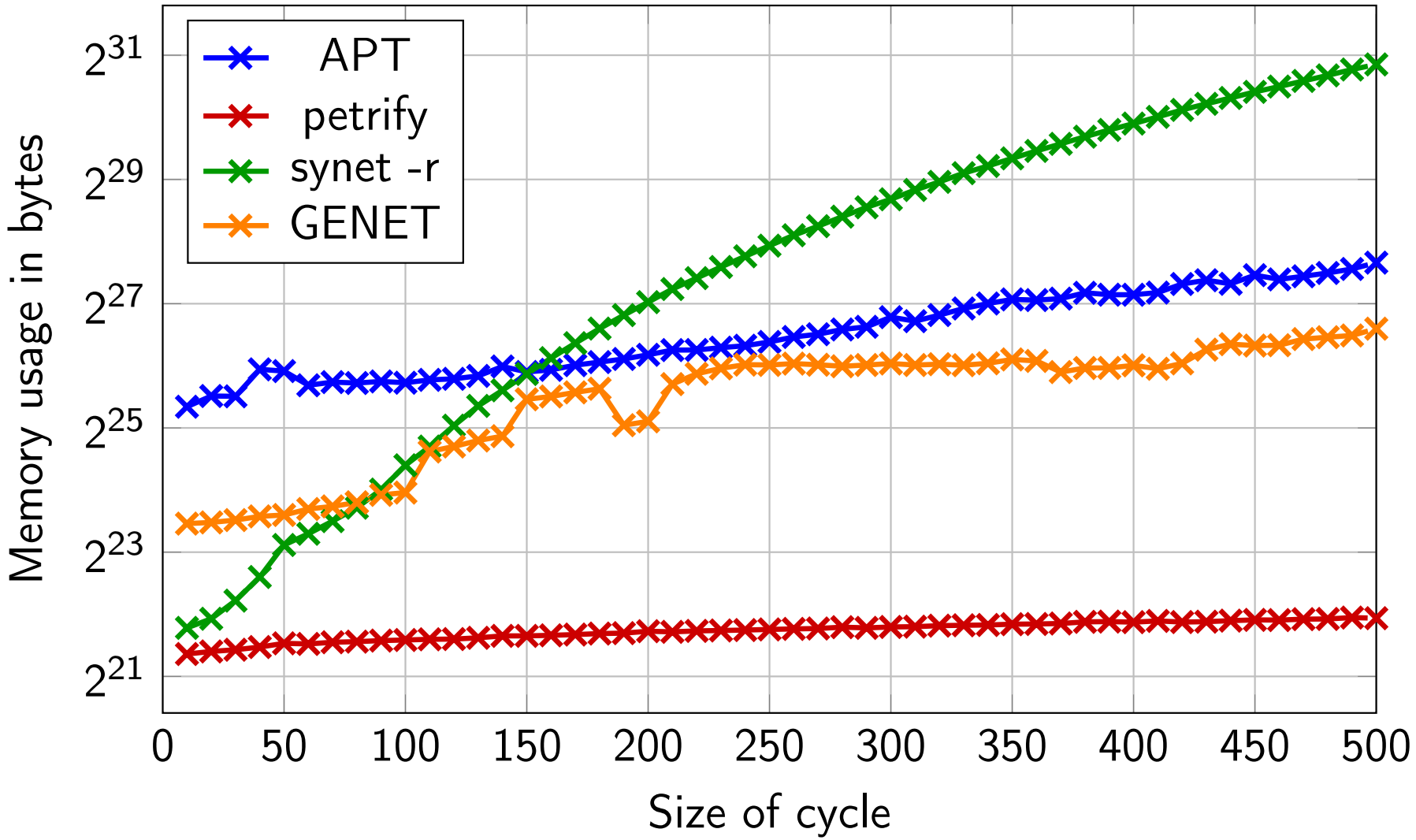
- Get APT on `http://github.com/CvO-theory/apt`
- Toolbox containing many algorithms
- APT is being extended
  - Optimize implementation
  - Regular expressions
  - Step reachability graph
  - Graphical user interface
  - ...

