FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Difference Constraints: An adequate Abstraction for Complexity Analysis of Imperative Programs

Florian Zuleger
Technische Universität Wien
FMCAD, 28.09.2015

Joint work with Moritz Sinn, Helmut Veith

Bounds and Complexity

```
foo(uint n)
x = n;
y = n;
while(x > 0)
t1:{ x--;
     y = y + 2;
}
z = y;
while(z > 0)
t2:  z--;
```

Local Bound(t1): x

Variable that decreases when t1 is executed.

Bounds and Complexity

```
foo(uint n)
x = n;
y = n;
while(x > 0)
t1:{ x--;
     y = y + 2;
}
z = y;
while(z > 0)
t2:  z--;
```

Local Bound(t1): x

Transition Bound(t1): n

# of visits to transition t1

Bounds and Complexity

```
foo(uint n)
x = n;
y = n;
while(x > 0)
t1:{ x--;
     y = y + 2;
}
z = y;
while(z > 0)
t2:  z--;
```

Local Bound(t1): x

Transition Bound(t1): n

Local Bound(t2): z

Variable Bound(y): 3n

Invariant of shape
$y \leq 3n$

Bounds and Complexity

```
foo(uint n)
x = n;
y = n;
while(x > 0)
t1:{ x--;
     y = y + 2;
}
z = y;
while(z > 0)
t2:  z--;
```

Local Bound(t1): x

Transition Bound(t1): n

Local Bound(t2): z

Variable Bound(y): 3n

Transition Bound(t2): 3n

Complexity: 4n

Bounds and Complexity

Bound Analysis:

- # of visits to a transition
- # of visits to multiple transitions
- # of iterations of a loop
- resource consumption of a program
- complexity of a program
- upper bound on the value of a variable

} Introduce a counter c and increment at places of interest

Intuition: *All these bound analysis problems are related and can be reduced to each other.*

Applications of Bound Analysis

Verification:

- Computing bounds on resource consumption (CPU time, memory, bandwidth,…)
- Termination analysis with quantitative information on program progress

Program understanding:

- Static profiling
- Understanding program performance

Bound Analysis and the Halting Problem

For imperative programs:

Halting Problem = termination analysis of loops

→Bound analysis is a hard problem!

```
while(n != 0)
    if (n % 2 = 0)
        n = n / 2;
    else
        n = 3n+1;
```

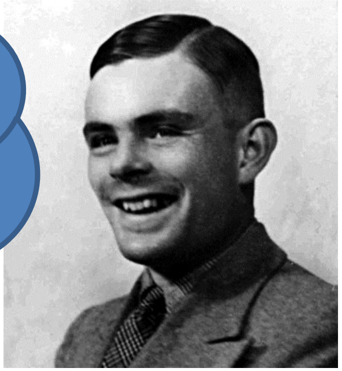
Typical programs?

```
while(n > 0) {
    m := n--;
    while(m > 0 && ?)
        m--;
}
```

Bound Analysis and the Halting Problem

Collatz Conjecture

Real-life Programs



ops

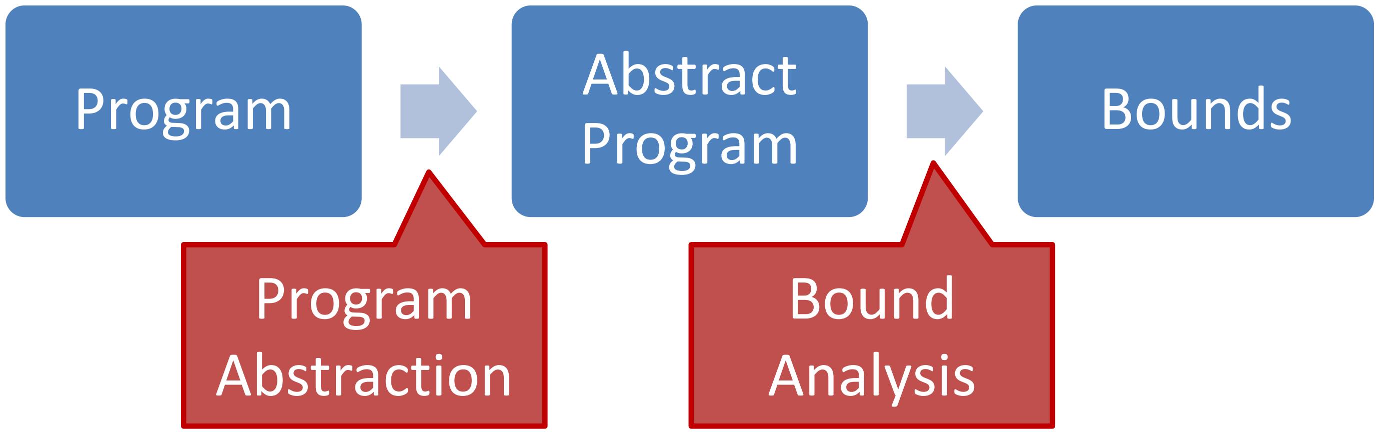
→B… is a hard pr…

```
while(n != 0)
    if (n % 2 = 0)
        n = n / 2;
    else
        n = 3n+1;
```

