中央研究院資訊科學研究所
INSTITUTE OF INFORMATION SCIENCE
ACADEMIA SINICA

Verifying Recursive Programs using Intra-procedural Analyzers

Yu-Fang Chen, Academia Sinica, Taiwan

joint work with

Chiao Hsieh, Ming-Hsien Tsai, Bow-Yaw Wang and Farn Wang

First of all

Thanks for the invitation and congratulation to Ed!

Difficulties of Program Verification

- Large/unbounded base types: int, float, string
- User-defined types/classes
- Pointers/aliasing + unbounded #'s of heap-allocated cells
- Procedure calls/recursion/calls through pointers/dynamic method lookup/overloading
- Concurrency + unbounded #'s of threads
- Templates/generics/include files
- Interrupts/exceptions/callbacks
- Use of secondary storage: files, databases
- Absent source code for: libraries, system calls, mobile code
- Size

Source: Turing Lecture of Edmund Clarke

Difficulties of Program Verification

- Large/unbounded base types: int, float, string
- User-defined typ
- Pointers/aliasing cells
- Procedure calls/ amic method lookup/overloading
- Concurrency + unbounded #’s of threads
- Templates/generics/include files
- Interrupts/exceptions/callbacks
- Use of secondary storage: files, databases
- Absent source code for: libraries, system calls, mobile code
- Size

Almost impossible to attack all features at the same time.

Source: Turing Lecture of Edmund Clarke

Difficulties of Program Verification

- Large/unbounded base types: int, float, string
- User-def
- Pointers/ ated cells
- Procedu dynamic method lookup/o
- Concurre
- Templat
- Interrupt

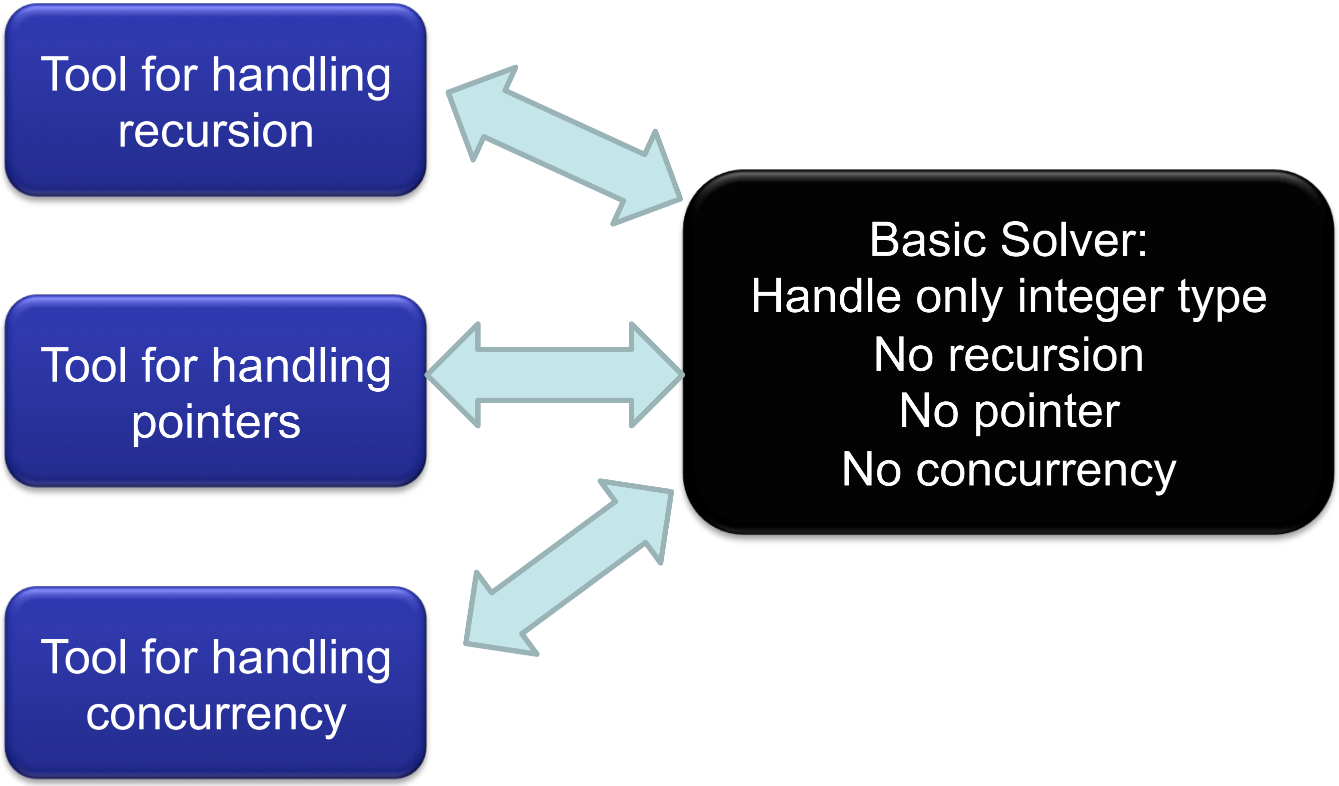
- Use of secondary storage: files, databases
- Absent source code for: libraries, system calls, mobile code
- Size

