SIZE-CHANGE TERMINATION

A partial survey

YE OLDE ART OF TERMINATION PROOFS

- Examples will use a simple functional language (hence, we ask whether recursion terminates).
- All values will be natural numbers.

```
add(x,y) =
  if x=0 then y
           else 1+add(x-1, y)
```

Argument: 1st parameter decreases in every call.

A slightly harder one

```
add(x,y) =
  if x=0 then y
          else 1+add(y, x-1)
```

Argument: 1st parameter decreases after two calls.

GCD program

```
gcd(x,y) =
   if x≤1 or x=y then x
      else if x<y then gcd(x, y-x)
                        else gcd(y, x-y)
```

Argument: larger of param's decreases in every call.

Ackermann's function

```
ack(x,y) =
   if x=0 then y+1
           else if y=0
                then ack(x-1, y)
                 else ack(x-1, ack(x, y-1))
```

Argument:

In every call, either x decreases or x stays put and y decreases.

$\Rightarrow$ the pair $\langle x,y \rangle$ decreases lexicographically.

Summary

All these examples (and many others) are based on impossibility of infinite descent

In every (hypothetical) chain of calls, something is shown to decrease indefinitely, which cannot really happen (because it's taken from a well-founded order).

Ingenuity is required either to define that "something" or to show the infinite descent

Note the two options:

- A (complex) combination of the data decreases certainly in every step.
    - (sum, pair of values…)

ranking function

- Combinations are not considered, but the proof of descent may be more clever
    - (consider two consecutive calls…)

analysis of paths

The SCT approach

[Lee, Jones & B, POPL 2001]

subject program

↓

initial analysis

↓

a set of graphs

↓

SCT tester

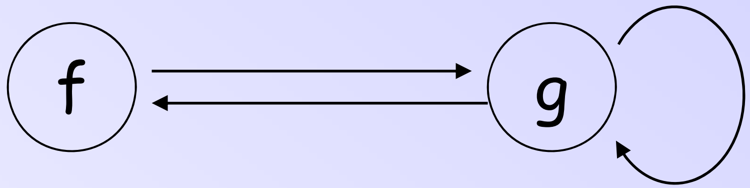
↓

"terminating!"

SCT is a purely combinatorial problem.

Products of initial analysis

Control-Flow Graph: possible transitions among "locations" in a program.



Functional programming context:
functions, calls

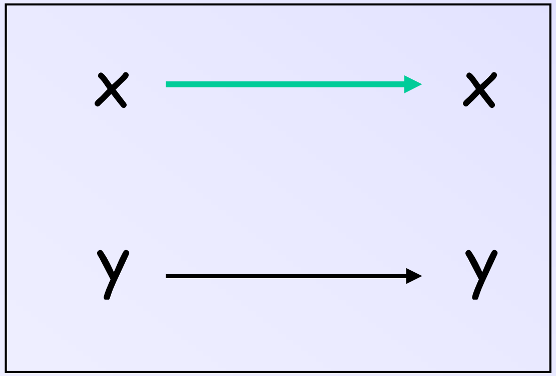
Imperative context:
flow-points, statements / basic blocks

Size-Change Graph

What's happening in a transition?

Consider call: add(x,y) = …add(x-1,y)…

Information: 1st param decreases. 2nd unchanged.



