Concolic Testing: An Application of Solvers

Vijay Ganesh
Affiliation: University of Waterloo

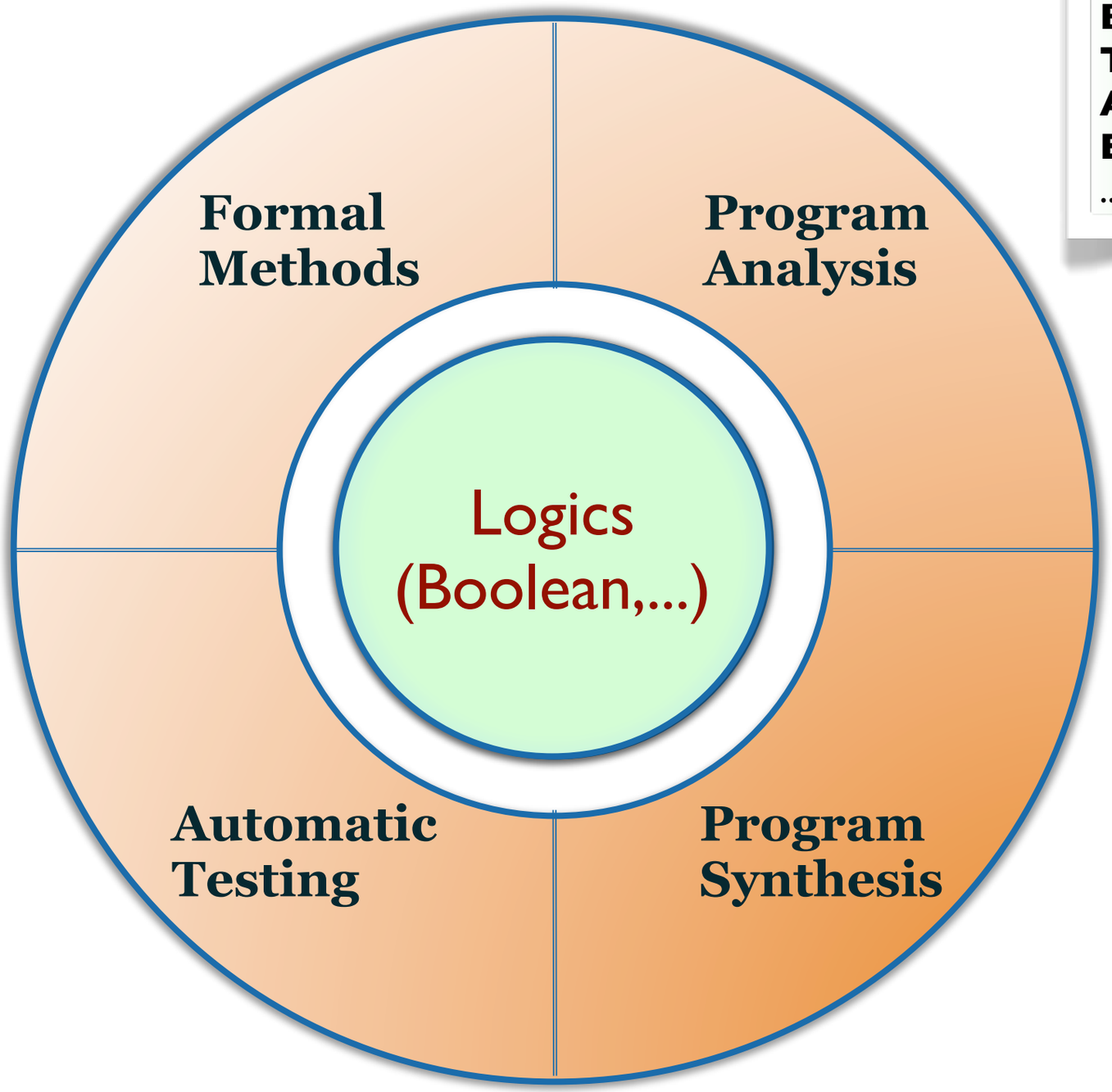
A Foundation for Software Engineering

Logic Abstractions of Computation

Bob Floyd (1967)
Tony Hoare (1968,70)
Amir Pnueli (1977)
Ed Clarke (1982)
...

