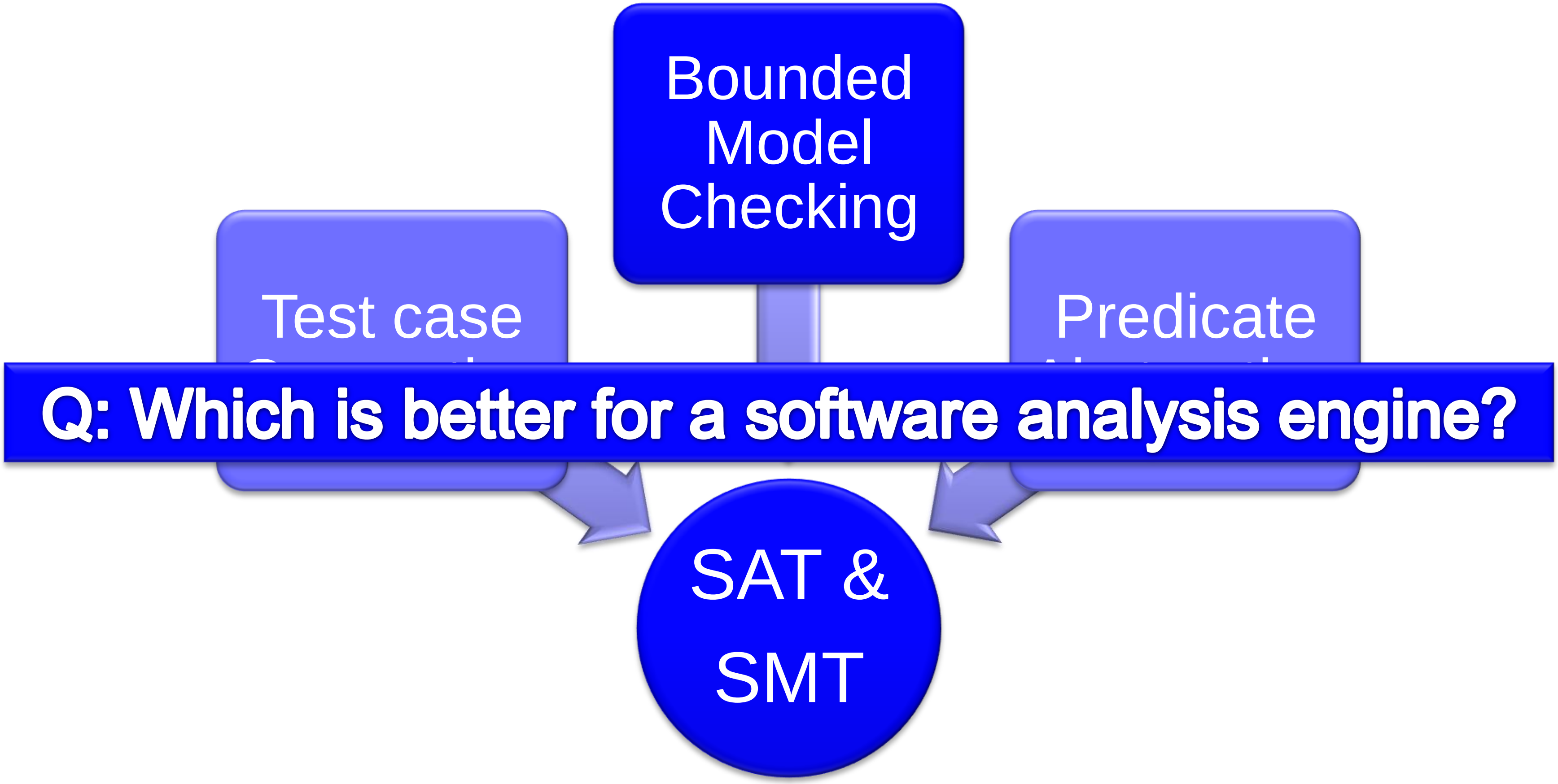
Comparison between SAT and SMT as a Software Analysis Engine

