BOUNDED MODEL CHECKING

KV Software Verification WS 18/19

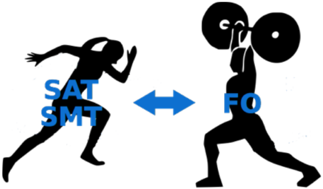
Martina Seidl
Institute for Formal Models and Verification

JOHANNES KEPLER
UNIVERSITY LINZ

Example: Verifiction with SMT

Expressiveness against Efficiency

- **SAT**: efficient, involved encodings
- **FO** (first-order logic): often too powerful
- satisfiability with respect to some theory is required
  (non-standard interpretations are not of interest)
  Example: $x + y < z \vee \neg(x + 1 \leq y \rightarrow x < z)$
- theory needs not be first-order axiomatizable
- specialized inference method for each theory



- **SMT**: sweetspot between SAT and FO
  - propositional logic + domain specific reasoning
  - in general more efficient than with general-purpose solvers with incorporated theory axioms

Satisfiability Modulo Theories (SMT)

$$f(x) \neq f(y) \;\wedge\; x + u = 3 \;\wedge\; v + y = 3 \;\wedge\; u = a[z] \;\wedge\; v = a[w] \;\wedge\; z = w$$

- formulas in first-order logic
  - usually without quantifiers, variables implicitly existentially quantified
  - but with sorted / typed symbols and
  - functions / constants / predicates are interpreted
  - SMT quantifier reasoning weaker than in first-order theorem proving (FO)
  - much richer language compared to propositional logic (SAT)
- many (industrial) applications
  - standardized language SMTLIB used in applications and competitions

The Program "Middle"

```
int middle (int x, int y, int z) {
  int m = z;
  if (y < z) {
    if (x < y)
      m = y;
    else if (x < z)
       m = y;
  } else {
    if (x > y)
      m = y;
    else if (x > z)
      m = x;
  }
  return m;
}
```

This program is supposed to return the middle (median) of three numbers.

The Program “Middle”

```
int middle (int x, int y, int z) {
  int m = z;
  if (y < z) {
    if (x < y)
      m = y;
    else if (x < z)
       m = y;
  } else {
    if (x > y)
      m = y;
    else if (x > z)
      m = x;
  }
  return m;
}
```

Some test cases:

```
middle (1, 2, 3) = 2
middle (1, 3, 2) = 2
middle (2, 3, 1) = 2
middle (3, 1, 2) = 2
middle (3, 2, 1) = 2
middle (1, 1, 1) = 1
middle (1, 1, 2) = 1
middle (1, 2, 1) = 1
middle (2, 1, 1) = 1
middle (1, 2, 2) = 2
middle (2, 1, 2) = 2
middle (2, 2, 1) = 2
```

The Program “Middle”

```
int middle (int x, int y, int z) {
  int m = z;
  if (y < z) {
    if (x < y)
      m = y;
    else if (x < z)
       m = y;
  } else {
    if (x > y)
      m = y;
    else if (x > z)
      m = x;
  }
  return m;
}
```

Missed test case:

`middle (2, 1, 3) = 1`

The Program "Middle"

```
int middle (int x, int y, int z) {
  int m = z;
  if (y < z) {
    if (x < y)
      m = y;
    else if (x < z)
       m = y;
  } else {
    if (x > y)
      m = y;
    else if (x > z)
      m = x;
  }
  return m;
}
```

Missed test case:

`middle (2, 1, 3) = 1`

BUG !

Specification for Middle

Let a be an array of size 3 indexed from 0 to 2.

$$
\boxed{
\begin{array}{l}
\quad a[i] = x \land a[j] = y \land a[k] = z \\
\land \\
\quad a[0] \leq a[1] \land a[1] \leq a[2] \\
\land \\
\quad i \neq j \land i \neq k \land j \neq k \\
\rightarrow \\
\quad m = a[1]
\end{array}
}
$$

Note: coming up with this specification is a manual process

Encoding of Middle in Logic

```
int m = z;
if (y < z) {
  if (x < y)
    m = y;
  else if (x < z)
    m = y;
} else {
  if (x > y)
    m = y;
  else if (x > z)
    m = x;
}
return m;
```

