Menhir : Generic High-Speed FPGA Model-Checker

Emilien FOURNIER

Emilien.fournier@ensta-bretagne.org

Ciprian TEODOROV

Ciprian.teodorov@ensta-bretagne.fr

Loic LAGADEC

Loic.lagadec@ensta-bretagne.fr

Context

- Model-checking
  - Generic, intuitive automated formal method..
  - .. but limited by the state-space explosion problem
- New trend : **Decomposition of the verification task**
  - ➢ Trades-off memory ressources for computation time
- Large research effort to accelerate model-checking
  - Shared Memory Systems, Distributed Systems, SIMD

Significant speedups, but no game-changing breakthrough

Model-Checking

- Hardware acceleration
  - PHAST : safety model-checker, 200x speedup
  - FPGASwarm : Swarm safety model-checker, 1000x speedup

- Problems
  - Different handcoded HDL models
  - No support for existing modeling languages
  - Implicit interleaving between the semantics and the algorithm

Difficult to use in practice and the results are hard to compare

Menhir : A Generic High-Speed FPGA Model-checking Engine

- Isolate the model semantics from the verification engine
  - Support for off-the-shelf modeling languages
  - Easy to change the verification algorithm

- Parametric verification engine
  - ➢ Multiple algorithms : continuum from **exhaustive** to **partial verification**
  - ➢ Allow precise performance characterization

- Menhir prototype [today] : 3 languages & 6 verification algorithms

Algorithm

- Definitions
  - F : **Frontier** states to be processed
  - K : **Known** states
  - N : Frontier's states **Neighborhood**
- Initialisation :
  - Initial states added to the Neighborhood
- Fixed-point iterations unrolling the state-space

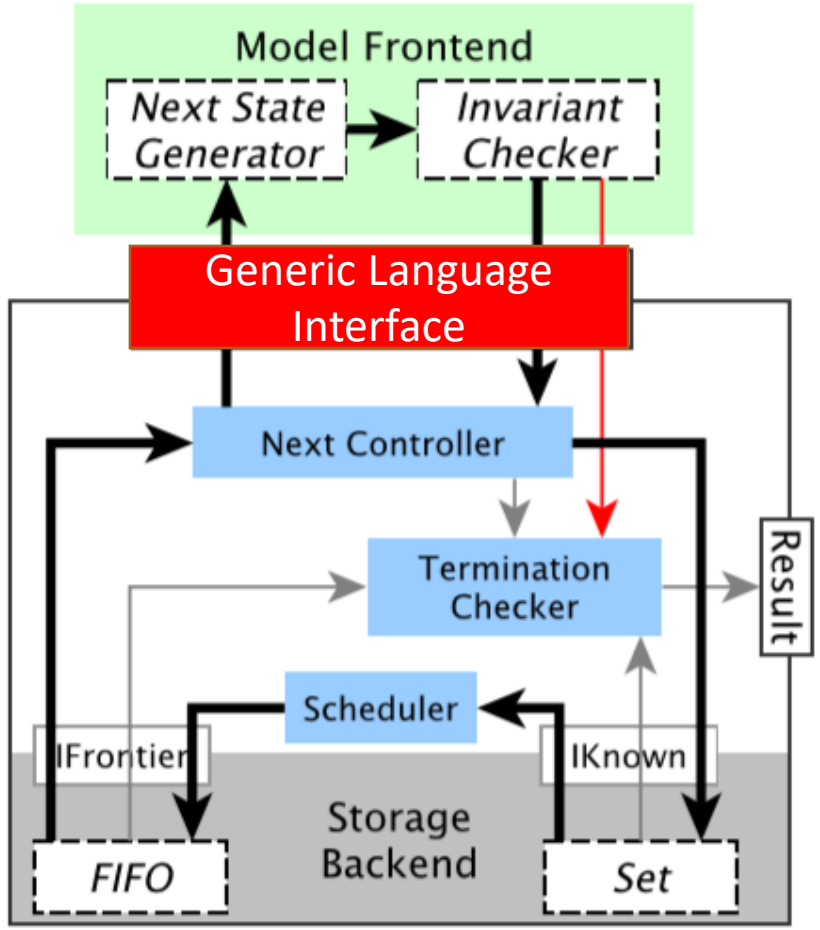
```
1  def safety_checker (m : 𝓜) : bool :=
2    𝒦 ← ∅
3    ℱ ← ∅
4    𝒩 ← m.initial
5    do
6      if ∃ n ∈ 𝒩, ¬ m.is_safe(n) then
7        return false
8      𝒦, ℱ ⇐ 𝒦 ∪ 𝒩, 𝒩 \ 𝒦
9      𝒩 ←{ n | ∀ x ∈ ℱ, n ∈ m.next(x) }
10   while ℱ ≠ ∅
11   return true
```