Typical programs?

```
while(n > 0) {
    m := n--;
    while(m > 0 && ?)
        m--;
    }
```

Methodological Approach



<u>Desired properties of our abstract program model:</u>

- simple
- good computational properties
- motivates further theoretical analysis

<u>Design goals of our analysis:</u>

- no refinement loop
- fail fast
- captures most common loop patterns

Minimal Requirements for Abstract Program Model?

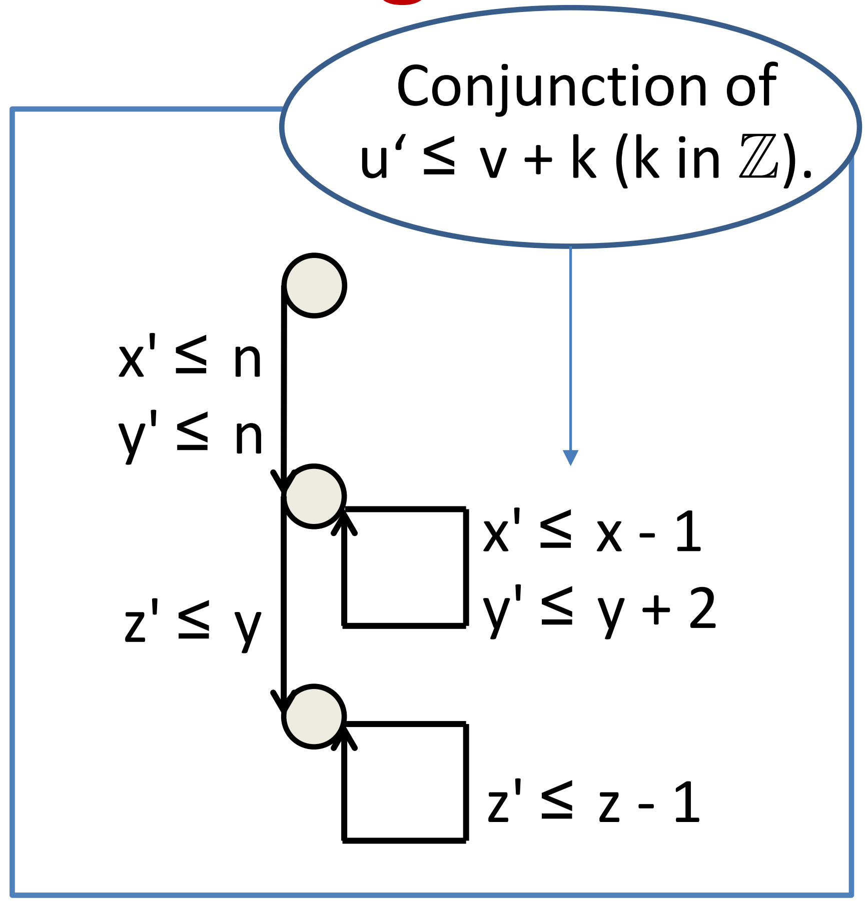
```
foo(uint n)
x = n;
y = n;
while(x > 0)
{     x--;
      y = y + 2;
}
z = y;
while(z > 0)
      z--;
```

We need to model:

- counter variables
  - increments/ decrements
  - resets
- finite control

Difference Constraint Programs

```
foo(uint n)
x = n;
y = n;
while(x > 0)
{     x--;
      y = y + 2;
}
z = y;
while(z > 0)
      z--;
```



Variables take values over $\mathbb{N}$.