old → new

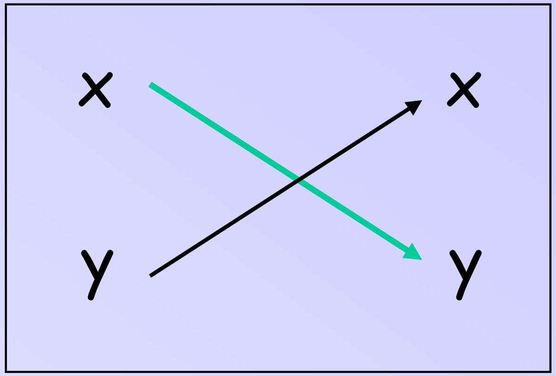
means: old > new

old → new

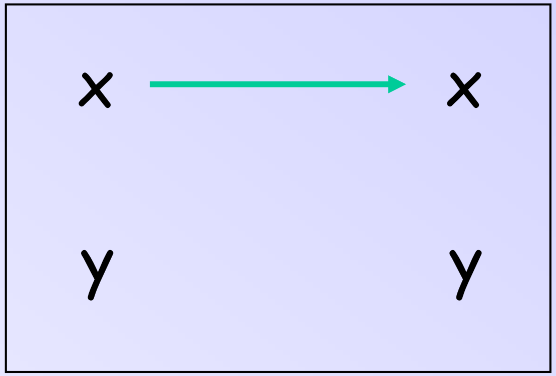
means: old ≥ new

size-change graphs

gcd(x,y) = …gcd(y,x-y)…

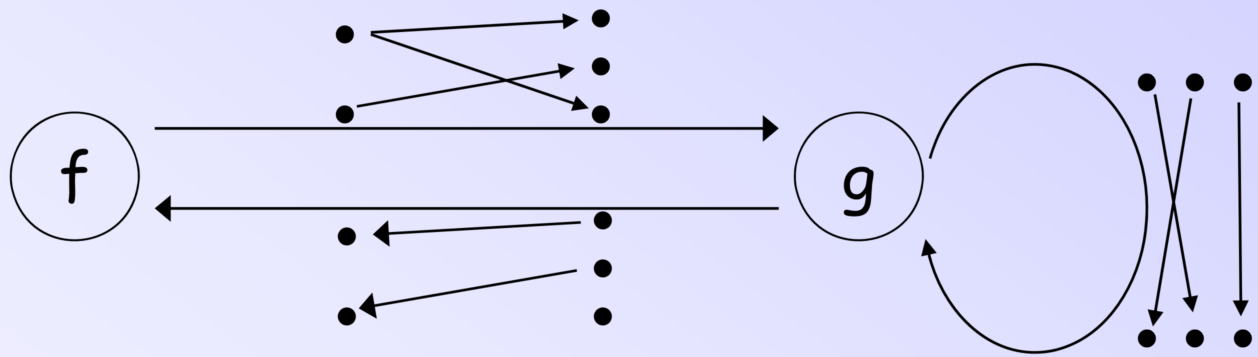


ack(x,y) =… ack(x-1, ack(…))



Analyzing SCT

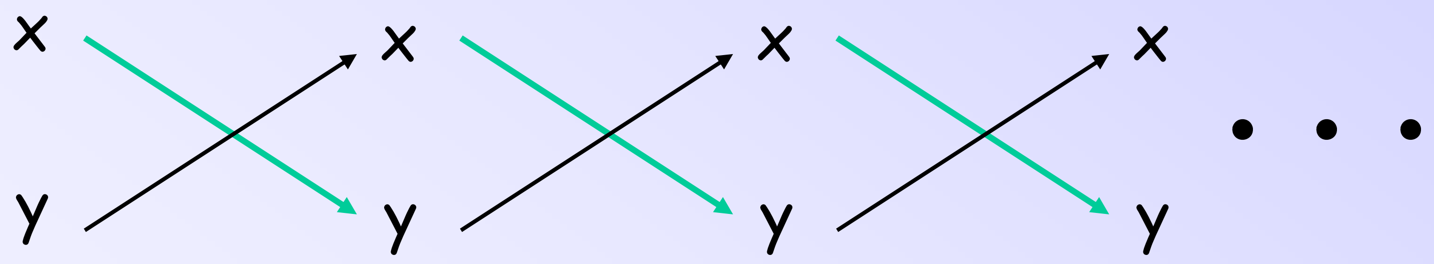
Size-Change Graphs "sit" on arcs of the CFG



Multipaths

A multipath results of concatenating SCG's along a CFG path.

