APPLICATION OF SMT: BOUNDED MODEL CHECKING

ANNOUNCEMENTS

- Assignment-1 (Theory) is released today.
  - Deadline: Feb 19.
  - Reminder: Use Latex for writing the solutions.
- Course Project
  - We will have one-on-one discussions next week.
- Assignment-1 (Tool) will be based on the Z3 Theorem Prover.
  - Likely to be released next week.

BOUNDED MODEL CHECKING

IN GENERAL...

- Given a transition system, a property and a bound k, Bounded Model Checking determines whether a state satisfying the property is reachable within k steps.
- We will demonstrate BMC using SMT for bug-finding in programs.
  - In this context, also called Symbolic Execution.
  - Basis of a number of highly successful automated bug-finding mechanisms—Concolic Testing, Whitebox fuzzing…

MICROSOFT ZUNE

- A portable media player introduced by Microsoft in 2008, discontinued in 2011.
- On December 31, 2008, all Zune devices went silent.
- On January 1, 2009, they miraculously started working again!
- We will automatically find the Zune bug using SMT.

Example adapted from Emina Torlak's Lecture

MICROSOFT ZUNE BUG

```
int daysToYear(int days)
{
  year = 2008;
  while (days > 365)
  {
      if (IsLeapYear(year))
      {
          if (days > 366)
          {
              days -= 366;
              year += 1;
          }
      }
      else
      {
          days -= 365;
          year += 1;
      }
  }
  return year;
}
```

- Why would this code get stuck on December 31, 2008?
- At `days=366`, the while loop iterates infinitely!
- How to solve this issue?
- Let us see how we can use SMT to automatically detect this bug.

HOW TO SPECIFY CORRECT EXECUTION?

```
int daysToYear(int days)
{
  year = 2008;
  while (days > 365)
  {
     if (IsLeapYear(year))
      {
          if (days > 366)
          {
             days -= 366;
             year += 1;
          }
      }
      else
      {
          days -= 365;
          year += 1;
      }
  }
  return year;
}
```

HOW TO SPECIFY CORRECT EXECUTION?

```
int daysToYear(int days)
{
  year = 2008;
  while (days > 365)
  {
      oldDays = days;
      if (IsLeapYear(year))
      {
          if (days > 366)
          {
              days -= 366;
              year += 1;
          }
      }
      else
      {
          days -= 365;
          year += 1;
      }
      assert(days < oldDays);
  }
  return year;
}
```

CONVERT TO SMT FORMULA - I

UNROLL LOOPS

```
year = 2008;
if (days > 365)
{
    oldDays = days;
    if (IsLeapYear(year))
    {
        if (days > 366)
        {
            days -= 366;
            year += 1;
        }
    }
    else
    {
        days -= 365;
        year += 1;
    }
    assert(days < oldDays);
    assert(days <= 365);
}
return year;
```

WE HAVE UNROLLED THE FIRST ITERATION OF THE LOOP

IF THIS ASSERTION IS VIOLATED, WE HAVE A VALID COUNTEREXAMPLE

If this assertion is violated then:

1. No counterexample involving one iteration exists
2. There may be counterexamples with more than one iteration

IF NONE OF THE ASSERTIONS ARE VIOLATED, NO COUNTEREXAMPLE EXISTS

CONVERT TO SMT FORMULA - II

CONVERT TO SSA FORM

```
year0 = 2008;
if (days0 > 365)
{
    oldDays0 = days0;
    if (IsLeapYear(year0))
    {
        if (days0 > 366)
        {
            days1 = days0 - 366;
            year1 = year0 + 1;
        }
    }
    else
    {
        days3 = days0 - 365;
        year3 = year0 + 1;
    }
    assert(days4 < oldDays0);
    assert(days4 <= 365);
}
return year5;
```