Presented by Yunho Kim

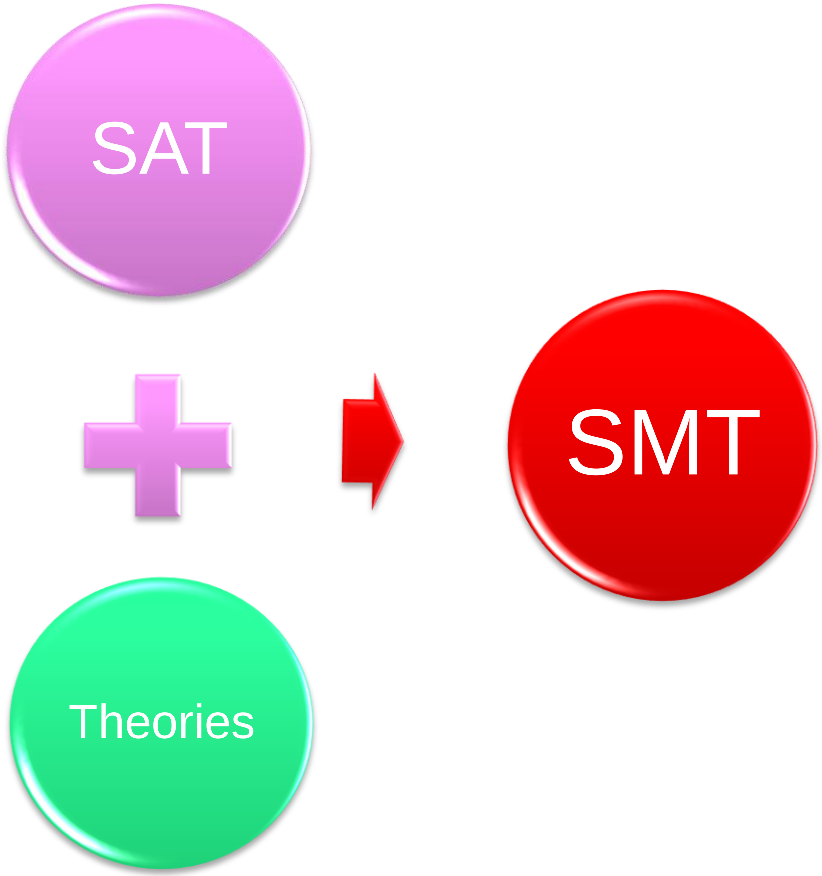
Provable Software Laboratory

CS Dept. KAIST

Introduction



Satisfiability Modulo Theory(1/2)



- Arithmetic
- Bit-vectors
- Arrays
- …

Satisfiability Modulo Theory(2/2)

Arithmetic

$$x+2 = y \Rightarrow f(select(store(a, x, 3), y-2) = f(y-x+1)$$

Array

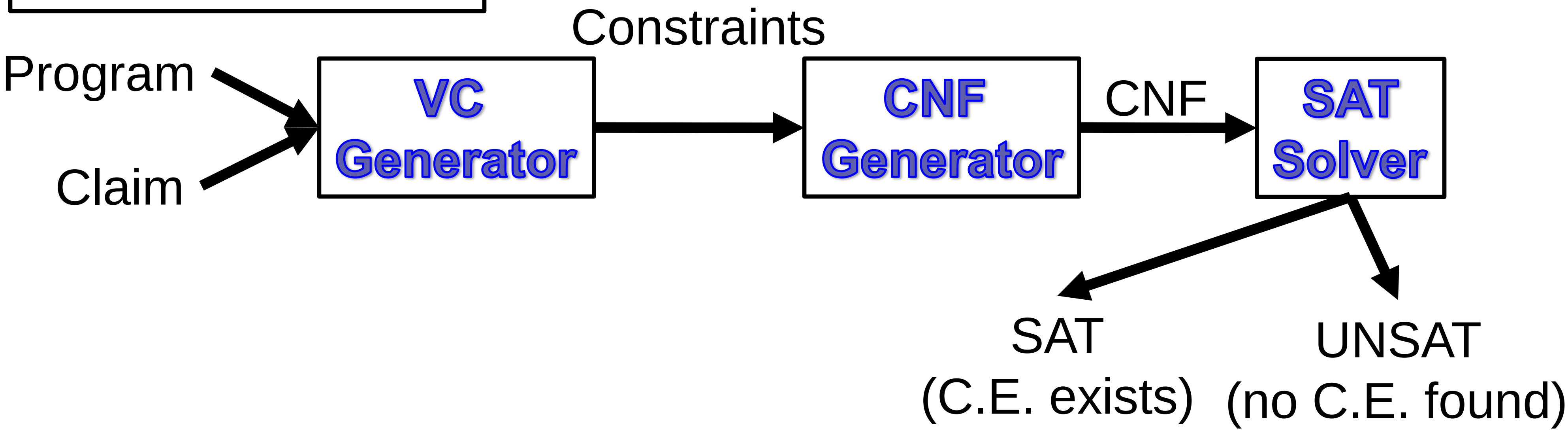
Uninterpreted Function

C Bounded Model Checking

```
int main(){
  int a[2], i, x;
  if (x==0)
    a[i] = 0;
  else
    a[i+2]=1;
  assert(a[i+1]==1);
}
```