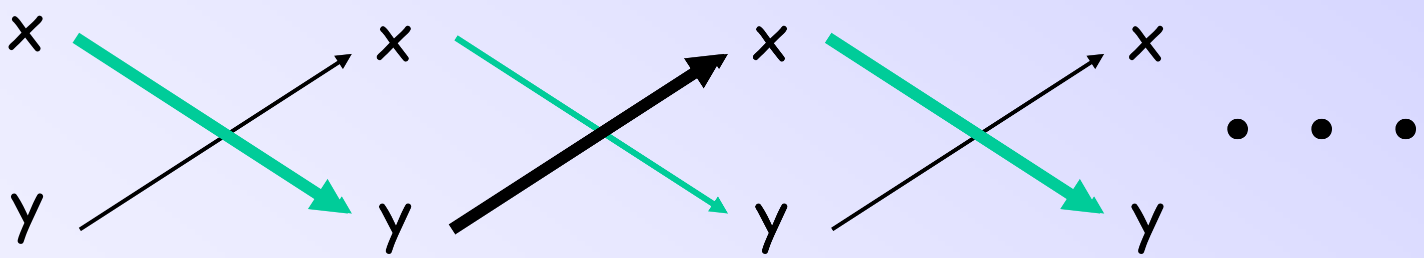
Example: a loop of add (2nd ver.) looks like that:



Threads

A thread is a (infinite) path in the multipath.

A thread is infinitely descending if it has infinitely many down-arcs.



SCT condition

A CFG/SCG-set satisfies SCT if every infinite multipath contains an infinitely descending thread.

This criterion is a sufficient condition for program termination.

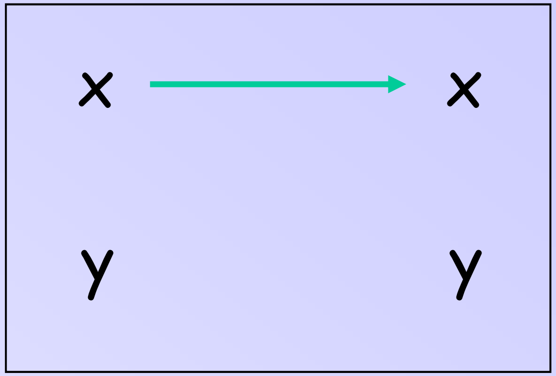
Assumptions:

Correct (safe) program representation

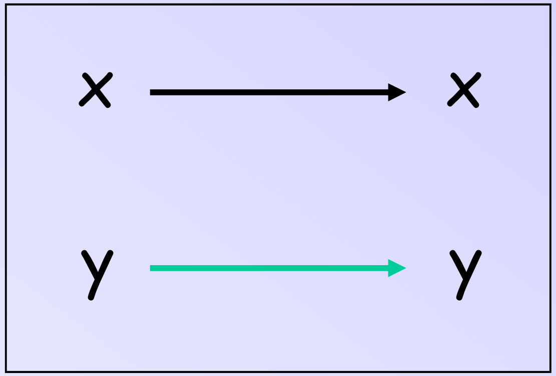
Well-founded data (no infinite descent)

An Example: ack

ack(x,y) =… ack(x-1, ack(…))



ack(x,y) =… ack(x, y-1)



Is SCT a decidable problem?

Proof #1: reduction to a question on Büchi automata.

Proof #2: the Closure Algorithm.

What is the complexity class of SCT ?

THM: the SCT problem is PSPACE-complete.

Upper bound: a variant of the Closure Algorithm

Hardness: reduction from a PSPACE-complete classic.

Some (Pre)History

LJB, POPL 2001

Sagiv, Logic Prog. Symp. 1991,

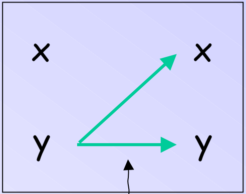
Lindenstrauss & Sagiv, ICLP 1997,

Codish & Taboch, JLP 1999

Dershowitz et al., AA 2001

The Closure Algorithm

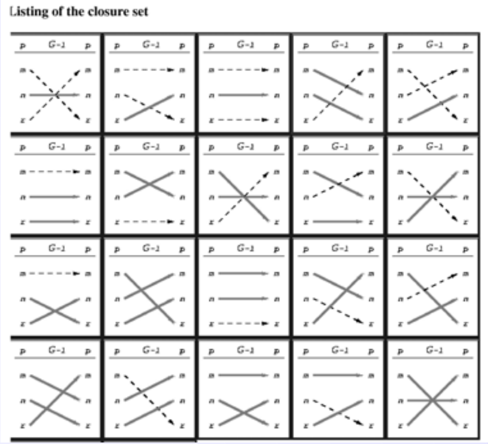
THM: SCT holds iff in the composition closure, every idempotent graph has an in-situ down-arc.