Encoding of Middle in Logic

```
int m = z;
if (y < z) {
  if (x < y)
    m = y;
  else if (x < z)
    m = y;
} else {
  if (x > y)
    m = y;
  else if (x > z)
    m = x;
}
return m;
```

$$
\begin{array}{l}
(y < z \land x < y \rightarrow m = y) \\
\land \\
(y < z \land x \geq y \land x < z \rightarrow m = y) \\
\land \\
(y < z \land x \geq y \land x \geq z \rightarrow m = z) \\
\land \\
(y \geq z \land x > y \rightarrow m = y) \\
\land \\
(y \geq z \land x \leq y \land x > z \rightarrow m = x) \\
\land \\
(y \geq z \land x \leq y \land x \leq z \rightarrow m = z)
\end{array}
$$

Encoding of Middle in Logic

```
int m = z;
if (y < z) {
  if (x < y)
    m = y;
  else if (x < z)
    m = y;
} else {
  if (x > y)
    m = y;
  else if (x > z)
    m = x;
}
return m;
```

$$
\begin{aligned}
&(y < z \land x < y \rightarrow m = y)\\
&\land\\
&(y < z \land x \geq y \land x < z \rightarrow m = y)\\
&\land\\
&(y < z \land x \geq y \land x \geq z \rightarrow m = z)\\
&\land\\
&(y \geq z \land x > y \rightarrow m = y)\\
&\land\\
&(y \geq z \land x \leq y \land x > z \rightarrow m = x)\\
&\land\\
&(y \geq z \land x \leq y \land x \leq z \rightarrow m = z)
\end{aligned}
$$

automatic

Checking Specification as SMT Problem

Let P be the encoding of the program, and S of the specification

- program is correct if "$P \rightarrow S$" is valid

Checking Specification as SMT Problem

Let P be the encoding of the program, and S of the specification

- program is correct if "$P \rightarrow S$" is valid
- program has a bug if "$P \rightarrow S$" is invalid

Checking Specification as SMT Problem

Let P be the encoding of the program, and S of the specification

- program is correct if "$P \rightarrow S$" is valid
- program has a bug if "$P \rightarrow S$" is invalid
- program has a bug if negation of "$P \rightarrow S$" is satisfiable

Checking Specification as SMT Problem

Let P be the encoding of the program, and S of the specification

- program is correct if "$P \to S$" is valid
- program has a bug if "$P \to S$" is invalid
- program has a bug if negation of "$P \to S$" is satisfiable
- program has a bug if "$P \wedge \neg S$" is satisfiable (has a model)

Checking Specification as SMT Problem

Let P be the encoding of the program, and S of the specification

- program has a bug if "$P \wedge \neg S$" is satisfiable (has a model)

$$
\left.
\begin{array}{ll}
(y < z \wedge x < y \rightarrow m = y) & \wedge \\
(y < z \wedge x \geq y \wedge x < z \rightarrow m = y) & \wedge \\
(y < z \wedge x \geq y \wedge x \geq z \rightarrow m = z) & \wedge \\
(y \geq z \wedge x > y \rightarrow m = y) & \wedge \\
(y \geq z \wedge x \leq y \wedge x > z \rightarrow m = x) & \wedge \\
(y \geq z \wedge x \leq y \wedge x \leq z \rightarrow m = z) & \wedge
\end{array}
\right\} P
$$

Checking Specification as SMT Problem

Let P be the encoding of the program, and S of the specification

- program has a bug if "$P \wedge \neg S$" is satisfiable (has a model)

$$
\left.\begin{array}{ll}
(y < z \wedge x < y \rightarrow m = y) & \wedge \\
(y < z \wedge x \geq y \wedge x < z \rightarrow m = y) & \wedge \\
(y < z \wedge x \geq y \wedge x \geq z \rightarrow m = z) & \wedge \\
(y \geq z \wedge x > y \rightarrow m = y) & \wedge \\
(y \geq z \wedge x \leq y \wedge x > z \rightarrow m = x) & \wedge \\
(y \geq z \wedge x \leq y \wedge x \leq z \rightarrow m = z) & \wedge
\end{array}\right\} P
$$

$$
a[i] = x \wedge a[j] = y \wedge a[k] = z \quad \wedge
$$

$$
\left.\begin{array}{ll}
a[0] \leq a[1] \wedge a[1] \leq a[2] & \wedge \\
i \neq j \wedge i \neq k \wedge j \neq k & \wedge \\
m \neq a[1] &
\end{array}\right\} \neg S
$$

