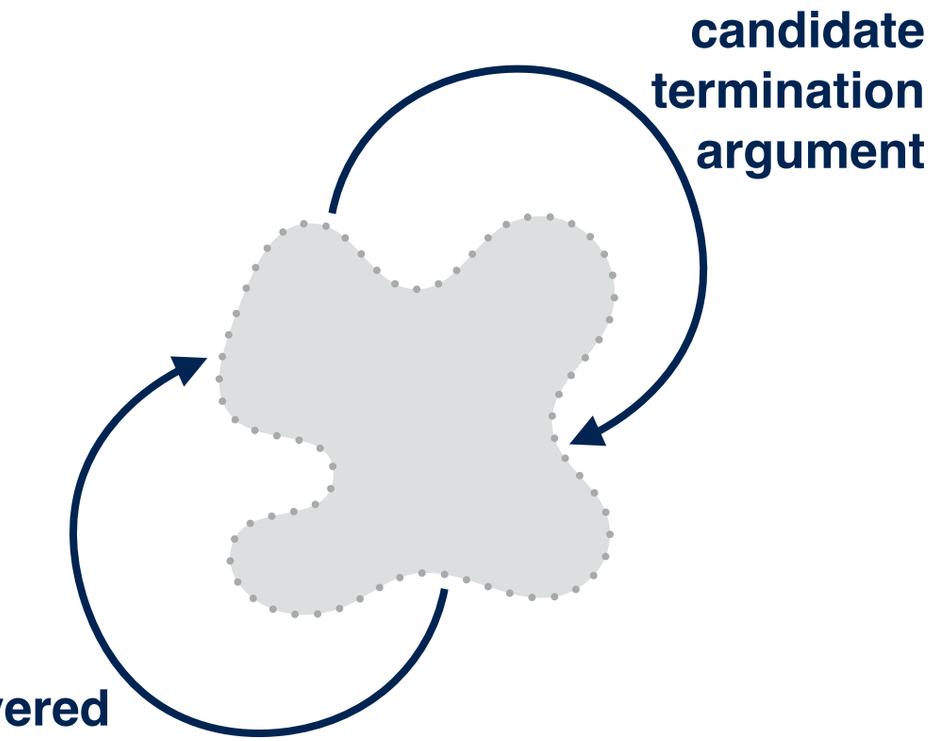
Synthesizing Ranking Functions From Bits and Pieces

Caterina Urban[1,2], Arie Gurfinkel[2], Temesghen Kahsai[3]

[1] ETH Zürich, Switzerland
[2] Carnegie Mellon University, USA
[3] NASA Ames Research Center, USA