Source: Turing Lecture of Edmund Clarke

The Proposal

Basic Solver:
Handle only integer type
No recursion
No pointer
No concurrency

The Proposal



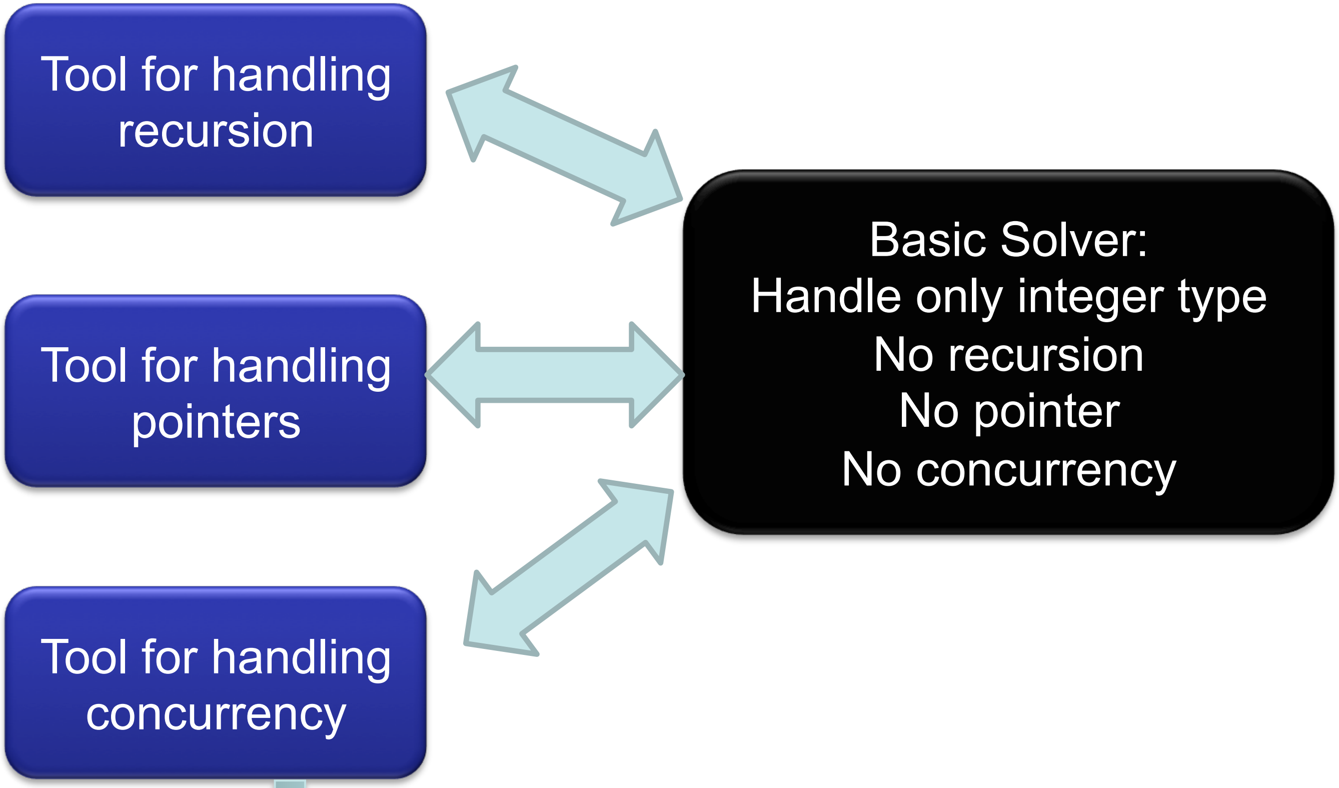
Tool for handling recursion

Tool for handling pointers

Tool for handling concurrency

Basic Solver:
Handle only integer type
No recursion
No pointer
No concurrency

The Proposal



Tool for handling recursion

Tool for handling pointers

Tool for handling concurrency

Basic Solver:
Handle only integer type
No recursion
No pointer
No concurrency

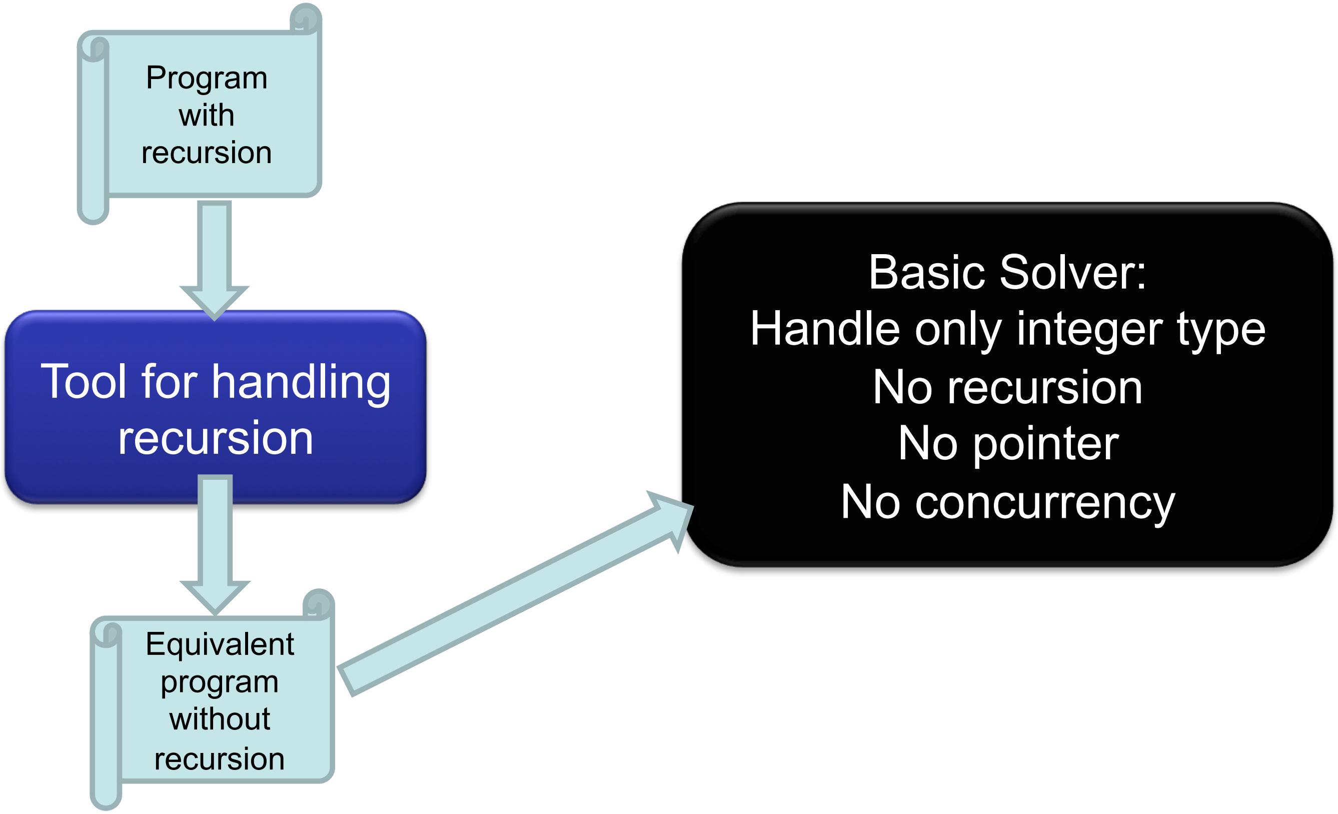
Akash Lal, Thomas W. Reps: Reducing Concurrent Analysis Under a Context Bound to Sequential Analysis. CAV 2008

Handling Recursive Programs

An approach to
verify recursive program using
non-recursive verifiers via
program transformation

Yu-Fang Chen, **Chiao Hsieh**, **Ming-Hsien Tsai**, **Bow-Yaw Wang** and **Farn Wang**,“Verifying Recursive Programs using Intraprocedural Analyzers”, SAS 2014

