TATA CONSULTANCY SERVICES
Experience certainty.

Model Checking

R. Venkatesh (Venky, r.venky@tcs.com

Tata Research Development and Design Centre, Pune, India

January 4, 2018

Acknowledgments

- Lecture 1 – Arie Gurfinkel and Daniel Kroening
- Lecture 2 – Priyanka Darke
- Lecture 3 – Divyesh Unadkat
- Lecture 4 – Shrawan Kumar

Course Outline

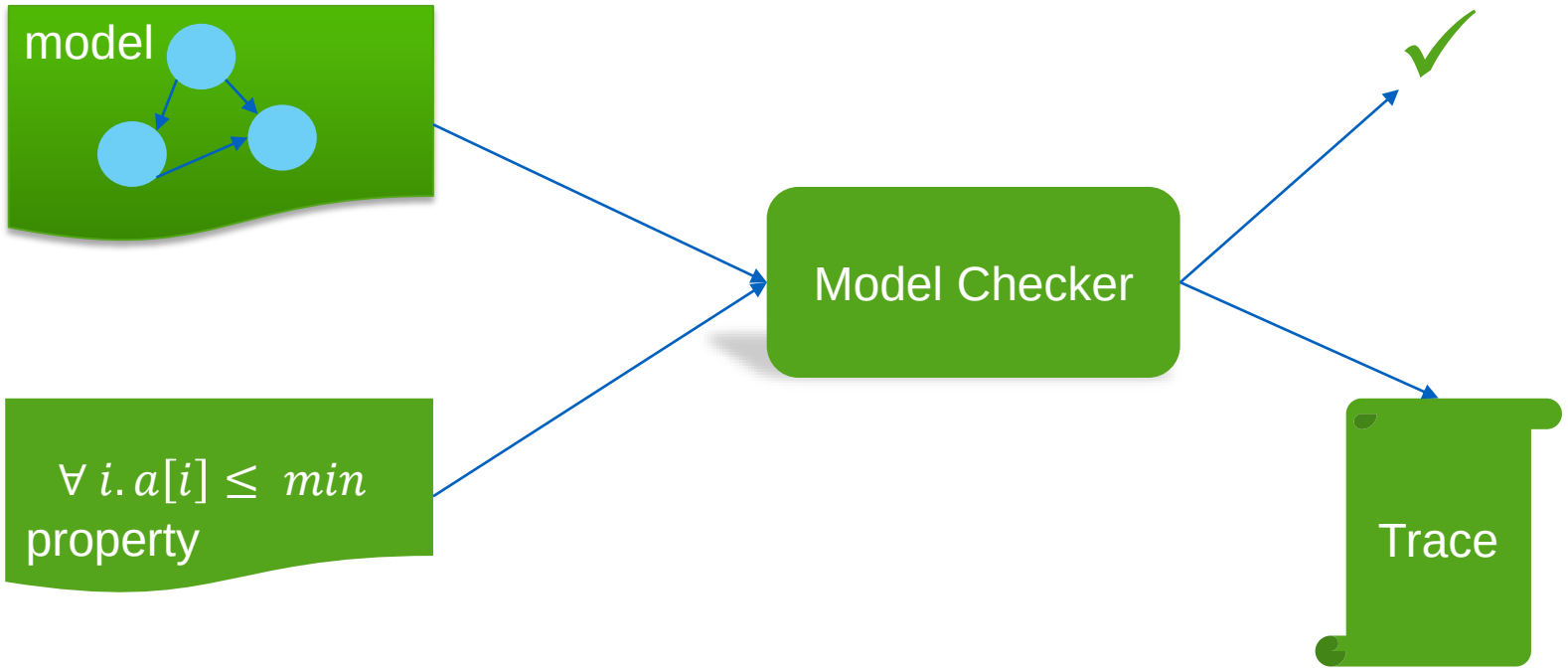
- Introduction to BMC and CBMC
  - basic
- Loop Abstraction for non-array loops
  - a simple practical idea
- Induction based proving for arrays
- Small witness proving for arrays

