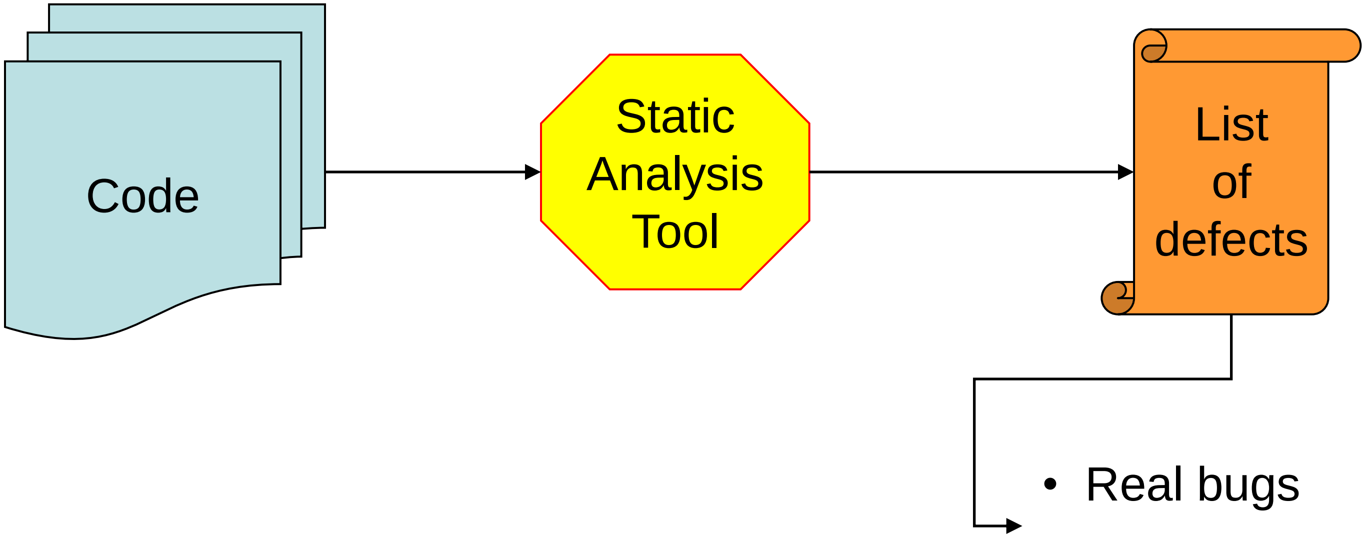
Theoretical Foundation of CodeHawk: Abstract Interpretation