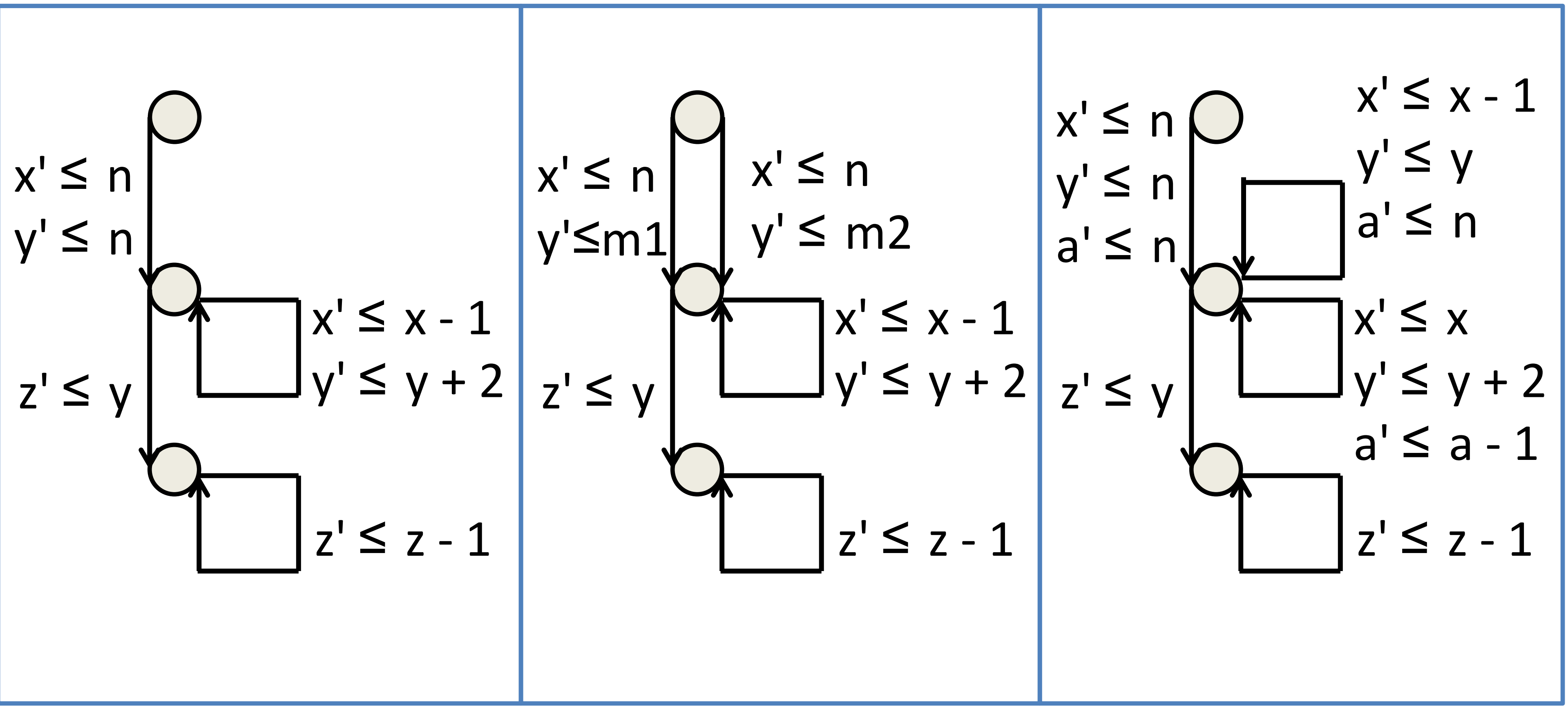
Difference Constraint Programs (DCPs)

- Introduced by Ben-Amram (2008)
- Termination is undecidable in general, but decidable for **deterministic** DCPs

  (**deterministic** = at most one constraint
  $u' \leq v + k$ for every variable u,
  this is a natural subclass: every variable assignement is abstracted to one constraint)