Menhir Verification Engine

- Pipeline structure

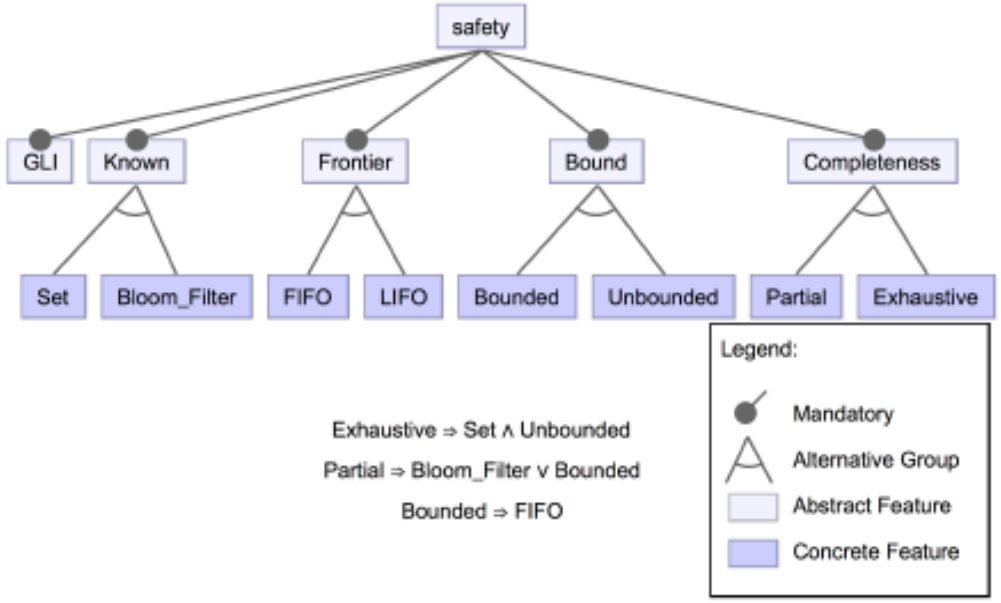
- Model Frontend
  - Isolated through the **Generic Language Interface** (GLI)
  - Encapsulates the model and the property
  - The only variable part between verification tasks

- Verification engine
  - Flexibility through generic interfaces
  - IKnown : Hardware set representation
  - IFrontier : Hardware priority queue representation



Verification Engine

- 6 safety algorithms implemented
  - Exhaustive safety
  - Partial safety (bitstate hashing)
  - Bounded model-checking



Experimental setup

- Zynq 7020
  - Dual-core arm A9 CPU, 666MHz
  - 7-series FPGA, 100MHz
    - 85K Logic Cells
    - 4.9Mb BRAM

