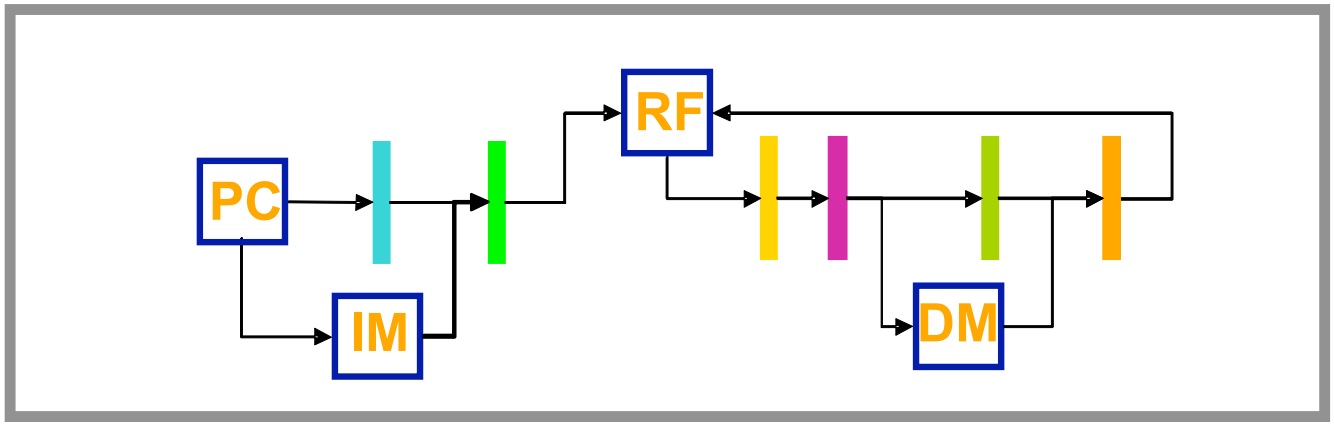
Automatic Memory Reductions for RTL Model Verification

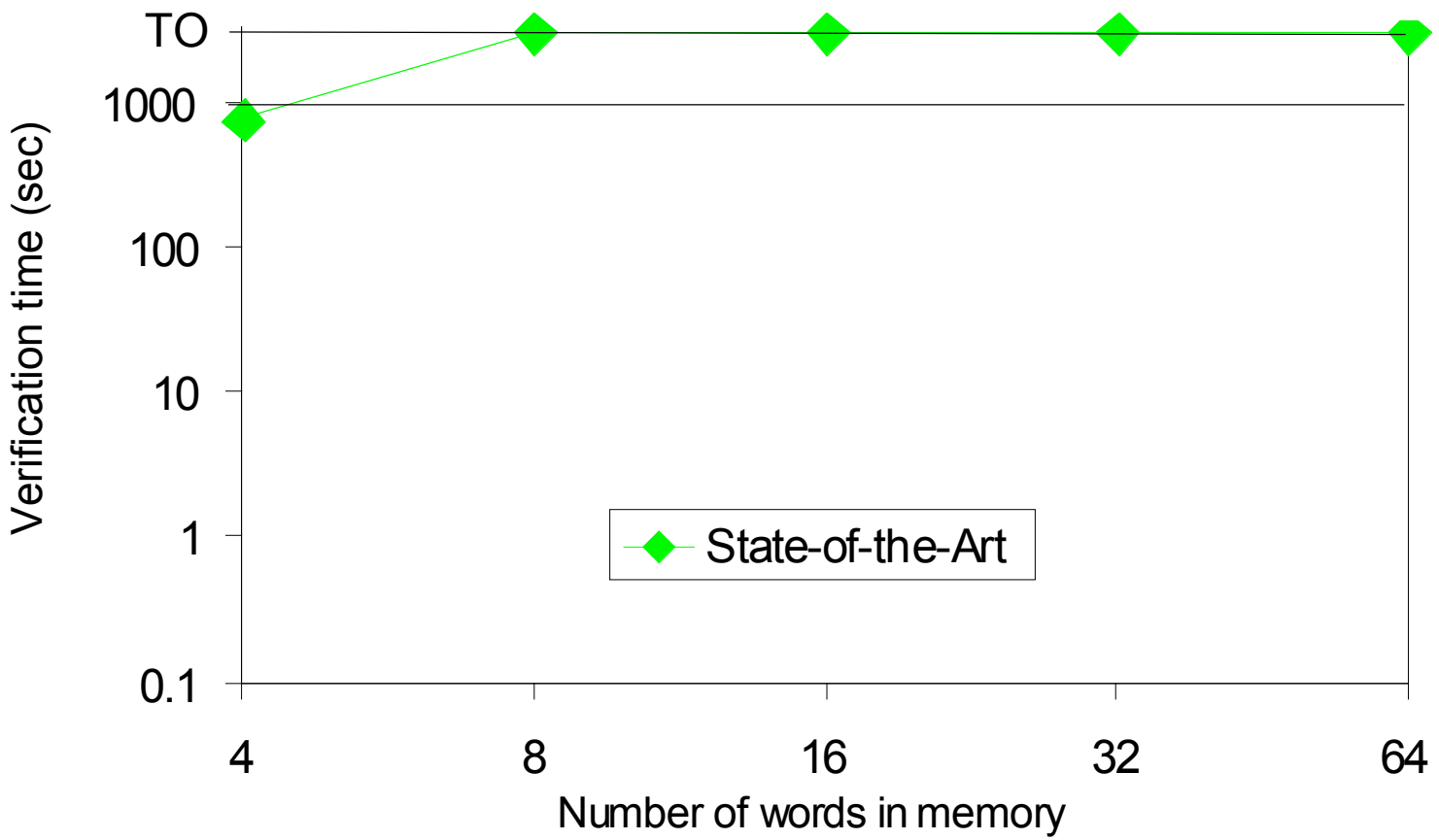
Panagiotis Manolios
Sudarshan K. Srinivasan
Daron Vroon