Cost of Software Bugs

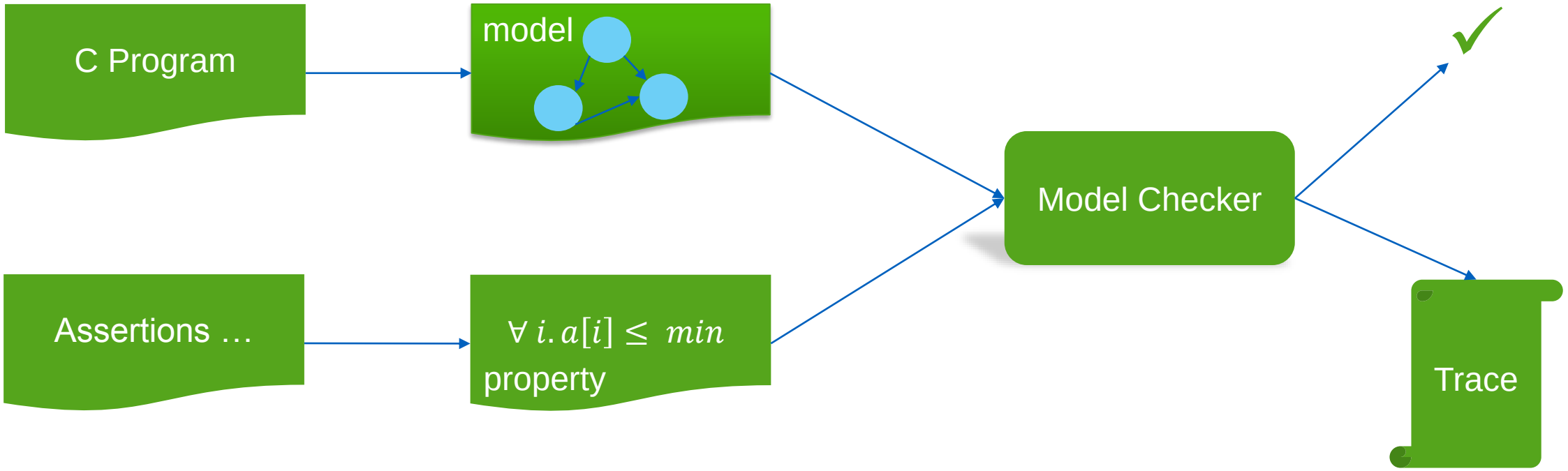
Company	Year	What & why	Source
Maquet	2011	Anesthesia systems	Fda.gov
BMW	2012	7-series vehicles – door latching problem	www.nconsumer.org
Volvo	2012	S80 vehicles – possible engine stall	www.nconsumer.org

24% of medical device recall in 2011 due to software failure (FDA) [slashdot.org]

Model Checking



Model Checking C Programs



Example Program

```
int x, y, z
int min

if (x < y )
   if ( x < z ) min = x
   else min = z
else if (y < z) min = y
else min = z

assert (min <= x && min <= y && min <= z )
```

x,y,z,min take non-deterministic value

Property to be checked

Main Problem : State Space Explosion

- Minimum of 3 Vars -2^{32^4} (x,y,z,min)

- Symbolic model checking
  - Symbolic representation
  - Set of states represented by formula in propositional logic

- Two main techniques
  - Binary Decision Diagrams (BDDs)
  - Satisfiability Checkers (SAT/SMT Solvers)

Why use a SMT Solver?

- SMT Solvers are very efficient
- Analysis is completely automated
- Analysis as good as the underlying SMT solver
- Allows support for many features of a programming language
  - bitwise operations, pointer arithmetic, dynamic memory, type casts

A (very) simple example (1)

Program

```
int x
int y=8,z=0,w=0
if (x)
  z = y - 1
else
  w = y + 1
assert (z == 7 ||
        w == 9)
```

Constraints

y = 8,
z = x ? y – 1 : 0,
w = x ? 0 :y + 1,
z != 7,
w != 9

UNSAT
no counterexample
assertion always holds!