Arnaud Venet

Kestrel Technology

arnaud@kestreltechnology.com

Static Analysis

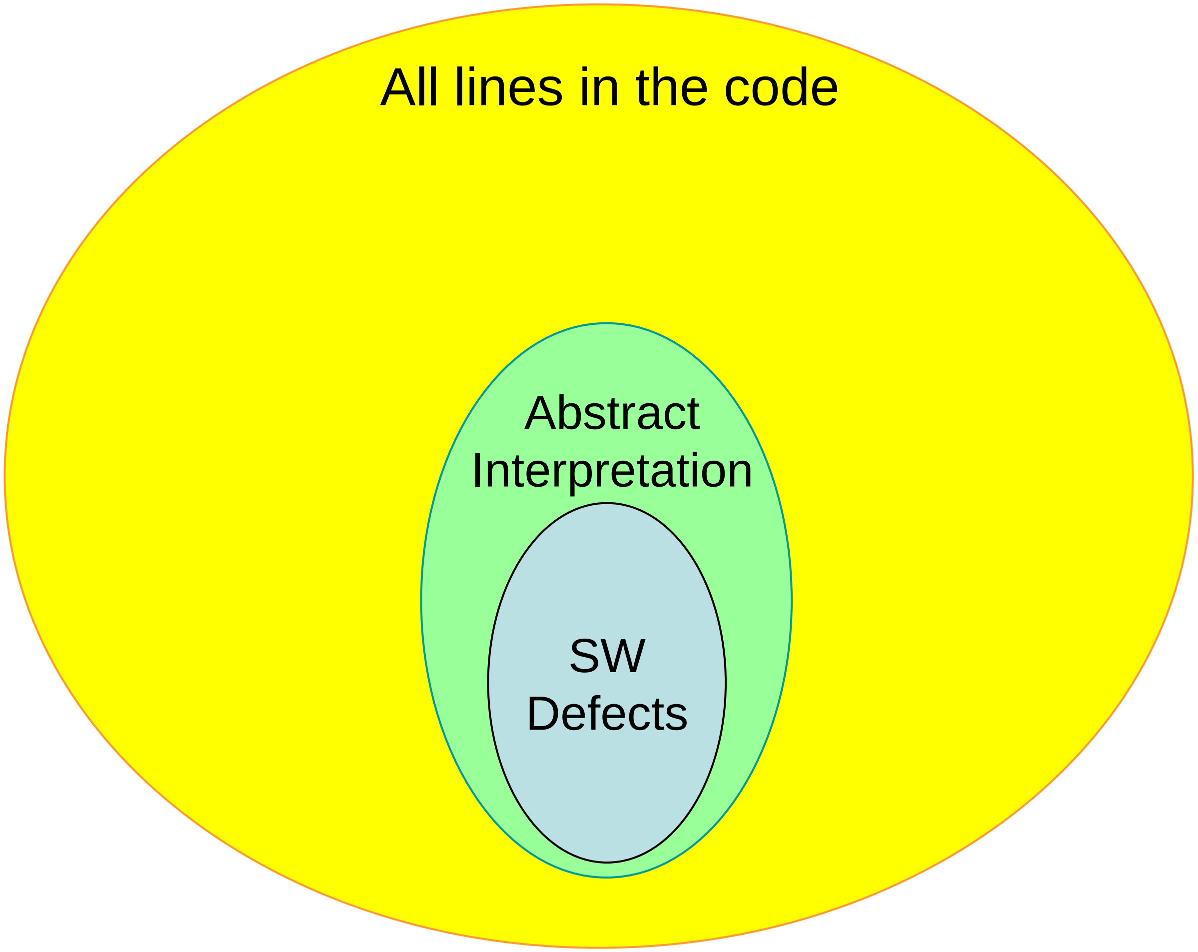


- Real bugs
- False alarms

Can we find all defects?

- Classic undecidability results in Computer Science (halting problem)
- We require soundness (no defects are missed)
  - A conservative approach is acceptable
- Abstract Interpretation is the enabling theory in CodeHawk
  - Sound
  - Tunably precise
  - Scalable
  - “Generatively general”