Appendix

Philosophers: ▸ Philosophers ▸ Time

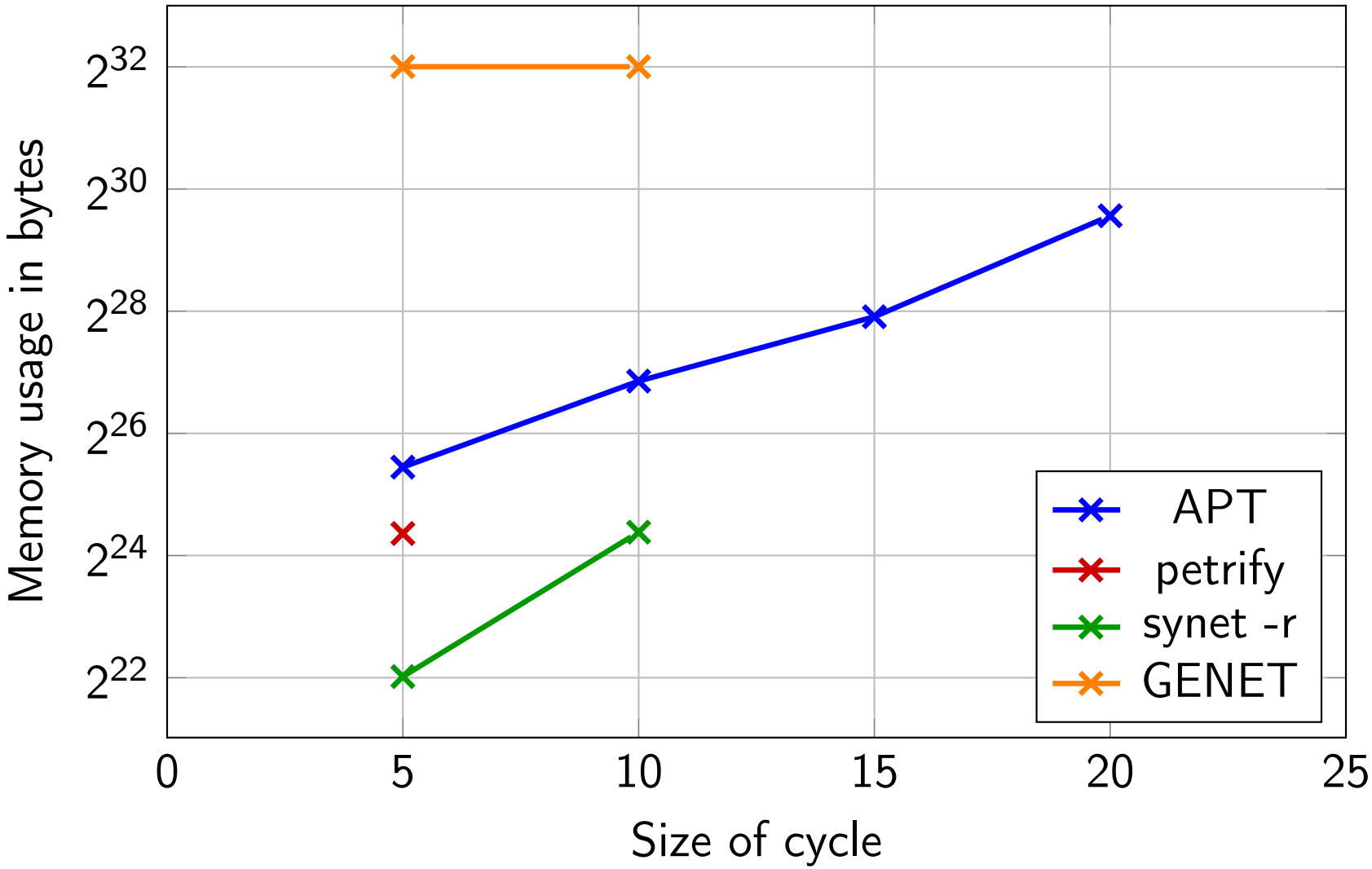
Memory usage: ▸ Cycles $k = 1$ ▸ Cycles $k = 5$ ▸ Bistate philosophers