Encoding with LIA in SMTLIB2

```
(set-logic QF_AUFLIA)
(declare-fun x () Int) (declare-fun y () Int) (declare-fun z () Int)
(declare-fun i () Int) (declare-fun j () Int) (declare-fun k () Int)
(declare-fun m () Int) (declare-fun a () (Array Int Int))
(assert (=> (and (< y z) (< x y) ) (= m y)))
(assert (=> (and (< y z) (>= x y) (< x z)) (= m y)))
(assert (=> (and (< y z) (>= x y) (>= x z)) (= m z)))
(assert (=> (and (>= y z) (> x y) ) (= m y)))
(assert (=> (and (>= y z) (<= x y) (> x z) ) (= m x)))
(assert (=> (and (>= y z) (<= x y) (<= x z)) (= m z)))
(assert (and (<= 0 i) (<= i 2) (<= 0 j) (<= j 2) (<= 0 k) (<= k 2)))
(assert (and (= (select a i) x) (= (select a j) y) (= (select a k) z)))
(assert (<= (select a 0) (select a 1) (select a 2)))
(assert (distinct i j k))
(assert (distinct m (select a 1)))

(check-sat) (get-model) (exit)
```

Encoding with LIA in SMTLIB2

```
(set-logic QF_AUFLIA)
(declare-fun x () Int) (declare-fun y () Int) (declare-fun z () Int)
(declare-fun i () Int) (declare-fun j () Int) (declare-fun k () Int)
(declare-fun m () Int) (declare-fun a () (Array Int Int))
(assert (=> (and (< y z) (< x y) ) (= m y)))
(assert (=> (and (< y z) (>= x y) (< x z)) (= m y)))
(assert (=> (and (< y z) (>= x y) (>= x z)) (= m z)))
(assert (=> (and (>= y z) (> x y) ) (= m y)))
(assert (=> (and (>= y z) (<= x y) (> x z) ) (= m x)))
(assert (=> (and (>= y z) (<= x y) (<= x z)) (= m z)))
(assert (and (<= 0 i) (<= i 2) (<= 0 j) (<= j 2) (<= 0 k) (<= k 2)))
(assert (and (= (select a i) x) (= (select a j) y) (= (select a k) z)))
(assert (<= (select a 0) (select a 1) (select a 2)))
(assert (distinct i j k))
(assert (distinct m (select a 1)))

(check-sat) (get-model) (exit)
```

Encoding with LIA in SMTLIB2

```
(set-logic QF_AUFLIA)
(declare-fun x () Int) (declare-fun y () Int) (declare-fun z () Int)
(declare-fun i () Int) (declare-fun j () Int) (declare-fun k () Int)
(declare-fun m () Int) (declare-fun a () (Array Int Int))
(assert (=> (and (< y z) (< x y) ) (= m y)))
(assert (=> (and (< y z) (>= x y) (< x z)) (= m y)))
(assert (=> (and (< y z) (>= x y) (>= x z)) (= m z)))
(assert (=> (and (>= y z) (> x y) ) (= m y)))
(assert (=> (and (>= y z) (<= x y) (> x z) ) (= m x)))
(assert (=> (and (>= y z) (<= x y) (<= x z)) (= m z)))
(assert (and (<= 0 i) (<= i 2) (<= 0 j) (<= j 2) (<= 0 k) (<= k 2)))
(assert (and (= (select a i) x) (= (select a j) y) (= (select a k) z)))
(assert (<= (select a 0) (select a 1) (select a 2)))
(assert (distinct i j k))
(assert (distinct m (select a 1)))

(check-sat) (get-model) (exit)
```