in-situ down-arc

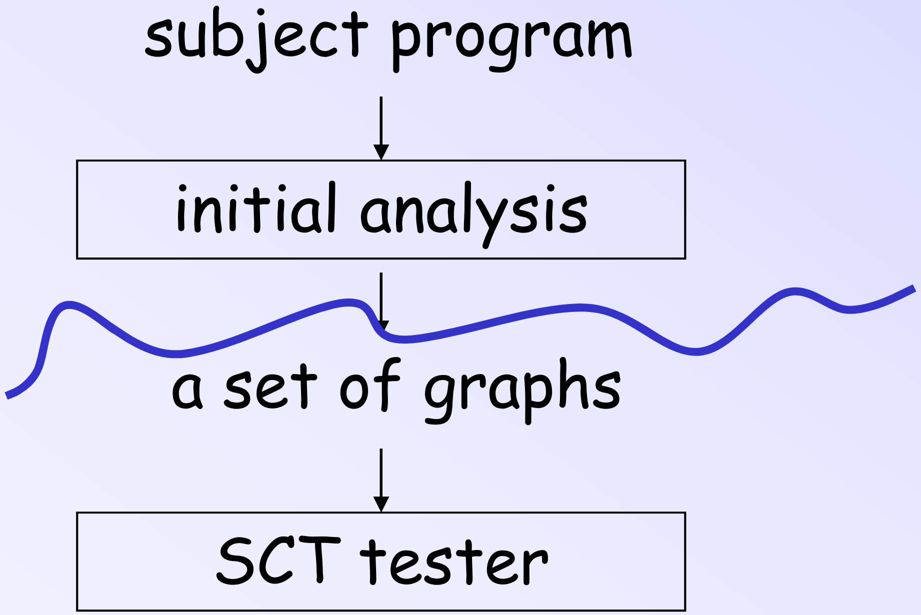
An Example

p(m, n, r) = if r>0 then p(m, r-1, n) else
if n>0 then p(r, n-1, m) else m



The Contibution of [LJB2001]

creation of an abstraction boundary



SCT is a purely combinatorial problem.

- solid theory
- it's not about Prolog

The next decade

- Contributions by

  Avery, Bohr, Codish, Dershowitz, Fogarty, Heizmann, Giesl, Jones, Krauss, Lagoon, Lee, Lindenstrauss, Manolios, Moyen, Podelski, Rybalchenko, Sagiv, Schneider-Kamp, Serebrenik, Sereni, Stuckey, Thiemann, Vardi, Vroon …

The next decade

- Systems applying SCT

- Better understanding the theory, in particular in a larger context of termination analysis

Semantics for Termination Analysis

(e.g., Codish-Taboch 99 for Prolog)

STEP 1:

A semantics $[| \square |]^{bin}$ that maps a program into its (infinite) set of "transitions"

Program P is terminating iff there is no infinite chain in $[|P|]^{bin}$

STEP 2: check it

Abstraction

Abstract Semantics for Termination Analysis

STEP 1:

An abstraction that maps a program into a (finite) set $P^{\#}$ of "abstract transitions" (an abstract program)

Abstract programs have a semantics that super-approximates the semantics of the source program.

If $P^{\#}$ is terminating then P is.

STEP 2: forget about P and study $P^{\#}$ instead.

Abstract Programs for Termination Analysis

1. Define the abstract state space $\mathcal{S}$.

   A typical state: $(f, x_1, \dots, x_n)$

   flow-point variables

   (add, 5, 4)

2. Choose a language for describing transitions in $\mathcal{S} \times \mathcal{S}$.

```
append(x,y) =
  case x of
     [] => y
     h::t =>  h::append(t, y)
```

concrete state: (append, [l,i,s,t], [a,n,o,t,h,e,r])

abstract state: (append, 4, 7)