A (very) simple example (2)

Program

```
int x;
int y=8,z=0,w=0;
if (x)
  z = y - 1;
else
  w = y + 1;
assert (z == 5 ||
        w == 9)
```

Constraints

y = 8,
z = x ? y – 1 : 0,
w = x ? 0 :y + 1,
z != 5,
w != 9

SAT
counterexample found!

y = 8, x = 1, w = 0, z = 7

CBMC: C Bounded Model Checker

- Developed at CMU by Daniel Kroening and Ed Clarke
- Available at: http://www.cprover.org/cbmc

- Supported platforms: Windows, Linux, OSX
- Has a command line, Eclipse CDT, and Visual Studio interfaces

- Scales to programs with over 30K LOC
- Found previously unknown bugs in MS Windows device drivers

Using CBMC from Command Line

- To see the list of claims

```
cbmc --show-claims -I include file.c
```

  - To check claim

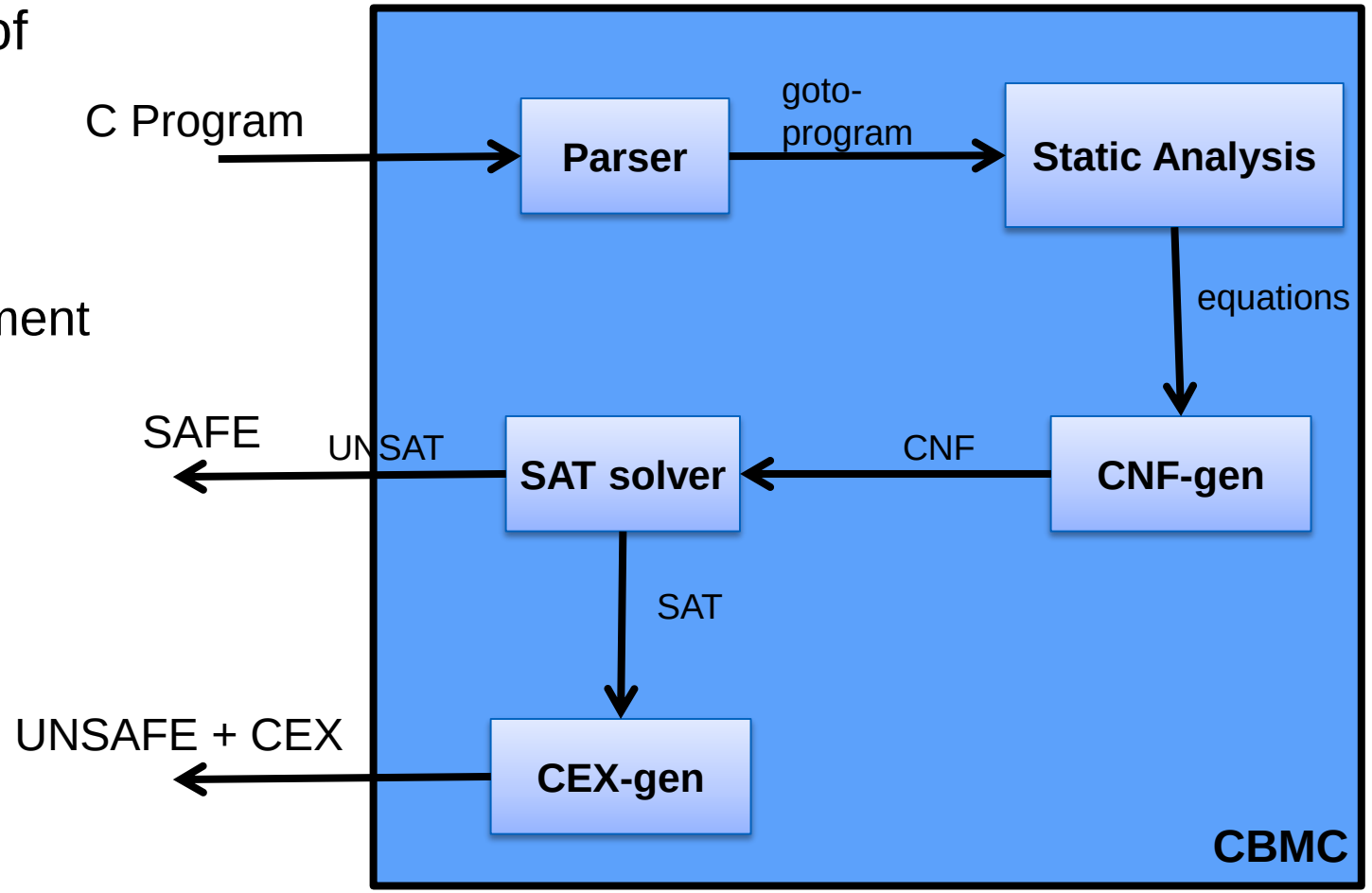
```
cbmc  file.c
```

- For help

  cbmc --help

How does it work

- Transform a programs into a set of equations
  - Simplify control flow
  - Unwind all of the loops
  - Convert into Single Static Assignment (SSA)
  - Convert into equations
  - Bit-blast
- Solve with a SAT Solver
- Convert SAT assignment into a counterexample



Using nondet for modeling

- Library spec:

  “foo is given non-deterministically, but is taken until returned”

- CMBC stub:

```
int nondet_int ();
int is_foo_taken = 0;
int grab_foo () {
  if (!is_foo_taken)
    is_foo_taken = nondet_int ();
  return is_foo_taken; }
```

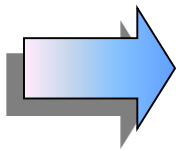
```
void return_foo ()
{ is_foo_taken = 0; }
```

Transforming Loop-Free Programs Into Equations (1)

