SIZE-CHANGE TERMINATION

A partial survey

mostly based on work by

Neil Jones, DIKU

DA∃D∀LUS

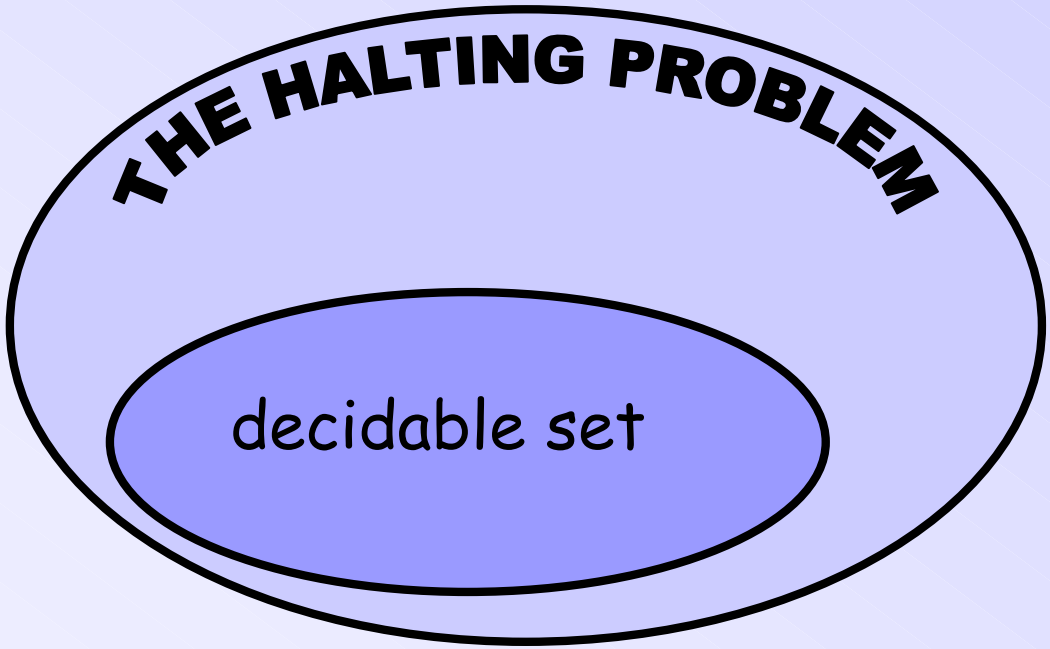
Chin Soon Lee, MPI

Amir Ben-Amram, MTA

BACKGROUND:
THE TERMINATION PROBLEM

- A cornerstone of program verification
- Also important for program generators, e.g., specializers
- … and certain interpreters

- As the problem is undecidable in general, we make do with conservative (safe) approximations.



One reason for hope:

many programmers are able to argue that their programs do terminate.

YE OLDE ART

OF TERMINATION PROOFS

- Examples will use a simple functional language (hence, it's infinite recursion we guard against).
- 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 (x,y) decreases lexicographically.

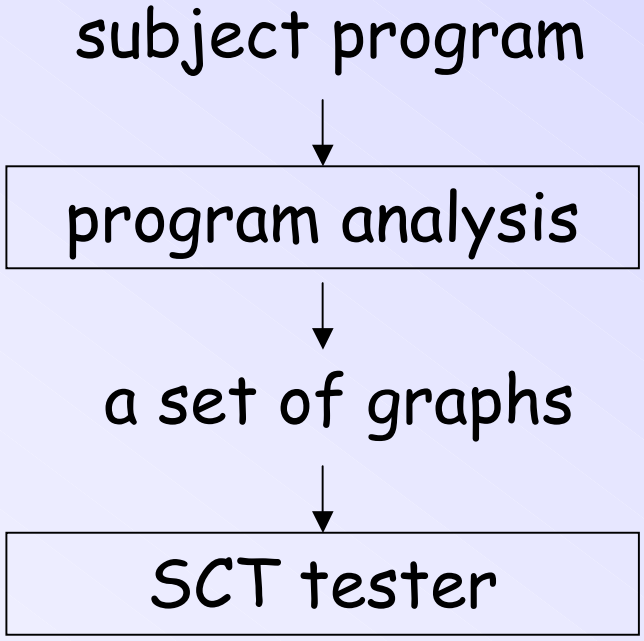
Summary

All these examples (and many others) are based on impossibility of infinite descent (Floyd 1967)

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" (sum, pair of param's...) or to show the infinite descent (consider two consecutive calls...)

