BOUNDED MODEL CHECKING

KV Software Verification WS 18/19

Martina Seidl
Institute for Formal Models and Verification
Example: Verification 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**: \( x + y < z \lor \neg(x + 1 \leq y \rightarrow x < z) \)

- 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)

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

- 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.
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”

```c
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”

```c
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.

\[
\begin{align*}
    a[i] &= x \land a[j] = y \land a[k] = z \\
    \land \\
    \land \\
    i &\neq j \land i \neq k \land j \neq k \\
    \rightarrow \\
    m &= a[1]
\end{align*}
\]

Note: coming up with this specification is a manual process
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;
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;

(y < z ∧ x < y → m = y)
∧
(y < z ∧ x ≥ y ∧ x < z → m = y)
∧
(y < z ∧ x ≥ y ∧ x ≥ z → m = z)
∧
(y ≥ z ∧ x > y → m = y)
∧
(y ≥ z ∧ x ≤ y ∧ x > z → m = x)
∧
(y ≥ z ∧ x ≤ y ∧ x ≤ z → m = z)
Encoding of Middle in Logic

```java
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;
```

\[(y < z \land x < y \rightarrow m = y)\]  
\[\land\]  
\[(y < z \land x \geq y \land x < z \rightarrow m = y)\]  
\[\land\]  
\[(y < z \land x \geq y \land x \geq z \rightarrow m = z)\]  
\[\land\]  
\[(y \geq z \land x > y \rightarrow m = y)\]  
\[\land\]  
\[(y \geq z \land x \leq y \land x > z \rightarrow m = x)\]  
\[\land\]  
\[(y \geq z \land x \leq y \land x \leq z \rightarrow m = z)\]
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

\[ \square \]
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 “$P \rightarrow S$” is valid
- program has a bug if “$P \rightarrow S$” is invalid
- program has a bug if negation of “$P \rightarrow S$” is satisfiable
- program has a bug if “$P \land \neg S$” 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 \( P \land \neg S \) is satisfiable (has a model)

\[
\begin{align*}
(y < z \land x < y \rightarrow m = y) & \quad \land \\
(y < z \land x \geq y \land x < z \rightarrow m = y) & \quad \land \\
(y < z \land x \geq y \land x \geq z \rightarrow m = z) & \quad \land \\
(y \geq z \land x > y \rightarrow m = y) & \quad \land \\
(y \geq z \land x \leq y \land x > z \rightarrow m = x) & \quad \land \\
(y \geq z \land x \leq y \land x \leq z \rightarrow m = z) & \quad \land \\
\end{align*}
\]
Checking Specification as SMT Problem

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

- program has a bug if \( P \land \neg S \) is satisfiable (has a model)

\[
(y < z \land x < y \rightarrow m = y) \land \\
(y < z \land x \geq y \land x < z \rightarrow m = y) \land \\
(y < z \land x \geq y \land x \geq z \rightarrow m = z) \land \\
(y \geq z \land x > y \rightarrow m = y) \land \\
(y \geq z \land x \leq y \land x > z \rightarrow m = x) \land \\
(y \geq z \land x \leq y \land x \leq z \rightarrow m = z) \land \\
\]

\[
\begin{align*}
  a[i] &= x \land a[j] = y \land a[k] = z \\
  i &\neq j \land i \neq k \land j \neq k \\
  m &\neq a[1]
\end{align*}
\]
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

Requirements

Formal Specification (Temporal Formula)

Model Checker

VERIFIED

Implementation

Model (Kripke Structure)

VERIFIED
Model Checking

Requirements

Formal Specification (Temporal Formula)

Model Checker

ERROR + Error Trace

Implementation

Model (Kripke Structure)

debug

Turing Award for


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

⇒ can be used for debugging

Propositional Satisfiability (SAT)

Given propositional formula $\phi$. Is there a satisfying truth assignment for $\phi$?

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

Example

Given: $\phi = (x_1 \lor \neg x_2) \land (\neg x_1 \lor x_3)$

Question: Is $\phi$ satisfiable?

Yes! For example: $x_1 = x_3 = \text{true}, x_2 = \text{false}$. 
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: \( I((x, y)) = \neg x \land \neg y \)
Transition Relation: \( T((x, y), (x', y')) = ((x' \iff x \lor y) \land (y' \iff y)) \lor ((x' \iff \neg y) \land (y' \iff x \lor \neg y)) \)
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 $Gp$ does not hold in $K$

- Observation: if $Gp$ does not hold in $K$, there is a **finite** counter-example.