Motivation

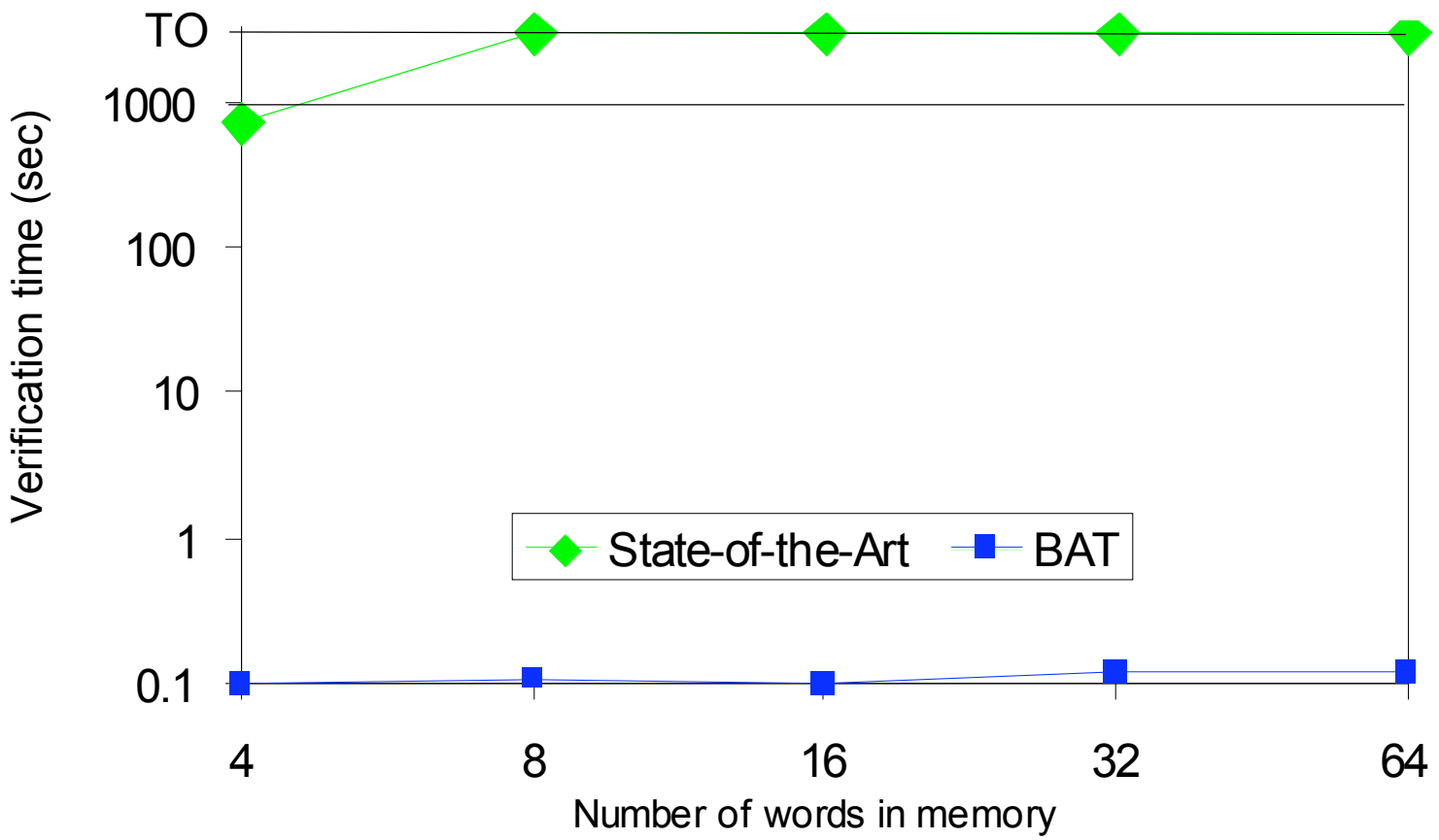


- Pipelined machine verification
  - State of the art: term level models
  - *The* major limitation
  - We really want to verify RTL-level models
  - RTL models too hard for state of the art
  - We developed BAT, Bit-level Analysis Tool

2 Stage Pipelined Machine



2 Stage Pipelined Machine



Contributions

- Automatic and efficient memory abstraction
  - Memories are first class objects
  - Memory comparisons allowed in all contexts
  - Memories can be passed to and returned from functions
- Incorporation of term-rewriting techniques
  - Decrease size of abstract memories
  - Drastic improvements in verification time
- Implemented in BAT
  - Extensive validation of techniques

Outline

- Specification Language
- Bit-level Analysis Tool (BAT)
- Memory Abstraction
  - Memory Reduction Algorithm
  - Memory Rewriting
- Results
- Conclusions

Outline

- Specification Language
- Bit-level Analysis Tool (BAT)
- Memory Abstraction
  - Memory Reduction Algorithm
  - Memory Rewriting
- Results
- Conclusions

BAT Specification Language

- Strongly typed
- Type inference
- Function definitions allowed
- Powerful Lisp-based language
- Syntax extensions enabled by Lisp
- Parameterized models are easy to define
- Target language for Verilog or VHDL
- Bounded model checker & *k*-induction engine

Memory Usage Example

Applicative Memory Operations:

`(get m a)` : Get the value in memory `m` at address `a`

`(set m a v)` : New memory, like `m`, except address `a` has value `v`

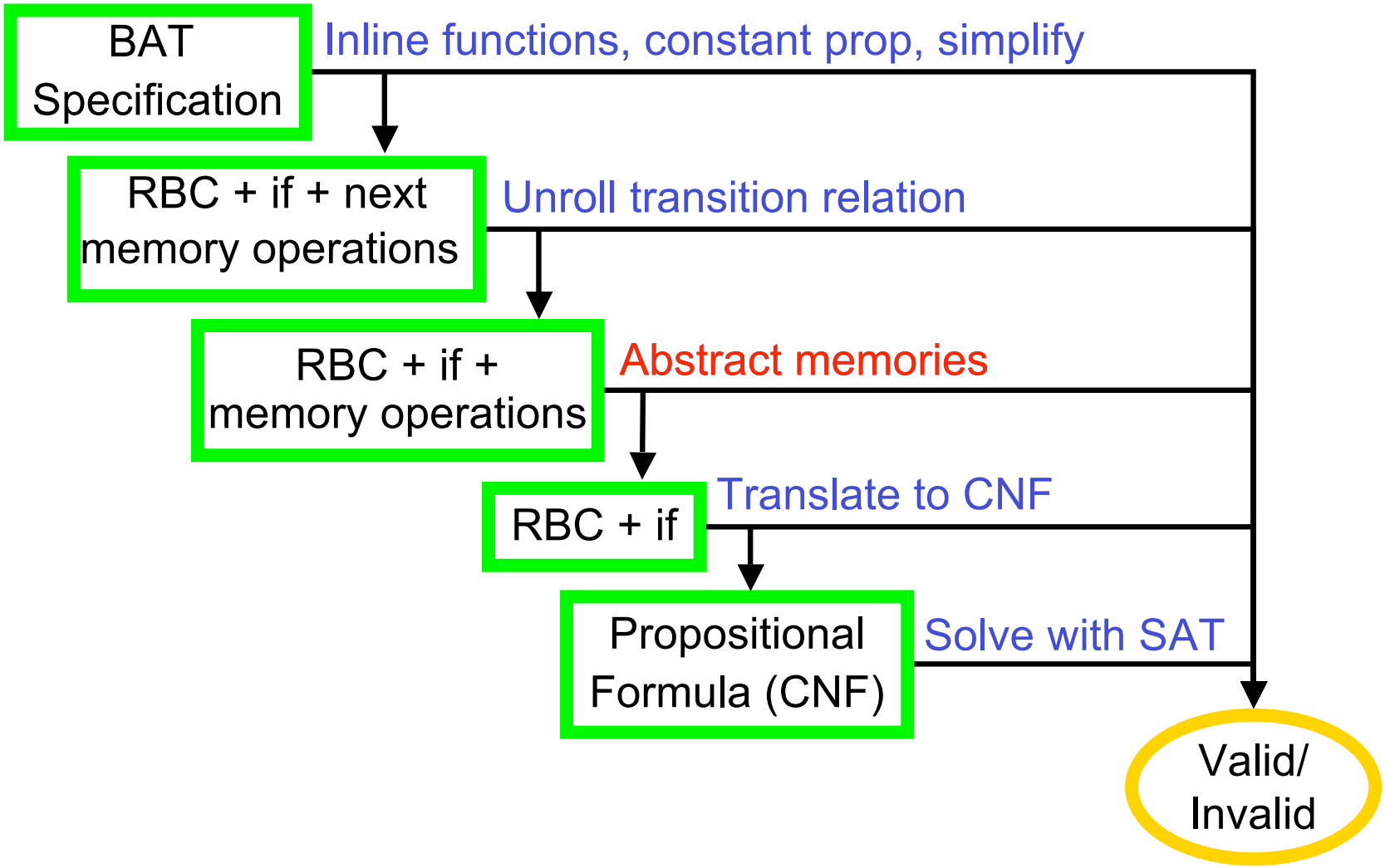
```
Vars:     (mem 8 4)
          (adr 3)
          (val 4)

Formula:(= (get (set mem adr val) adr)
           val)
```

Outline

- Specification Language
- Bit-level Analysis Tool (BAT)
- Memory Abstraction
  - Memory Reduction Algorithm
  - Memory Rewriting
- Results
- Conclusions

Bit-level Analysis Tool (BAT)



Outline

- Specification Language
- Bit-level Analysis Tool (BAT)
- Memory Abstraction
  - Memory Reduction Algorithm
  - Memory Rewriting
- Results
- Conclusions

Previous Work

- Ganai et al. approach
  - Malay K. Ganai, Aarti Gupta, Pranav Ashar: Verification of Embedded Memory Systems using Efficient Memory Modeling. DATE 2005.
- UCLID approach
  - Randal E. Bryant, Shuvendu K. Lahiri, Sanjit A. Seshia: Modeling and Verifying Systems Using a Logic of Counter Arithmetic with Lambda Expressions and Uninterpreted Functions. CAV 2002.
- Limited to reads and writes
- Memories are *not* first class objects
  - Cannot be passed to functions
  - Cannot be directly compared

Limitations of Previous Work

In some contexts, memories can be compared for equality using memory reads

$$(= \quad (\text{set } m_1\ a_1\ v_1) \quad (\text{set } m_2\ a_1\ v_2) \quad)$$

Limitations of Previous Work

In some contexts, memories can be compared for equality using memory reads

```
(= (get (set m1 a1 v1) a)
   (get (set m2 a1 v2) a))
```

Limitations of Previous Work

In some contexts, memories can be compared for equality using memory reads

```
(= (get (set m1 a1 v1) a)
   (get (set m2 a1 v2) a))
```

Memories cannot be compared in all contexts

```
(not (= (get (set m1 a1 v1) a)
        (get (set m2 a1 v2) a)))
```

BAT Memory Abstraction

Memories are treated as first class objects

```
(=      (set m1 a1 v1)
        (set m2 a1 v2)   )
```

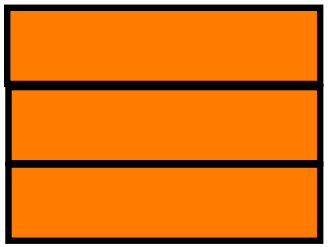
Memories can be directly compared in all contexts

```
(not (=      (set m1 a1 v1)
             (set m2 a1 v2)   ))
```

BAT Memory Abstraction

$(\texttt{get}\ (\texttt{set}\ (\texttt{set}\ \texttt{m}\ a_1\ v_1)\ a_2\ v_2)\ a_3)$