A Foundation for Software Engineering

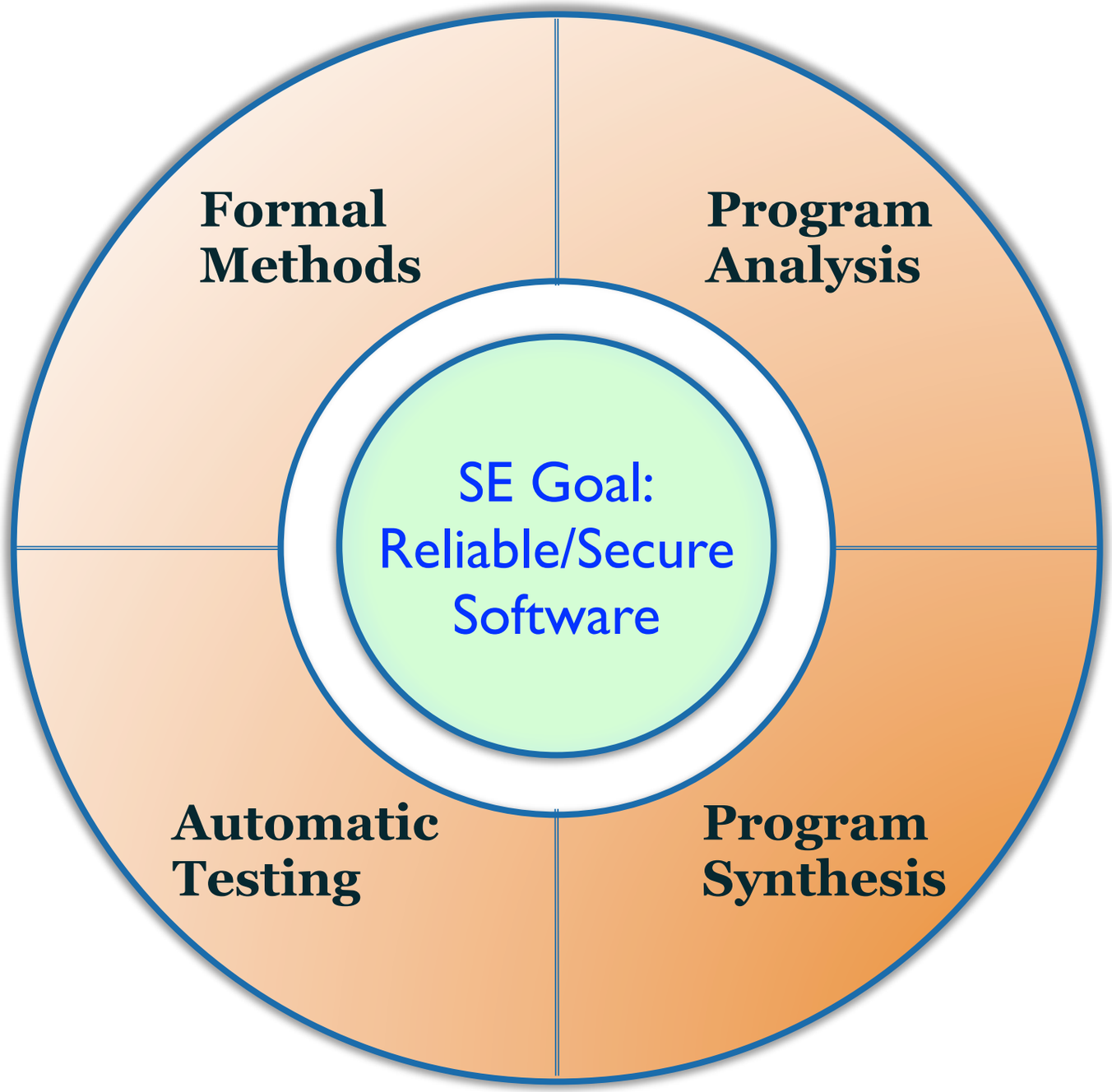
Logic Abstractions of Computation



Bob Floyd (1967)
Tony Hoare (1968,70)
Amir Pnueli (1977)
Ed Clarke (1982)
...