A language to describe transitions

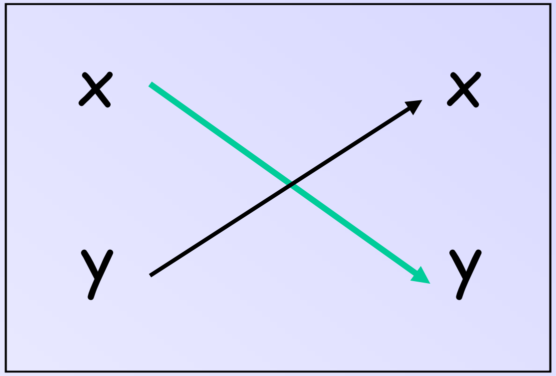
- Fix a logical theory
- Fix a class of formulas for this theory that define relations over

$$x_1, \dots, x_n, x'_1, \dots, x'_n$$

(state and new state)

Size-Change Graphs

$$gcd(x,y) \to gcd(y,x\text{-}y)$$



$$x > y' \wedge y \geq x'$$

The Size-Change Graph abstraction is based on the theory of well-ordered sets

and its transitions are conjunctions of atomic predicates from:

$$x > y'$$

$$x \geq y'$$

where x,y are any state variables.

The Secret of Success

SCT is an abstraction which is useful, but simple enough to get results.

Result #1:

A "size-change program" terminates iff it satisfies the SCT condition.

So termination is decidable.

Highlights of SCT theory

- Analysis of complexity (PSPACE complete; time complexity $2^{O(n\log n)}$).
- 3 algorithms to decide termination (and then some more)
- Each algorithm has a story

Algorithm 1 (POPL 2001):

Reduction to a problem about Büchi automata.

Fogarty, Vardi (TACAS '09,'10) went from there to study the efficiency of algorithms on such automata.

Algorithm 2 (POPL 2001):

the Closure Algorithm.

Podelski, Rybalchenko (LICS '04) formulated a general notion of "disjunctive transition invariants" that justifies a whole class of similar algorithms.

Algorithm 3 (CAV '09 - LMCS '10):

Generating a global ranking function

= A combination of the variables that decreases in **every** transition

So, with SCT, a program terminates $\Rightarrow$ a ranking function can be generated.

More expressive abstractions

The Size-Change Graph abstraction:

the theory of well-ordered sets

atomic predicates from:

$$x > y'$$

$$x \geq y'$$

A richer language allows for handling more programs

More expressive abstractions

The Monotonicity Constraint abstraction:

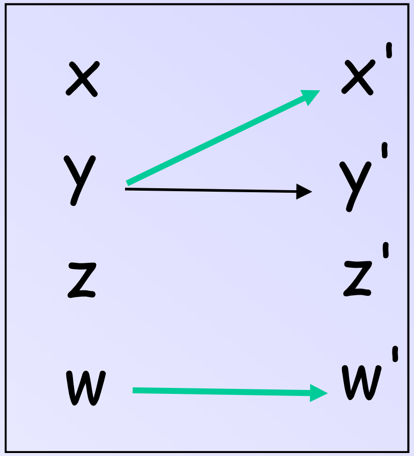
the theory of well-ordered sets

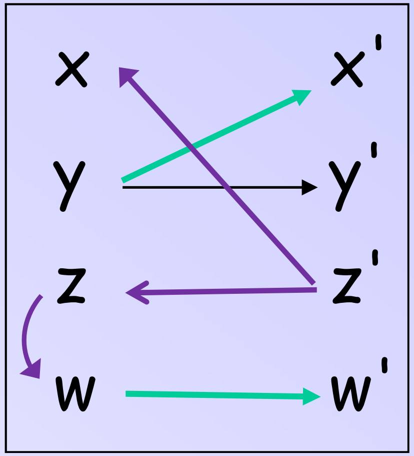
atomic predicates from:

$$x > y, \quad x \geq y, \quad x = y$$

where x,y range over all state variables.

Monotonicity Constraints





$y > x'$	$x \le z'$
$y \ge y'$	$z < z'$
$w > w'$	$z > w$

Monotonicity Constraint theory

- Broadly speaking – all the results from SCT theory have been successfully extended.
- In particular, termination is decidable, and ranking functions can be automatically found.

Codish et al. '05, B. '09/'10

- Order constraints over the integers

(instead of a well-ordered set)

```
mid(x,y) =
  if x>=y then y
          else mid(x+1, y-1)
```

$x < x' \wedge y > y'$

$x < x' \wedge y > y' \wedge x \geq y$

- We still have decidability etc.

and a little more (e.g., execution time bounds)