- Easy to transform when every variable is only assigned once!

Program

```
x = a;
y = x + 1;
z = y - 1;
```



Constraints

```
x = a &&
y = x + 1 &&
z = y – 1 &&
```

Transforming Loop-Free Programs Into Equations (2)

- When a variable is assigned multiple times,
- use a new variable for the RHS of each assignment

Program

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

ρ

SSA Program

$$x_1 = x_0 + y_0;$$
$$x_2 = x_1 * 2;$$
$$a_1[i_0] = 100;$$

What about conditionals?

Program

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

w = x;
```

SSA Program

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

w1 = x??;
```

What should ‘x’ be?

What about conditionals?

Program

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

w = x;
```

SSA Program

```
if (v0)
  x0 = y0;
else
  x1 = z0;
x2 = v0 ? x0 : x1;
w1 = x2
```

- For each join point, add new variables with selectors

Adding Unbounded Arrays

$$v_\alpha[a] = e \quad \rho \quad v_\alpha = \lambda i : \begin{cases} \rho(e) & : \ i = \rho(a) \\ v_{\alpha-1}[i] & : \ \text{otherwise} \end{cases}$$

- Arrays are updated “whole array” at a time

A[1] = 5;
A[2] = 10;
A[k] = 20;

A_1=λ i : i == 1 ? 5 : A_0[i]
A_2=λ i : i == 2 ? 10 : A_1[i]
A_3=λ i : i == k ? 20 : A_2[i]

Examples:

A_2[2] == 10 A_2[1]==5 A_2[3] == A_0[3]
A_3[2] == (k==2 ? 20 : 10)

Uses only as much space as there are uses of the array!

From Programming to Modeling

- Extend C programming language with 3 modeling features

- Assertions
  - assert(e) – aborts an execution when e is false, no-op otherwise