SCT handles all these examples automatically.

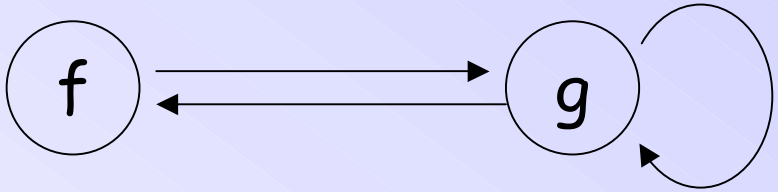


SCT is a purely combinatorial problem.

Supporting program analyses are outside the scope of this work.

Products of program analysis

Control-Flow Graph: possible calls in a program.

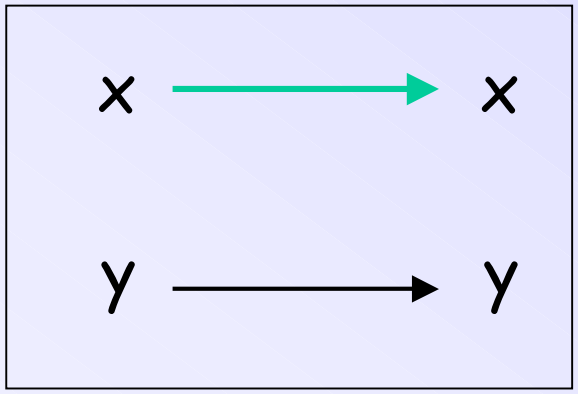


SCT is based on conservative analysis – every path is considered.

Size-Change Graph: what's happenning in a call?

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

Information: 1st param decreases. 2nd unchanged.



pre → post

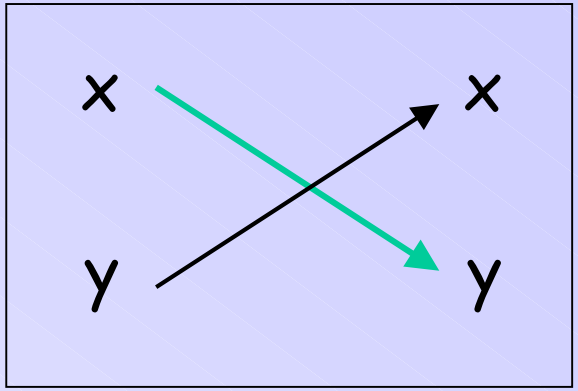
means: pre > post

pre → post

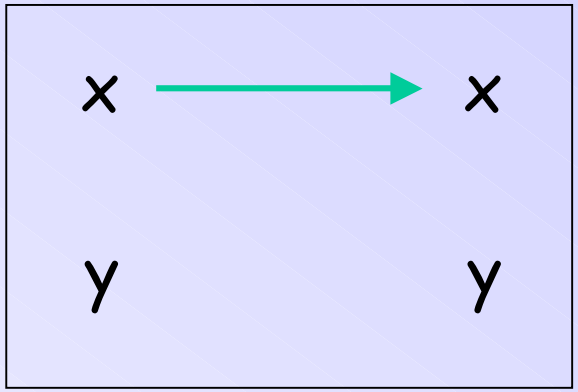
means: pre ≥ post

size-change graphs

gcd(x,y) = ...gcd(y,x-y)...