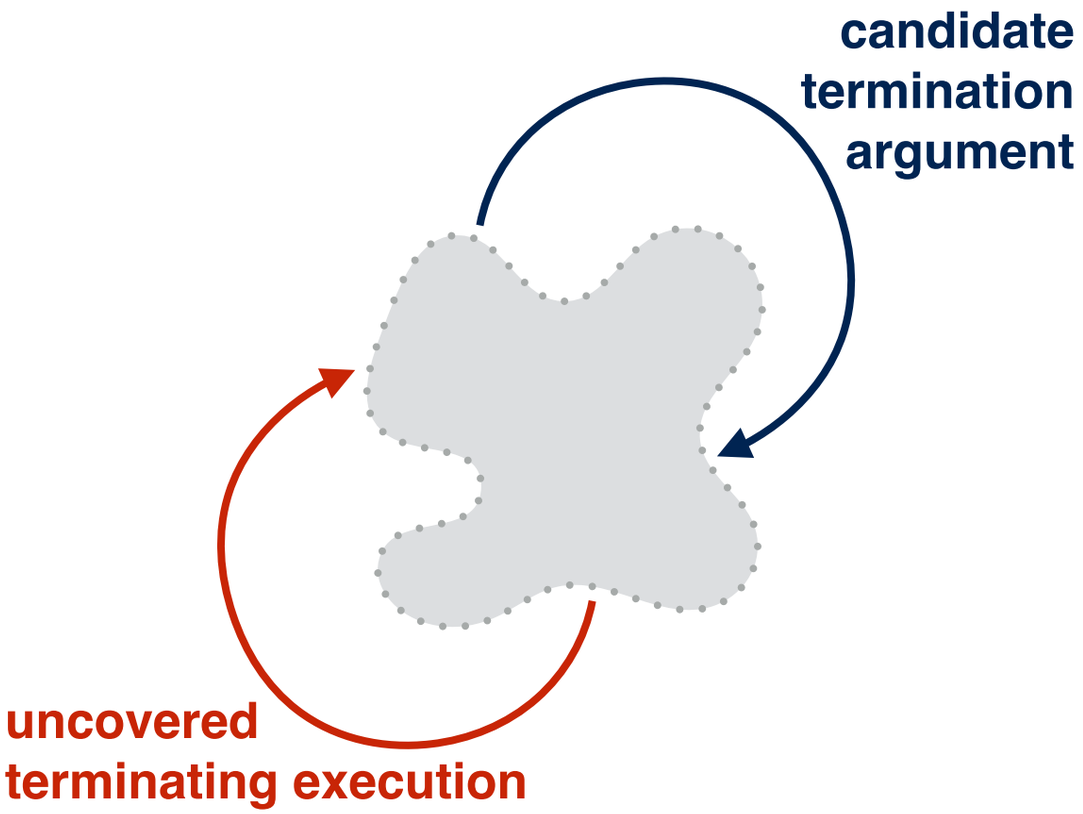
Proving Termination via Safety

the state of the art…



- Terminator/T2
- Ultimate Büchi Automizer
- …

what we propose…

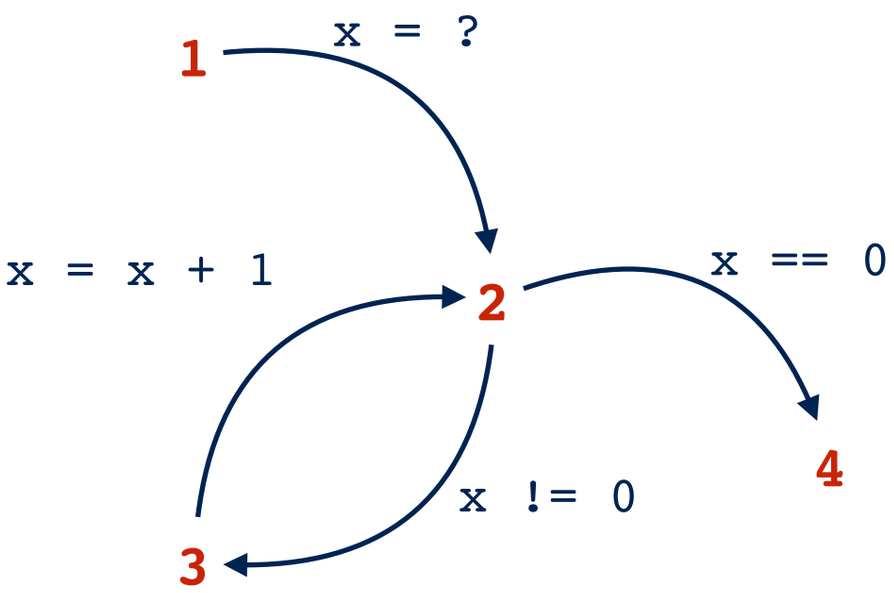


- **lightweight**, fast
- more **informative**

innovative use of safety verification techniques **for** proving **termination**

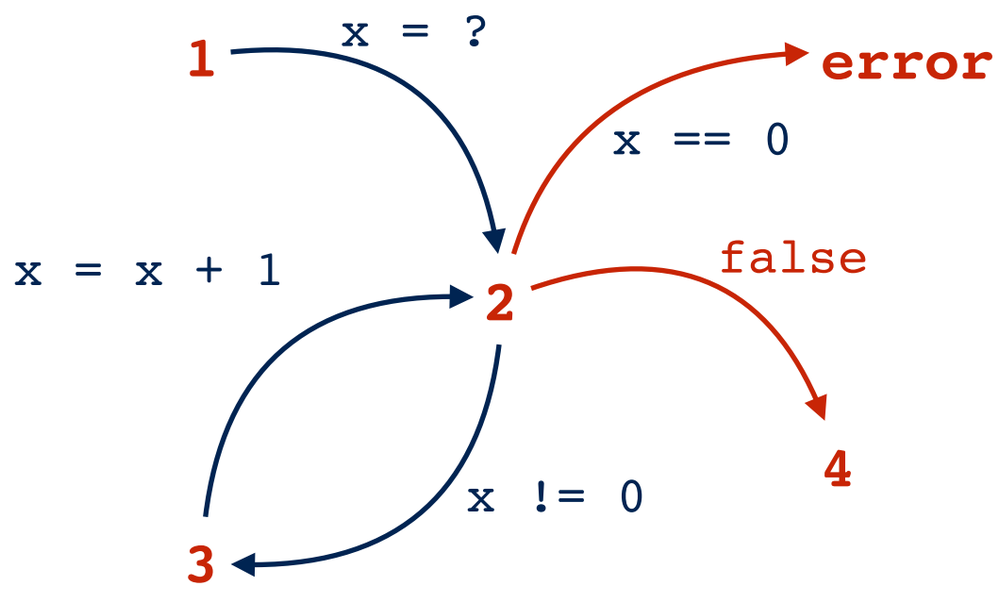
Checking Non-Termination via Error Unreachability

```
int 1x = ?;
while 2(x != 0) {
  3x = x + 1;
}
4
```




```
int 1x = ?;
while 2(x != 0) {
  3x = x + 1;
}
assert(false);4
```

unreachable error location = non-termination



counterexamples = terminating executions

Checking a Ranking Function via Error Unreachability


- **strictly decreasing**
- **bounded** from below

```
int x = ?;
while (x != 0) {
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
```

$$r(x) = \begin{cases} -x & x \leq 0 \\ 21 - x & 0 < x < 10 \\ x + 1 & 10 \leq x \end{cases}$$

```
int x = ?;
int r = max{-x, 21-x, x+1};
while (x != 0) {
  r = r - 1;
  assert(r >= 0);
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
```

$$r(x) = max\{-x, 21 - x, x + 1\}$$

Checking a Ranking Function via Error Unreachability


- **strictly decreasing**
- **bounded** from below

```
int x = ?;
while (x != 0) {
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
```

```
int x = ?;
int r = max{-x, 21-x, x+1};
while (x != 0) {
  r = r - 1;
  assert(r >= 0);
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
```

counterexamples = non-terminating executions or uncovered terminating executions

Uncovered Terminating Executions

```
int x = ?;
int r = max{-x, 21-x, x+1};
while (x != 0) {
  r = r - 1;
  assert(r >= 0);
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
```

```
int x = ?;
int r = max{-x,21-x,x+1};
while (x != 0) {
  r = r - 1;
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
assert(r >= 0);
```

counterexamples = uncovered terminating executions

Ranking Function Synthesis

```
int x = ?;
while (x != 0) {
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
```

Ranking Function Synthesis

```
int x = ?;
while (x != 0) {
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
```

```
int x = ?;
int r = 0;
while (x != 0) {
  r = r - 1;
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
assert(r >= 0);
```

r = 0

candidate
ranking function

T_{TERM}

Ranking Function Synthesis

```
int x = ?;
while (x != 0) {
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
```

```
int x = ?;
int r = 0;
while (x != 0) {
  r = r - 1;
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
assert(r >= 0);
```