- **<u>we show:</u>** DCPs can model interesting bound analysis problems

Invariant Analysis Problems



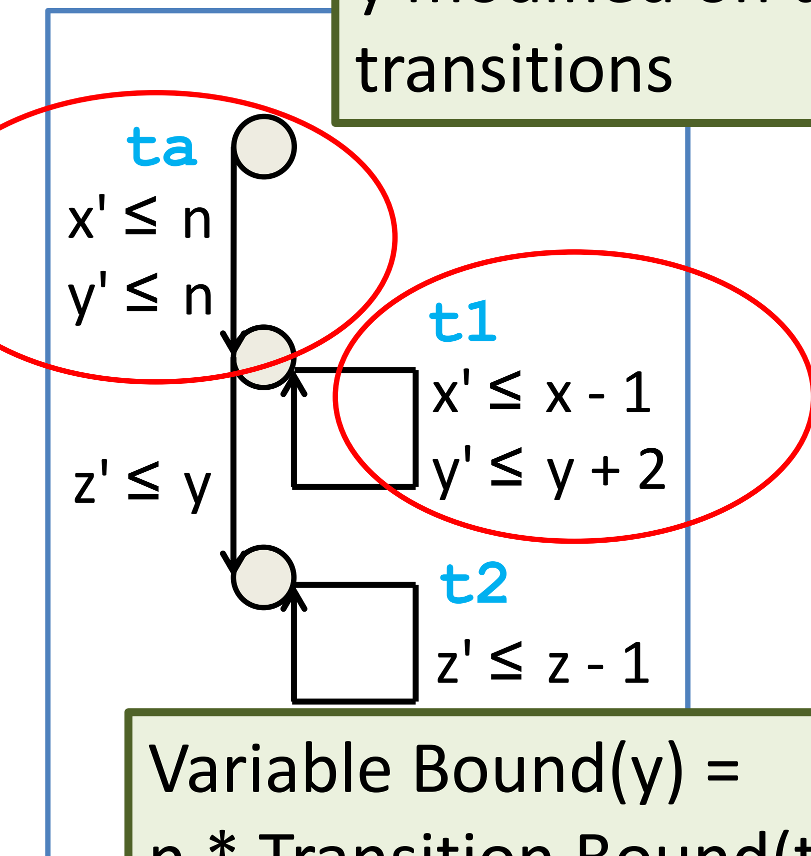
Variable Bound(y): $3n$

Variable Bound(y): $n + 2\max\{m1,m2\}$

Variable Bound(y): $3n + 2n^2$

Bound Analysis Algorithm: Intuition

y modified on two transitions



Variable Bound(y) =
n * Transition Bound(ta) +
2 * Transition Bound(t1)

Local Bound(t1): x

Transition Bound(t1): n

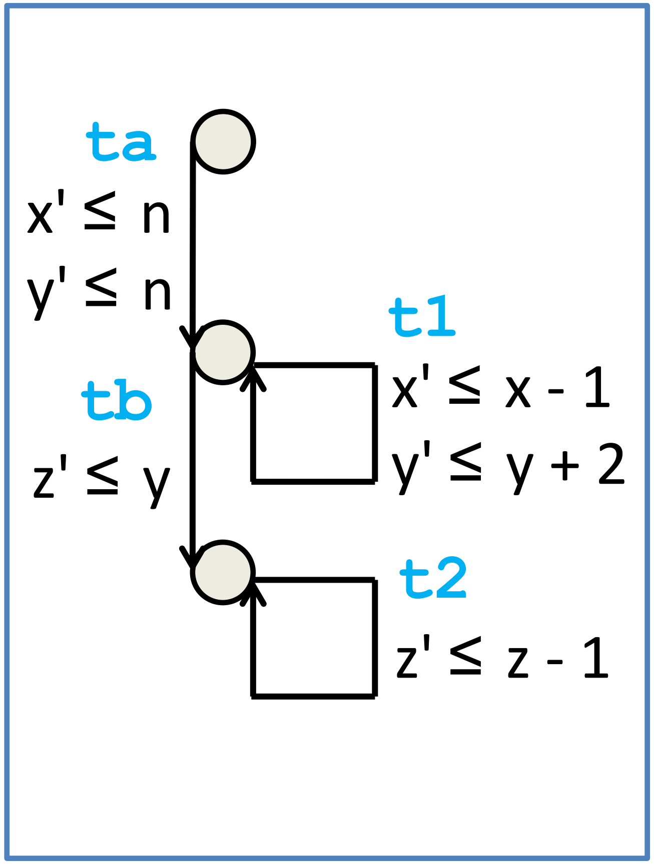
Local Bound(t2): z

Variable Bound(y): 3n

Transition Bound(t2): 3n

Mutual recursion between Variable Bound and Transition Bound

Bound Analysis Algorithm



We assume a **local bound** for every transition, which **decreases** when the transition is executed:

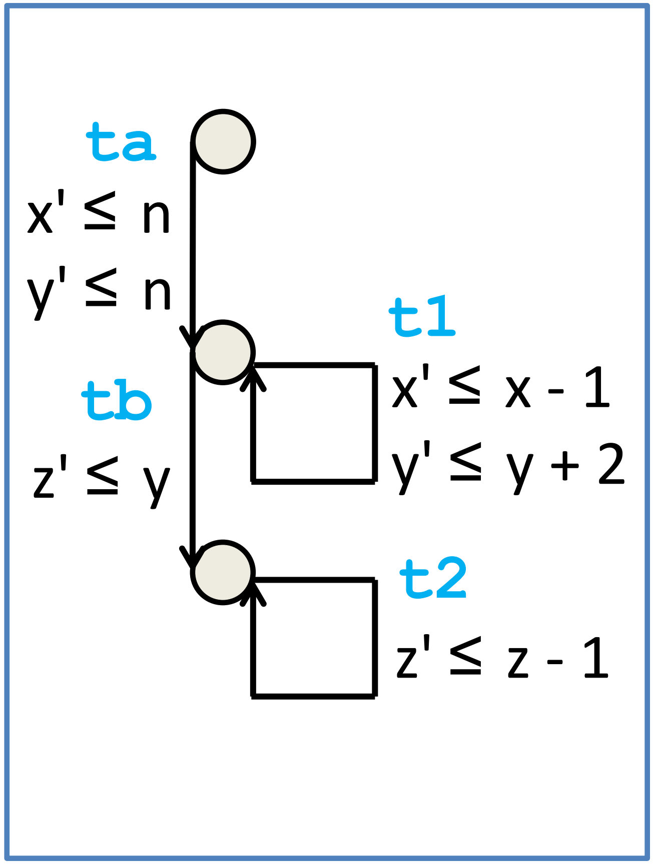
LB(t1) = x
LB(t2) = z
LB(ta) = 1
LB(tb) = 1

We define **increments** and **resets**:

inc(t1,y) = 2

reset(ta,z) = y
reset(tb,x) = n
reset(tb,y) = n

Bound Analysis Algorithm



TB(t2) =

= VB(reset(tb,LB(t2)) * TB(tb) =

= VB(reset(tb,z)) * 1 =

= VB(y) =

= VB(reset(ta,y)) * TB(ta) +

inc(t1,y) * TB(t1) =

= n * 1 +

2 * VB(reset(ta,LB(t1)) * TB(ta) =

= n + 2 * VB(x) * 1 = 3n

Bound Analysis Algorithm

Let P be a set of transitions,
where each transition t of P has local bound LB(t).

For all $t \in P$ we define

$TB(t) = \Sigma_{s \in P}$ inc(s,LB(t))*TB(s) + $\Sigma_{s \in P}$ VB(reset(s,LB(t)))*TB(s)

$VB(t) = \Sigma_{s \in P}$ inc(s,LB(t))*TB(s) + $\max_{s \in P}$ VB(reset(s,LB(t)))*TB(s)

Mutual recursion terminates, if there are no cycles.
(is the case for reasonable programs).

Invariant Analysis Problems

Alternative Method for Invariant Computation:

- demand-driven
- compositional
- no fixed point computation needed

$y' \leq y + 2$

Complementary technique for invariant computation

Related Work: has also been observed in earlier work on bound analysis

- SPEED
- KoAT
- Loopus 2014

Variable Bound(y):	Variable Bound(y):	Variable Bound(y):
$3n$	$n + 2\max\{m1,m2\}$	$3n + 2n^2$

Amoritzed Complexity Analysis

```
foo(uint n)
x = n;
r = 0;
while(x > 0)
{t1:  x--;
      r++;
      if(?) {
            p = r;
            while(p > 0)
            t2:   p--;
            r = 0;
      }
}
```

Local Bound(t1): x

Transition Bound(t1): n

Local Bound(t2): p

Variable Bound(p): n

Transition Bound(t2): n^2?