- Parametric model

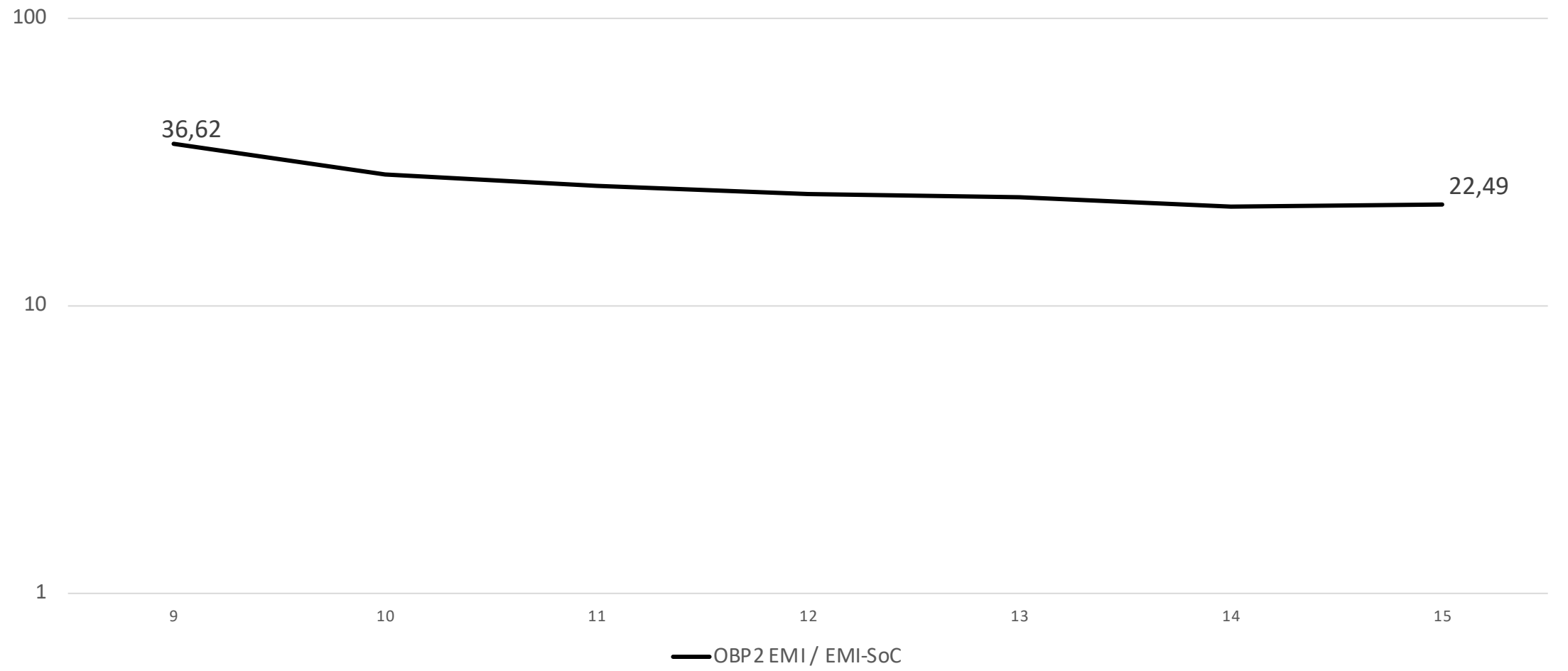
$$\begin{aligned}&\texttt{def}\ \ \texttt{nbits}\ (w \in \mathbb{N}^+)\ :\ \ \mathcal{M}_H\ \mathcal{C}\ \ :=\\&\langle \textbf{initial} \leftarrow \{\ (n,T)\mid \forall\ i\in[0,w),\ n_i=0\ \},\\&\ \textbf{next} \leftarrow \lambda s,\ \{\ (n,T)\mid \exists\ i\in[0,w),\ n_i=\neg s_i\ \}\rangle\end{aligned}$$

Experimental setup

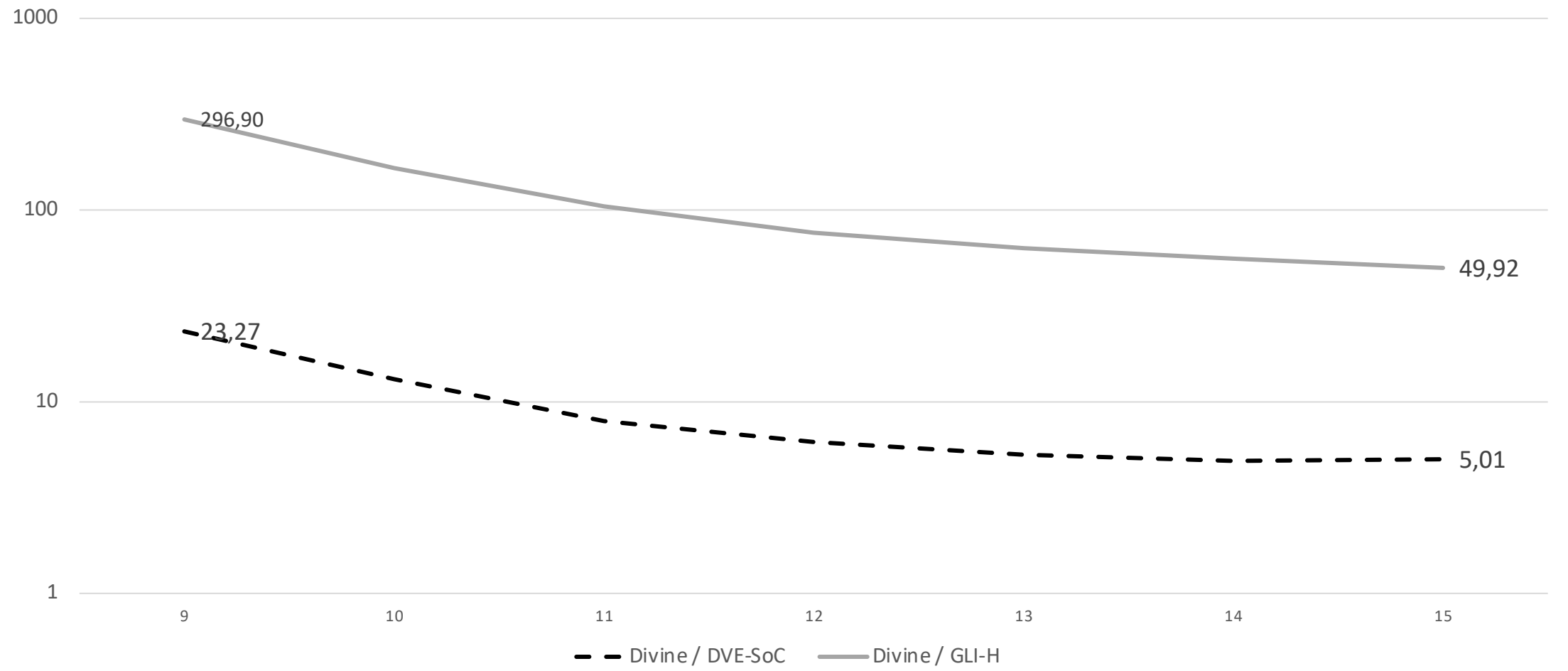
- Baseline :
  - Divine3 model-checker running on one arm-A9 core
  - OBP2 running on the arm