The Idea



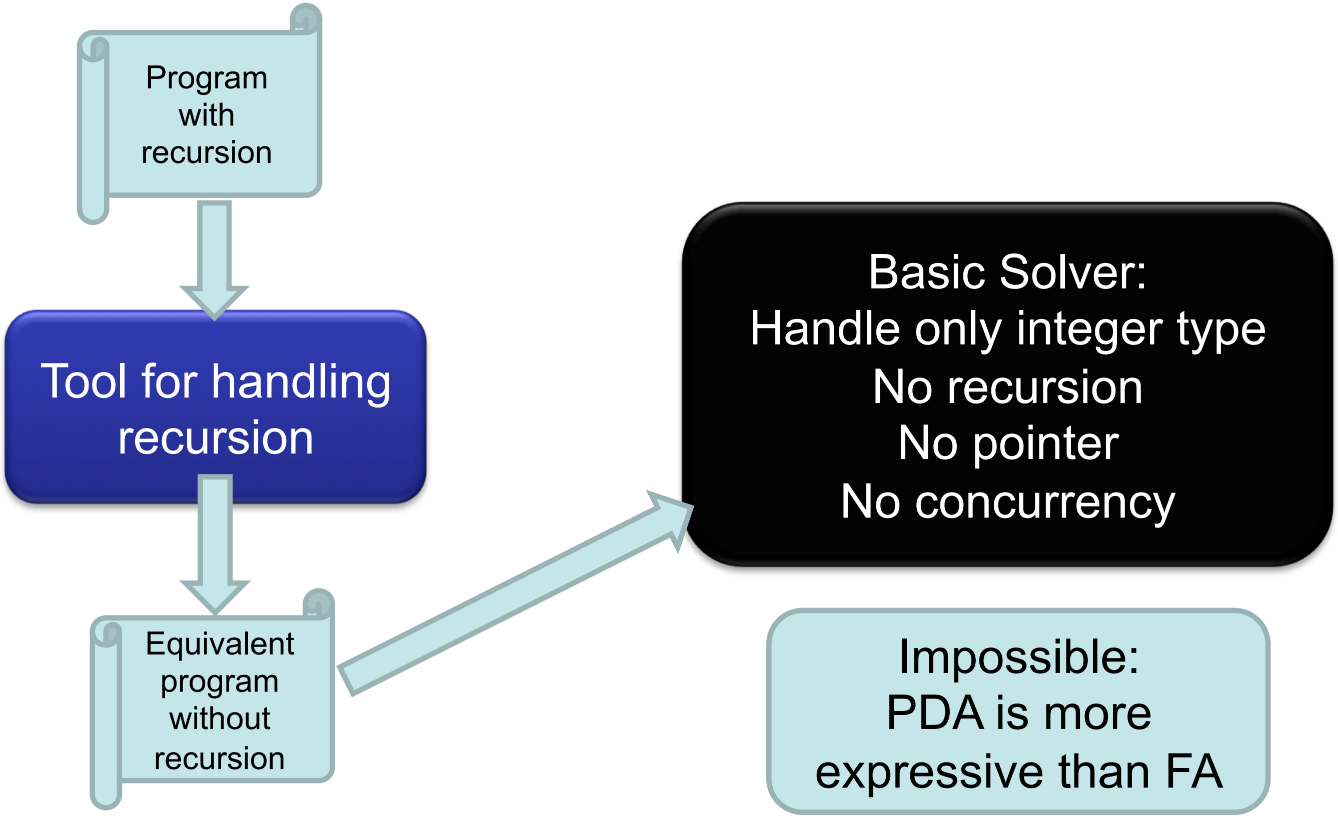
Program with recursion

Tool for handling recursion

Equivalent program without recursion

Basic Solver:
Handle only integer type
No recursion
No pointer
No concurrency

The Idea



Program with recursion

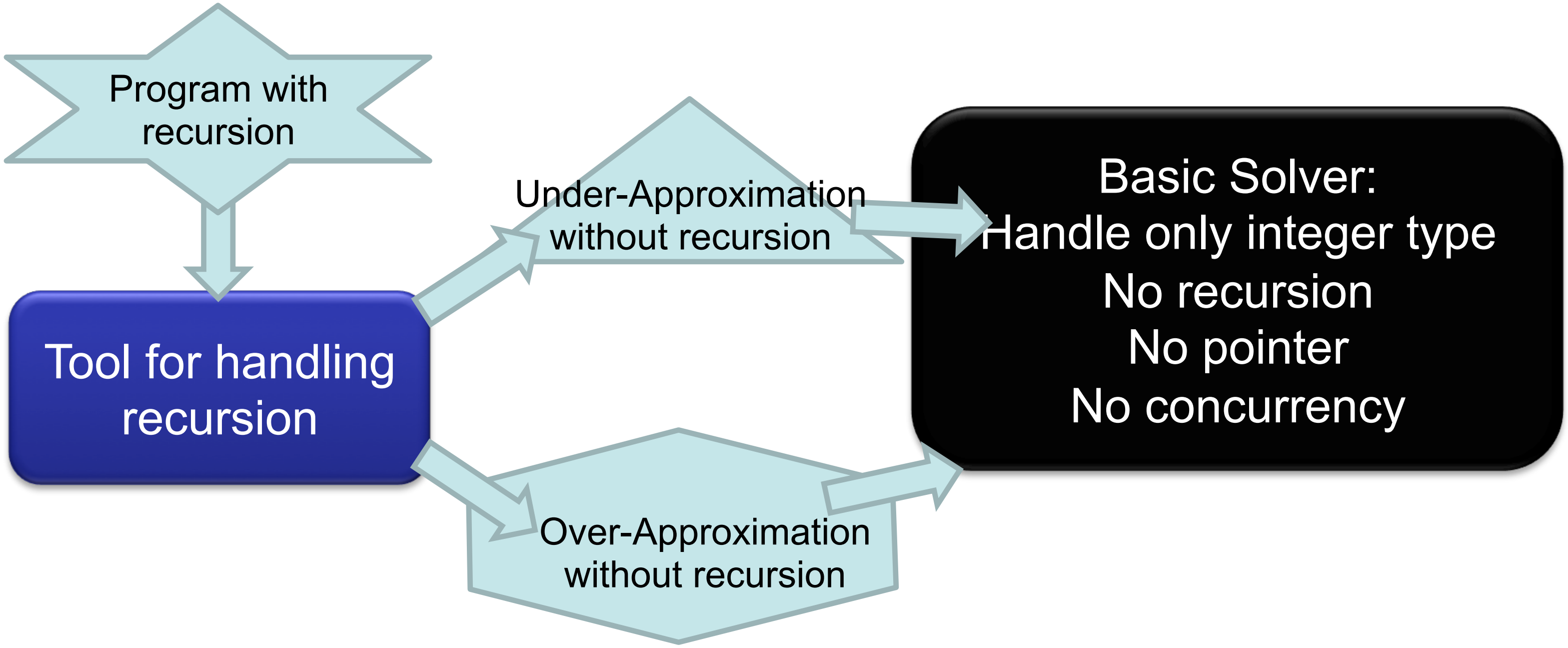
Tool for handling recursion

Equivalent program without recursion

Basic Solver:
Handle only integer type
No recursion
No pointer
No concurrency

Impossible:
PDA is more expressive than FA

The Idea



Program with recursion