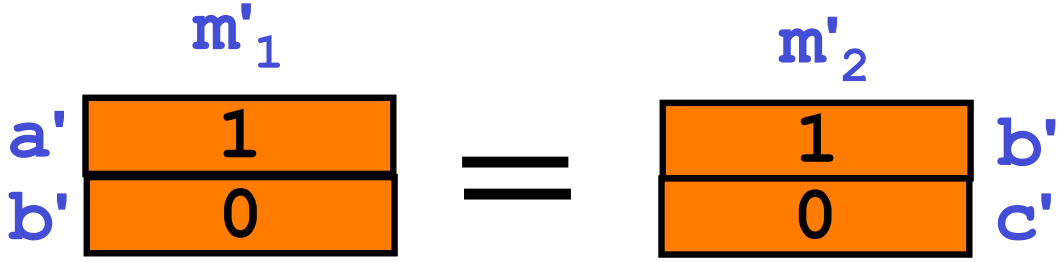
Abstracted memory



- Determine number of unique gets and sets (*n*)
- Generate abstract memory consisting of *n* words
- Apply abstraction to original addresses
- Note: size of abstract addresses is lg(*n*)

BAT Memory Abstraction

```
(not (= (set (set m1 b 0) a 1)
        (set (set m2 b 1) c 0)))
```



- Cannot abstract memories m_1 and m_2 in isolation.
- Memories have to be abstracted together if they are:
  - Compared for equality
  - Appear in the same context

Memory Equivalence Classes

Base memories of a memory expression:

1. *base((if e m_1 m_2)) = base(m_1) U base(m_2)*
2. *base((set m a v)) = base(m)*
3. *base(m) = {m}, where m is a variable*

Memory Equivalence Classes

Base memories of a memory expression:

1. *base((if e m_1 m_2)) = base(m_1) U base(m_2)*
2. *base((set m a v)) = base(m)*
3. *base(m) = {m}, where m is a variable*

base((set (set m_1 b 0) a 1)) = {m_1}

Memory Equivalence Classes

Base memories of a memory expression:

1. *base((if e m_1 m_2)) = base(m_1) U base(m_2)*
2. *base((set m a v)) = base(m)*
3. *base(m) = {m}, where m is a variable*

base((set (set m_1 b 0) a 1)) = {m_1}

base((if (= m_1 m_2) m_3 (if e_2 m_4 m_5))) = {m_3,m_4,m_5}

Memory Equivalence Classes

Base memories of a memory expression:

1. *base((if e m_1 m_2)) = base(m_1) U base(m_2)*
2. *base((set m a v)) = base(m)*
3. *base(m) = {m}, where m is a variable*

Equality test relation R_f = {m_1,m_2} such that

1. *e is in f and m_1, m_2 are in base(e), or*
2. *(= e_1 e_2) is in f, m_1 is in base(e_1), and m_2 is in base(e_2)*

Memory Equivalence Classes

Base memories of a memory expression:

1. *base((if e m_1 m_2)) = base(m_1) U base(m_2)*
2. *base((set m a v)) = base(m)*
3. *base(m) = {m}, where m is a variable*

Equality test relation R_f = {m_1,m_2} such that

1. *e is in f and m_1, m_2 are in base(e), or*
2. *(= e_1 e_2) is in f, m_1 is in base(e_1), and m_2 is in base(e_2)*

(and (= m_1 m_2)(= m_2 m_3))

Memory Equivalence Classes

Base memories of a memory expression:

1. *base((if e m_1 m_2)) = base(m_1) U base(m_2)*
2. *base((set m a v)) = base(m)*
3. *base(m) = {m}, where m is a variable*

Equality test relation $R_f = \{m_1, m_2\}$ such that

1. *e is in f and m_1, m_2 are in base(e), or*
2. *(= e_1 e_2) is in f, m_1 is in base(e_1), and m_2 is in base(e_2)*

E_f is the transitive closure of R_f

E_f is an equivalence relation

Memory variables partitioned into ≡-classes induced by E_f

Memory Equivalence Classes

- Abstract memory **m** with memory **m′** that has *n* words:
  - *n*: total number of accesses to all memory variables in the equivalence class (*C*) of m.

1.(= m_1 (if e m_2 m_3))
2.(and (= m_1 m_2)(= m_2 m_3))

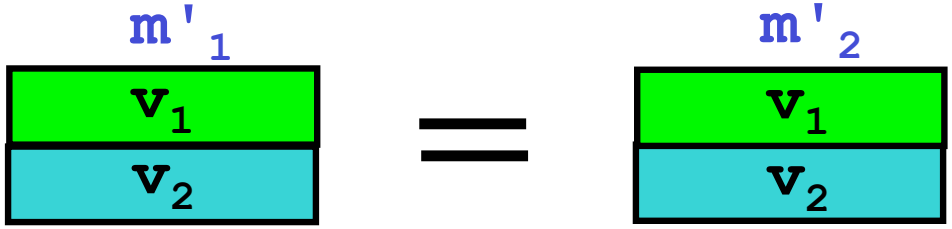
- Example:
  - m_1: 10 accesses;
  - m_2: 10 accesses;
  - m_3: 5 accesses
  - m_1, m_2, m_3 are abstracted using memories with 25 words.

BAT Memory Abstraction

$$(= \ (\text{set} \ (\text{set} \ m_1 \ a_1 \ v_1) \ a_2 \ v_2) \\ (\text{set} \ (\text{set} \ m_2 \ a_1 \ v_1) \ a_2 \ v_2))$$

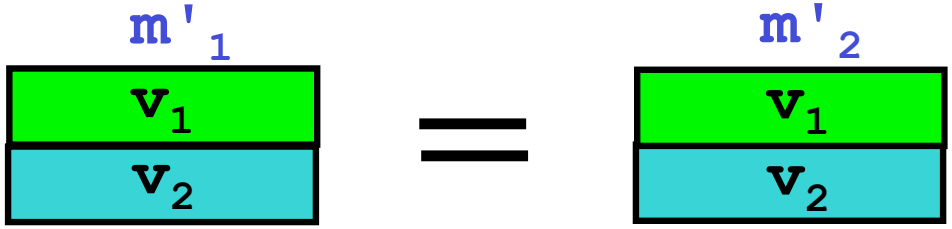
BAT Memory Abstraction

$$(= \ (set \ (set \ m_1 \ a_1 \ v_1) \ a_2 \ v_2) \ (set \ (set \ m_2 \ a_1 \ v_1) \ a_2 \ v_2))$$