- Scenarios :
  - EMI-UML
    - C-compiled baremetal interpreter for UML
  - DVE
    - C-compiled model from DVE
    - Used by Divine3
  - GLI native
    - VHDL handwriten model
    - 1 Clk « response time »

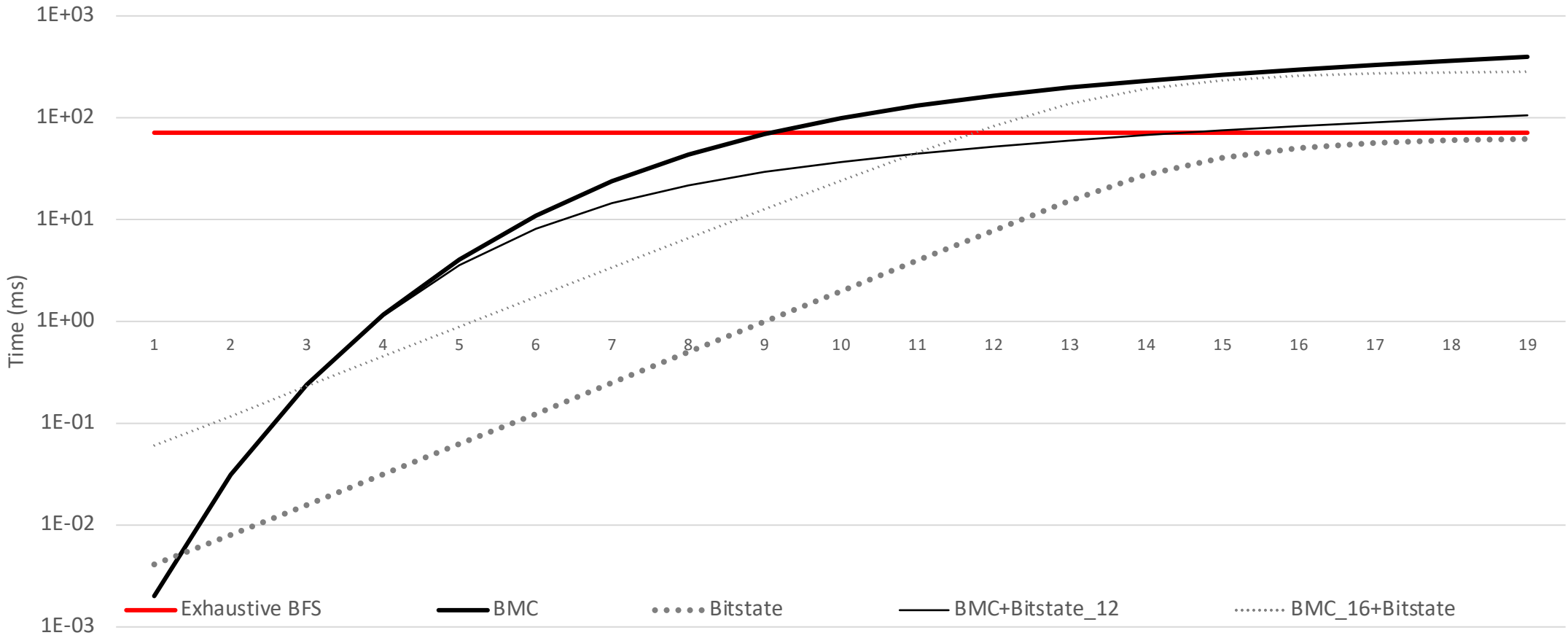
Results : Exhaustive verification



Results : Exhaustive verification



Results : Algorithms comparison



Conclusion

- Highly modular hardware model-checker
- 50x speedup vs high-performance software model-checker (Divine 3)
- Future work :
  - Performance optimisation
  - LTL model-checking support