REPLACE EVERY ASSIGNMENT TO A VARIABLE BY A NEW VARIABLE INSTANCE, AND REPLACE USES TO APPROPRIATE VARIABLE INSTANCES

CONVERT TO SMT FORMULA - II

CONVERT TO SSA FORM

```
year0 = 2008;
if (days0 > 365)
{
    oldDays0 = days0;
    if (IsLeapYear(year0))
    {
        if (days0 > 366)
        {
            days1 = days0 - 366;
            year1 = year0 + 1;
        }
    }
    else
    {
        days3 = days0 - 365;
        year3 = year0 + 1;
    }
    assert(days4 < oldDays0);
    assert(days4 <= 365);
}
return year5;
```

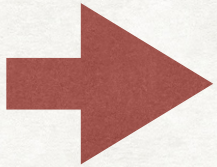
≡

```
year0 = 2008;
bool g0 = (days0 > 365);
oldDays0 = days0;
bool g1 = (IsLeapYear(year0));
bool g2 = (days0 > 366));
days1 = days0 - 366;
year1 = year0 + 1;
days2 = φ(g1 && g2, days1, days0);
year2 = φ(g1 && g2, year1, year0);
days3 = days0 - 365;
year3 = year0 + 1;
days4 = φ(g1, days2, days3);
year4 = φ(g1, year2, year3);
assert(days4 < oldDays0);
assert(days4 <= 365);
year5 = φ(g0, year4, year0);
return year5;
```

CONVERT TO SMT FORMULA - III

CONVERT TO EQUATIONS

```
year₀ = 2008;
bool g₀ = (days₀ > 365);
oldDays₀ = days₀;
bool g₁ = (IsLeapYear(year₀));
bool g₂ = (days₀ > 366));
days₁ = days₀ - 366;
year₁ = year₀ + 1;
days₂ = φ(g₁ && g₂, days₁, days₀);
year₂ = φ(g₁ && g₂, year₁, year₀);
days₃ = days₀ - 365;
year₃ = year₀ + 1;
days₄ = φ(g₁, days₂, days₃);
year₄ = φ(g₁, year₂, year₃);
assert(days₄ < oldDays₀);
assert(days₄ <= 365);
year₅ = φ(g₀, year₀, year₄);
return year₅;
```



$$
\begin{aligned}
&year_0 = 2008 \wedge \\
&g_0 = (days_0 > 365) \wedge \\
&oldDays_0 = days_0 \wedge \\
&g_1 = (IsLeapYear(year_0)) \wedge \\
&g_2 = (days_0 > 366)) \wedge \\
&days_1 = days_0 - 366 \wedge \\
&year_1 = year_0 + 1 \wedge \\
&days_2 = ite(g_1 \,\&\&\, g_2, days_1, days_0) \wedge \\
&year_2 = ite(g_1 \,\&\&\, g_2, year_1, year_0) \wedge \\
&days_3 = days_0 - 365 \wedge \\
&year_3 = year_0 + 1 \wedge \\
&days_4 = ite(g_1, days_2, days_3) \wedge \\
&year_4 = ite(g_1, year_2, year_3) \wedge \\
&(\neg(days_4 < oldDays_0) \vee \\
&\neg(days_4 <= 365))
\end{aligned}
$$

FINAL SMT FORMULA

$$
\begin{aligned}
&year_0 = 2008 \wedge \\
&g_0 = (days_0 > 365) \wedge \\
&oldDays_0 = days_0 \wedge \\
&g_1 = (IsLeapYear(year_0)) \wedge \\
&g_2 = (days_0 > 366)) \wedge \\
&days_1 = days_0 - 366 \wedge \\
&year_1 = year_0 + 1 \wedge \\
&days_2 = ite(g_1 \,\&\&\, g_2, days_1, days_0) \wedge \\
&year_2 = ite(g_1 \,\&\&\, g_2, year_1, year_0) \wedge \\
&days_3 = days_0 - 365 \wedge \\
&year_3 = year_0 + 1 \wedge \\
&days_4 = ite(g_1, days_2, days_3) \wedge \\
&year_4 = ite(g_1, year_2, year_3) \wedge \\
&(\neg(days_4 < oldDays_0) \vee \\
&\neg(days_4 <= 365))
\end{aligned}
$$