Encoding with LIA in SMTLIB2

```
(set-logic QF_AUFLIA)
(declare-fun x () Int) (declare-fun y () Int) (declare-fun z () Int)
(declare-fun i () Int) (declare-fun j () Int) (declare-fun k () Int)
(declare-fun m () Int) (declare-fun a () (Array Int Int))
(assert (=> (and (< y z) (< x y) ) (= m y)))
(assert (=> (and (< y z) (>= x y) (< x z)) (= m y)))
(assert (=> (and (< y z) (>= x y) (>= x z)) (= m z)))
(assert (=> (and (>= y z) (> x y) ) (= m y)))
(assert (=> (and (>= y z) (<= x y) (> x z) ) (= m x)))
(assert (=> (and (>= y z) (<= x y) (<= x z)) (= m z)))
(assert (and (<= 0 i) (<= i 2) (<= 0 j) (<= j 2) (<= 0 k) (<= k 2)))
(assert (and (= (select a i) x) (= (select a j) y) (= (select a k) z)))
(assert (<= (select a 0) (select a 1) (select a 2)))
(assert (distinct i j k))
(assert (distinct m (select a 1)))

(check-sat) (get-model) (exit)
```

Encoding with LIA in SMTLIB2

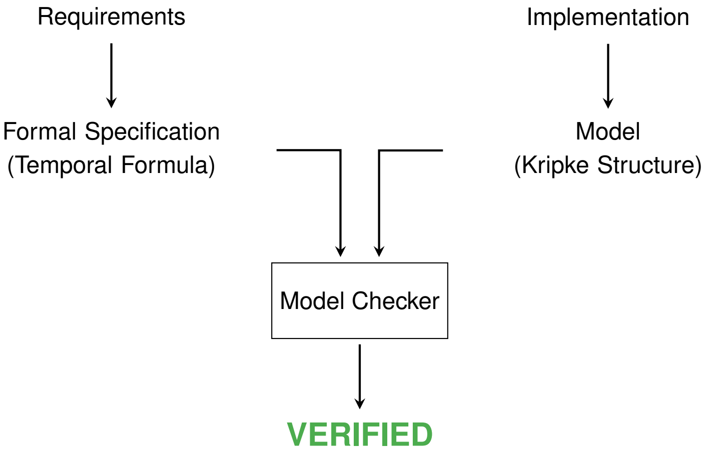
```
(set-logic QF_AUFLIA)
(declare-fun x () Int) (declare-fun y () Int) (declare-fun z () Int)
(declare-fun i () Int) (declare-fun j () Int) (declare-fun k () Int)
(declare-fun m () Int) (declare-fun a () (Array Int Int))
(assert (=> (and (< y z) (< x y) ) (= m y)))
(assert (=> (and (< y z) (>= x y) (< x z)) (= m y)))
(assert (=> (and (< y z) (>= x y) (>= x z)) (= m z)))
(assert (=> (and (>= y z) (> x y) ) (= m y)))
(assert (=> (and (>= y z) (<= x y) (> x z) ) (= m x)))
(assert (=> (and (>= y z) (<= x y) (<= x z)) (= m z)))
(assert (and (<= 0 i) (<= i 2) (<= 0 j) (<= j 2) (<= 0 k) (<= k 2)))
(assert (and (= (select a i) x) (= (select a j) y) (= (select a k) z)))
(assert (<= (select a 0) (select a 1) (select a 2)))
(assert (distinct i j k))
(assert (distinct m (select a 1)))

(check-sat) (get-model) (exit)
```