```
void assert (_Bool b) { if (!b)  exit(); }
```

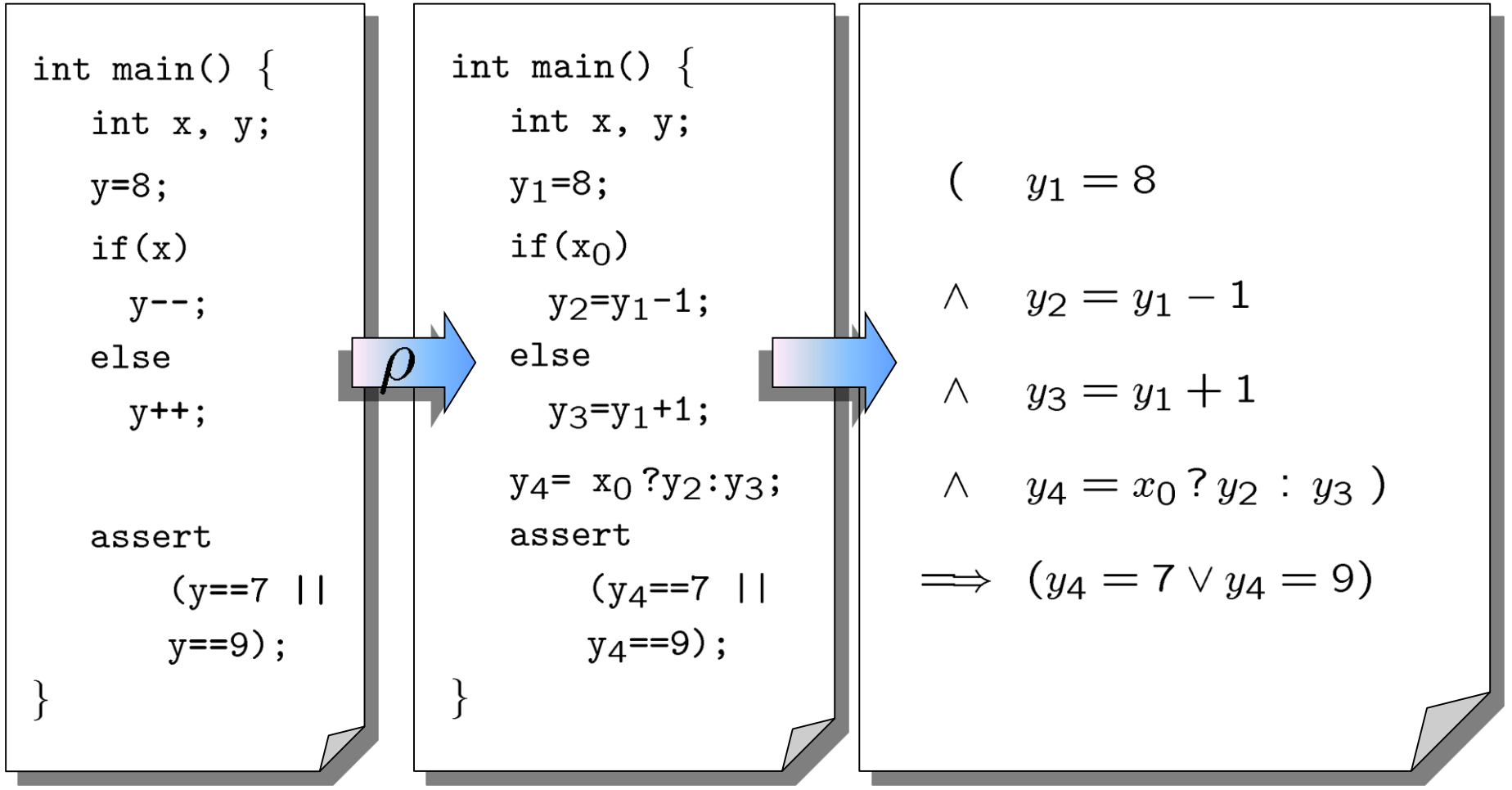
- Non-determinism
  - nondet_int() – returns a non-deterministic integer value

```
int nondet_int () { int x; return x; }
```

- Assumptions
  - assume(e) – “ignores” execution when e is false, no-op otherwise