- Idea: consider paths of fixed length $k$
  - encode problem to propositional formula $\phi$
  - pass problem to SAT solver
  - $\phi$ is true $\iff$ model of $\phi$ is counter-example
  - if $\phi$ is false, then increase $k$
A bounded model checking (BMC) problem for Kripke structure $K$ and safety property $Gp$ is encoded by

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

where

- $I(s_0)$ is true $\iff$ $s_0$ is an initial state
- $T$ is the transition relation of $K$
- $B(s_k)$ is true $\iff$ $s_k$ 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
    → 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:
```c
void f (...) {
    ...
    while (cond) {
        body;
    }  
    rest;  
}
```

with unwounded loop:
```c
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
From C to SAT/SMT: SSA

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

example:

\[x = x + y;\]
\[x = x \ast 2;\]
\[a[i] = 100;\]

is translated to

\[x_1 = x_0 + y_0;\]
\[x_2 = x_1 \ast 2;\]
\[a_1[i_0] = 100;\]

from which the following SMT formula can be derived

\[(x_1 = x_0 + y_0) \land (x_2 = x_1 \ast 2) \land (a_1[i_0] = 100)\]
From C to SAT/SMT: Conditionals

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

original program:

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

w = x;
```

rewritten program:

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

x2 = v0 ? x0 : x1;

w1 = x2;
```
From C to SAT/SMT: Example

```c
int main () {
    int x, y;
    y = 1;
    if (x)
        y–;
    else
        y++;
    assert
        (y==2 || y==3);
}
```

⇒

```c
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);
}
```

\[
((y_1 = 1) \land (y_2 = y_1 - 1) \land (y_3 = y_1 + 1) \land (y_4 = x_0 ? y_2 : y_3))
\]

\[\rightarrow ((y_4 = 2) \lor (y_4 = 3))\]
Arrays

- functions “read” and “write”: \( \text{read}(a, i), \text{write}(a, i, v) \)
- axioms

  - array congruence
    \[
    \forall a, i, j : i = j \rightarrow \text{read}(a, i) = \text{read}(a, j)
    \]
  
  - read over write 1
    \[
    \forall a, v, i, j : i = j \rightarrow \text{read}(\text{write}(a, i, v), j) = v
    \]
  
  - read over write 2
    \[
    \forall a, v, i, j : i \neq j \rightarrow \text{read}(\text{write}(a, i, v), j) = \text{read}(a, j)
    \]

- used to model memory (HW and SW)
Array to EUF Example

- eagerly reduce arrays to uninterpreted functions:

\[
\text{read(write}(a, i, v), j) \text{ replaced by } (i = j \ ? \ v : \text{read}(a, j))
\]
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:
  
  \[
  i \neq j \land u = \text{read(}\text{write}(a, i, v), j) \land v = \text{read}(a, j) \land u \neq v
  \]
Array to EUF Example

- eagerly reduce arrays to uninterpreted functions:

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

- Example:

  \[
  i \neq j \land u = \text{read}(\text{write}(a, i, v), j) \land v = \text{read}(a, j) \land u \neq v
  \]

  \[
  i \neq j \land u = (i = j \text{ ? } v : \text{read}(a, j)) \land v = \text{read}(a, j) \land u \neq v
  \]
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:

  \[
  i \neq j \land u = \text{read}(\text{write}(a, i, v), j) \land v = \text{read}(a, j) \land u \neq v
  \]

  \[
  i \neq j \land u = (i = j \ ? \ v : \text{read}(a, j)) \land v = \text{read}(a, j) \land u \neq v
  \]

  \[
  i \neq j \land u = \text{read}(a, j) \land v = \text{read}(a, j) \land u \neq v
  \]
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:

  \[
  i \neq j \land u = \text{read}(\text{write}(a, i, v), j) \land v = \text{read}(a, j) \land u \neq v
  \]

  \[
  i \neq j \land u = (i = j \ ? \ v : \text{read}(a, j)) \land v = \text{read}(a, j) \land u \neq v
  \]

  \[
  i \neq j \land u = \text{read}(a, j) \land v = \text{read}(a, j) \land u \neq v
  \]

  \[
  i \neq j \land u = \text{read}(a, j) = \text{read}(a, j) = v \land u \neq v
  \]
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:

  \[
  i \neq j \land u = \text{read(\text{write}(a, i, v), j)} \land v = \text{read}(a, j) \land u \neq v \\
  i \neq j \land u = (i = j \ ? \ v : \text{read}(a, j)) \land v = \text{read}(a, j) \land u \neq v \\
  i \neq j \land u = \text{read}(a, j) \land v = \text{read}(a, j) \land u \neq v \\
  i \neq j \land u = \text{read}(a, j) = \text{read}(a, j) = v \land u \neq v
  \]

  UNSATISFIABLE