Encoding with LIA in SMTLIB2

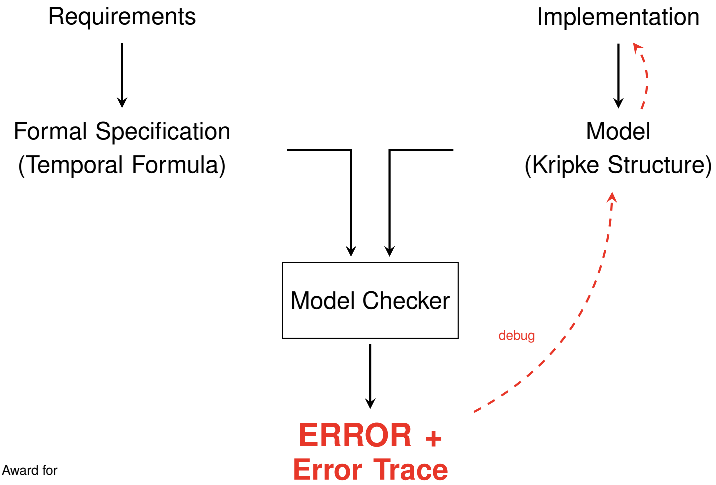
```
(set-logic QF_AUFLIA)
(declare-fun x () Int) (declare-fun y () Int) (declare-fun z () Int)
(declare-fun i () Int) (declare-fun j () Int) (declare-fun k () Int)
(declare-fun m () Int) (declare-fun a () (Array Int Int))
(assert (=> (and (< y z) (< x y) ) (= m y)))
(assert (=> (and (< y z) (>= x y) (< x z)) (= m y)))
(assert (=> (and (< y z) (>= x y) (>= x z)) (= m z)))
(assert (=> (and (>= y z) (> x y) ) (= m y)))
(assert (=> (and (>= y z) (<= x y) (> x z) ) (= m x)))
(assert (=> (and (>= y z) (<= x y) (<= x z)) (= m z)))
(assert (and (<= 0 i) (<= i 2) (<= 0 j) (<= j 2) (<= 0 k) (<= k 2)))
(assert (and (= (select a i) x) (= (select a j) y) (= (select a k) z)))
(assert (<= (select a 0) (select a 1) (select a 2)))
(assert (distinct i j k))
(assert (distinct m (select a 1)))

(check-sat) (get-model) (exit)
```

Bounded Model Checking

Model Checking



Model Checking



Turing Award for

- Queille, J. P.; Sifakis, J. (1982), "Specification and verification of concurrent systems in CESAR", International Symposium on Programming, citations: 1900
- Edmund M. Clarke, E. Allen Emerson: "Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic". Logic of Programs 1981: 52-71, citations: 3895

Types of Model Checking

General question: Given a system K and a property p, does p hold for K (i.e., for all initial states of K) ?

- Explicit state model checking
  - enumeration of the state space
  - state explosion problem

- Symbolic model checking
  - representation of model checking problem as logical formula (e.g., in propositional logic (SAT) or QBF)

Bounded Model Checking

basic idea: search for a counter-example of bounded length k

- encoding in propositional logic (or extensions)
- use SAT solvers to find such a counter-example:
  formula is satisfiable iff a bug is found, i.e., an execution of program that violates the claim.
- benefits:
  - bit-precise encoding of the real semantics
  - powerful SAT solvers
  - difficulty of the problem is controllable (by selection of k)
- drawback: incomplete for k that is too small

$\Rightarrow$ can be used for debugging

Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Yunshan Zhu (1999) Symbolic Model Checking without BDDs. TACAS 193-207, citations: 2580

Propositional Satisfiability (SAT)

Given propositional formula ϕ. Is there a satisfying truth assignment for ϕ?

- SAT solvers are very powerful solving tools
- Using SAT as a “programming language” is very successful in many domains

Example

Given: $\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3)$

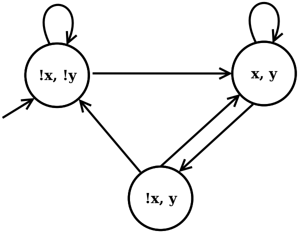
Question: Is ϕ satisfiable?

Yes! For example: $x_1 = x_3 =$ true, $x_2 =$ false.

Symbolic System Representation

Kripke Structure: Description of the System

States:	$\{s_1, s_2, s_3\}$
Initial state:	$\{s_1\}$
Transition Relation:	$\{(s_1, s_1), (s_1, s_2), (s_2, s_2), (s_2, s_3), (s_3, s_1), (s_3, s_2)\}$
Propositions:	x, y
Labeling:	$\{(s_1, \{\neg x, \neg y\}), (s_2, \{x, y\}), (s_3, \{\neg x, y\})\}$,



Translation to SAT

Initial state: $I((x, y)) = \neg x \wedge \neg y$

Transition Relation: $$T((x, y), (x', y')) = ((x' \Leftrightarrow x \vee y) \wedge (y' \Leftrightarrow y)) \vee ((x' \Leftrightarrow \neg y) \wedge (y' \Leftrightarrow x \vee \neg y))$$

Bounded Model Checking (Safety)

- Given a Kripke structure K. Is there a path of length k to a **bad state** s, i.e., a certain property p is violated in s?

- In other words: there is a path where $\mathsf{G}p$ does not hold in K

- Observation: if $\mathsf{G}p$ does not hold in K, there is a **finite counter-example**.