```
void __CPROVER_assume (_Bool e) { while (!e) ;  }
```

Example


int main() {
int x, y;
y=8;
if(x)
y--;
else
y++;
assert
(y==7 ||
y==9);
}
ρ
int main() {
int x, y;
y1=8;
if(x0)
y2=y1-1;
else
y3=y1+1;
y4= x0 ?y2:y3;
assert
(y4==7 ||
y4==9);
}
(y1 = 8
∧ y2 = y1 − 1
∧ y3 = y1 + 1
∧ y4 = x0 ? y2 : y3)
⟹ (y4 = 7 ∨ y4 = 9)

Loop Unwinding

- All loops are unwound
  - can use different unwinding bounds for different loops
  - to check whether unwinding is sufficient special “unwinding assertion” claims are added
- If a program satisfies all of its claims and all unwinding assertions then it is correct!
- Same for backward `goto` jumps and recursive functions

Loop Unwinding

```
void f(...) {
  ...
  while(cond) {
    Body;
  }
  Remainder;
}
```

while() loops are unwound iteratively

Break / continue replaced by goto

Loop Unwinding

```
void f(...) {
  ...
  if(cond) {
    Body;
    while(cond) {
      Body;
    }
  }
  Remainder;
}
```

while() loops are unwound iteratively

Break / continue replaced by goto

Loop Unwinding

```
void f(...) {
  ...
  if(cond) {
    Body;
    if(cond) {
      Body;
      while(cond) {
        Body;
      }
    }
  }
  Remainder;
}
```

while() loops are unwound iteratively

Break / continue replaced by goto

Bounded unwinding

```
void f(...) {
  ...
  if(cond) {
    Body;
    if(cond) {
      Body;
      if(cond) {
        Body;
        assume(!cond)
      }
    }
  }
  Remainder;
}
```

while() loops are unwound iteratively

Break / continue replaced by goto

Assume inserted after last iteration: longer program runs are not considered for analysis

Unwinding assertion

```
void f(...) {
  ...
  if(cond) {
    Body;
    if(cond) {
      Body;
      if(cond) {
        Body;
        while(cond) {
          Body;
        }
      }
    }
  }
  Remainder;
}
```

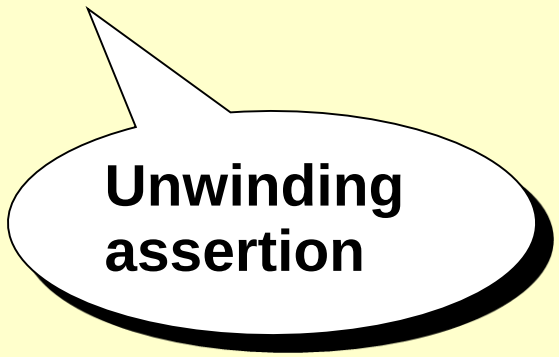
while() loops are unwound iteratively

Break / continue replaced by goto

Assertion inserted after last iteration: violated if program runs longer than bound permits

Unwinding assertion

```
void f(...) {
  ...
  if(cond) {
    Body;
    if(cond) {
      Body;
      if(cond) {
        Body;
        assert(!cond);
      }
    }
  }
  Remainder;
}
```



- while() loops are unwound iteratively
- Break / continue replaced by goto
- Assertion inserted after last iteration: violated if program runs longer than bound permits
- Positive correctness result!

Example: Sufficient Loop Unwinding

```
void f(...) {
  j = 1
  while (j <= 2)
    j = j + 1;
  Remainder;
}
```

unwind = 3