BAT Memory Abstraction

$$(= \ (\text{set} \ (\text{set} \ m_1 \ a_1 \ v_1) \ a_2 \ v_2) \\ (\text{set} \ (\text{set} \ m_2 \ a_1 \ v_1) \ a_2 \ v_2))$$



Problem: Original formula not equisatisfiable with memory abstracted formula

BAT Memory Abstraction

- Abstract memories are comprised of two components
  - *m'* : memory with *n* words
  - *b* : bit-vector with lg(C_n) bits
- Bit-vector b is used to represent the unconstrained words of m
- Algorithm for abstracting *f* with *f'* appears in paper
- Theorem: *f* is satisfiable iff *f'* is satisfiable
- Size of formula generated depends on sets and gets

Outline

- Specification Language
- Bit-level Analysis Tool (BAT)
- Memory Abstraction
  - Memory Reduction Algorithm
  - Memory Rewriting
- Results
- Conclusions

Memory Rewriting: Example

(set m a_0
 (get (set (set m a_1 v_1) a_2 v_2) a_0))

= [RW1]

(set m a_0
 (get (set m a_1 v_1) a_0))

= [RW1]

(set m a_0
 (get m a_0))

= [RW2]

m

$a_0 \neq a_1$; $a_0 \neq a_2$; $a_1 \neq a_2$

Memory Rewriting

Rewrite Rule 1

$$(\texttt{get}\ (\texttt{set}\ \texttt{m}\ \texttt{a}_1\ \texttt{v})\ \texttt{a}_2) \quad =$$

Memory Rewriting

Rewrite Rule 1

$$(\mathtt{get}\ (\mathtt{set}\ \mathtt{m}\ \mathtt{a}_1\ \mathtt{v})\ \mathtt{a}_2) = \mathtt{a}_1 = \mathtt{a}_2 : \mathtt{v}$$

Memory Rewriting

Rewrite Rule 1

$$(\text{get}\ (\text{set}\ m\ a_1\ v)\ a_2) =$$

$$a_1 = a_2 \quad : v$$

$$a_1 \neq a_2 \quad : (\text{get}\ m\ a_2)$$

Memory Rewriting

Rewrite Rule 1

```
(get (set m a1 v) a2)   =
 a1 = a2     : v
 a1 ≠ a2     : (get m a2)
(if (= a1 a2) v (get m a2))
```

Memory Rewriting

Rewrite Rule 1

```
(get (set m a1 v) a2)   =
 a1 = a2     : v
 a1 ≠ a2     : (get m a2)
(if (= a1 a2) v (get m a2))
```

Rewrite Rule 2

```
(set m a1 (get m a2))   =
```

Memory Rewriting

Rewrite Rule 1

```
(get (set m a1 v) a2)   =
 a1 = a2     : v
 a1 ≠ a2     : (get m a2)
(if (= a1 a2) v (get m a2))
```

Rewrite Rule 2

```
(set m a1 (get m a2))   =
 a1 = a2 : m
```

Memory Rewriting

Rewrite Rule 3

$(\texttt{get}\ (\texttt{if}\ e_1\ m_1\ m_2)\ a_2) \quad =$

Memory Rewriting

Rewrite Rule 3

$$(\texttt{get}\ (\texttt{if}\ e_1\ m_1\ m_2)\ a_2) = (\texttt{if}\ e_1\ (\texttt{get}\ m_1\ a_2)\ (\texttt{get}\ m_2\ a_2))$$

Memory Rewriting

Rewrite Rule 3

$$(\texttt{get}\ (\texttt{if}\ e_1\ m_1\ m_2)\ a_2) = (\texttt{if}\ e_1\ (\texttt{get}\ m_1\ a_2)\ (\texttt{get}\ m_2\ a_2))$$