- Satisfiability or Validity?
- Which theories are used?
  - Linear Integer Arithmetic
  - Equality

FINAL SMT FORMULA

$$
\begin{aligned}
&year_0 = 2008 \wedge \\
&g_0 = (days_0 > 365) \wedge \\
&oldDays_0 = days_0 \wedge \\
&g_1 = (IsLeapYear(year_0)) \wedge \\
&g_2 = (days_0 > 366)) \wedge \\
&days_1 = days_0 - 366 \wedge \\
&year_1 = year_0 + 1 \wedge \\
&days_2 = ite(g_1 \,\&\&\, g_2, days_1, days_0) \wedge \\
&year_2 = ite(g_1 \,\&\&\, g_2, year_1, year_0) \wedge \\
&days_3 = days_0 - 365 \wedge \\
&year_3 = year_0 + 1 \wedge \\
&days_4 = ite(g_1, days_2, days_3) \wedge \\
&year_4 = ite(g_1, year_2, year_3) \wedge \\
&(\neg(days_4 < oldDays_0) \vee \\
&\neg(days_4 <= 365))
\end{aligned}
$$

- Satisfiability or Validity?
- Which theories are used?
  - Linear Integer Arithmetic
  - Equality

SATISFIABLE FOR $DAYS_0$=366 AND ISLEAPYEAR(2008)=⊤

Z3

INTRODUCTION

- Z3 is a constraint-solver/theorem-prover developed at Microsoft Research.
- Basic Operation:
  - It takes as input a formula [PL/FOL/SMT].
  - Outputs SAT/UNSAT.
- Supports a whole range of theories (including all theories we have seen).
- Open-source (written in C++)
  - Latest version available at Z3 Github page (https://github.com/Z3Prover/z3).

INPUT/OUTPUT FORMAT

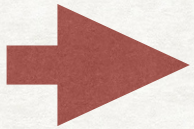
1. APIs for Python, C++, Java, etc.
   - API functions for declaring variables, constants, predicates, functions, and for constructing formula.
   - API functions for accessing a satisfying interpretation (in case of SAT).
2. SMT-LIB 2.0
   - Standard input format for all SMT solvers
   - Formula written in SMT-LIB 2.0 can be directly provided to the Z3 executable.

INPUT FORMAT

- Z3 expects input formula in Many Sorted First Order Logic (MSFOL).
  - ‘sort’ is similar to type. Variables, constants, functions, predicates must be given appropriate types.
  - Built-in sorts: Bool, Integer, Real, Array,…
  - Users can also define new sorts.

SMT-LIB EXAMPLE

$year_0 = 2008 \wedge$
$g_0 = (days_0 > 365) \wedge$
$oldDays_0 = days_0 \wedge$
$g_1 = (IsLeapYear(year_0)) \wedge$
$g_2 = (days_0 > 366)) \wedge$
$days_1 = days_0 - 366 \wedge$
$year_1 = year_0 + 1 \wedge$
$days_2 = ite(g_1 \&\& g_2, days_1, days_0) \wedge$
$year_2 = ite(g_1 \&\& g_2, year_1, year_0) \wedge$
$days_3 = days_0 - 365 \wedge$
$year_3 = year_0 + 1 \wedge$
$days_4 = ite(g_1, days_2, days_3) \wedge$
$year_4 = ite(g_1, year_2, year_3) \wedge$
$(\neg(days_4 < oldDays_0) \vee \neg(days_4 <= 365))$



```
(declare-const year0 Int)
(declare-const g0 Bool)
(declare-fun IsLeapYear (Int)
Bool)
.
.
(assert (= year0 2008))
(assert (= g0 (> days0 365)))
.
.
(assert (not (<= days4 365)))

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

TUTORIALS

- For SMT-LIB
  - https://jfmc.github.io/z3-play/
- For Python API
  - http://theory.stanford.edu/~nikolaj/programmingz3.html
- Download, Installation instructions
  - https://github.com/Z3Prover/z3

TOOL ASSIGNMENT-1 WILL BE BASED ON Z3