Synthesizing cycles with $k = 1$ tokens[6]: Memory usage



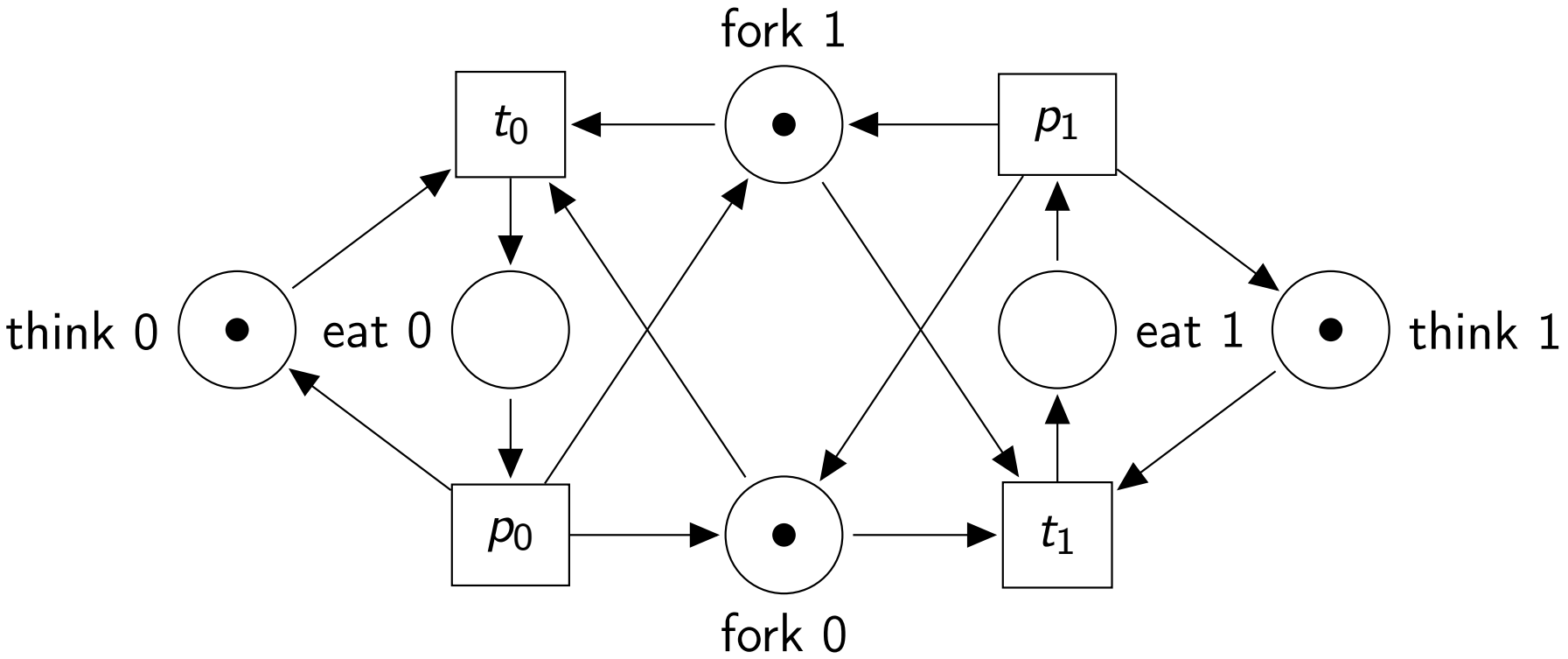
[6] Upper time limit: 600 sec

Synthesizing cycles with $k = 5$ tokens[7]: Memory usage