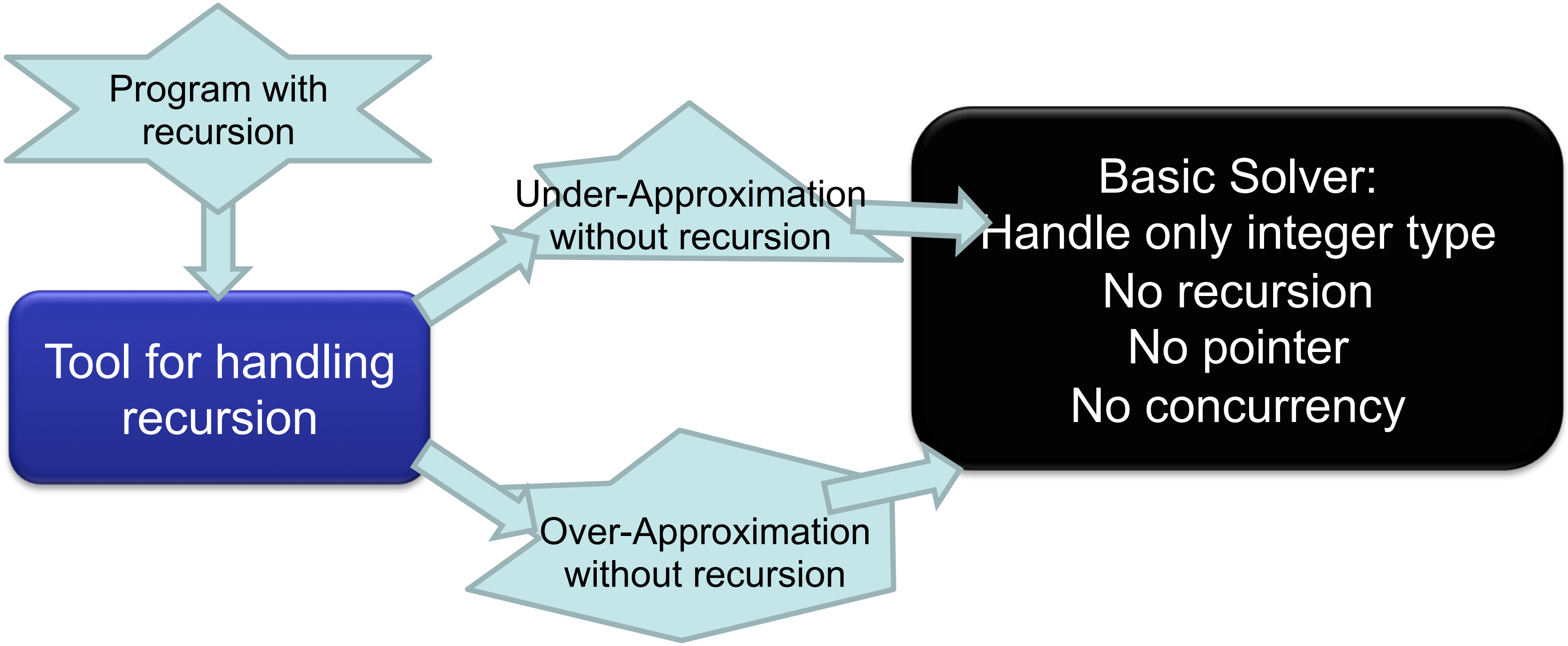
Tool for handling recursion

Under-Approximation without recursion

Over-Approximation without recursion

Basic Solver:
Handle only integer type
No recursion
No pointer
No concurrency

The Idea



Program with recursion

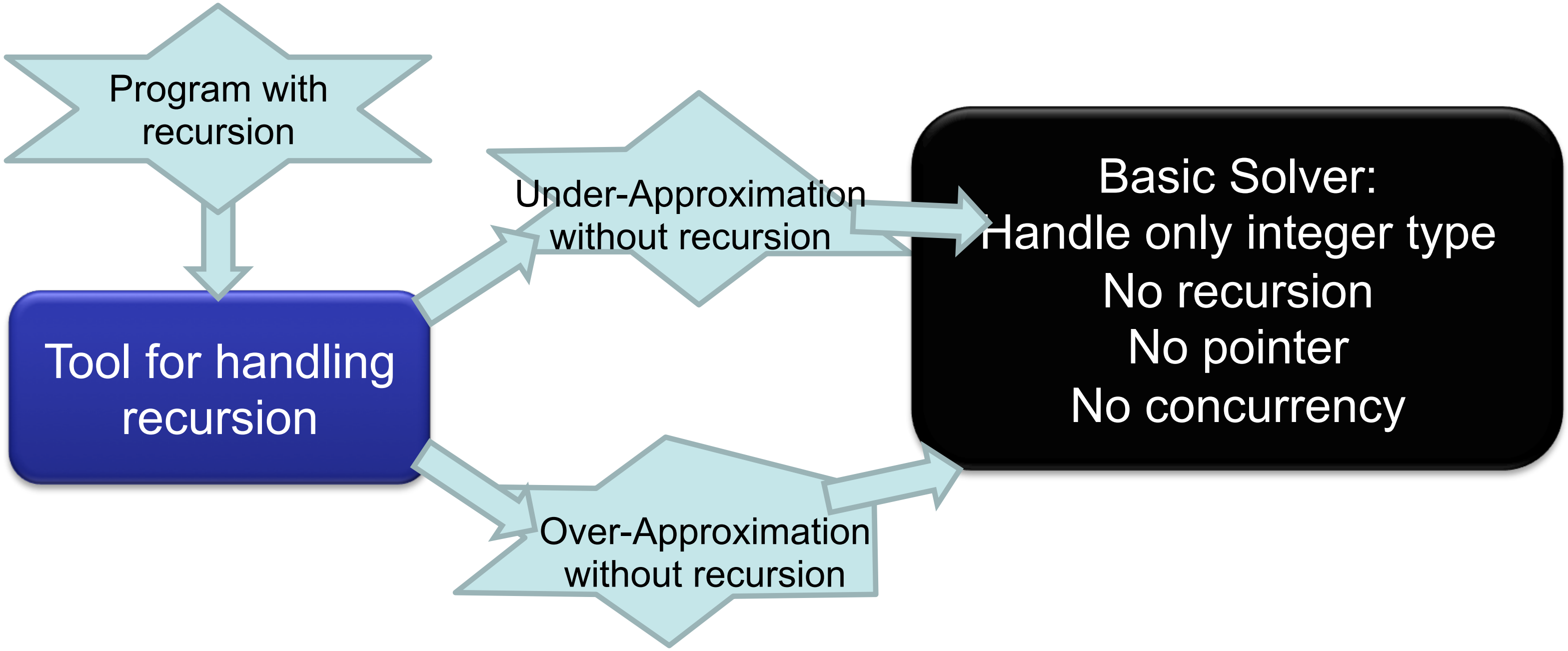
Tool for handling recursion

Under-Approximation without recursion

Over-Approximation without recursion

Basic Solver:
Handle only integer type
No recursion
No pointer
No concurrency

The Idea



Program with recursion

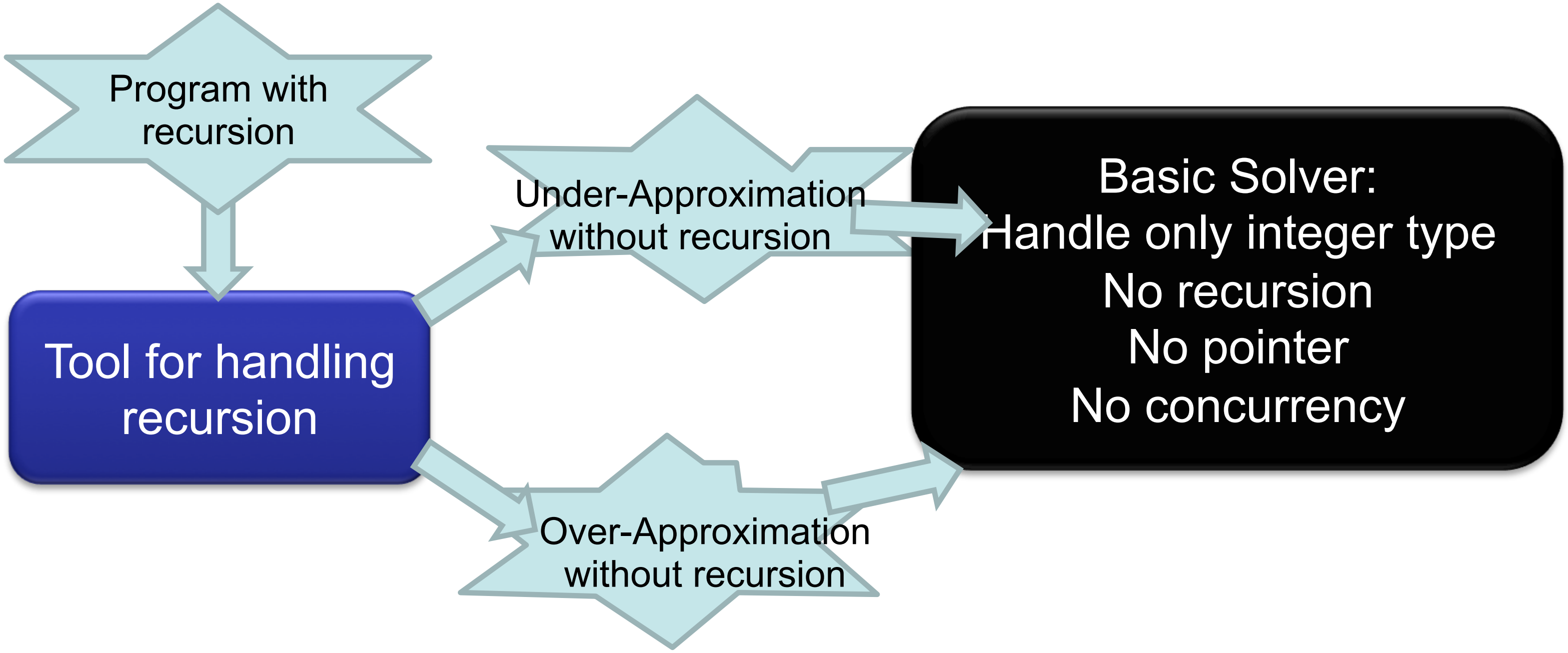