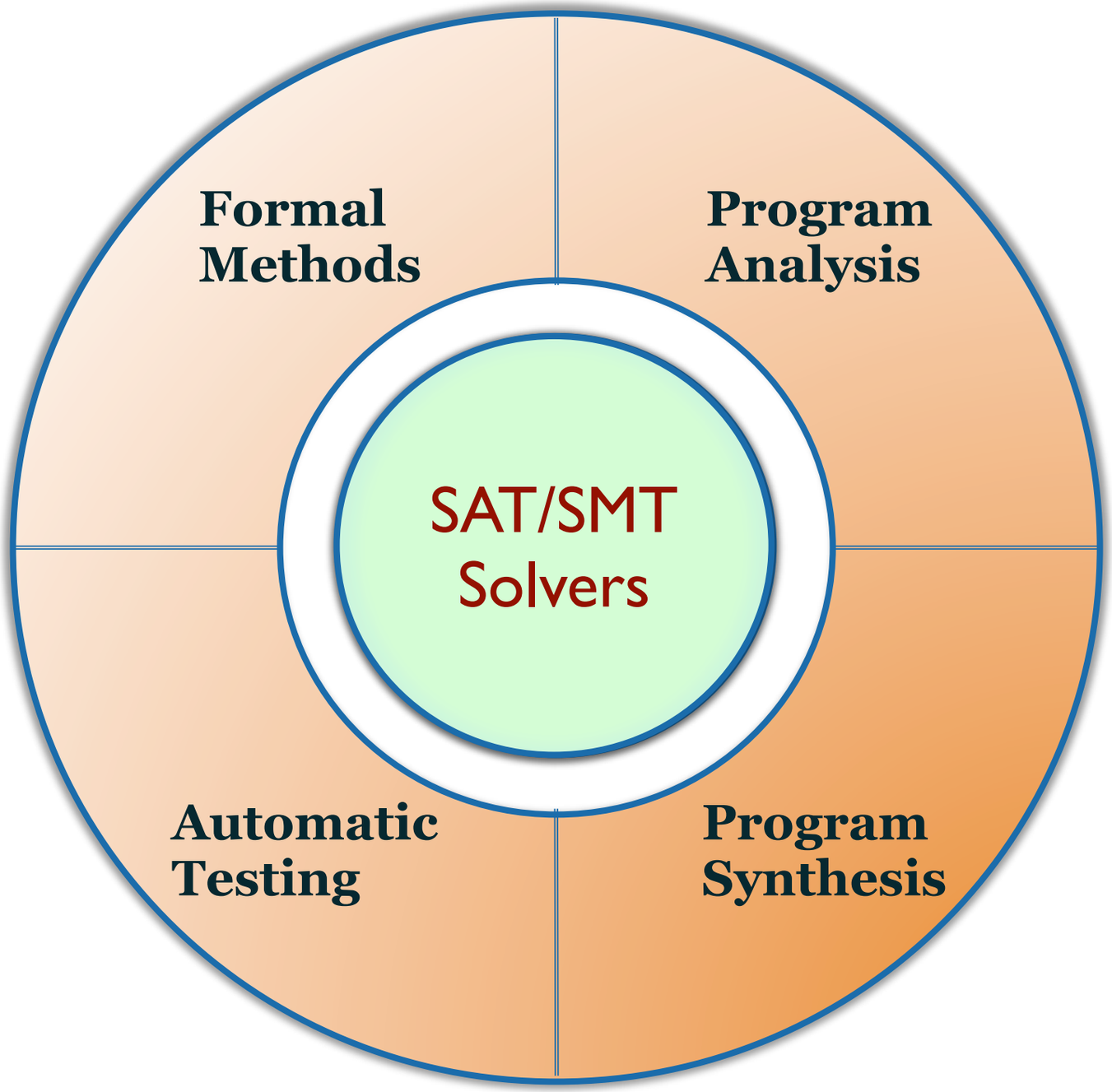
Software Engineering & SAT/SMT Solvers

An Indispensable Tactic for Any Strategy



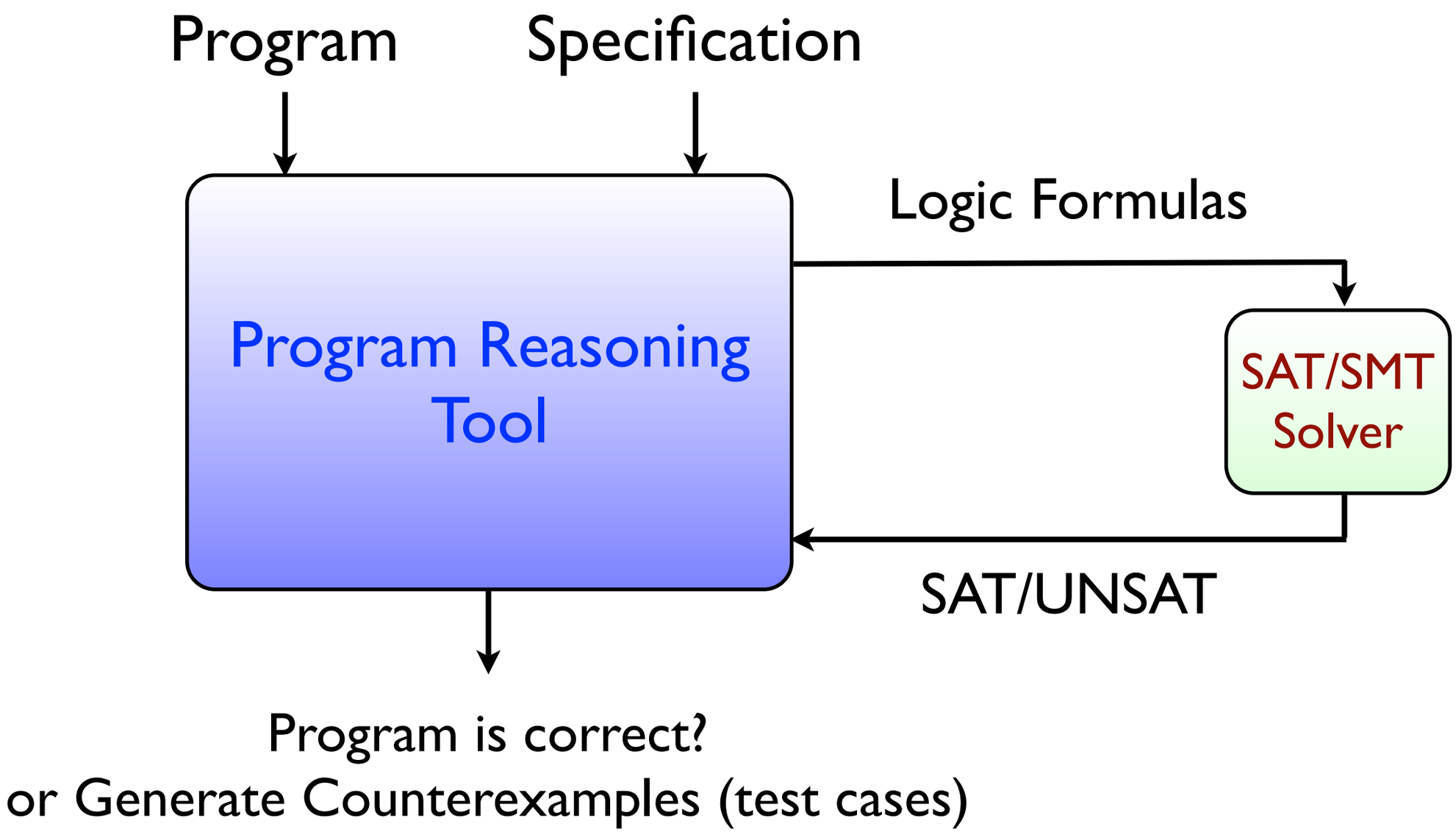
Software Engineering & SAT/SMT Solvers

An Indispensable Tactic for Any Strategy



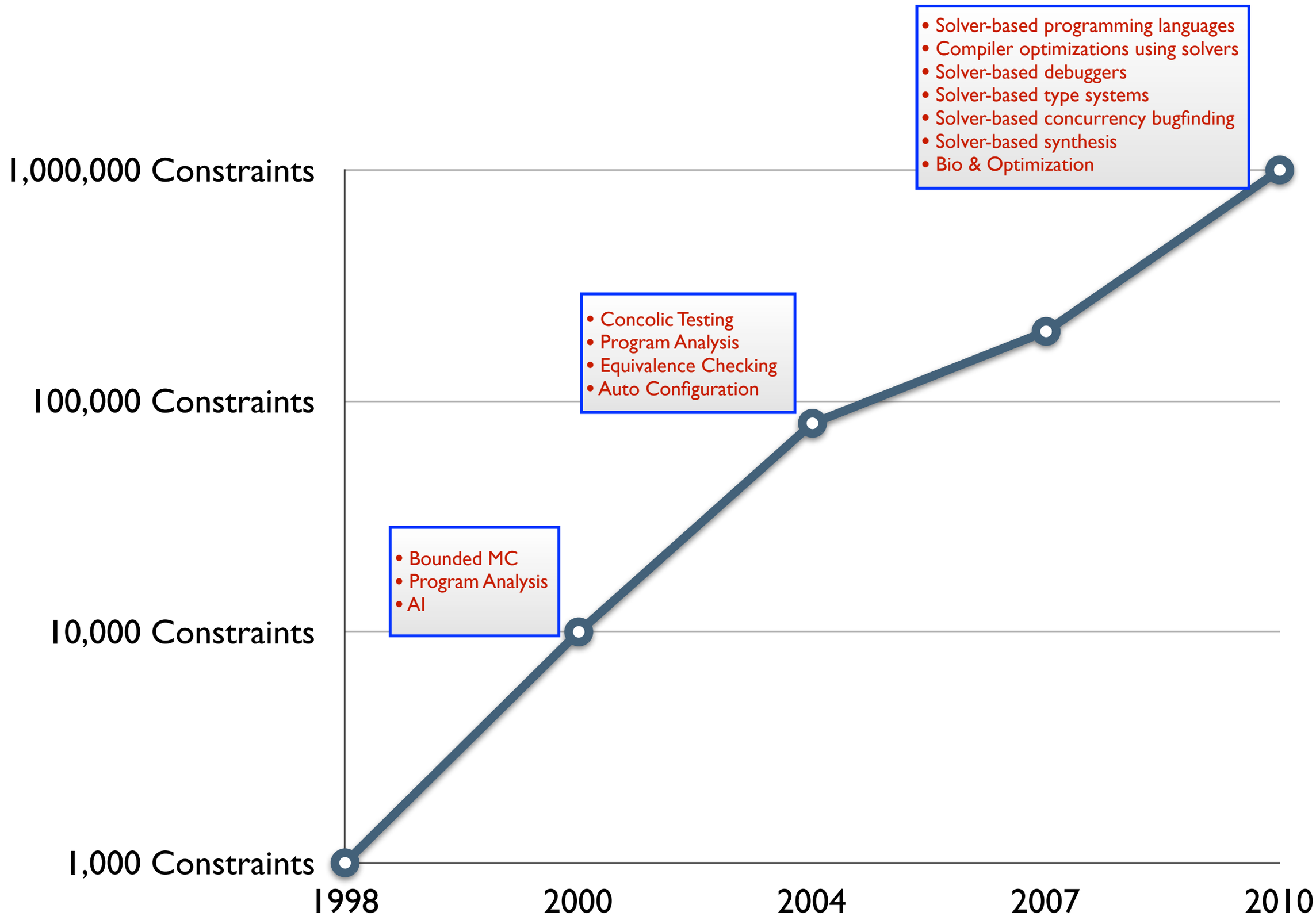
Software Engineering using Solvers

Engineering, Usability, Novelty



SAT/SMT Solver Research Story

A 1000x Improvement: Democratization of Logic



The SAT/SMT Problem

- Rich logics (Modular arithmetic, Arrays, Strings,...)
- NP-complete, PSPACE-complete,...
- Practical, scalable, usable, automatic
- Enable novel software reliability approaches

