Termination

Guest lecture: Amir Ben-Amram
amirben@cs.mta.ac.il
Partial vs Total Correctness

\[
\{P\} \ S \ \{Q\}
\]
\{input provided\} PROGRAM \{output correct\} → partial correctness

Total correctness: \{P\} \ S \ \{\Downarrow \ Q\}
Given precondition P, statement S terminates and satisfies the post-condition.

Safety vs Liveness
Safety property "this bad event cannot happen"
Liveness property "this good event eventually happens"
Zune chokes on leap-year bug

The bug disabled the players on Dec. 31, the last day of a leap year

By Robert McMillan

December 31, 2006 12:00 PM ET      Add a comment

IDG News Service - Microsoft Corp.'s Zune 30GB music player just wasn't ready for a leap year.

That's what owners of the devices discovered Wednesday morning when they awoke to find their players frozen and unworkable.

The problem turned out to be "a bug in the internal clock driver related to the way the device handles a leap year," Microsoft Zune spokesman Matt Akers said in a posting to Zune forums Wednesday. The issue does not affect all Zune players, but all models of the Zune 30GB are potentially affected, he said.

Zune is Microsoft's alternative to Apple's popular iPod devices.

The bug disabled the players on Dec. 31, the last day of a leap year.

Microsoft expects that the bug will resolve itself by Jan. 1, when the device's internal clock will reset itself.

"By [Thursday] you should allow the battery to fully run out of power before the unit can restart successfully, then simply ensure that your device is recharged, then turn it back on," Akers said. "If you're a Zune Pass subscriber, you may need to sync your device with your PC to refresh the rights to the subscription content you have downloaded to your device."

source: IDG
What happened?

year = ORIGINYEAR; /* = 1980 */

while (days > 365)
{
    if (IsLeapYear(year))
    {
        if (days > 366)
        {
            days -= 366;
            year += 1;
        }
    }
    else
    {
        days -= 365;
        year += 1;
    }
}
Proving termination

Turing Told us that it is Undecidable

But Turing also introduced the practical way of proving termination using a **ranking function** aka "progress measure"
Proving termination for a WHILE loop

while (x ≤ y) {
    x := x + 1
}

ρ(x, y) = y - x

A ranking function ρ is:
• A function of the state variables ρ(σ)
• Maps a state to a value in a set \( W \) (example: \( \mathbb{N} \ ))
• Every iteration decreases the value ρ(σ) > ρ(σ')
  - Termination proof: every execution generates a decreasing sequence

Generalizations:
• Other sets \( W \) can be used as long as there are no infinite decreasing chains - well-founded set
• Other types of loops, recursion
Termination proofs in Dafny

method Mul(x: int, y: int) returns (r: int)
    requires 0 <= x && 0 <= y
    decreases x
{
    if x == 0 {
        r := 0;
    } else {
        if y > 0 {
            r := 0;
            // r := Mul(x, y-1);
        }
        var m := Mul(x-1, y);
        r := m + x;
    }  
}
function gcd(m:int, n: int) : int
// gcd of positive numbers
  requires n >= 1 && m >= 1
{
  if n == 1 || m == 1 then 1
  else if n == m then m
  else if m > n then
    gcd(m-n, n)
  else
    gcd(n-m, m)
}
method gcd(x: int, y: int) returns (g: int)
  requires 0 < x && 0 < y
{
  var n: int := x;
  var m: int := y;
  m := y;
  while m != n && m > 1 && n > 1
  {
    if m < n {
      n := n - m;
    } else {
      m := m - n;
    }
  }
  if m == 1 {
    g := m;
  } else {
    g := n;
  }
}
Termination proofs in Dafny

```daffny
function Ackermann(m: int, n: int): int
{
   if m <= 0 then
      n + 1
   else if n <= 0 then
      Ackermann(m - 1, 1)
   else
      Ackermann(m - 1, Ackermann(m, n - 1))
}
```
Kinds of ranking function

Numeric: $\rho(\sigma)$ is a **number** and decreases at least by 1

Lexicographic: $\rho(\sigma)$ is a **tuple** and decreases lexicographically

**Theorem:** if $W$ is a well-founded set than $W^n$ (under lexicographic order) is well-founded
Automatic termination provers

- Well-studied subject
- Different methods, different types of programs, many tools (at least two called Terminator)

Logic programs: since the 1980's. Termilog - Sagiv & Lindenstrauss, ~1990
Functional programs: the 1990's, in particular in connection with program transformation tools
Imperative programs: the 2000's
  Java: Julia, COSTA
  C: many, including Microsoft tools Terminator and T2 - used to verify device drivers

- Another class of tools - non-termination finders
Finding ranking functions automatically

Given: a loop transition relation (in a particular form)
Find: a ranking function (of a specific kind)

DECIDABLE cases:
• transition relation expressed by linear constraints
• variables integer, rational or real
• ranking functions are linear or lexicographic-linear

\[
\rho(x_1, \ldots, x_4) = \langle x_1 + x_2, \ x_3 - 2x_4 \rangle
\]

while \(1 \leq z\) {
\[
\begin{align*}
    x &:= x + 1; \\
    y &:= y - x; \\
    z &:= z - 1;
\end{align*}
\]
\}

\[
\begin{align*}
    z \geq 1 \land \\
    x' &= x + 1 \land \\
    y' &= y - x - 1 \land \\
    z' &= z - 1
\end{align*}
\]
The template method