Intuition



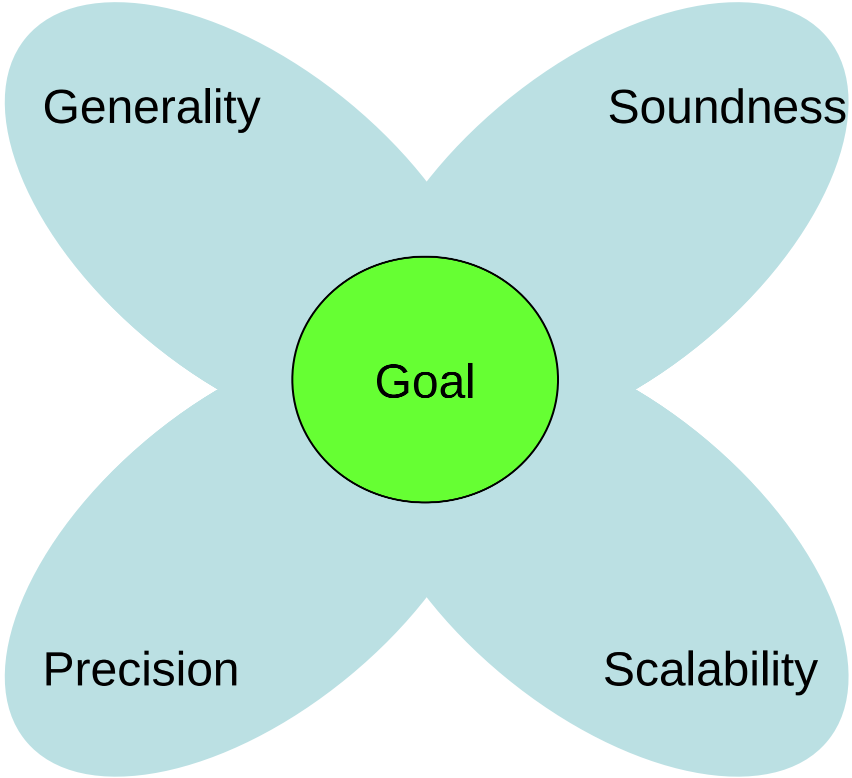
Abstract Interpretation

- Computes an envelope of all data in the program
- Mathematical assurance
- Static analyzers based on Abstract interpretation are difficult to engineer
- KT’s expertise: building scalable and effective abstract interpreters

Properties vs. defects

- An application might be defect-free but not carry the desired property
  - resource issues (memory, execution time)
  - separation
  - range of output data
  - vulnerability to attack
  - forbidden functionality
  - compliance with a policy
- Abstract Interpretation covers those families of properties as well

Static Analyzer Wish List



- Experience shows you can have any three.
- We want an approach to have all four.

Objectives

- Go over a detailed example
  - Understand how the technology works
- Achievements and challenges in the engineering of abstract interpreters
  - What it means to build an analyzer based on Abstract Interpretation