Dynamic Systematic Testing

Some History

Dynamic Systematic Testing
Some History

- ☑ Symbolic execution for testing first proposed by Lori Clarke (1975)
  ACM SIGSOFT Outstanding Researcher Award 2012

Dynamic Systematic Testing
Some History

- ☑ Symbolic execution for testing first proposed by Lori Clarke (1975)
  ACM SIGSOFT Outstanding Researcher Award 2012

- ☑ Follow up work by J.C. King (1976)

Dynamic Systematic Testing
Some History

- ☑ Symbolic execution for testing first proposed by Lori Clarke (1975)
  ACM SIGSOFT Outstanding Researcher Award 2012

- ☑ Follow up work by J.C. King (1976)

- ☑ Rediscovered/modified in the context of powerful solvers, analysis and appropriate concretizations by two independent groups

Dynamic Systematic Testing
Some History

- ☑ Symbolic execution for testing first proposed by Lori Clarke (1975)
  ACM SIGSOFT Outstanding Researcher Award 2012

- ☑ Follow up work by J.C. King (1976)

- ☑ Rediscovered/modified in the context of powerful solvers, analysis and appropriate concretizations by two independent groups

  - ☑ Patrice Godefroid and Koushik Sen (2005)

Dynamic Systematic Testing
Some History

- ☑ Symbolic execution for testing first proposed by Lori Clarke (1975)
  ACM SIGSOFT Outstanding Researcher Award 2012

- ☑ Follow up work by J.C. King (1976)

- ☑ Rediscovered/modified in the context of powerful solvers, analysis and appropriate concretizations by two independent groups

  - ☑ Patrice Godefroid and Koushik Sen (2005)

  - ☑ Dawson Engler et al. (2005)

Dynamic Systematic Testing
Some History

- ☑ Symbolic execution for testing first proposed by Lori Clarke (1975)
  ACM SIGSOFT Outstanding Researcher Award 2012

- ☑ Follow up work by J.C. King (1976)

- ☑ Rediscovered/modified in the context of powerful solvers, analysis and appropriate concretizations by two independent groups

  - ☑ Patrice Godefroid and Koushik Sen (2005)

  - ☑ Dawson Engler et al. (2005)

- ☑ Many follow up works by George Candea, Dawn Song, David Molnar, researchers in the audience,...

Dynamic Systematic Testing
Some History

- ☑ Symbolic execution for testing first proposed by Lori Clarke (1975)
  ACM SIGSOFT Outstanding Researcher Award 2012

- ☑ Follow up work by J.C. King (1976)

- ☑ Rediscovered/modified in the context of powerful solvers, analysis and appropriate concretizations by two independent groups

  - ☑ Patrice Godefroid and Koushik Sen (2005)