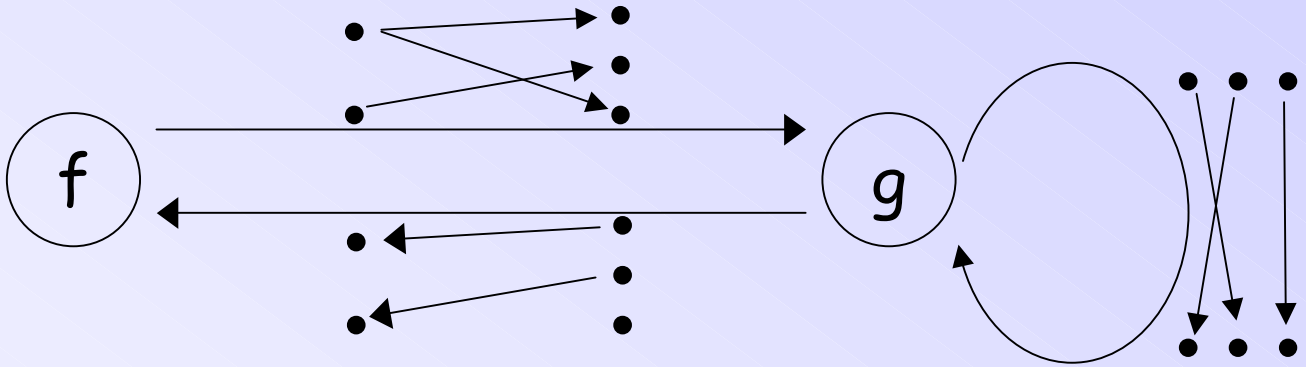


ack(x,y) =... ack(x-1, ack(...))



Analyzing SCT

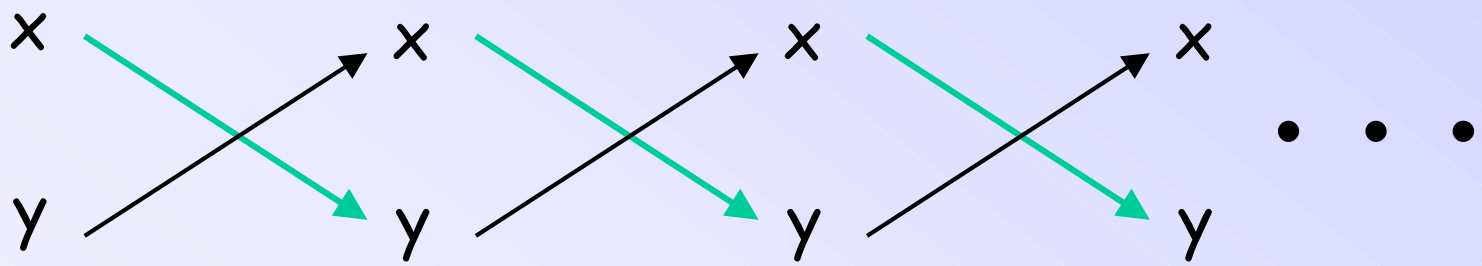
Size-Change Graphs "sit" on arcs of the CFG



Multipaths

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

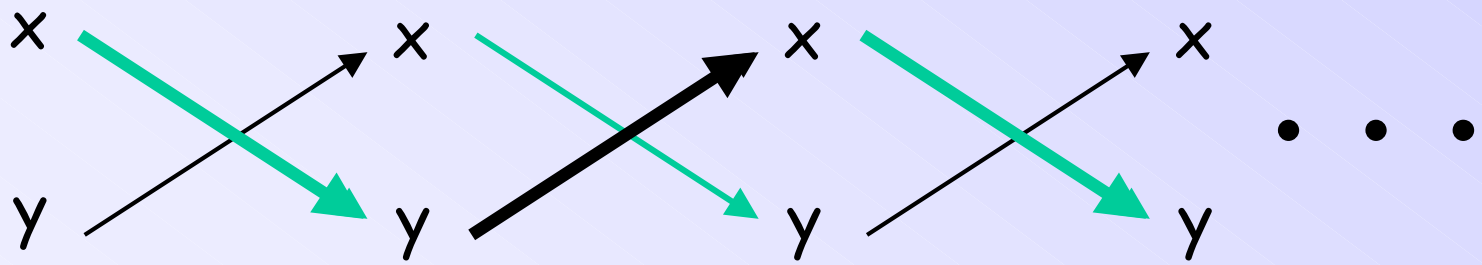
Example: a loop of add (2nd ver.) looks like that:



Threads

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

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



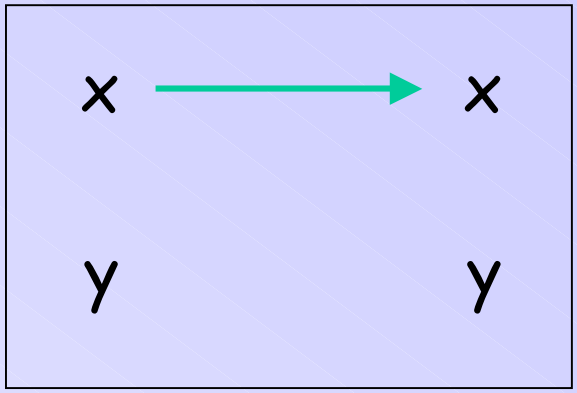
Size-Change Termination

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

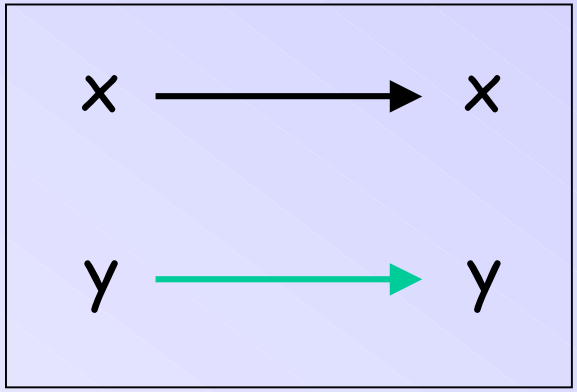
SCT is a sufficient condition for program termination.

An Example: ack

ack(x,y) =... ack(x-1, ack(...))



ack(x,y) =... ack(x, y-1)



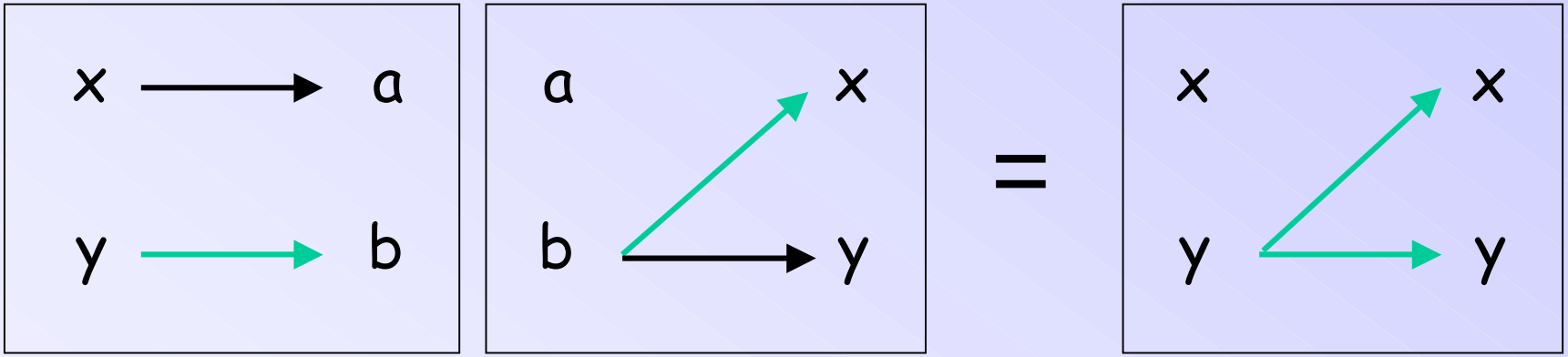
Is SCT decidable?

Proof #1: redfine it as a problem about Büchi automata.

Proof #2: the Closure Algorithm.

The Closure Algorithm

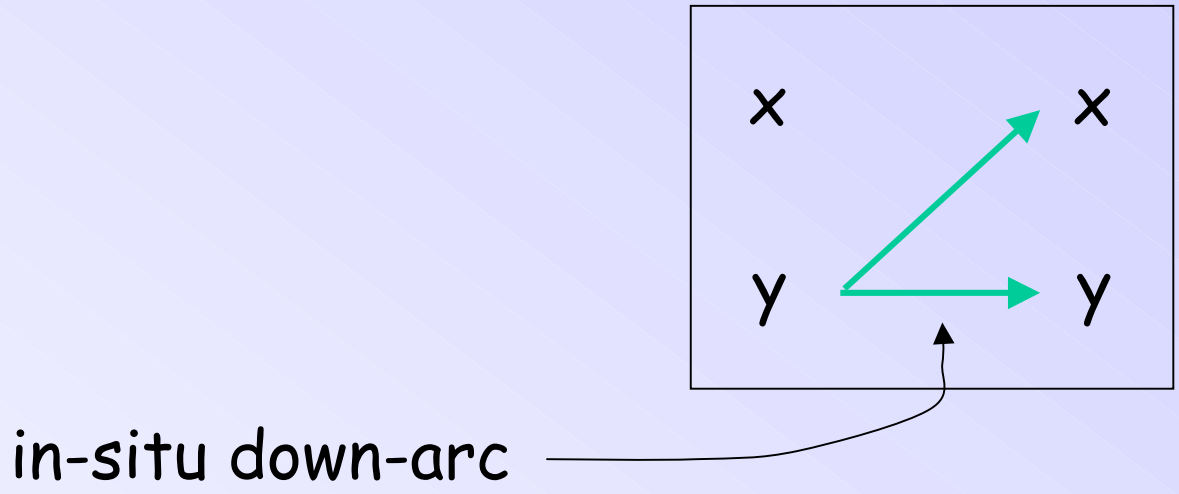
- Define the composition of size-change graphs