Tool for handling recursion

Under-Approximation without recursion

Over-Approximation without recursion

Basic Solver:
Handle only integer type
No recursion
No pointer
No concurrency

The Idea



Program with recursion

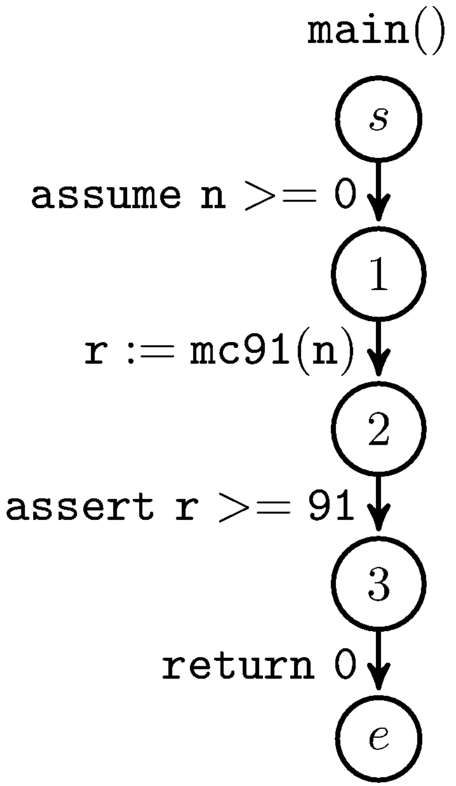
Tool for handling recursion

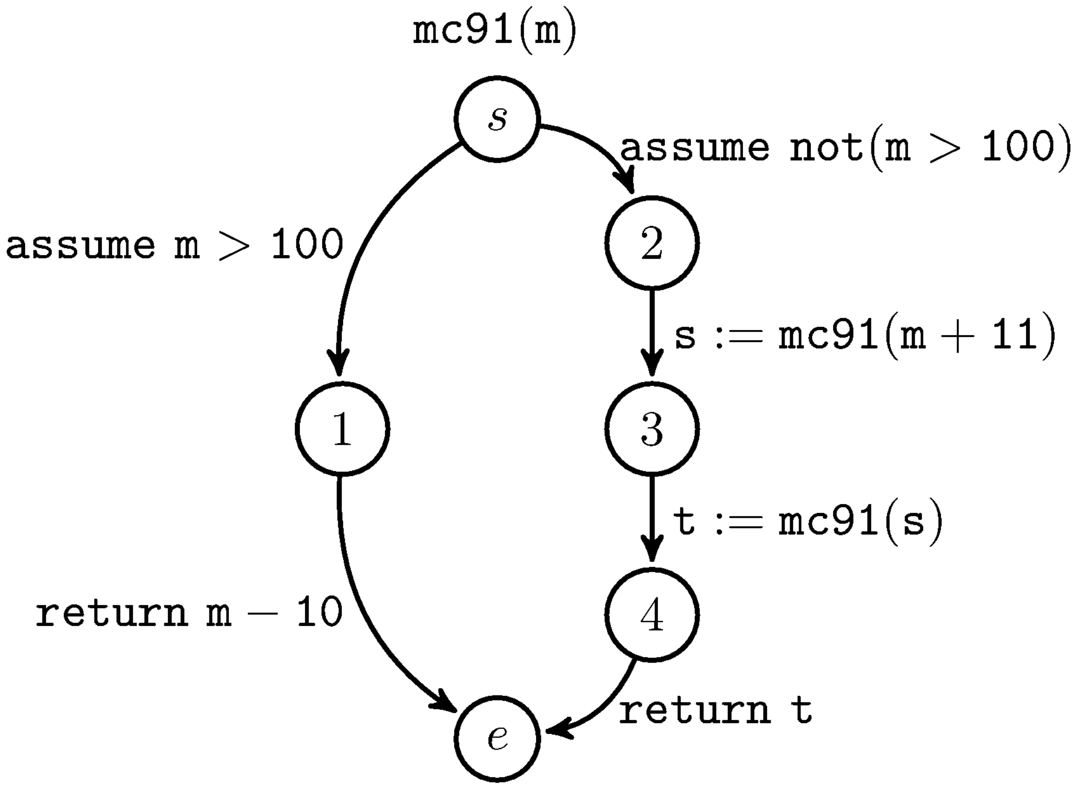
Under-Approximation without recursion

Over-Approximation without recursion

Basic Solver:
Handle only integer type
No recursion
No pointer
No concurrency

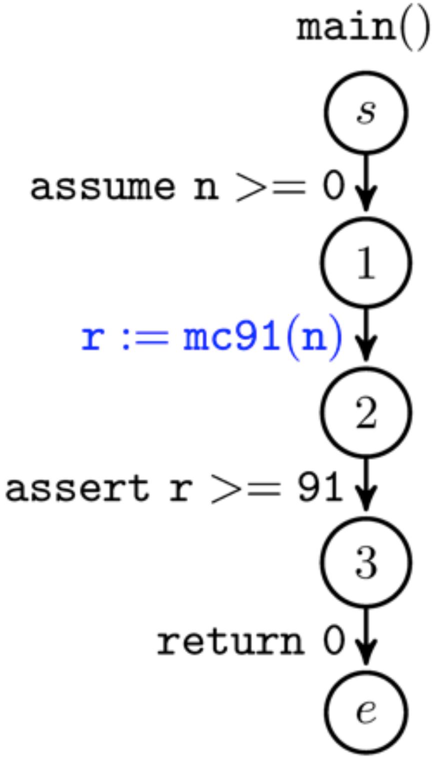