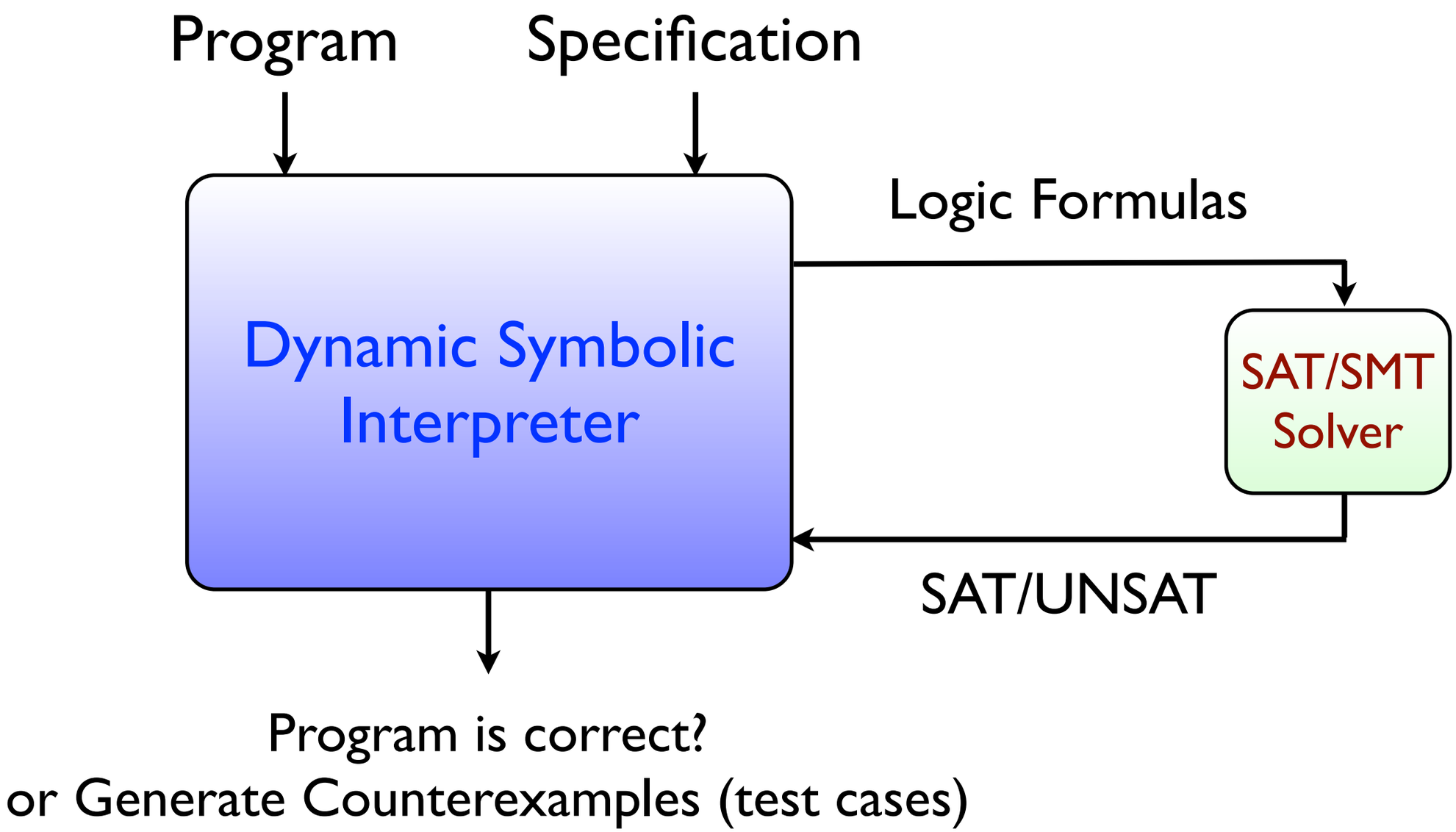
  - ☑ Dawson Engler et al. (2005)

- ☑ Many follow up works by George Candea, Dawn Song, David Molnar, researchers in the audience,...

- ☑ Beyond testing: Fault localization, repair, security,...