m_1: 10 accesses

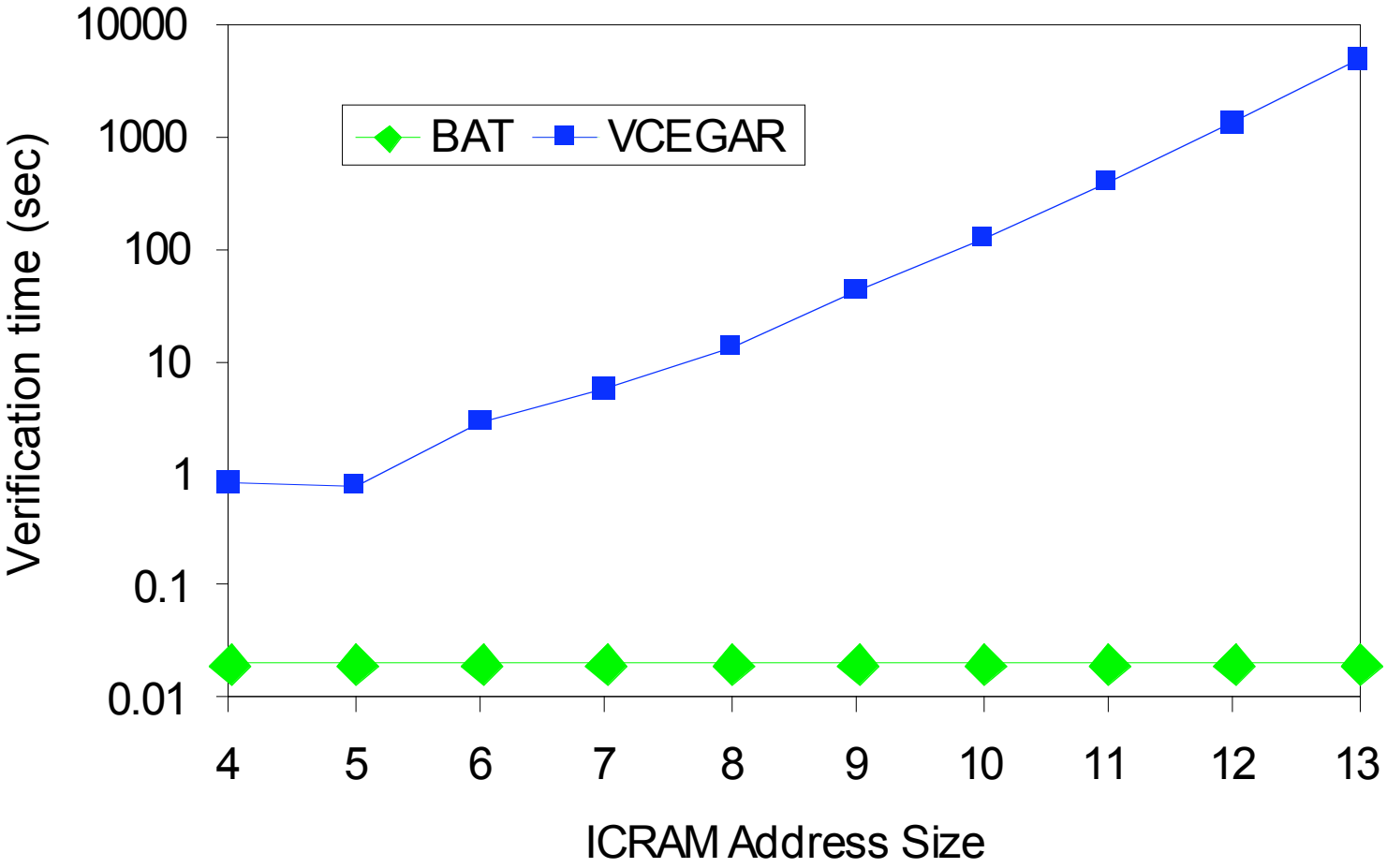
m_2: 10 accesses

Abstraction of m_1 and m_2 reduced from 20 to 10

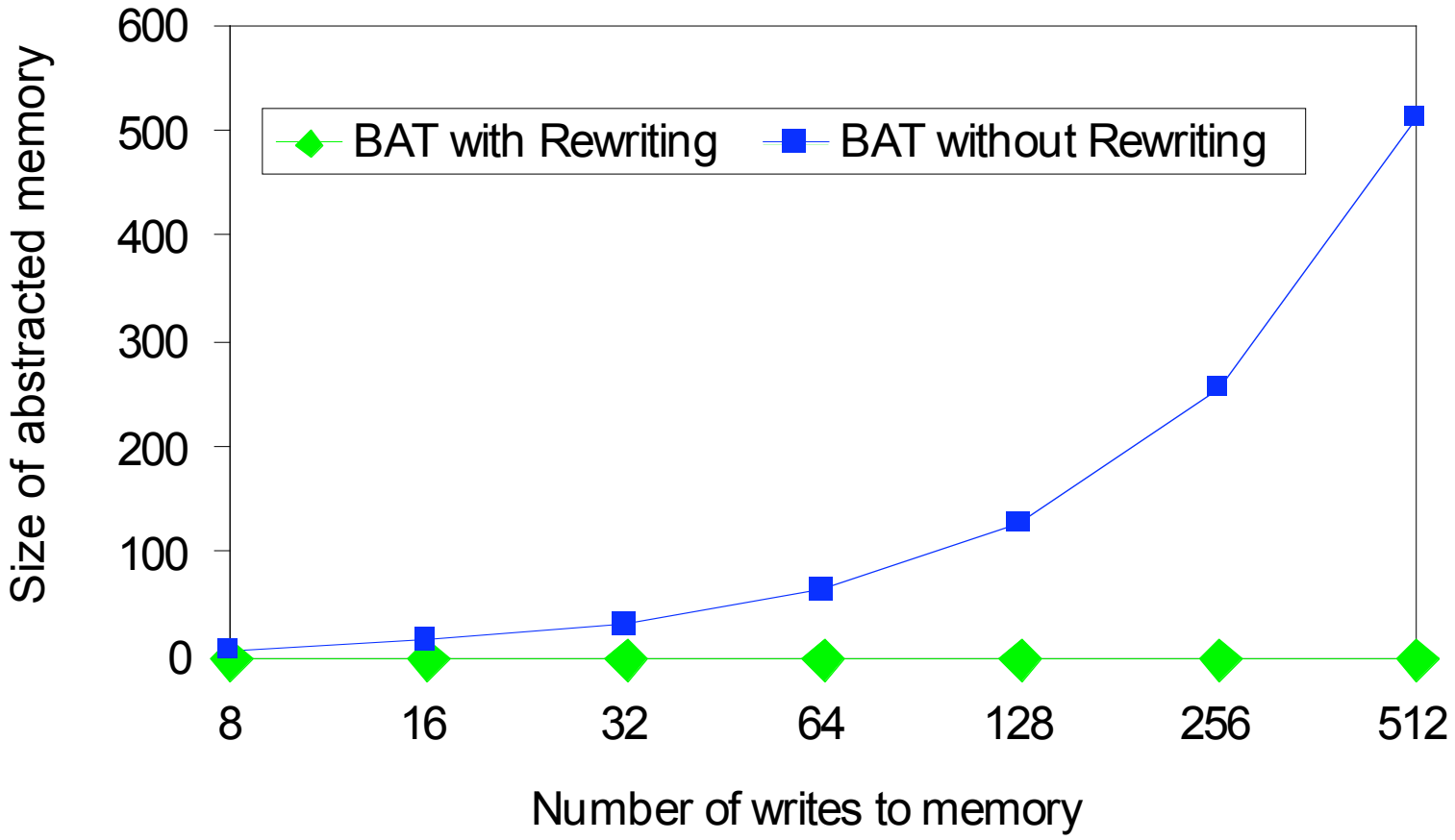
Outline

- Specification Language
- Bit-level Analysis Tool (BAT)
- Memory Abstraction
  - Memory Reduction Algorithm
  - Memory Rewriting
- Results
- Conclusions