Given: the transition relation $\tau$ as a formula on $(x, x')$ 

$\mathbf{x} = \langle x_1, \ldots, x_n \rangle$

A template for a desired ranking function E.g.: $\rho(x) = a \cdot x + a_0$

- the coefficients $a$ are symbolic variables

Solve: $\forall (x, x') \quad \tau(x, x') \rightarrow \rho(x) \geq 0$

$\land \tau(x, x') \rightarrow \rho(x') \leq \rho(x) - 1$
### Algorithms & complexity (for linear constraints)

<table>
<thead>
<tr>
<th>Function kind</th>
<th>Domain</th>
<th>( \mathbb{Q} / \mathbb{R} )</th>
<th>( \mathbb{Z} )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>linear</strong></td>
<td></td>
<td>Polynomial time</td>
<td>coNP-complete</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Colón and Sipma 2001; Feautrier 1992; Mesnard and Serebrenik 2008; Podelski &amp; Rybalchenko 2004; Sohn and Van Gelder 1991</td>
<td>Ben-Amram and Genaim 2013</td>
</tr>
<tr>
<td><strong>lexicographic - linear</strong></td>
<td></td>
<td>Polynomial time</td>
<td>coNP-complete</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alias et al., 2010; Ben-Amram and Genaim 2014</td>
<td>Ben-Amram and Genaim 2014</td>
</tr>
</tbody>
</table>
Enter the loop in the corresponding text area, set the required options, and click on Analyze. The result will be displayed at the bottom of this page.

<table>
<thead>
<tr>
<th>state variables</th>
<th>post-state variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>!vars x y</td>
<td>!pvars xp yp</td>
</tr>
<tr>
<td>!path x=1 y&gt;=0</td>
<td>!path x=0 y&gt;=0 xp=1-x yp=y-x</td>
</tr>
</tbody>
</table>

ALGORITHM

- LinRF(Z)
- LinRF(Q)
- LexLinRF(Z)
- LexLinRF(Q)

OPTIONS

- Check PTIME cases
- Use closure for octagons instead of integer hull
- Compute the integer hull if PTIME cases fail
- Use the generators algorithm for LRFs
- Generate LRFs/LLRFs with integer coefficients
- Consolidate (weak) LLRFs
- Verbose mode
method Main(n: int) {
    var x: int := 0;
    var y: int := n;
    while y >= 0 {
        x := 1-x;
        y := y-x;
    }
}
\( \mathbb{Z} \) vs \( \mathbb{Q} \)

Treating integer variables as if they are rational (or real) is safe - but may be too safe

\[
x_2 - x_1 \leq 0, \quad x_1 + x_2 \geq 1
\]

\[
x'_2 = x_2 - 2x_1 + 1, \quad x'_1 = x_1
\]

Non-terminating over \( \mathbb{Q} \): \( (\frac{1}{2}, \frac{1}{2}) \)

Terminating over \( \mathbb{Z} \)
\[ \mathbb{Z} \text{ vs } \mathbb{Q} \]

\[ x_2 - x_1 \leq 0, \quad x_1 + x_2 \geq 1; \]
\[ x'_2 = x_2 - 2x_1 + 1, \quad x'_1 = x_1 \]

Non-terminating over \( \mathbb{Q} \): \( (\frac{1}{2}, \frac{1}{2}) \)
Terminating over \( \mathbb{Z} \)

Geometrically: the set of states is a \textit{polyhedron}

To avoid false alarms for integer variables, we have to compute a smaller polyhedron

\textit{The Integer Hull}: smallest polyhedron that includes all the integer points
Termination & ranking functions

- Turing proposed ranking functions as a way to prove termination
- For certain kinds of loops (linear constraints) and certain kind of ranking functions (linear, lexicographic-linear) we have complete solution

- End-to-end tools that prove termination:
  Julia  http://www.juliasoft.com/
  AProVE  http://aprove.informatik.rwth-aachen.de/
  COSTA  http://costa.ls.fi.upm.es/~costa/costa/costa.php
Obtaining the linear-constraint loop

- Symbolic evaluation / SSA form
- Non-numeric variables:
  - Ignore them if they do not affect the loop conditions
  - Abstract them if they are important

```plaintext
function RecursiveSearch(a: array<int>, value: int) =

    if (a[1] == value) then 1
    else if (a.len < 2) then None
    else 1 + RecursiveSearch(a[2:], value)
```

```plaintext
while (1 <= z) {
    x := x + 1;
y := y - x;
z := z - 1;
}
```

\[ z \geq 1 \land \\
x' = x + 1 \land \\
y' = y - x - 1 \land \\
z' = z - 1 \land \\
a_{\text{len}} \geq 1 \land a_{\text{len}}' = a_{\text{len}} - 1 \]
Obtaining the linear-constraint loop

- Symbolic evaluation / SSA form
- Non-numeric variables:
  - Ignore them if they do not affect the loop conditions
  - Abstract them if they are important
- Non-linear operations:
  - Use constraints (inequalities) to represent them

```c
while (1 <= z) {
  x := x + 1;
  y := y - x;
  z := z - 1;
}
```

```c
z \geq 1 \land
x' = x+1 \land
y' = y - x - 1 \land
z' = z - 1
```

```c
function BinarySearch(low: int, hi: int) =
  mid := (low + hi) / 2
  ... BinarySearch( mid, hi )
  ...
```

```c
2*mid' \leq low+hi
\land 2*mid' \geq low+hi-1
```
Project suggestions

Write a termination prover for a small language, using Z3

Extend a symbolic-checking tool (CBMC?) to check "decreases" assertions

Think about the converse problem: how to find a non-termination bug using tools like BMC and symbolic execution. Implement something.