- Idea: consider paths of fixed length k
  - encode problem to propositional formula ϕ
  - pass problem to SAT solver
  - ϕ is true $\Leftrightarrow$ model of ϕ is counter-example
  - if ϕ is false, then increase k

Bounded Model Checking (Safety)

A bounded model checking (BMC) problem for Kripke structure K and safety property $\mathsf{G}p$ is encoded by

$$I(s_0) \wedge \mathcal{T}(s_0, s_1) \wedge \mathcal{T}(s_1, s_2) \wedge \ldots \wedge \mathcal{T}(s_{k-1}, s_k) \wedge B(s_k)$$

where

- $I(s_0)$ is true $\Leftrightarrow$ s_0 is an initial state
- $\mathcal{T}$ is the transition relation of K
- $B(s_k)$ is true $\Leftrightarrow$ s_k is a bad state, i.e., $\neg p$ holds in s_k

Bounded Model Checking for Software

Bounded Model Checking of ANSI-C Programs

- idea:
  - unwind program into equation
  - check equation using SAT/SMT
- benefits:
  - completely automated
  - treatment of pointers and dynamic memory is possible
- properties:
  - simple assertions
  - run time errors (pointers/arrays)
  - run time guarantees (WCET)

for example implemented in tool CBMC

A tool for checking ANSI-C programs E Clarke, D Kroening, F Lerda Tools and Algorithms for the Construction and Analysis of Systems, 168-176, citations: 1339

From C to SAT/SMT

- removal of side effects
  example: `j=i++` is rewritten to `j=i; i=i+1`
- control flow is made explicit
  example: `continue, break` are replaced by `goto`
- transformation of loops to `while (...) ...`
- `while (...) ...` loops are unwound
  - all loops must be bounded
    $\rightarrow$ analysis may become incomplete
  - constant loop bounds are found automatically, others must be specified by user
  - to ensure sufficient unwinding, “unwinding assertions” are added

From C to SAT/SMT: Loop Unwinding

original function:

```
void f (...) {
  ...
  while (cond) {
    body;
  }
  rest;
}
```

with unwounded loop:

```
void f (...) {
  ...
  if (cond) {
    body;
    if (cond) {
      body;
      if (cond) {
        body;
        assert(!cond);
      }
    }
  }
  rest;
}
```

after last iteration an assertion is added:
violated if program runs longer than bound permits

JKU

From C to SAT/SMT: SSA

single static assignment (SSA) form: fresh variable for LHS of each assignment

example:

```
x = x + y;
x = x * 2;
a[i] = 100;
```

is translated to

```
x1 = x0 + y0;
x2 = x1 * 2;
a1[i0] = 100;
```

from which the following SMT formula can be derived

$$(x_1 = x_0 + y_0) \wedge (x_2 = x_1 * 2) \wedge (a_1[i_0] = 100)$$

From C to SAT/SMT: Conditionals

- for each join point, new variables with selectors are added
- example:

original program:

```
if (v)
  x = y;
else
  x = z;

w = x;
```

$\Rightarrow$

rewritten program:

```
if (v0)
  x0 = y0;
else
  x1 = z0;

x2 = v0 ? x0 : x1;

w1 = x2;
```

From C to SAT/SMT: Example

```
int main () {
  int x, y;
  y = 1;
  if (x)
    y-;
  else
    y++;

  assert
    (y==2 || y==3);


}
```

$\Rightarrow$

```
int main () {
  int x, y;
  y1 = 1;
  if(x0)
    y2 = y1-1;
  else
    y3 = y1+1;

  y4 = x0 ? y2 : y3;

  assert
    (y4==2 || y4==3);
}
```

$\Rightarrow$

$$((y_1 = 1) \wedge (y_2 = y_1 - 1) \wedge (y_3 = y_1 + 1) \wedge (y_4 = x_0 ? y_2 : y_3))$$

$$\rightarrow ((y_4 = 2) \vee (y_4 = 3))$$

Arrays

- functions "read" and "write": read(a, i), write(a, i, v)
- axioms

array congruence

$$\forall a, i, j\colon i = j \rightarrow \text{read}(a, i) = \text{read}(a, j)$$

read over write 1

$$\forall a, v, i, j\colon i = j \rightarrow \text{read}(\text{write}(a, i, v), j) = v$$

read over write 2

$$\forall a, v, i, j\colon i \neq j \rightarrow \text{read}(\text{write}(a, i, v), j) = \text{read}(a, j)$$