- form the composition closure of the given graphs – a finite set!

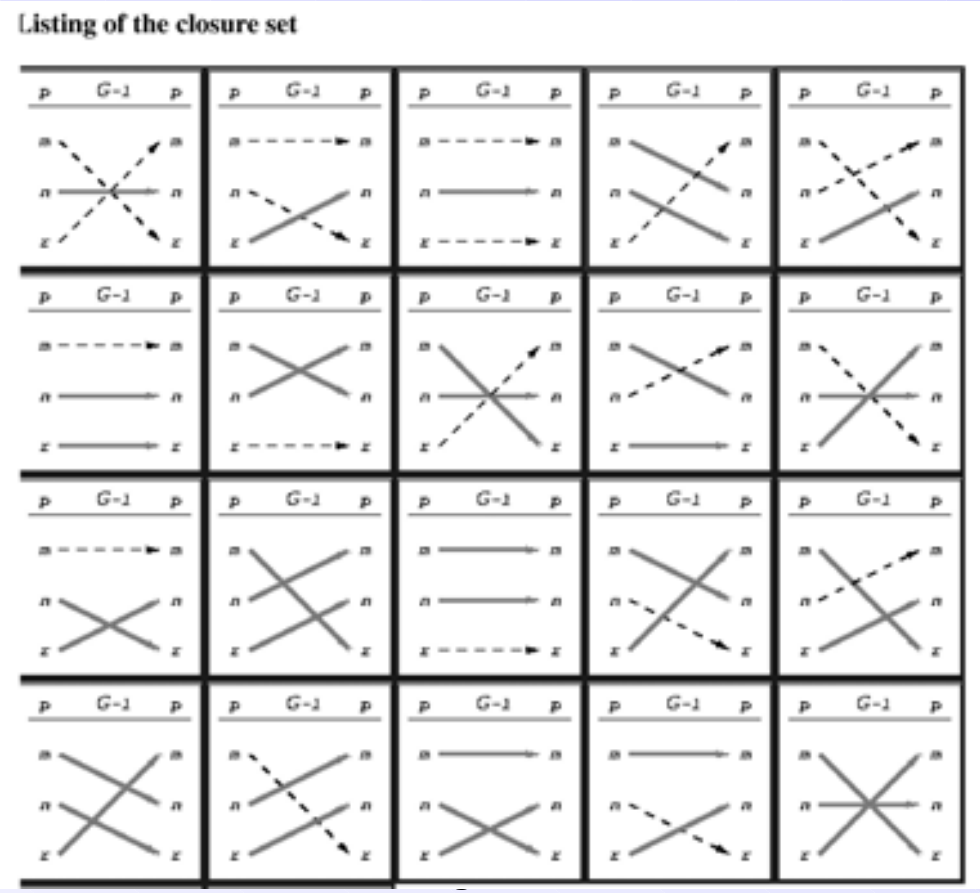
The Closure Algorithm

THM: SCT holds iff in the composition closure, every idempotent graph has an 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



A Brief History

LJB, POPL 2001

Sagiv, Logic Prog. Symp. 1991,

Lindenstrauss & Sagiv, ICLP 1997,

Codish & Taboch, JLP 1999

THE REST OF THE TALK

- The Complexity-Theoretic Quest
- The Recursion-Theoretic Quest
- The Algorithmic Quest