r = 0

candidate
ranking function

T_{TERM}

Ranking Function Synthesis

Terminating Trace Sample

x = -1

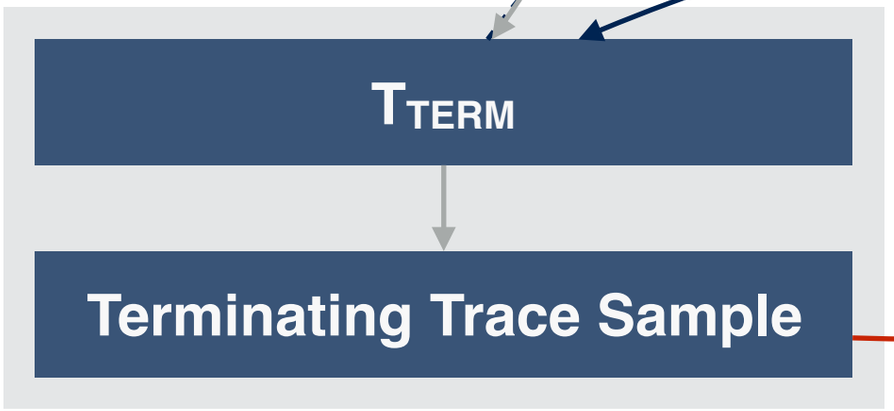
uncovered
terminating execution

Ranking Function Synthesis

```
int x = ?;
while (x != 0) {
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
```

```
int x = ?;
int r = 0;
while (x != 0) {
  r = r - 1;
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
assert(r >= 0);
```

r = 0



candidate ranking function

Ranking Function Synthesis

x = -1

uncovered terminating execution

- **piecewise-defined**
- lexicographic
- multiphase

Ranking Function Synthesis

```
int x = ?;
while (x != 0) {
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
```

```
int x = ?;
int r = 1;
while (x != 0) {
  r = r - 1;
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
assert(r >= 0);
```

r = 1

T_{TERM}

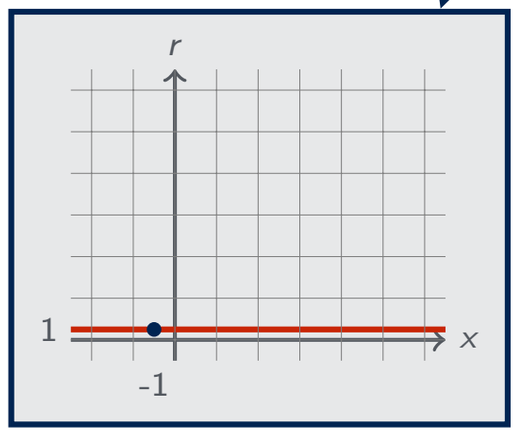
Terminating Trace Sample

x = -1

uncovered
terminating execution

candidate
ranking function

Ranking Function Synthesis



Ranking Function Synthesis

```
int x = ?;
while (x != 0) {
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
```

```
int x = ?;
int r = 1;
while (x != 0) {
  r = r - 1;
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
assert(r >= 0);
```

r = 1

T_{TERM}

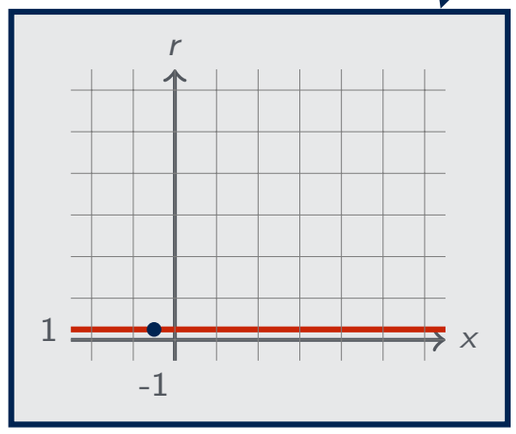
Terminating Trace Sample

x = -2

uncovered
terminating execution

candidate
ranking function

Ranking Function Synthesis



Ranking Function Synthesis

```
int x = ?;
while (x != 0) {
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
```

```
int x = ?;
int r = max{0,-x};
while (x != 0) {
  r = r - 1;
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
assert(r >= 0);
```