An Example: McCarthy 91



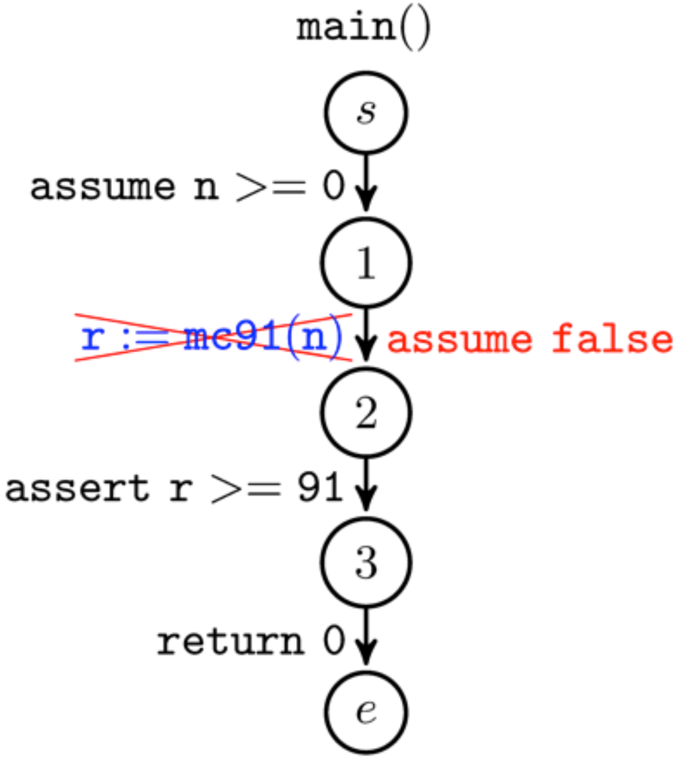


Goal: verify assertion safely
Assumption: Formal parameters are **read-only**

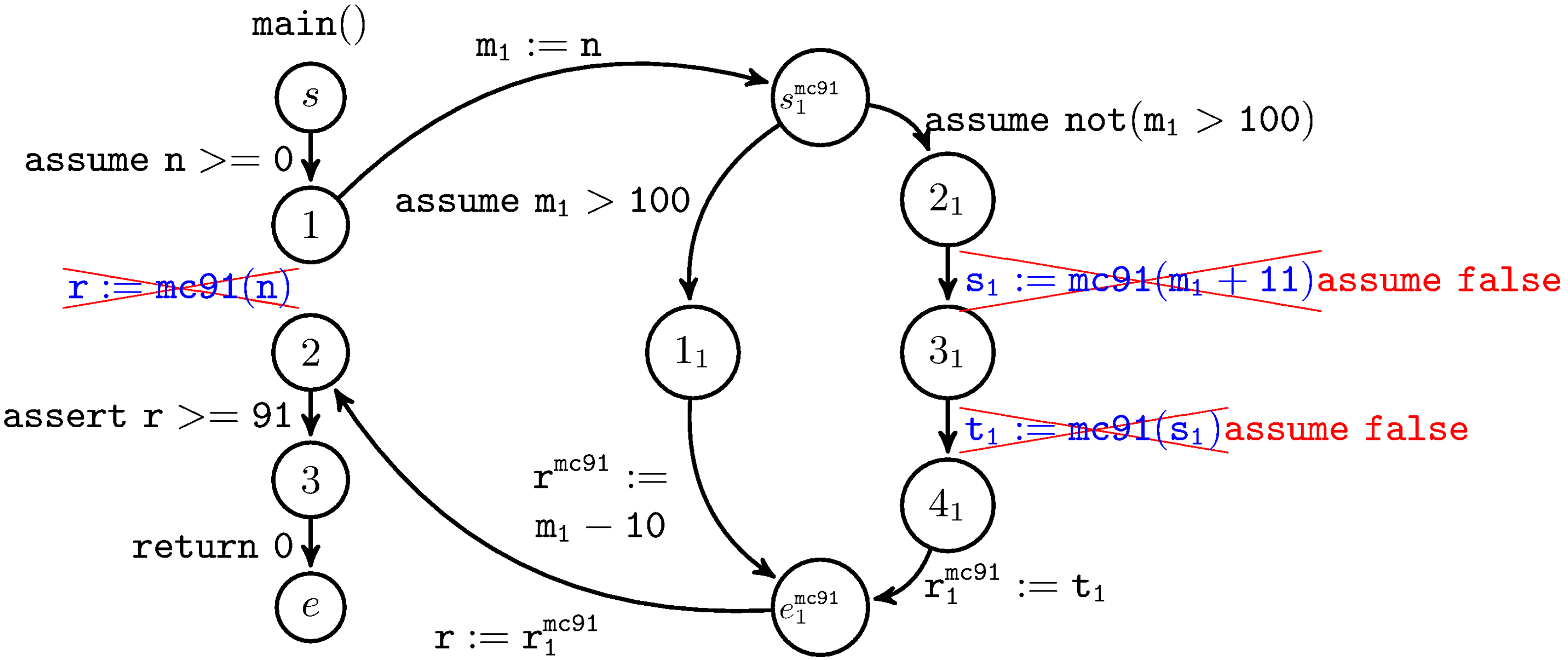
Construct Under-Approximation



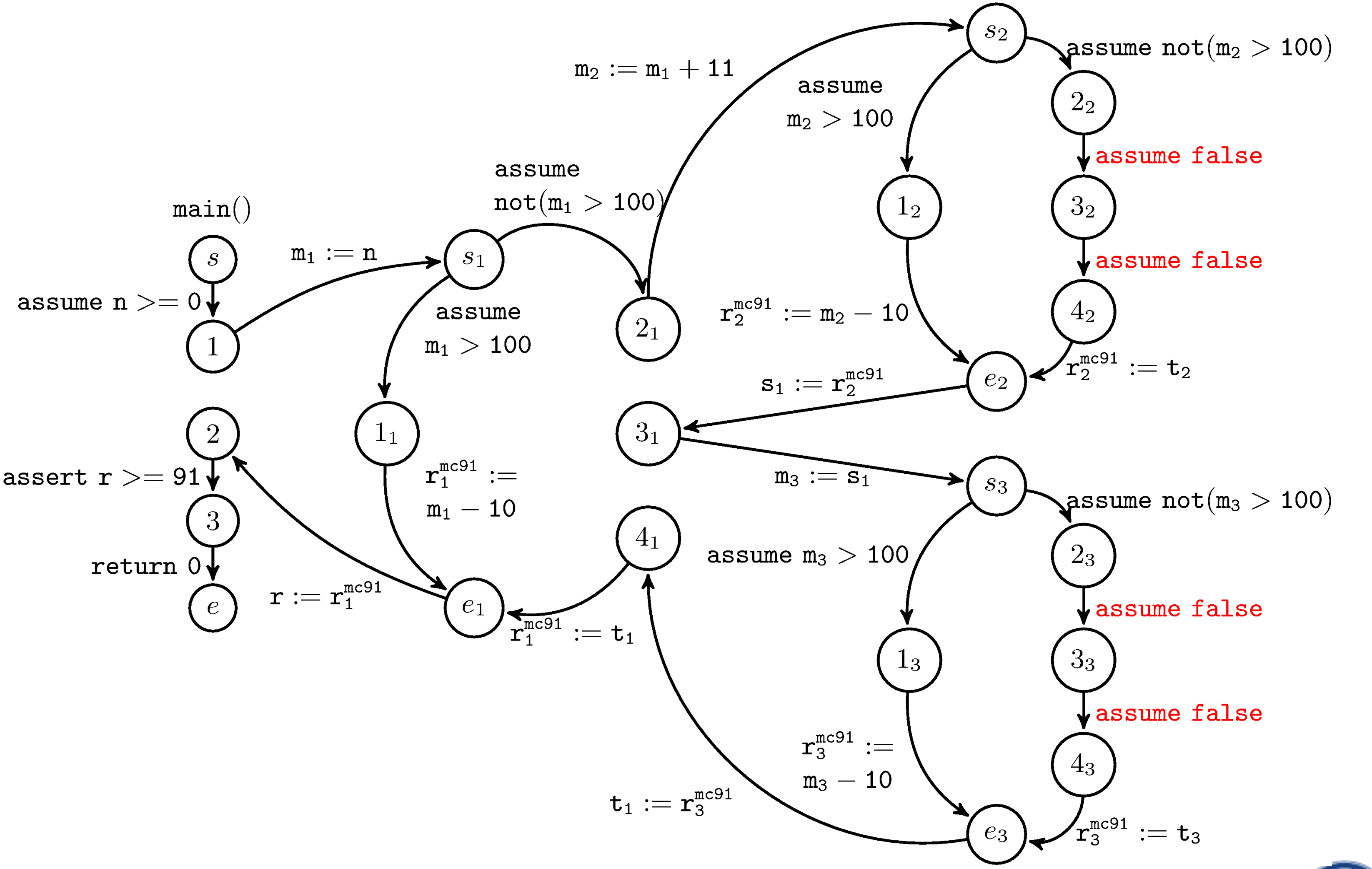
Construct Under-Approximation



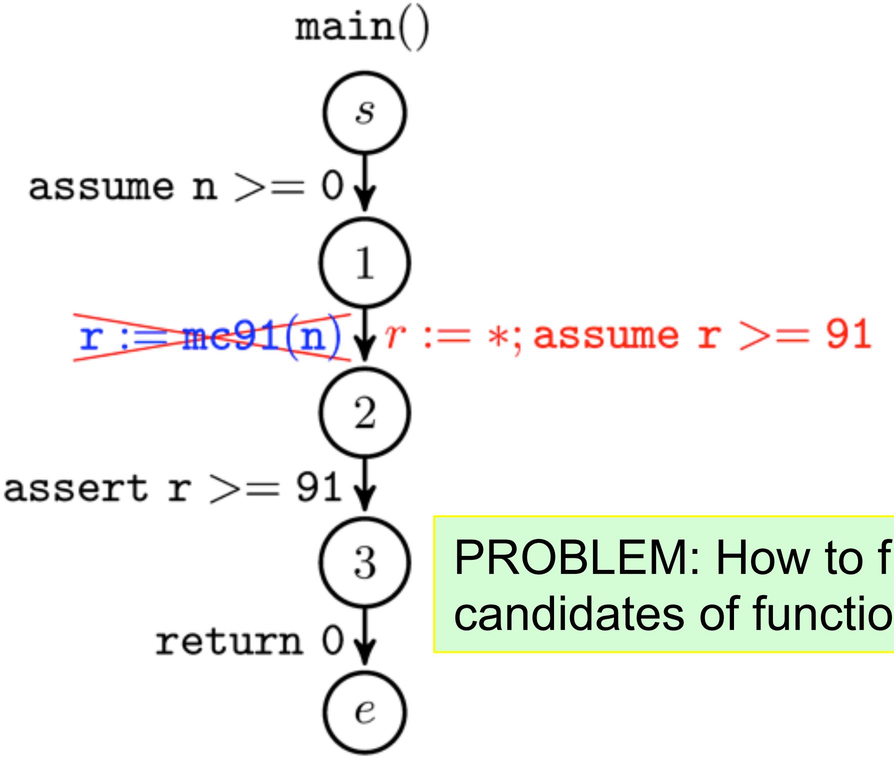
Refine the Approximation by Unwinding



More Accurate Refinement



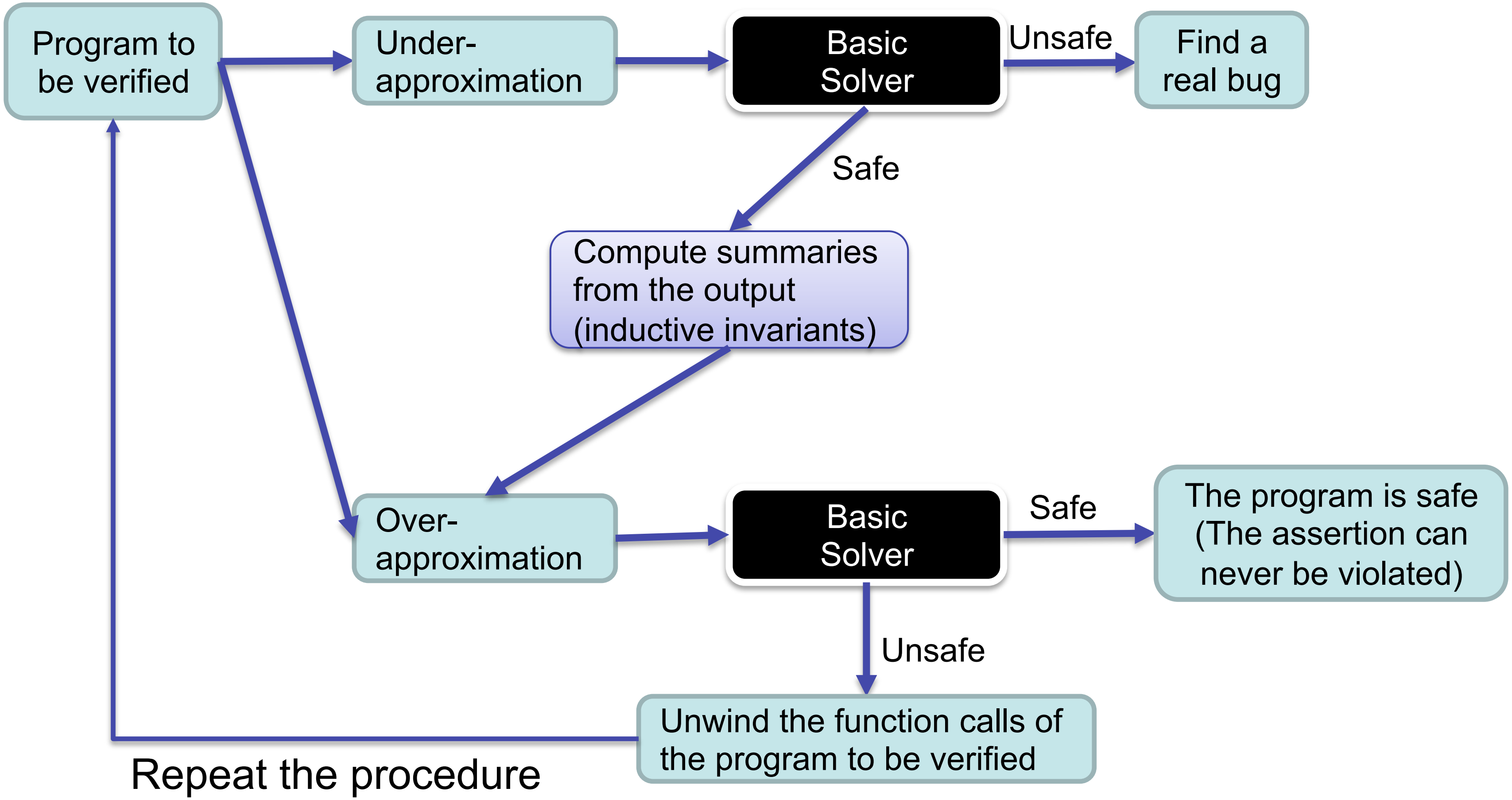
Construct Over-Approximation



PROBLEM: How to find reasonable candidates of function summaries?