- used to model memory (HW and SW)

Array to EUF Example

- eagerly reduce arrays to uninterpreted functions:

$$\text{read}(\text{write}(a, i, v), j) \quad \text{replaced by} \quad (i = j \ ? \ v : \text{read}(a, j))$$

Array to EUF Example

- eagerly reduce arrays to uninterpreted functions:

$$\mathsf{read}(\mathsf{write}(a,i,v),j) \quad \text{replaced by} \quad (i = j \ ? \ v : \mathsf{read}(a,j))$$

- Example:

$$i \neq j \ \wedge \ u = \mathsf{read}(\mathsf{write}(a,i,v),j) \ \wedge \ v = \mathsf{read}(a,j) \ \wedge \ u \neq v$$

Array to EUF Example

- eagerly reduce arrays to uninterpreted functions:

$$\text{read}(\text{write}(a,i,v),j) \quad \text{replaced by} \quad (i = j \ ? \ v : \text{read}(a,j))$$

- Example:

$$i \neq j \ \wedge \ u = \text{read}(\text{write}(a,i,v),j) \ \wedge \ v = \text{read}(a,j) \ \wedge \ u \neq v$$

$$i \neq j \ \wedge \ u = (i = j \ ? \ v : \text{read}(a,j)) \ \wedge \ v = \text{read}(a,j) \ \wedge \ u \neq v$$

Array to EUF Example

- eagerly reduce arrays to uninterpreted functions:

$$\mathsf{read}(\mathsf{write}(a,i,v),j) \quad \text{replaced by} \quad (i = j \ ? \ v : \mathsf{read}(a,j))$$

- Example:

$$i \neq j \ \wedge \ u = \mathsf{read}(\mathsf{write}(a,i,v),j) \ \wedge \ v = \mathsf{read}(a,j) \ \wedge \ u \neq v$$

$$i \neq j \ \wedge \ u = (i = j \ ? \ v : \mathsf{read}(a,j)) \ \wedge \ v = \mathsf{read}(a,j) \ \wedge \ u \neq v$$

$$i \neq j \ \wedge \ u = \mathsf{read}(a,j) \ \wedge \ v = \mathsf{read}(a,j) \ \wedge \ u \neq v$$

Array to EUF Example

- eagerly reduce arrays to uninterpreted functions:

$$\text{read}(\text{write}(a,i,v),j) \quad \text{replaced by} \quad (i=j\ ?\ v:\text{read}(a,j))$$

- Example:

$$i \neq j \ \wedge\ u = \text{read}(\text{write}(a,i,v),j) \ \wedge\ v = \text{read}(a,j) \ \wedge\ u \neq v$$

$$i \neq j \ \wedge\ u = (i=j\ ?\ v:\text{read}(a,j)) \ \wedge\ v = \text{read}(a,j) \ \wedge\ u \neq v$$

$$i \neq j \ \wedge\ u = \text{read}(a,j) \ \wedge\ v = \text{read}(a,j) \ \wedge\ u \neq v$$

$$i \neq j \ \wedge\ u = \text{read}(a,j) = \text{read}(a,j) = v \ \wedge\ u \neq v$$

Array to EUF Example

- eagerly reduce arrays to uninterpreted functions:

$$\text{read}(\text{write}(a,i,v),j) \quad \text{replaced by} \quad (i=j\ ?\ v : \text{read}(a,j))$$

- Example:

$$i \neq j \ \wedge \ u = \text{read}(\text{write}(a,i,v),j) \ \wedge \ v = \text{read}(a,j) \ \wedge \ u \neq v$$

$$i \neq j \ \wedge \ u = (i=j\ ?\ v : \text{read}(a,j)) \ \wedge \ v = \text{read}(a,j) \ \wedge \ u \neq v$$

$$i \neq j \ \wedge \ u = \text{read}(a,j) \ \wedge \ v = \text{read}(a,j) \ \wedge \ u \neq v$$

$$i \neq j \ \wedge \ u = \text{read}(a,j) = \text{read}(a,j) = v \ \wedge \ u \neq v$$

UNSATISFIABLE