Additional related work

The Complexity-Theoretic Quest [LJB 2001]

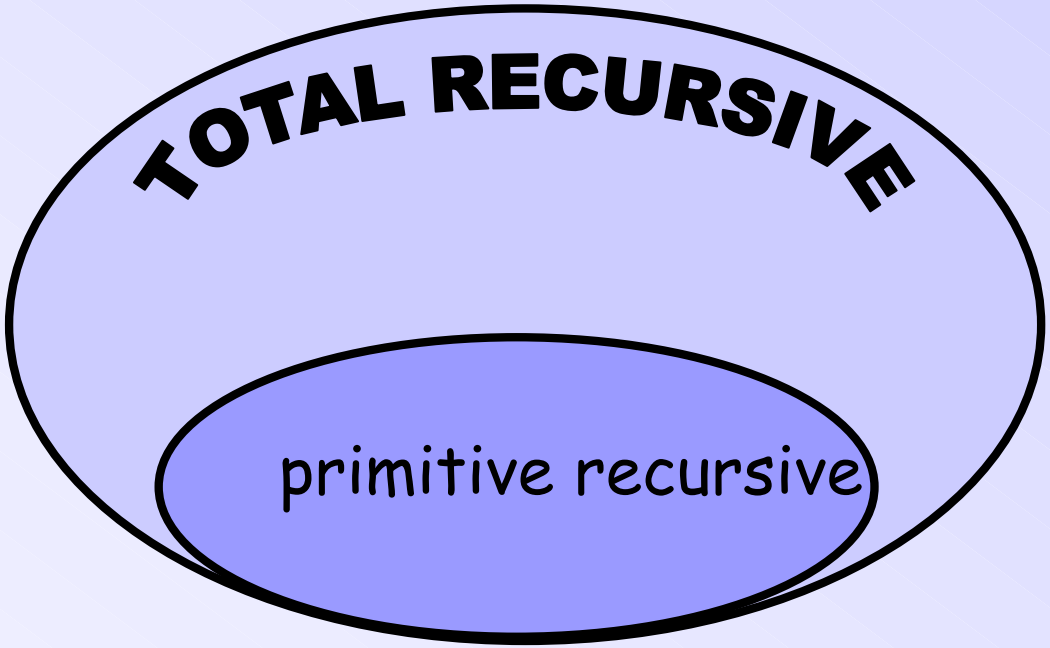
THM: the SCT problem is PSPACE-complete.

Upper bound: provided by the Closure Algorithm (implemented in a small-space manner).

Hardness: a reduction from Termination of Boolean Programs. Such a program has a fixed number of memory bits and therefore its termination problem is PSPACE (and easily proved complete).

The Recursion-Theoretic Quest [B 2003]

Recursion Theory deals with classes of functions, of which the best known are:



The Recursion-Theoretic Quest

- No usefully-defined programming system captures exactly the Total Recursive functions.
- Primitive Recursive functions are captured by a very simple type of recursive programs.

  add(x,y) = ...add(x-1,y)...

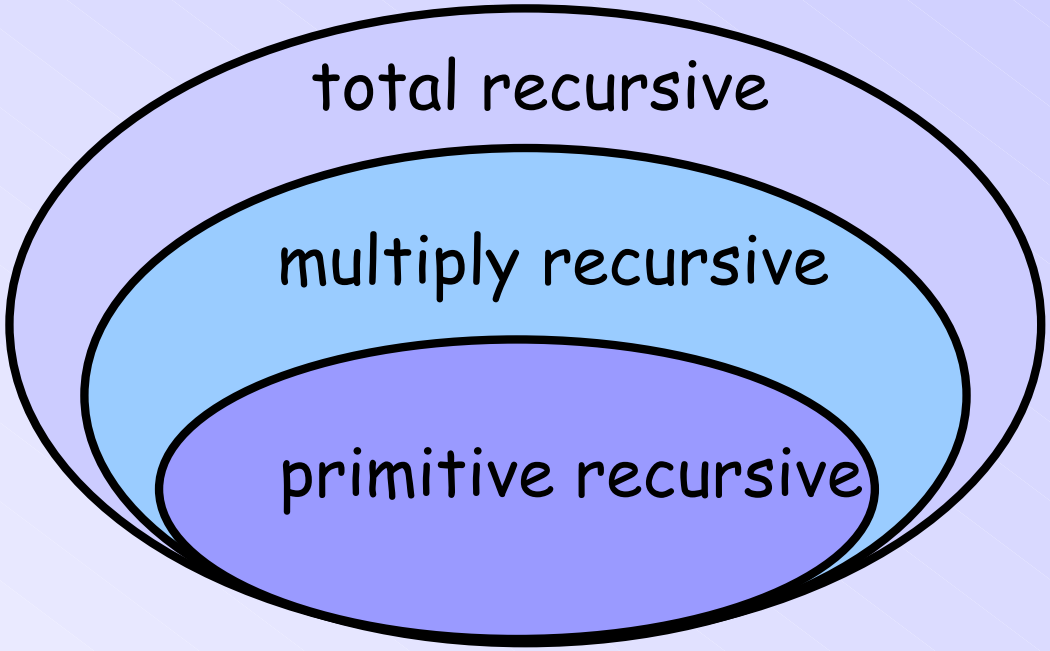
- Ackermann's function is "the" classic example of a total recursive function which is not PR.
- But the ack program is easy to prove terminating.