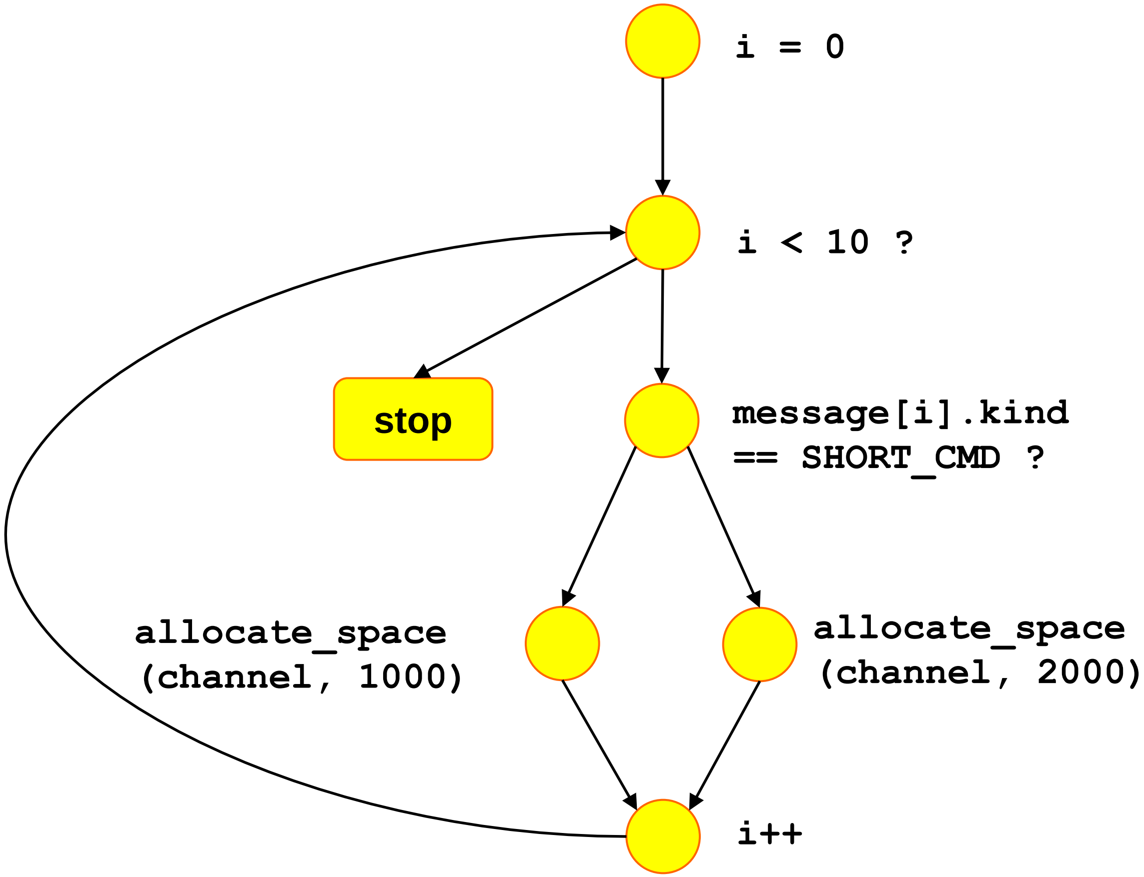
Detailed example

Buffer overflow

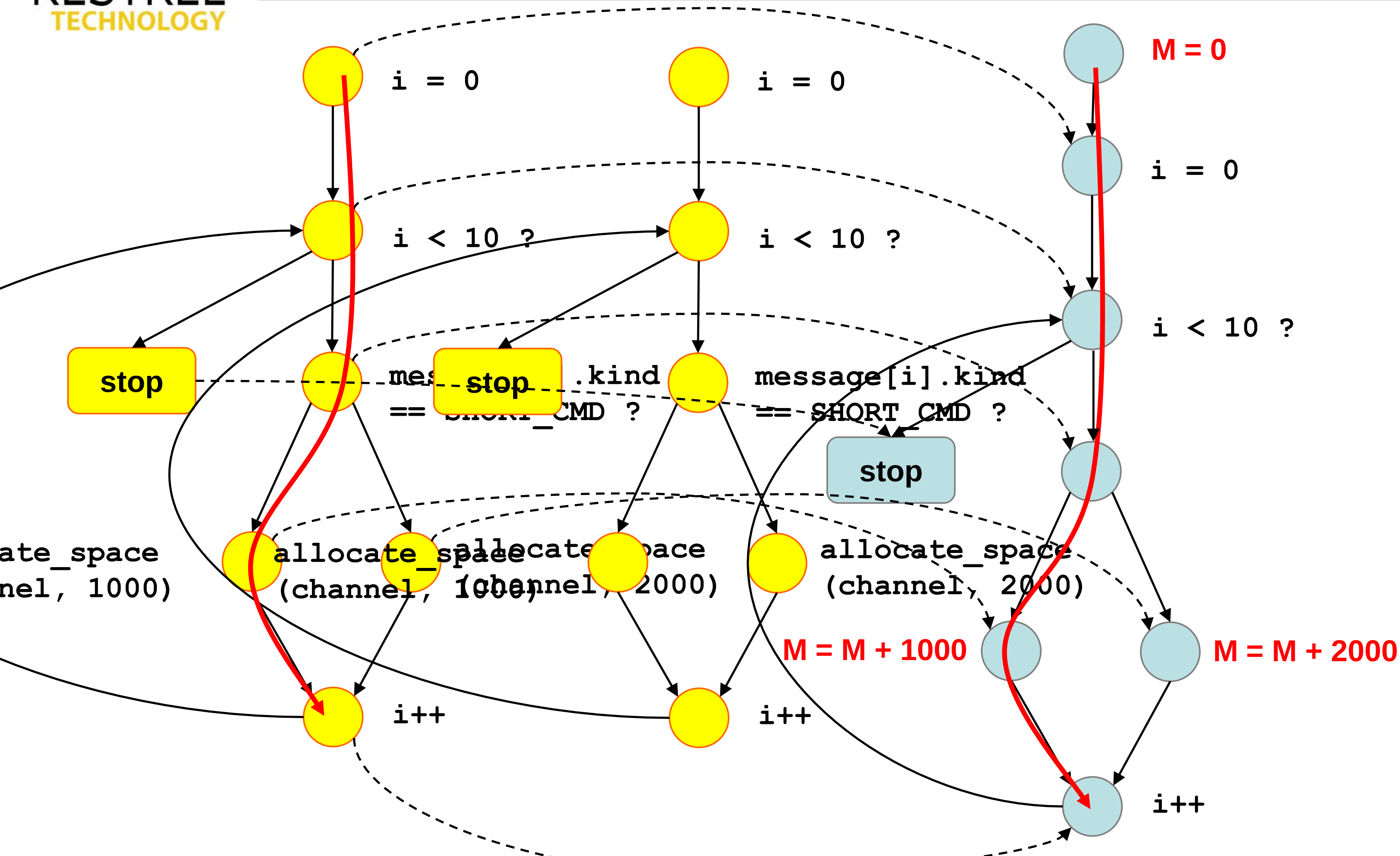
```
for(i = 0; i < 10; i++) {
  if(message[i].kind == SHORT_CMD)
    allocate_space (channel, 1000);
  else
    allocate_space (channel, 2000);
}
```