Amoritzed Complexity Analysis

```
foo(uint n)
x = n;
r = 0;
while(x > 0)
{t1:   x--;
       r++;
       if(?) {
             p = r;
             while(p > 0)
             t2:   p--;
             r = 0;
       }
```

r ist reset after the inner loop → every increment **r++** leads to one loop iteration

Local Bound(t1): x

Transition Bound(t1): n

Local Bound(t2): p

Variable Bound(p): n

Transition Bound(t2): n

We call **r = 0** a **context** for `p = r.`

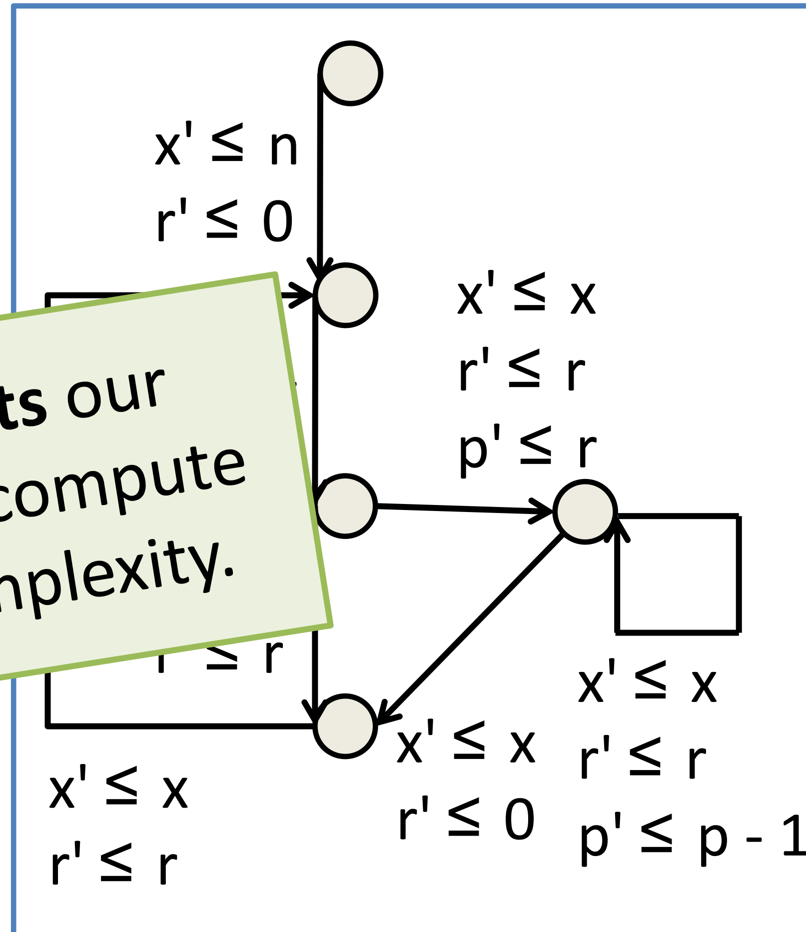
Complexity: 2n

Amoritzed Complexity Analysis

```
foo(uint n)
x = n;
r = 0;
while(x > 0)
{       x--;
        r++;
        if(?)
                p
                wh

                r       0;
        }
}
```

Using **contexts** our algorithm can compute the linear complexity.



Complexity: 2n

Amortized Complexity in Real Code

Amortization due to

Dependencies between increments and resets

15 Examples found during our experiments

Examples: forsyte.at/software/loopus

Implementation

- **Tool:** „Loopus“ based on LLVM and Z3
- **Evaluation:** CBench („Collective Benchmark“)
  - C programs
  - 1027 files with > 200 kLoc, > 4000 loops
  - 1751 functions
- **Comparison to the tools:**
  - KoAT (TACAS 2014)
  - CoFloCo (APLAS 2014)
  - Loopus 2014 (CAV 2014)
- **First experimental comparison on real world code**

Experimental Results: Function Complexity

	Succ	1	n	n^2	n^3	$n^{>3}$	2^n	Time	TimeOut
Loopus 15	806	205	489	97	13	2	0	15m	6
Loopus 14	431	200	188	43	0	0	0	40m	20
KoAT	430	253	138	35	2	0	2	5.6h	161
CoFloCo	386	200	148	38	0	0	0	4.7h	217

Loopus 15: - More Complexity Results
- In Shorter Time

More details: forsyte.at/software/loopus

Summary

Contributions to bound/complexity analysis:

- notions of increment/decrement and reset
- first algorithm based on DCPs
- we demonstrate the scalability and applicability of our algorithm

- DCPs are an interesting model for further research