r = max{0,-x}

candidate ranking function

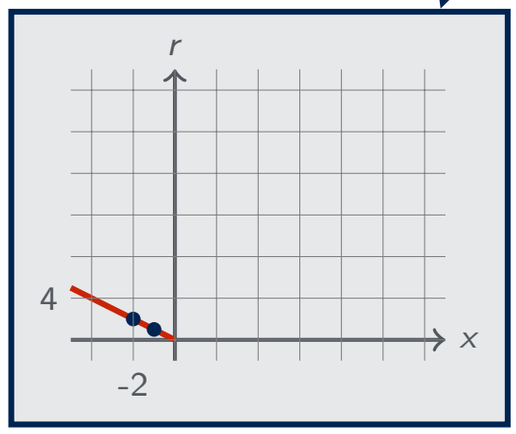
T_{TERM}

Terminating Trace Sample

x = -2

uncovered terminating execution

Ranking Function Synthesis



Ranking Function Synthesis

```
int x = ?;
while (x != 0) {
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
```

```
int x = ?;
int r = max{0,-x};
while (x != 0) {
  r = r - 1;
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
assert(r >= 0);
```

r = max{0,-x}

candidate ranking function

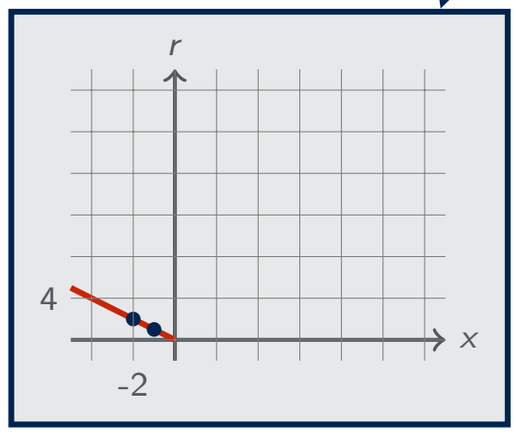
T_{TERM}

Terminating Trace Sample

Ranking Function Synthesis

x = 10

uncovered terminating execution



Ranking Function Synthesis

```
int x = ?;
while (x != 0) {
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
```

```
int x = ?;
int r = max{11,-x};
while (x != 0) {
  r = r - 1;
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
assert(r >= 0);
```

r = max{11,-x}

candidate ranking function

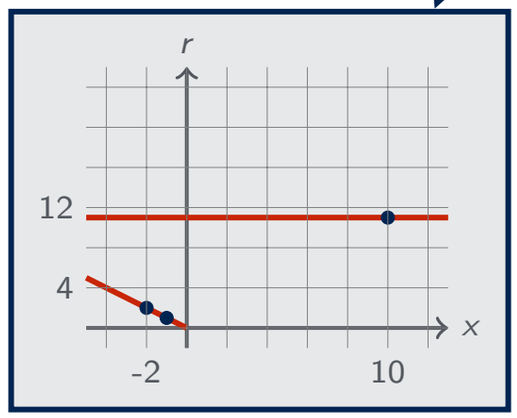
T_{TERM}

Ranking Function Synthesis

Terminating Trace Sample

x = 10

uncovered terminating execution



Ranking Function Synthesis

```
int x = ?;
while (x != 0) {
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
```

```
int x = ?;
int r = max{11,-x};
while (x != 0) {
  r = r - 1;
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
assert(r >= 0);
```

r = max{11,-x}

candidate ranking function

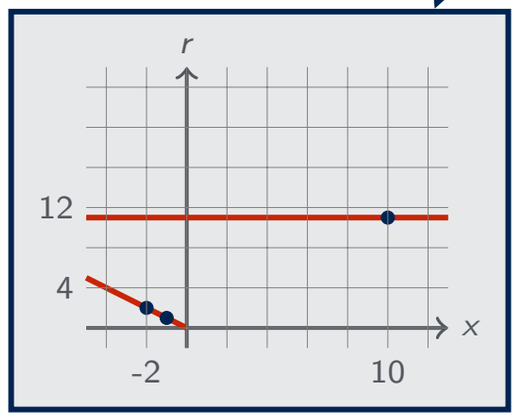
T_{TERM}

Ranking Function Synthesis

Terminating Trace Sample

x = 11

uncovered terminating execution



Ranking Function Synthesis

```
int x = ?;
while (x != 0) {
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
```

```
int x = ?;
int r = max{-x,x+1};
while (x != 0) {
  r = r - 1;
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
assert(r >= 0);
```