Can we exceed the channel's buffer capacity?

Control Flow Graph



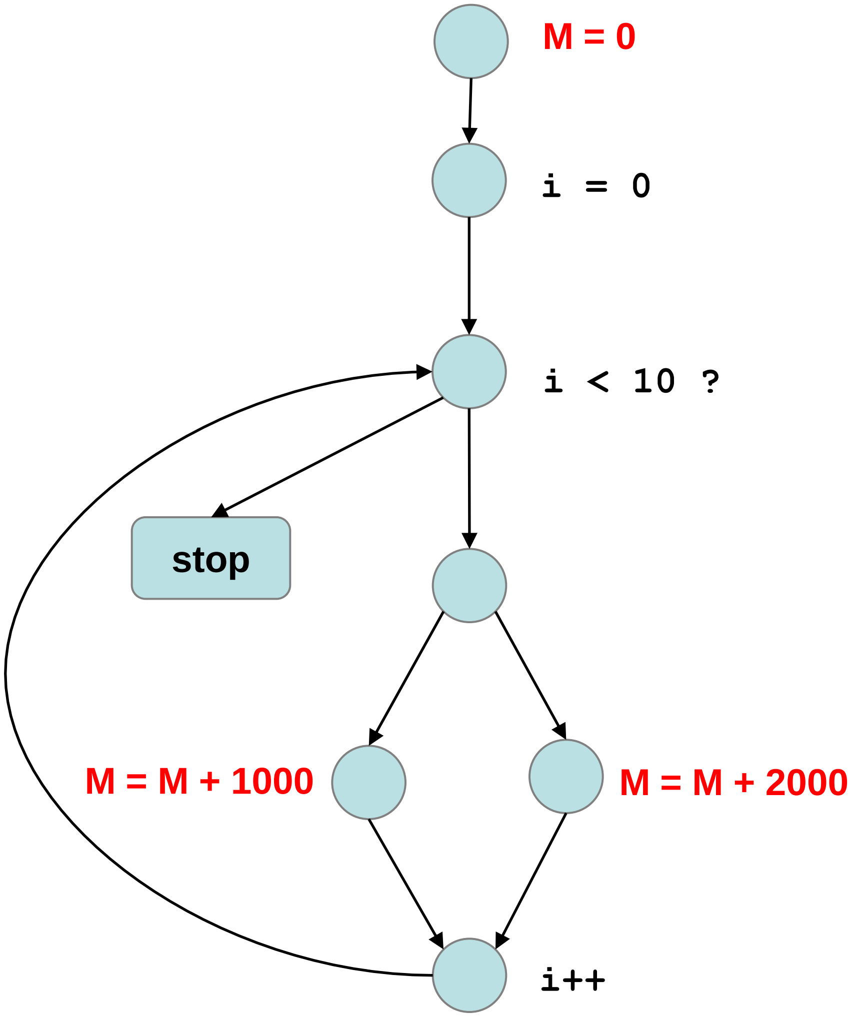
Analytic model of the code

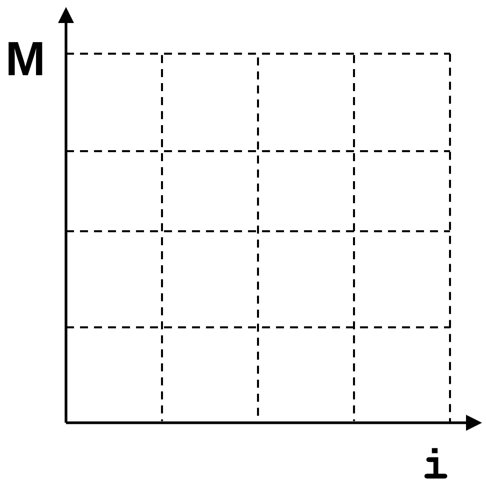


Analysis process intuition

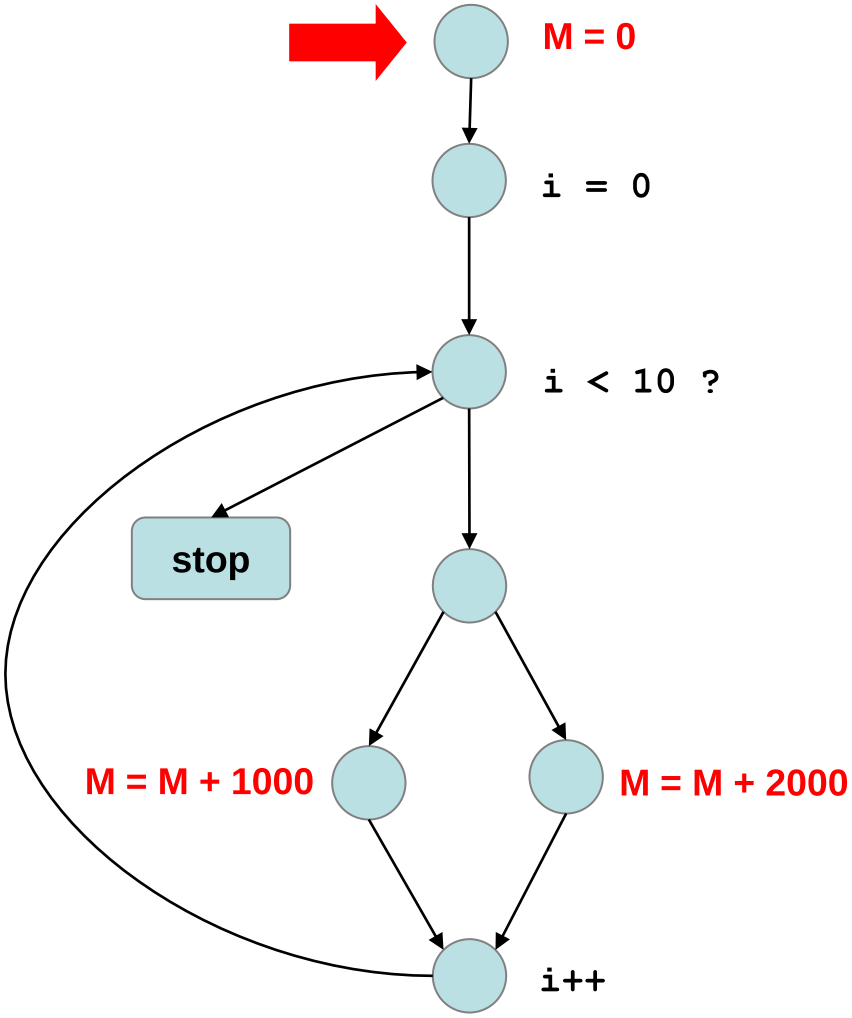
- We mimic the execution of the program
- We collect all possible data values