Dynamic Symbolic Testing

Symbolic/Concrete Execution + Solvers



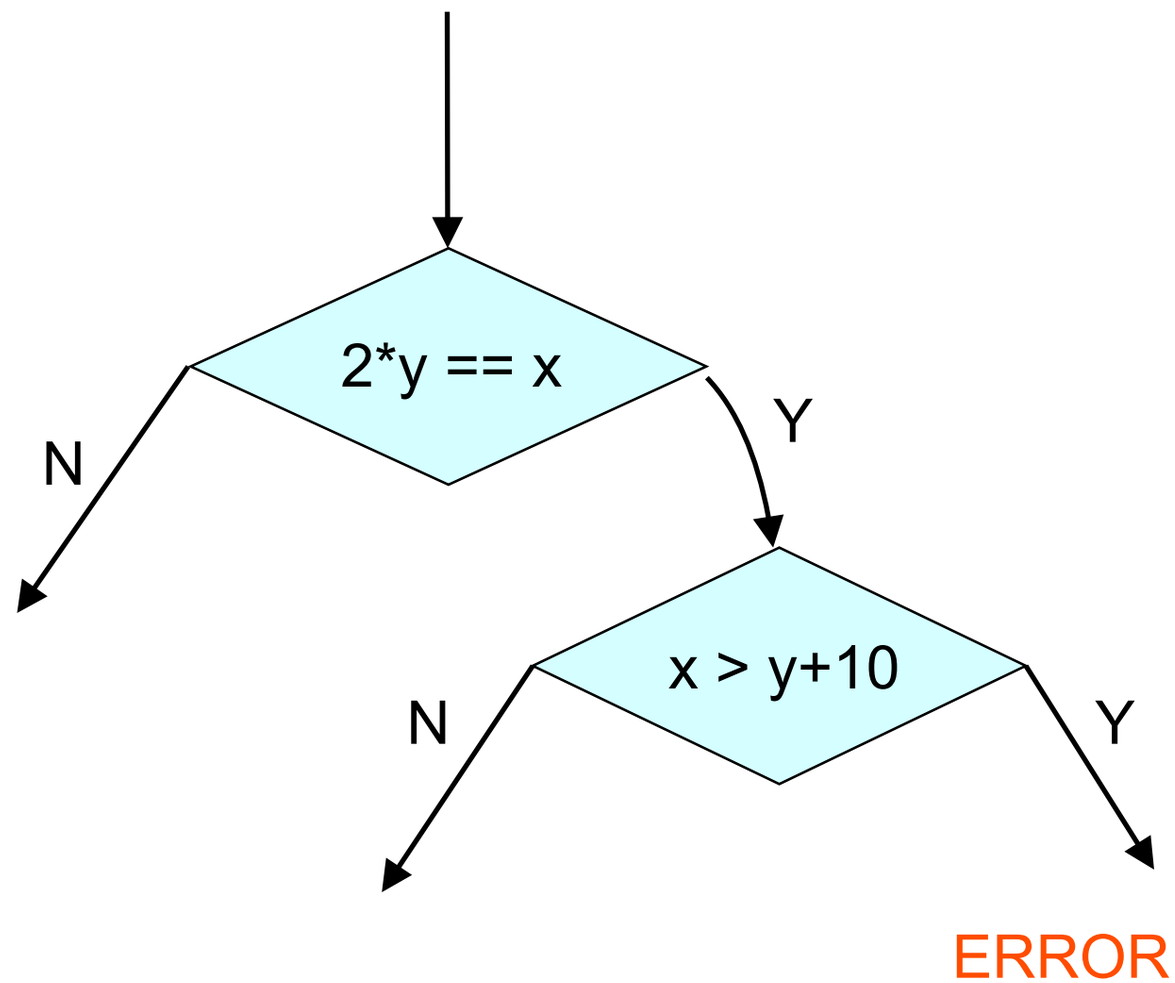
Concolic Testing: Example

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```

The concolic testing slides are courtesy Koushik Sen

Concolic Testing: Example

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```



Concolic Testing Approach

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```

Concrete Execution	Symbolic Execution	
concrete state	symbolic state	path condition
$x = 22, y = 7$	$x = x_0, y = y_0$	

Concolic Testing Approach

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```

Concrete Execution	Symbolic Execution	
concrete state	symbolic state	path condition
$x = 22, y = 7,$	$x = x_0, y = y_0,$	
$z = 14$	$z = 2*y_0$	