Assume we have a summary {true} r := mc91(n) { r≥91}

Basic Flow



Program to be verified

Under-approximation

Basic Solver

Unsafe

Find a real bug

Safe

Compute summaries from the output (inductive invariants)

Over-approximation

Basic Solver

Safe

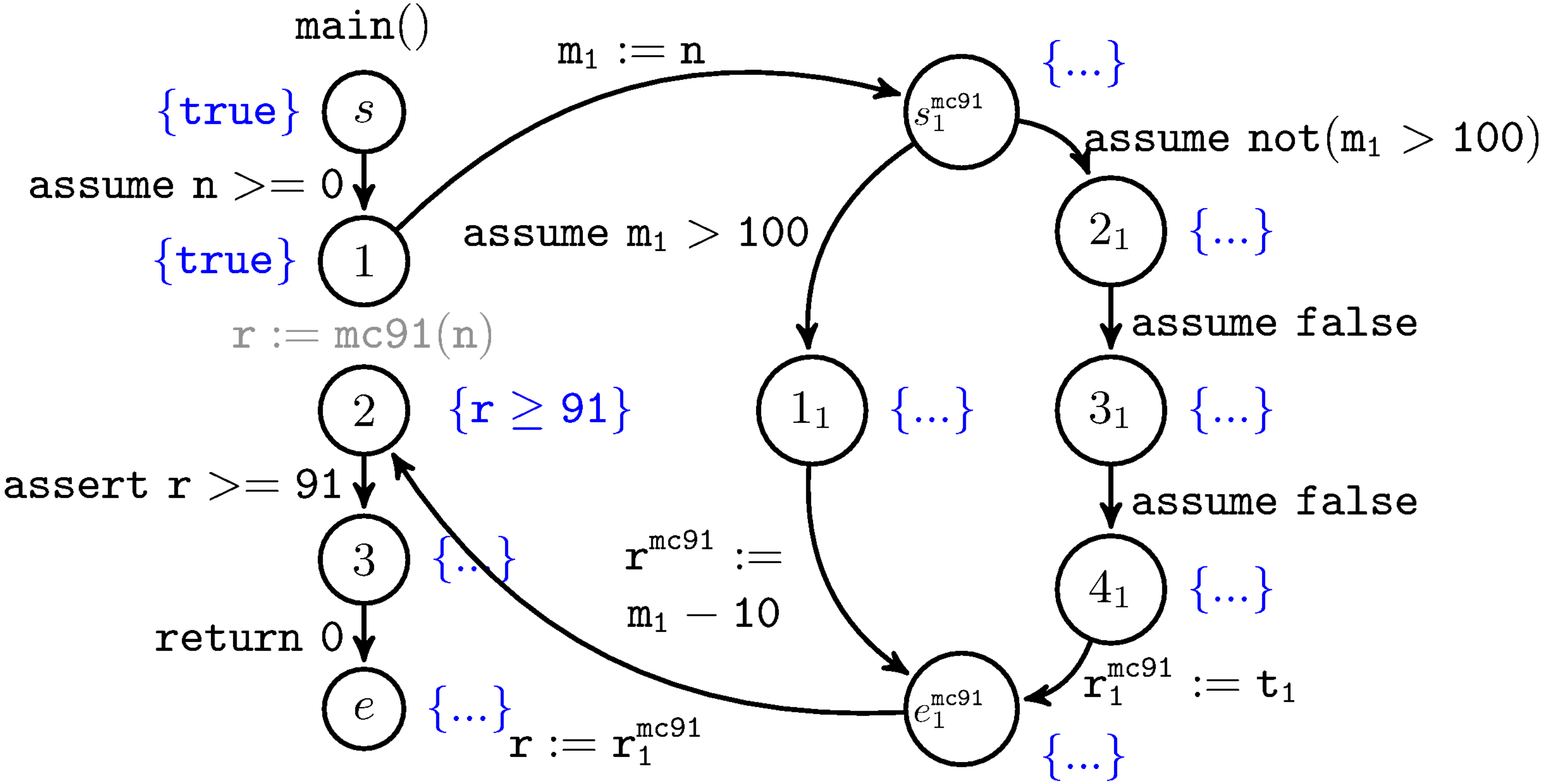
The program is safe (The assertion can never be violated)

Unsafe

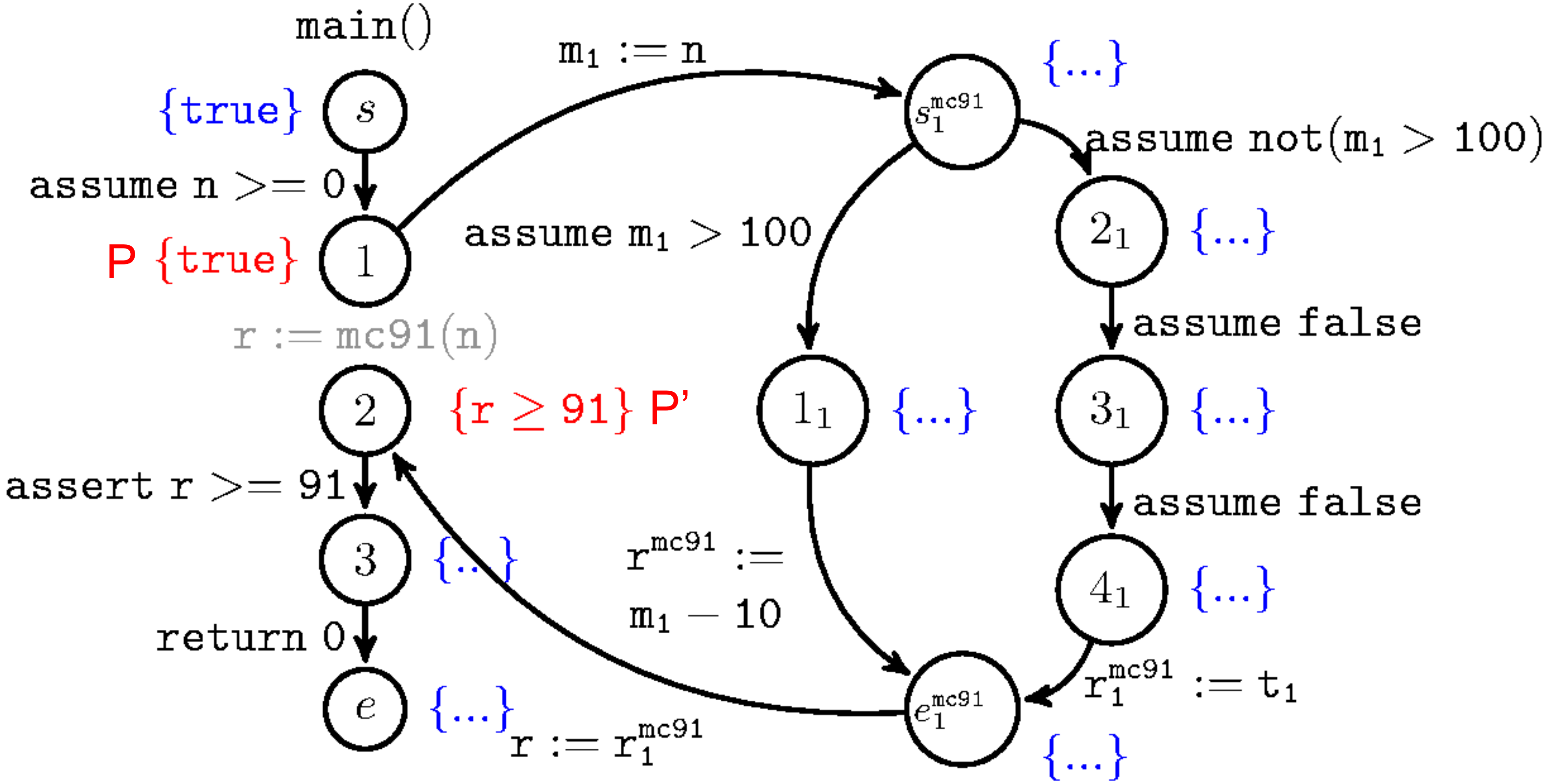
Unwind the function calls of the program to be verified