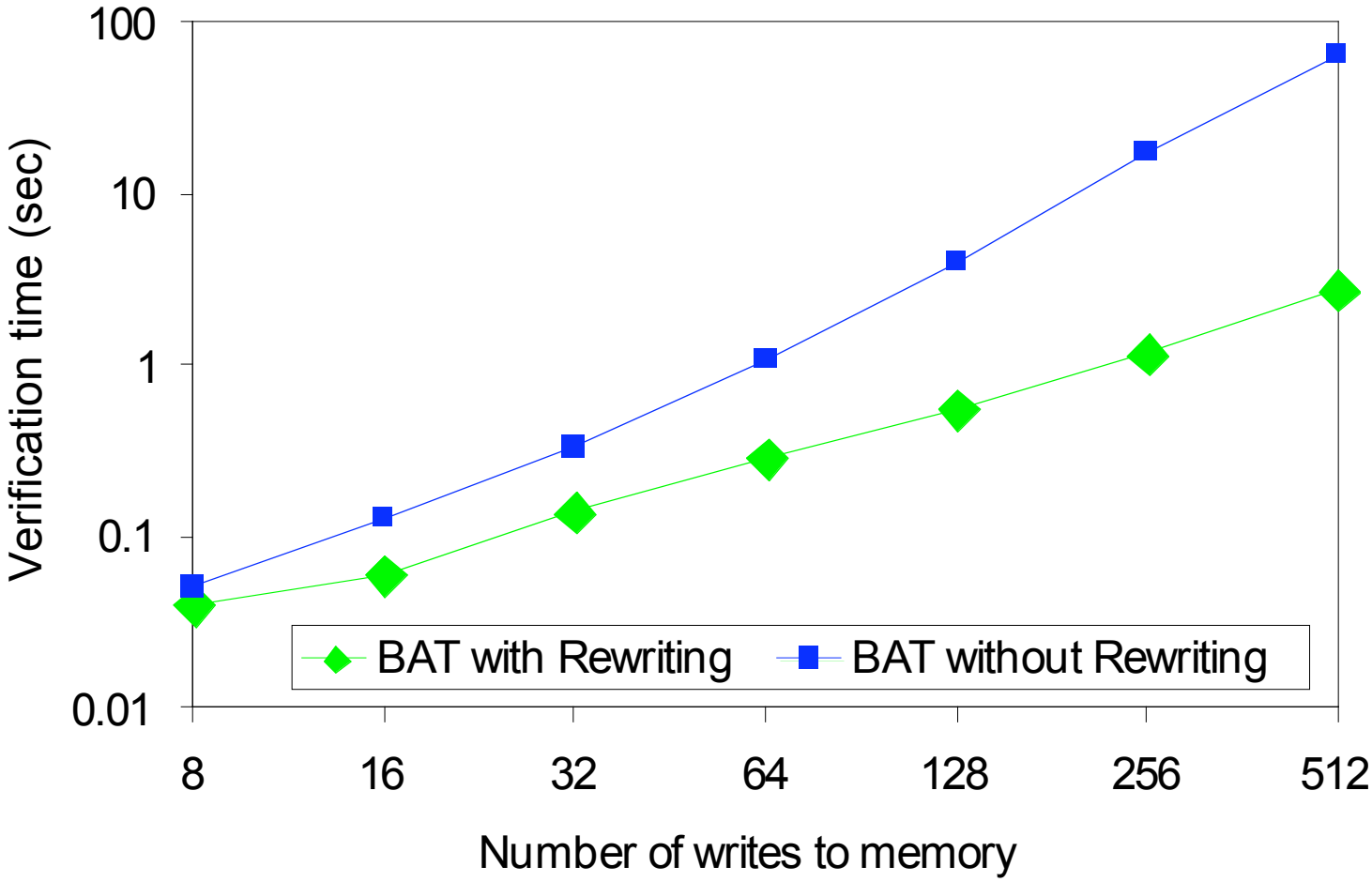
ICRAM Benchmarks



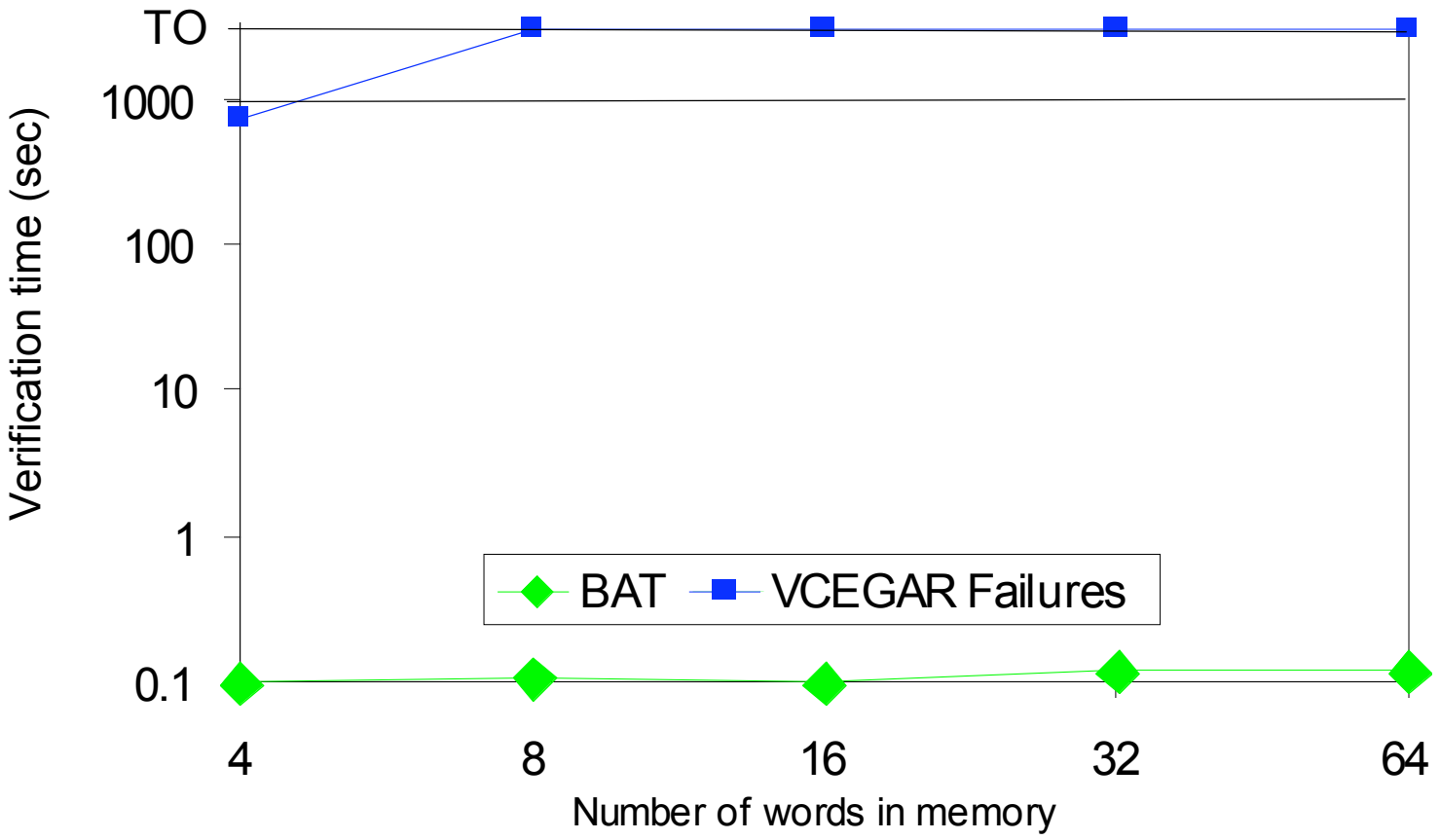
Out-of-Order Benchmarks



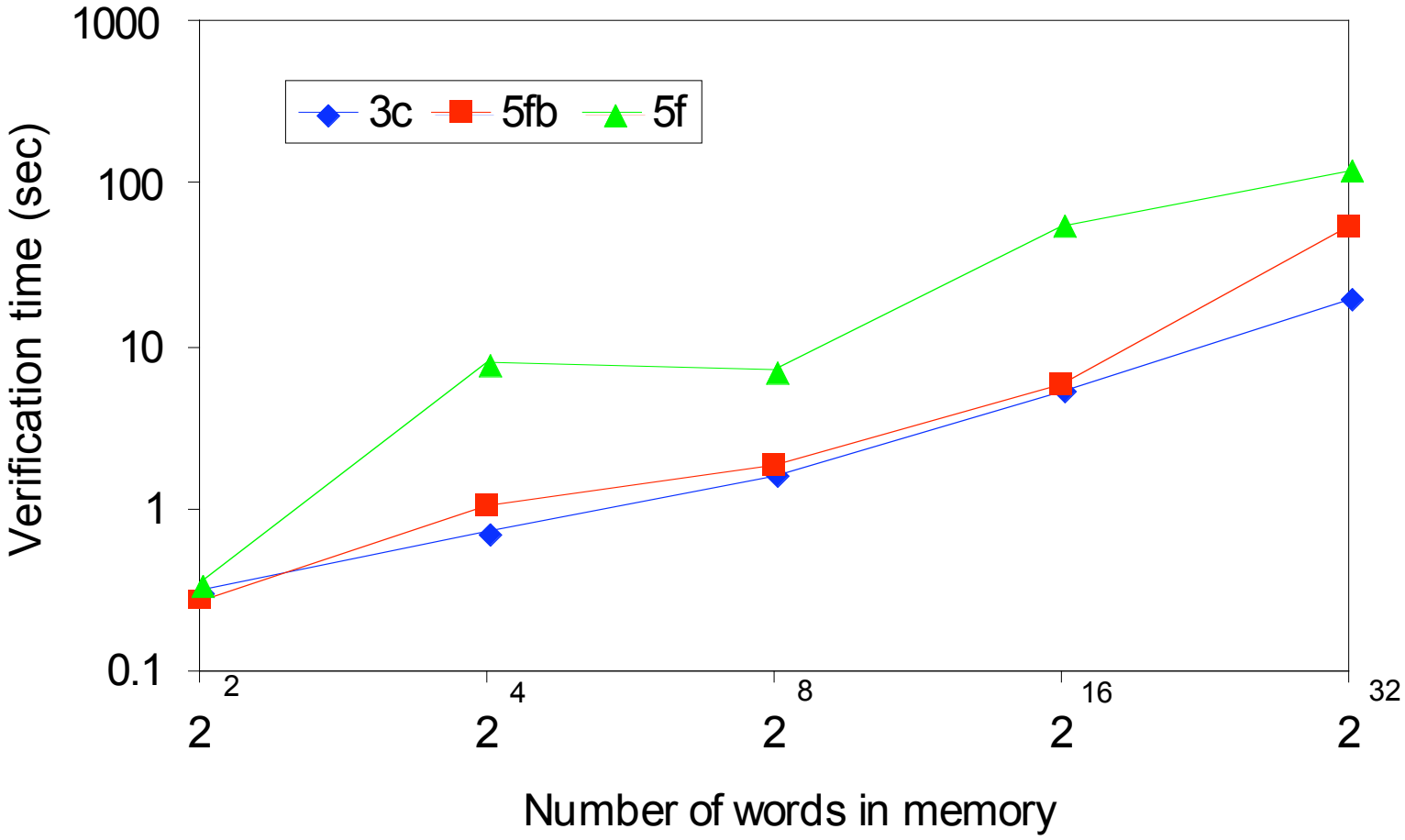
Out-of-Order Benchmarks



2 Stage Pipelined Machine



Pipelined Machine Benchmarks



Outline

- Specification Language
- Bit-level Analysis Tool (BAT)
- Memory Abstraction
  - Memory Reduction Algorithm
  - Memory Rewriting
- Results
- Conclusions

Conclusions and Future Work

- Conclusions
  - BAT: Tool for bit-level verification
  - New automatic memory abstraction algorithm
  - Memories are first class objects
  - Introduced effective term-rewriting techniques
  - Can verify 32-bit, 5 stage pipelines automatically
- Future Work
  - Automatically translate Verilog/VHDL to BAT
  - Develop similar abstractions to reduce data path
  - Integrate additional term-rewriting techniques

Automatic Memory Reductions for RTL Model Verification

Panagiotis Manolios
Sudarshan K. Srinivasan
Daron Vroon