r = max{-x,x+1}

candidate ranking function

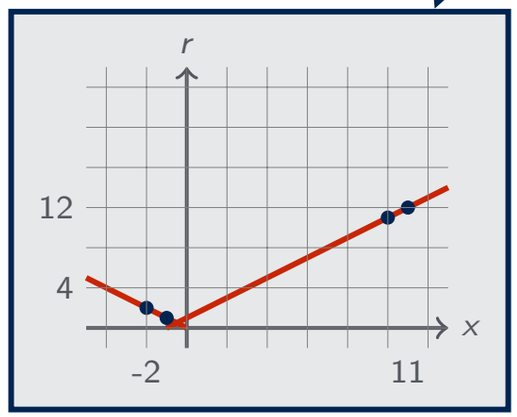
T_{TERM}

Terminating Trace Sample

Ranking Function Synthesis

x = 11

uncovered terminating execution



Ranking Function Synthesis

```
int x = ?;
while (x != 0) {
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
```

```
int x = ?;
int r = max{-x,x+1};
while (x != 0) {
  r = r - 1;
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
assert(r >= 0);
```

r = max{-x,x+1}

candidate ranking function

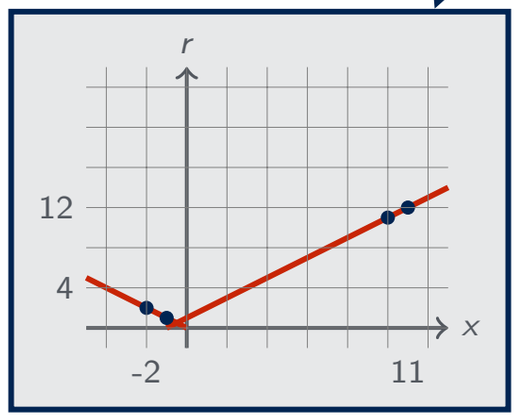
T_{TERM}

Terminating Trace Sample

Ranking Function Synthesis

x = 9

uncovered terminating execution



Ranking Function Synthesis

```
int x = ?;
while (x != 0) {
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
```

```
int x = ?;
int r = max{-x,21-x,x+1};
while (x != 0) {
  r = r - 1;
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
assert(r >= 0);
```

r = max{-x,21-x,x+1}

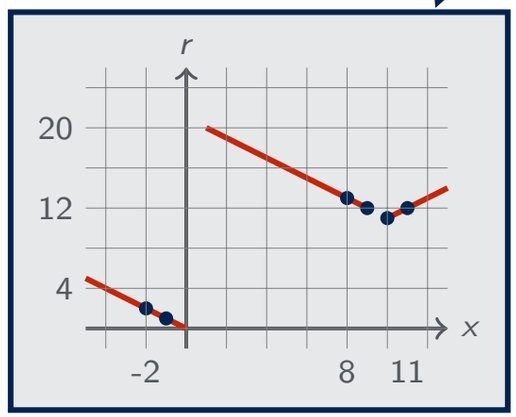
candidate ranking function

T_{TERM}

Terminating Trace Sample

Ranking Function Synthesis

uncovered terminating execution



Ranking Function Synthesis

```
int x = ?;
while (x != 0) {
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
```

```
int x = ?;
int r = max{-x,21-x,x+1};
while (x != 0) {
  r = r - 1;
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
assert(r >= 0);
```