Repeat the procedure

Inductive Invariant from Basic Solver



Candidate from the Inductive Invariant

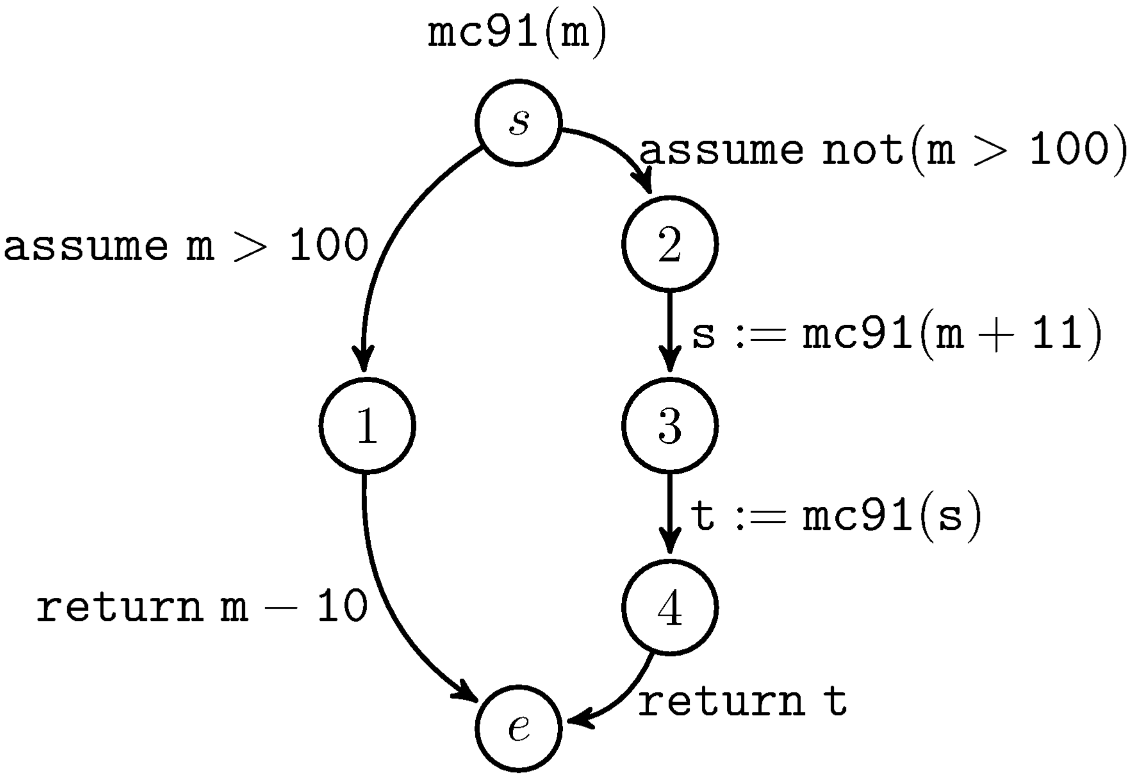


Generate a candidate of summary: true → $r^{mc91} \geq 91$

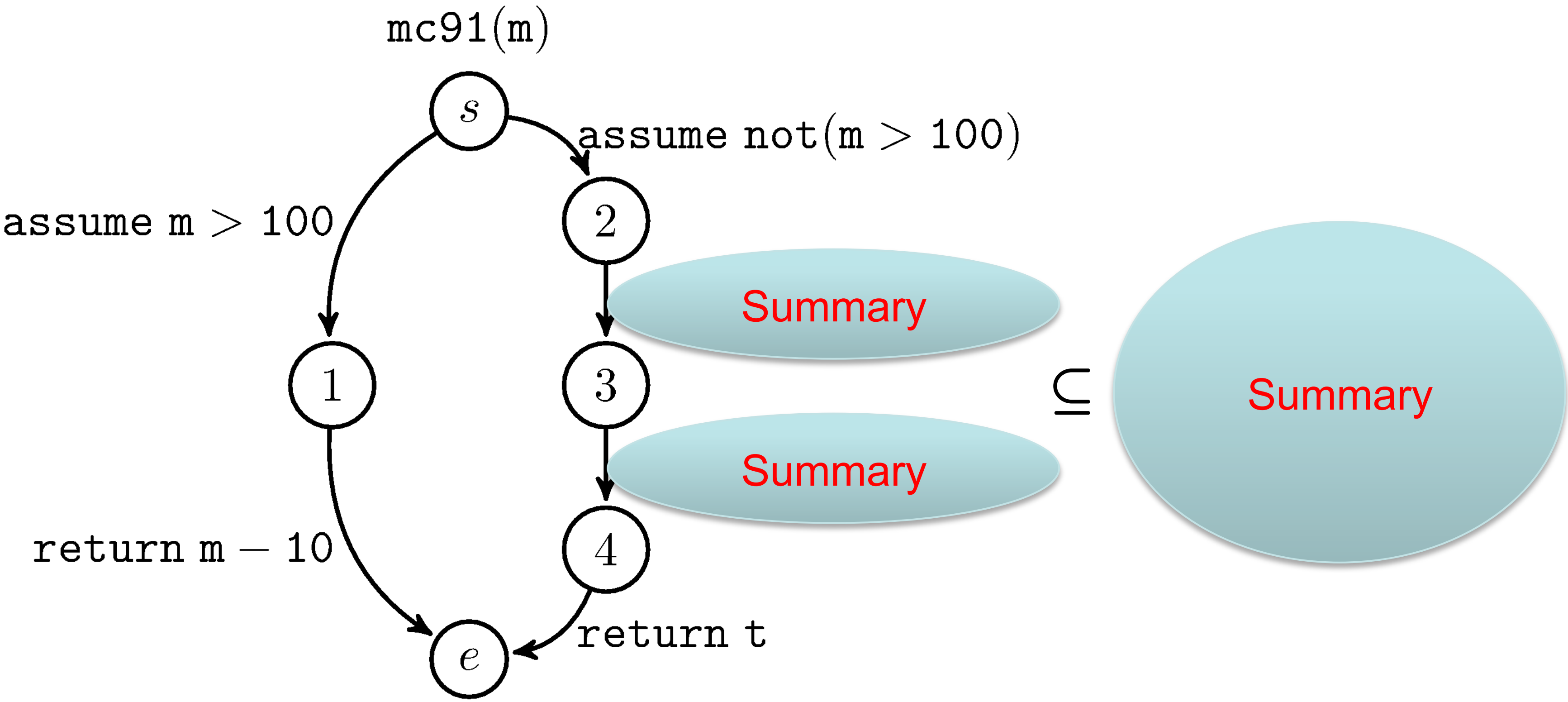
$\forall N_{FR}$. P →P', where N_{FR} means all variables other than formal parameters and return variables

Some renaming is needed. r^{mc91} is the return variable of the function mc91.

Check the Summary

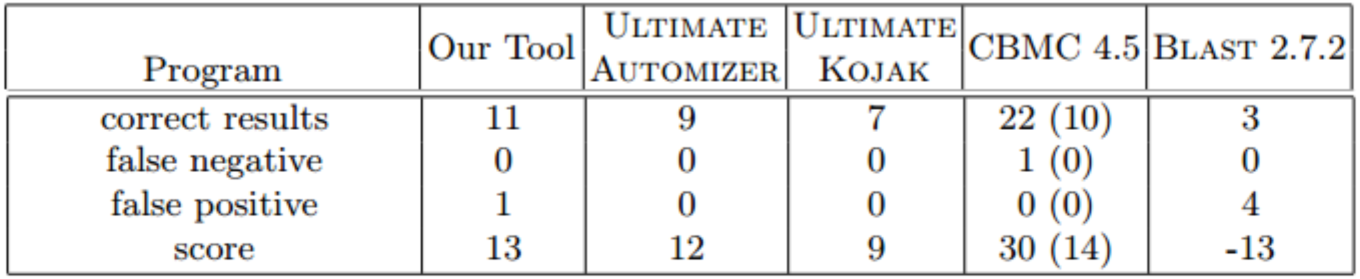


Check the Summary



Experimental Results

- Benchmarks: the Recursive category of the 2014 Competition on Software Verification



Program	Our Tool	Ultimate Automizer	Ultimate Kojak	CBMC 4.5	Blast 2.7.2
correct results	11	9	7	22 (10)	3
false negative	0	0	0	1 (0)	0
false positive	1	0	0	0 (0)	4
score	13	12	9	30 (14)	-13

- Blast is the most well-known tool in software verification.
- The rest are the top 3 tools in the competition.

Our Advantages

- A light-weight, modular approach for recursive program verification.
  - Our implementation has 2k lines, CPAChecker has 170k lines
- The performance of the implementation is comparable to those specialized for handling recursion