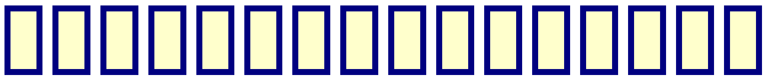
```
guard₁ == (x₀ == 0)
a₁ == (a₀ WITH [i₀:=0])
a₂ == a₀
a₃ == (a₂ WITH [2+i₀:=1])
a₄ == (guard₁ ? a₁ : a₃)
t₁ == (a₄[1+i₀]==1)
```

$(\neg x_4 \lor \neg x_{36}) \land$
$(\neg x_5 \lor \neg x_{36}) \land$
…
$(\neg x_{134} \lor \neg x_{135}) \land$
$(x_{38} \lor \neg x_{135}) \land$
$(x_{134} \lor \neg x_{38} \lor x_{135}) \lor$
…



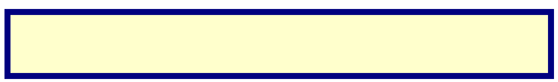
Encoding Approach(1/2)

- Bit-level encoding



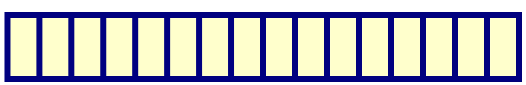
  - Every bit is represented individually

- Word-level encoding



  - E.g., unbounded integers

- Bit-vector encoding



  - Captures true semantics of hardware and software
  - Has more structures for abstraction than with bits

Encoding Approach(2/2)

```
1 guard1 == (x0 == 0)
2 a1 == (a0 WITH [i0:=0])
3 a2 == a0
4 a3 == (a2 WITH [2+i0:=1])
5 a4 == (guard1 ? a1 : a3)
6 t1 == (a4[1+i0]==1)
```

Bit-level encoding

$(\neg x_4 \lor \neg x_{36}) \land$
$(\neg x_5 \lor \neg x_{36}) \land$
…
$(\neg x_{134} \lor \neg x_{135}) \land$
$(x_{38} \lor \neg x_{135}) \land$
…

Word-level encoding

$(guard_1 \Leftrightarrow x_0 = 0) \land$
$a_1 = store(a, i_0, 0) \land$
…
$(T_1 \Leftrightarrow select(a_4, 1+i_0)=1)$

Bit-vector encoding

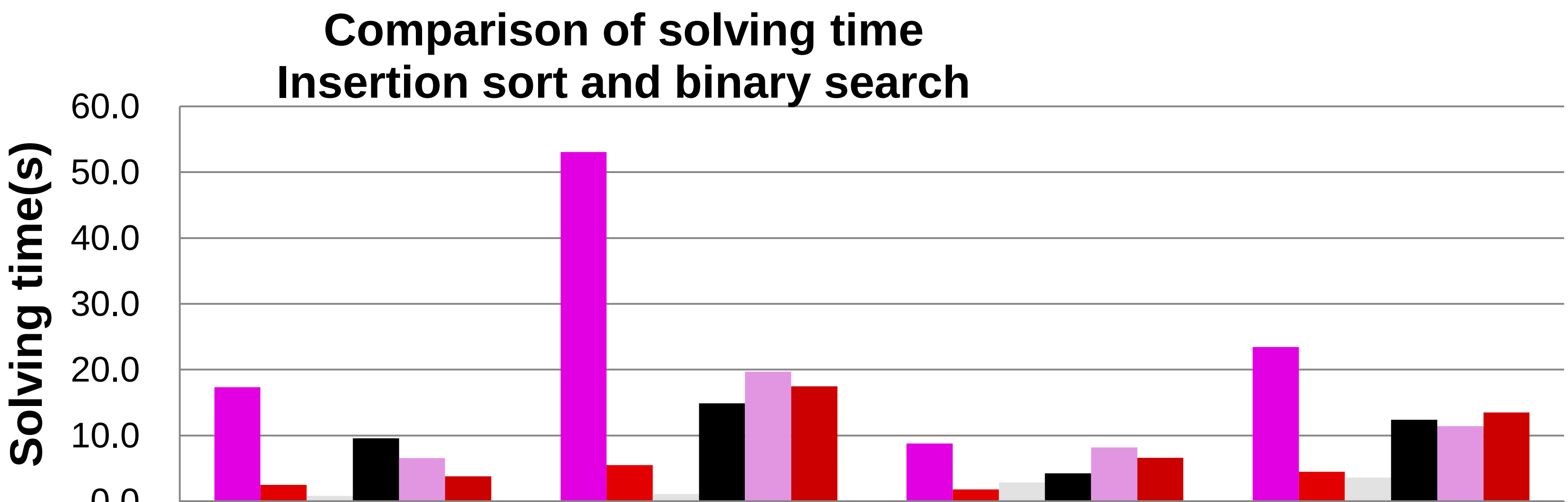
$(guard_1 \Leftrightarrow x_0 = bv0[32]) \land$
$a_1 = store(a, i_0, bv0[32]) \land$
…
$(T_1 \Leftrightarrow select(a_4,(bvadd\ bv1[32]\ i_0))$
$=bv1[32])$