The Multiply-Recursive Functions

DEF: A multiply-recursive function is one defined using (nested) recursion which obeys lexicographic descent.

```
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))
```

The Multiply-Recursive Functions



Lexicographic descent is easily captured by SCT.

What is the class of SCT-computable functions?

Conjecture (Jones, 2000):

It's the same as the multiply-recursive class.

> Note – SCT programs can have very complicated forms of descent.

Now, it's a theorem.

Proof (B 2003):

Essentially an algorithm to compile every SCT program into one that has lexicographic descent.

The Algorithmic Quest [BL]

THM: the SCT problem is PSPACE-complete.

Conclusion: the exponential worst-case behaviour is unbeatable.

The heuristic approach: an efficient algorithm that works for (many) practical problem instances.

SCP:
Size-Change termination in Polynomial time

(cubic - often quadratic)

(-) We lack an elegant, complete theoretical characterization of the instances it handles.

(+) The proof of the pudding is in the eating.

Experimental Evaluation of SCP

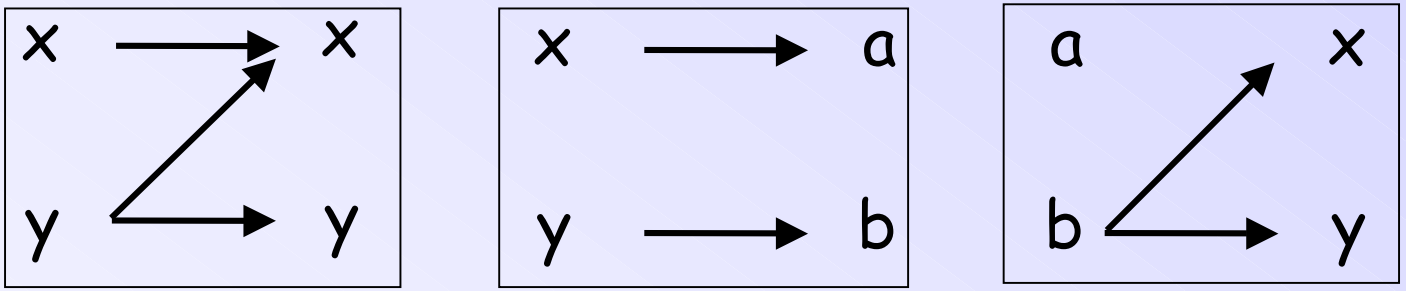
- A benchmark of Prolog queries assembled by several researchers of termination in the Prolog context.
- Size-change information obtained (with very little adjustment) from the Prolog analyzer Terminweb of Codish and Taboch (1999).

QUERIES	SCT correct	SCP correct
123	118	118

A glimpse of the algorithm

DEF: A set of parameter names is a Thread Preserver (TP) is a thread reaching one of these parameters can be continued throughout any possible multipath.

Example: {y,b} is a TP in the following set of graphs



LEMMA: if a set of size-change graphs has a TP consisting entirely of down-arcs, the set satisfies SCT.

This can be tested in linear time.

The algorithm tests some other, more subtle conditions that also guarantee SCT, all based on the computation of Thread Preservers.

THANK YOU