```
void f(...) {
  j = 1
  if(j <= 2) {
    j = j + 1;
    if(j <= 2) {
      j = j + 1;
      if(j <= 2) {
        j = j + 1;
        assert(!(j <= 2));
        }
      }
    }
  }
  Remainder;
}
```

Example: Insufficient Loop Unwinding

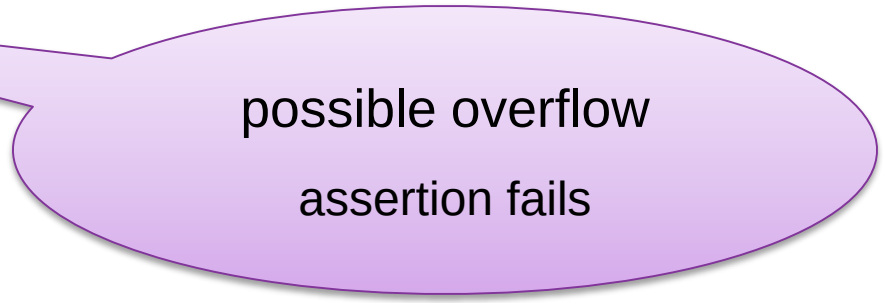
```
void f(...) {
  j = 1
  while (j <= 10)
    j = j + 1;
  Remainder;
}
```

unwind = 3

```
void f(...) {
  j = 1
  if(j <= 10) {
    j = j + 1;
    if(j <= 10) {
      j = j + 1;
      if(j <= 10) {
        j = j + 1;
        assert(!(j <= 10));
        }
      }
    }
  }
  Remainder;
}
```

Accurate Modeling of C Types

```
int x, y;
void main (void)
{
  x = nondet_int ();

  assume (x > 10);
  y = x + 1;

  assert (y > x);
}
```



Dangers of unrestricted assumptions

- Assumptions can lead to vacuous satisfaction

```
if (x > 0) {
  __CPROVER_assume (x < 0);
  assert (0); }
```

This program is passed by CBMC!

Assume must either be checked with assert or used as an idiom:

```
x = nondet_int ();
y = nondet_int ();
__CPROVER_assume (x < y);
```

Min3

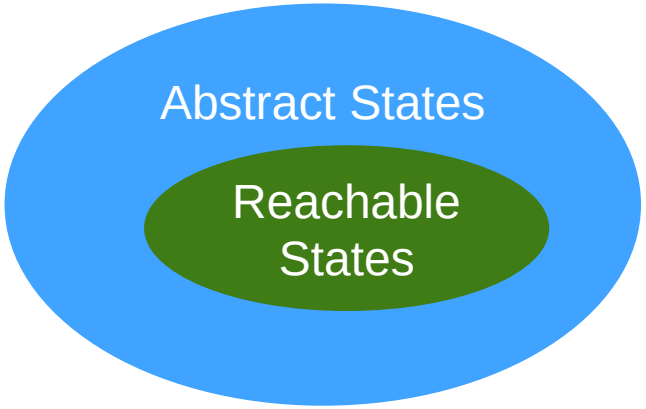
```
int x, y, z
int min

if (x < y )
   if ( x < z ) min = x
   else min = z
else if (y < z) min = y
else min = z

assert (min <= x && min <= y && min <= z )
```

x,y,z,min take non-deterministic value

Property to be checked

Proving Properties with CBMC

Invariants and Abstractions



If a property is satisfied by an abstract set of states then it is satisfied by the reachable states too.

Invariant : A property that holds for every run of the program
An Invariant defines an abstract set of states

Loop Invariant :

while(B) S

$\{I \ \wedge B\}\, S\, \{I\}$
$\{I\}\, while(B) S\, \{I \wedge \neg B\}$

Abstraction & Invariants

```
found = 0
while (nondet() && found == 0)
      i = nondet()
      if ( i == 1)
           found = 1

assert ( found == 0 || i == 1)
```

```
tmp = nondet()
found = nondet()
/* abstraction */
if (tmp && found != 1)
   assume ( (found == 0 || i == 1) && found == 0)
   i = nondet()
   if ( i == 1)
      found = 1
   assert ( found == 0 || i == 1) // Invariant

assume((tmp != 0  || found == 1) && (found == 0 || i == 1))

assert ( found == 0 || i == 1)
```

Exercise Problems

- Write a program and corresponding property to find the minimum and second minimum element from an array of integers.

iSIGCSE

- Community of CS Education researchers in the country
- Workshops, Conferences
- School, UG etc.
- Interested?
  - membership - goo.gl/kAx6o7