Benchmark

- We benchmark three examples to compare each encoding scheme
  - Insertion sort (20LOC, 2-level loop)
  - Binary search (54LOC, 1-level loop)
  - Multi-sector read function in the proprietary flash device driver (157LOC, 4-level loop)

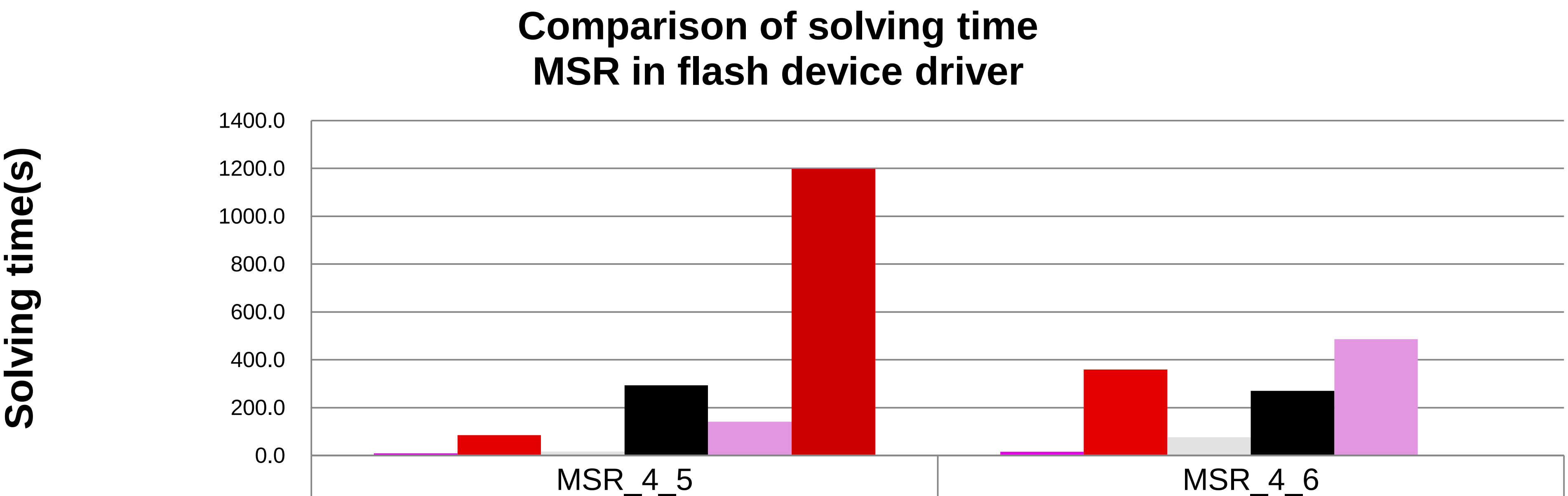
- We use four state-of-the-art SAT and SMT solvers
  - MiniSAT 1.14(integrated with CBMC)
  - Z3 (2008 SMT-competition winner of linear arithmetic category)
  - Yices
  - Boolector (2008 SMT-competition winner of bit-vector category)

Results(1/2)



	Sort_5	Sort_7	Search_5	Search_7
Bit-level MiniSAT	17.4	53.1	8.8	23.5
Word-level Yices	2.5	5.5	1.8	4.5
Word-level Z3	0.8	1.1	2.9	3.6
Bit-vector Yices	9.6	14.9	4.2	12.4
Bit-vector Z3	6.5	19.7	8.2	11.4
Bit-vector Boolector	3.8	17.5	6.6	13.5

Results(2/2)



	MSR_4_5	MSR_4_6
Bit-level MiniSAT	8.6	15.8
Word-level Yices	85.1	359.7
Word-level Z3	16.1	76.3
Bit-vector Yices	293.4	270.4
Bit-vector Z3	141.4	486.4
Bit-vector Boolector	1196.8	

Conclusion

- SAT & SMT is a hot issue in software verification
- Constraint encoding scheme can vary solving performance dramatically
- We need to investigate more efficient encoding method to improve scalability