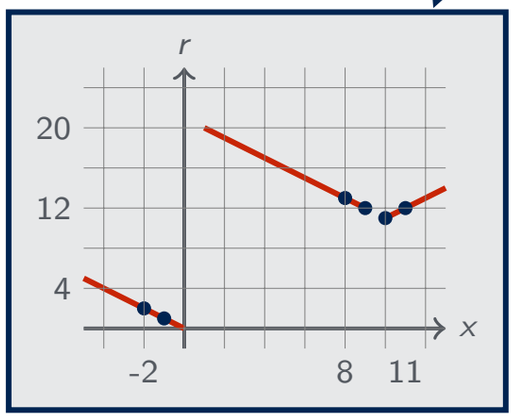
r = max{-x,21-x,x+1}

candidate ranking function

T_{TERM}

Terminating Trace Sample

Ranking Function Synthesis



Ranking Function Synthesis

```
int x = ?;
while (x != 0) {
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
```

```
int x = ?;
int r = max{-x,21-x,x+1};
while (x != 0) {
  r = r - 1;
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
assert(r >= 0);
```

r = max{-x,21-x,x+1}

candidate ranking function

Ranking Function Synthesis

T_{TERM}

Terminating Trace Sample

T_{RANK}

Ranking Function Check

✔ rank

```
int x = ?;
int r = max{-x, 21-x, x+1};
while (x != 0) {
  r = r - 1;
  assert(r >= 0);
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
```

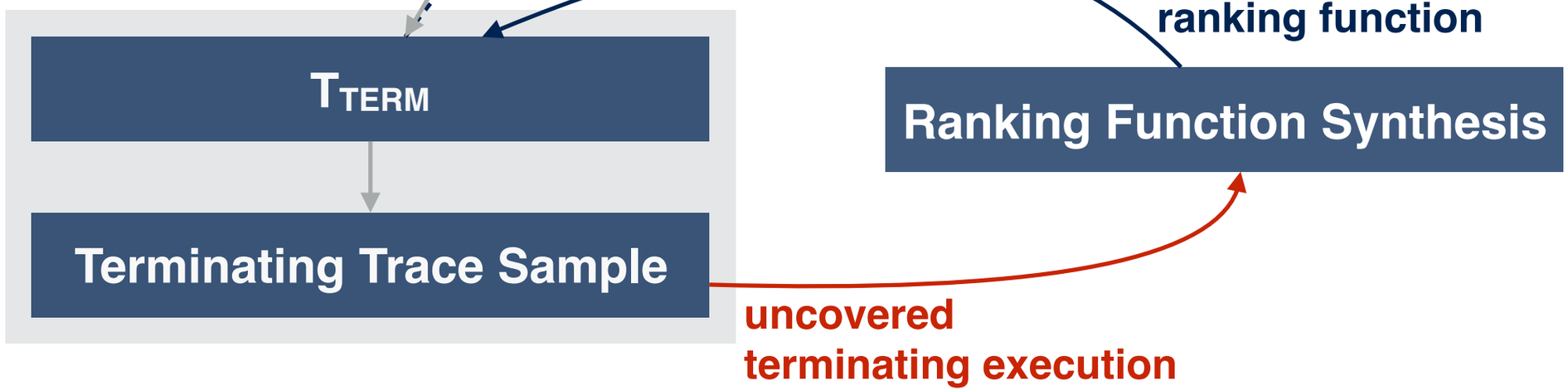
Non-Terminating Executions

```
int x = ?;
while (x != 0) {
  x = x + 1;
}
```

Non-Terminating Executions

```
int x = ?;
while (x != 0) {
  x = x + 1;
}
```

```
int x = ?;
int r = max{0,-x};
while (x != 0) {
  r = r - 1;
  x = x + 1;
}
assert(r >= 0);
```