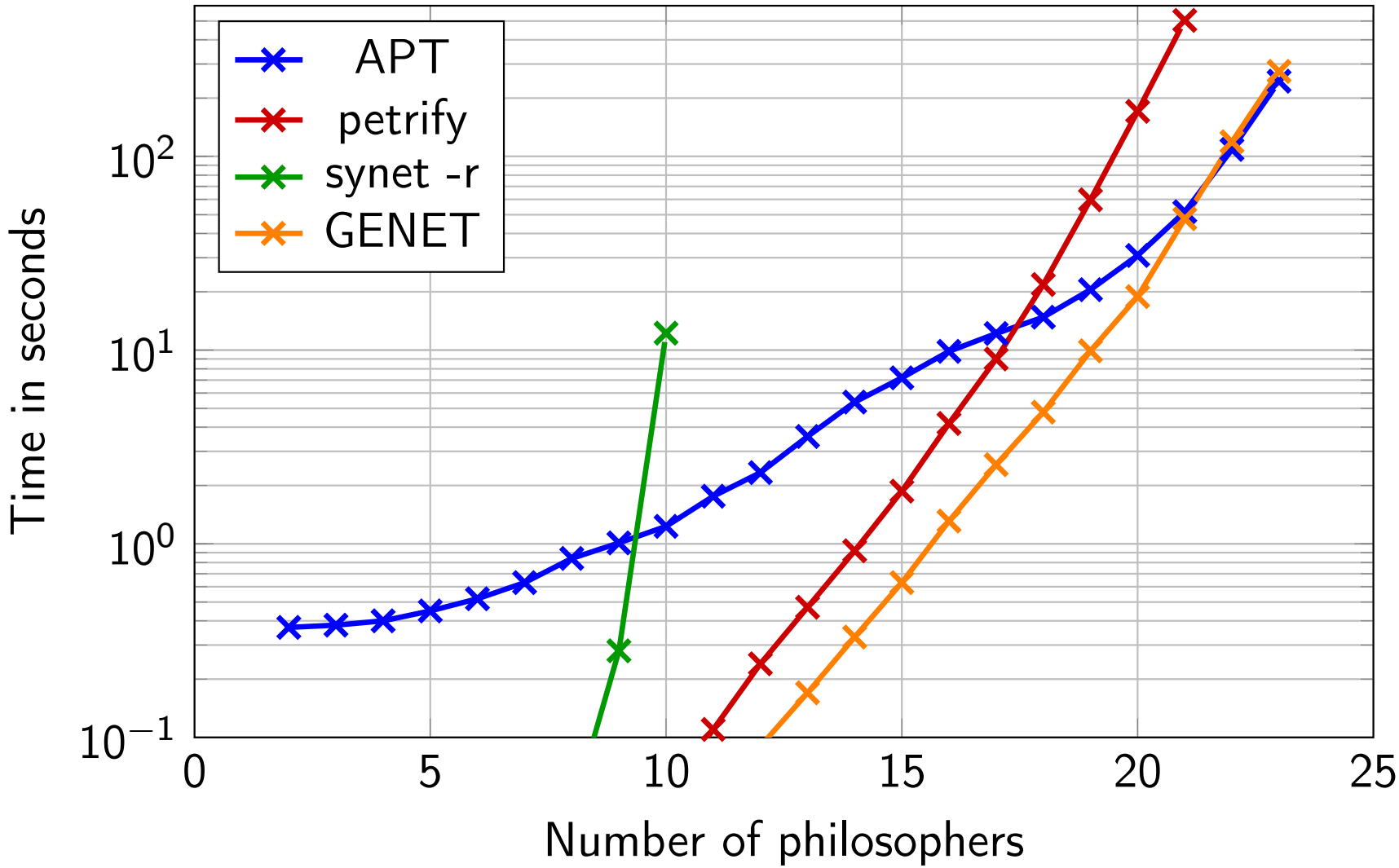


[7] Upper time limit: 600 sec

Two-state dining philosophers with $n = 2$

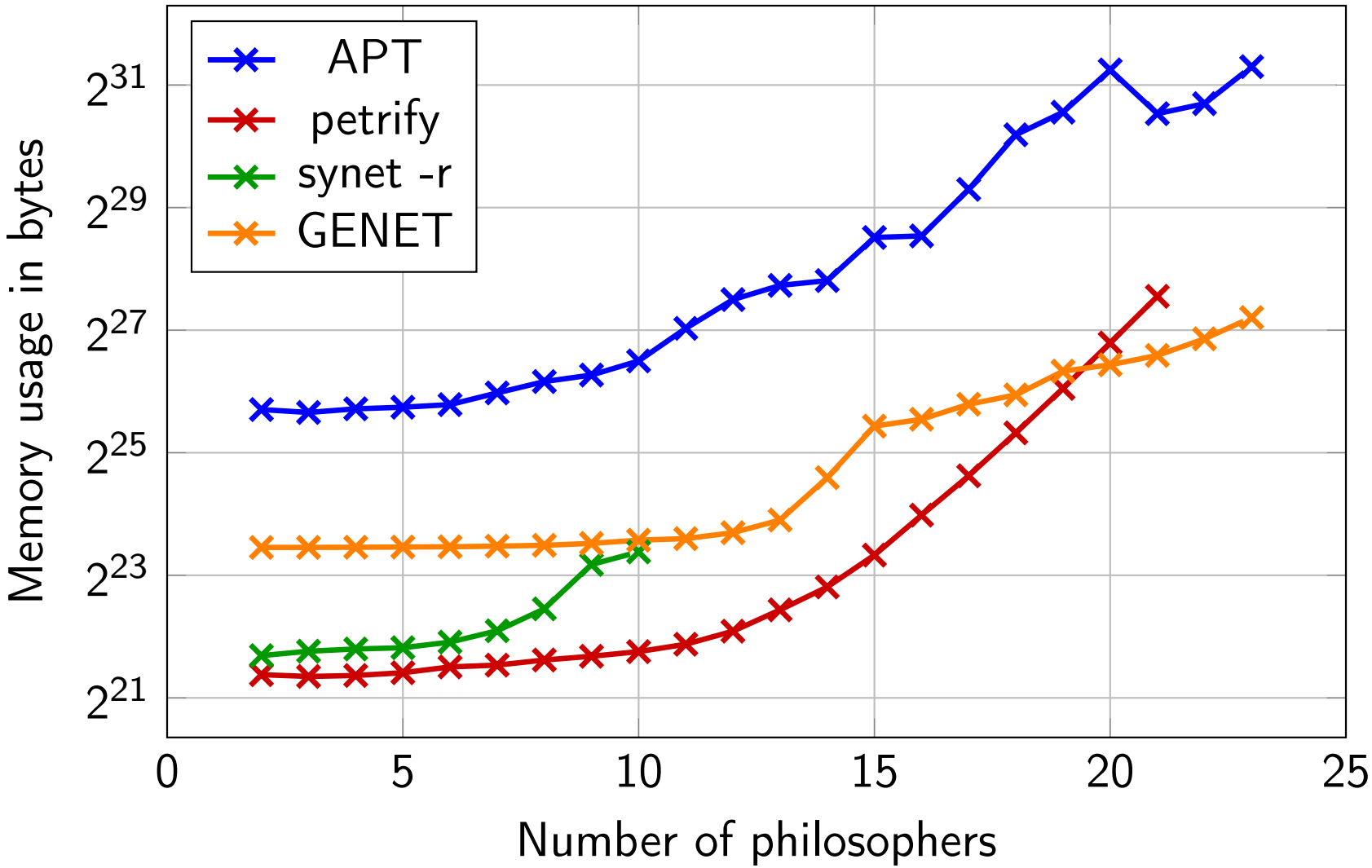


Synthesizing two-state dining philosophers[8]: Run time



[8] Upper time limit: 600 sec

Synthesizing two-state dining philosophers[9]: Memory usage



[9] Upper time limit: 600 sec