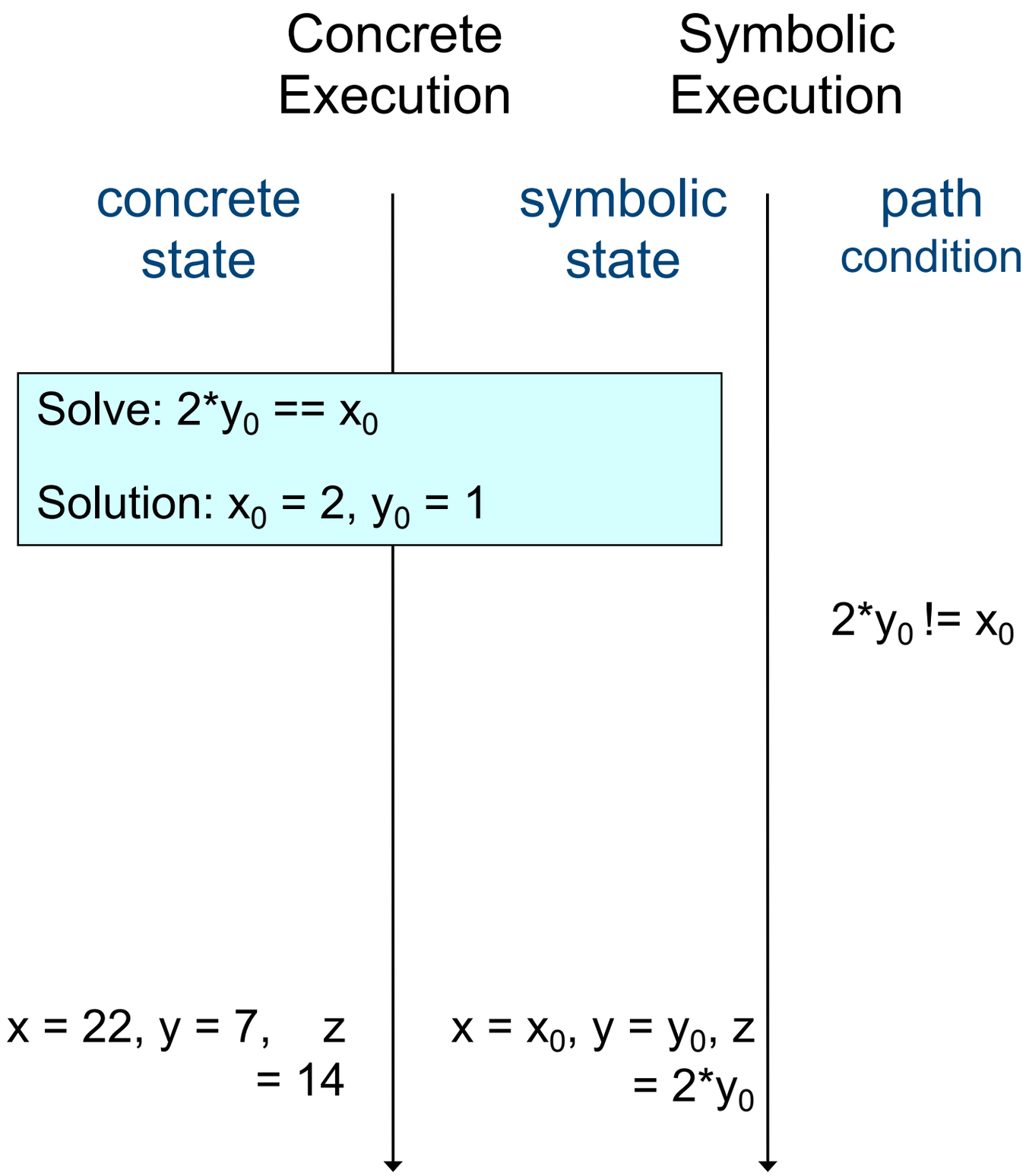
Concolic Testing Approach

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```

Concrete Execution	Symbolic Execution	
concrete state	symbolic state	path condition
$x = 22, y = 7, z = 14$	$x = x_0, y = y_0, z = 2*y_0$	$2*y_0 != x_0$

Concolic Testing Approach

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```



Concolic Testing Approach

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```

Concrete Execution	Symbolic Execution	
concrete state	symbolic state	path condition
$x = 2, y = 1,$ $z = 2$	$x = x_0, y = y_0,$ $z = 2*y_0$	$2*y_0 == x_0$

Concolic Testing Approach

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```

Concrete Execution	Symbolic Execution	
concrete state	symbolic state	path condition
$x = 2, y = 1, z = 2$	$x = x_0, y = y_0, z = 2*y_0$	$2*y_0 == x_0$
		$x_0 > y_0+10$

Concolic Testing Approach

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```

Concrete Execution	Symbolic Execution	
concrete state	symbolic state	path condition
$x = 2, y = 1, z = 2$	$x = x_0, y = y_0, z = 2*y_0$	$2*y_0 == x_0$
		$x_0 \cdot y_0+10$

Solve: $(2*y_0 == x_0)$ **AND** $(x_0 > y_0 + 10)$

Solution: $x_0 = 30$, $y_0 = 15$

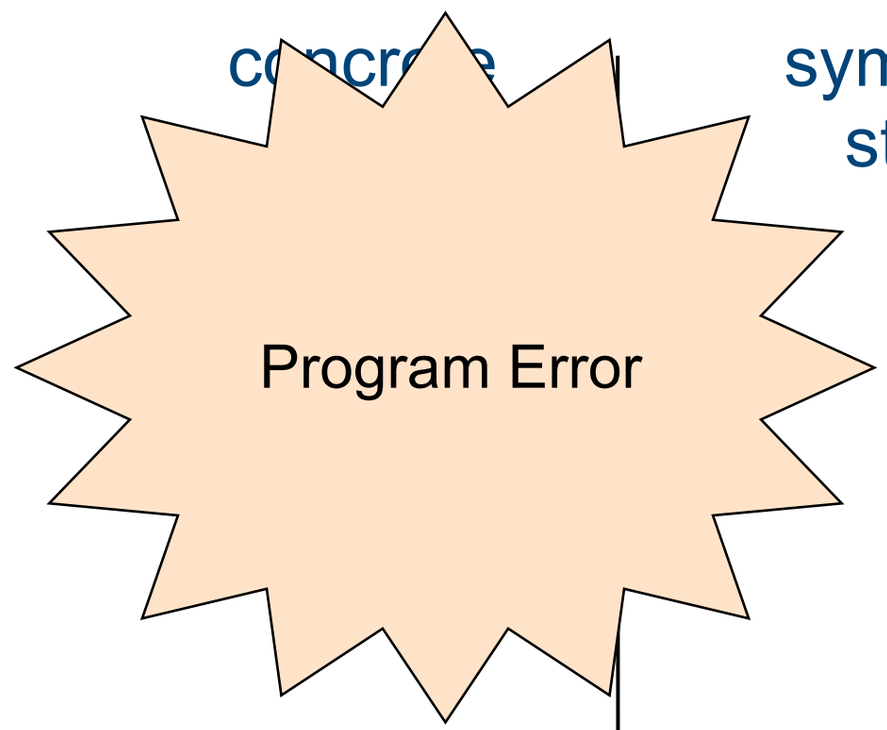
Concolic Testing Approach

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```

Concrete Execution	Symbolic Execution	
concrete state	symbolic state	path condition
$x = 30, y = 15$	$x = x_0, y = y_0$	

Concolic Testing Approach

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```



Concrete Execution	Symbolic Execution	
concrete state	symbolic state	path condition
		$2*y_0 == x_0$
		$x_0 > y_0+10$
$x = 30, y = 15$	$x = x_0, y = y_0$	

Explicit Path (not State) Model Checking

- Traverse all execution paths one by one to detect errors
  - assertion violations
  - program crash
  - uncaught exceptions
- combine with valgrind to discover memory errors