Non-Terminating Executions

```
int x = ?;
while (x != 0) {
  x = x + 1;
}
```

```
int x = ?;
int r = max{0,-x};
while (x != 0) {
  r = r - 1;
  x = x + 1;
}
assert(r >= 0);
```

y = max{0,-x}

candidate
ranking function

T_{TERM}

Ranking Function Synthesis

Terminating Trace Sample

Non-Terminating Executions

```
int x = ?;
while (x != 0) {
  x = x + 1;
}
```

```
int x = ?;
int r = max{0,-x};
while (x != 0) {
  r = r - 1;
  x = x + 1;
}
assert(r >= 0);
```

y = max{0,-x}

candidate ranking function

T_{TERM}

Terminating Trace Sample

T_{RANK}

Ranking Function Check

✗ x = 1

Ranking Function Synthesis

```
int x = ?;
int r = max{0,-x};
while (x != 0) {
  r = r - 1;
  assert(r >= 0);
  x = x + 1;
}
```

Experimental Evaluation

SeaHorn

- **190 terminating C programs** from SV-COMP 2015
- comparison with **participants to SV-COMP 2015**

	Tot	Time
AProVE	129	10.77s
FuncTion	111	0.55s
HIPTnT+	152	0.62s
SeaHorn	135	1.71s
Ultimate	109	8.45s

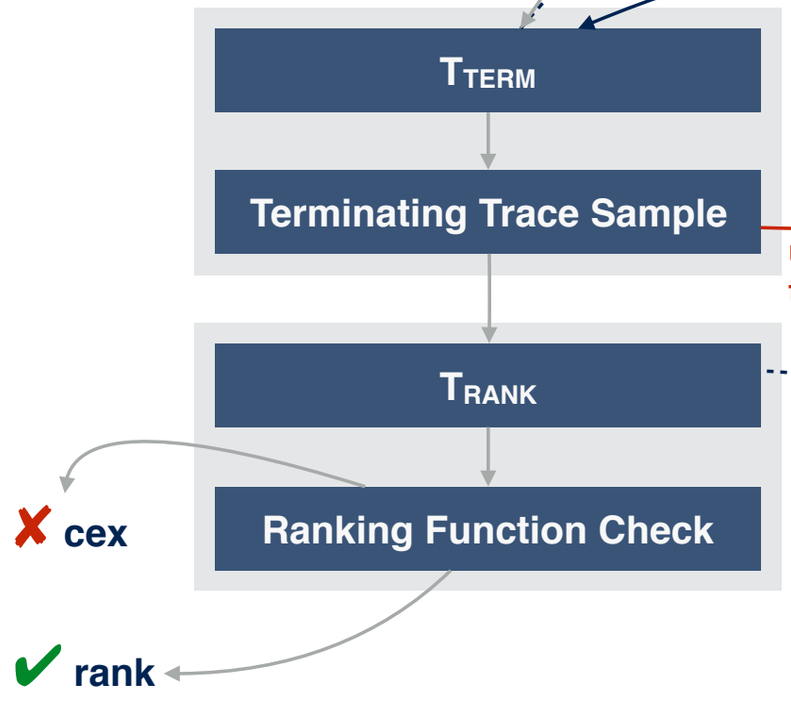
	SeaHorn	
AProVE	39	33
FuncTion	50	26
HIPTnT+	16	33
Ultimate	55	29

- **5** programs proved **only by SeaHorn**
- **1** program proved **only by AProVE**
- **1** program proved **only by FuncTion**
- **2** programs proved **only by HIPTnT+**
- **5** programs proved **only by Ultimate**
- **6** programs proved **by none**

http://seahorn.github.io

```
int x = ?;
while (x != 0) {
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
```

```
int x = ?;
int r = max{-x,21-x,x+1};
while (x != 0) {
  r = r - 1;
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
assert(r >= 0);
```



candidate ranking function

Ranking Function Synthesis

- piecewise-defined
- lexicographic
- multiphase

uncovered terminating execution

```
int x = ?;
int r = max{-x, 21-x, x+1};
while (x != 0) {
  r = r - 1;
  assert(r >= 0);
  if (x < 10) {
    x = x + 1;
  } else { x = -x; }
}
```

Future Work

- synthesis of **non-linear** ranking functions
- inference of **preconditions**
- other **liveness** properties

Thanks!