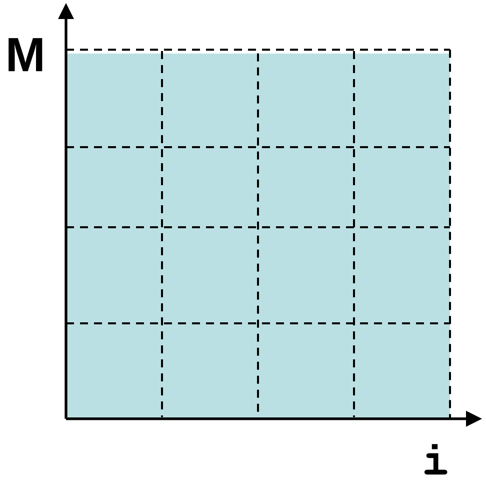
Analyzing the model



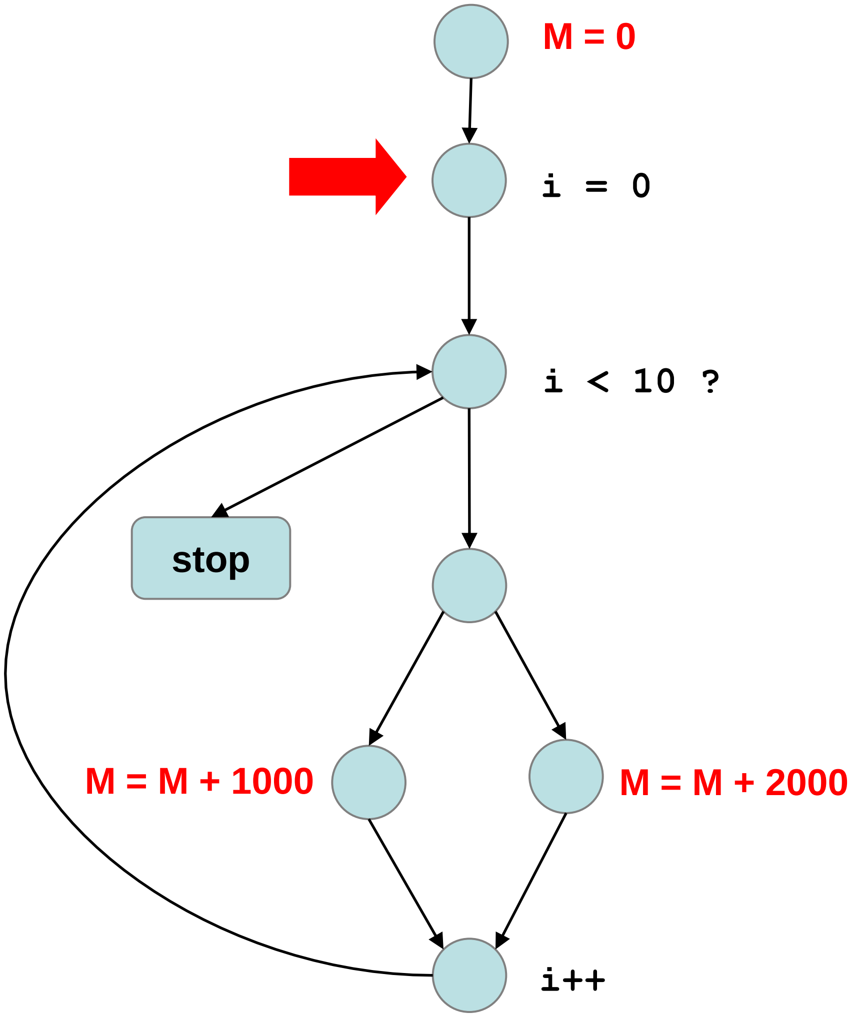


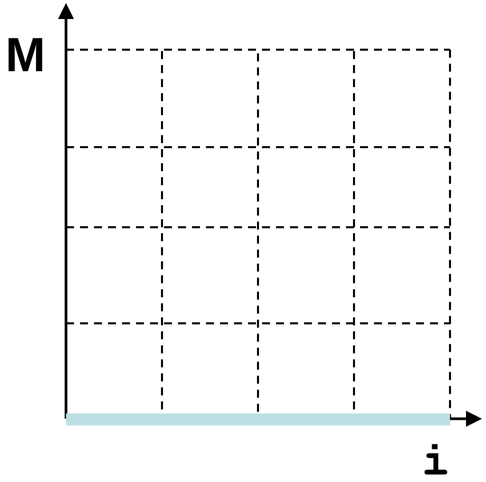
Initially



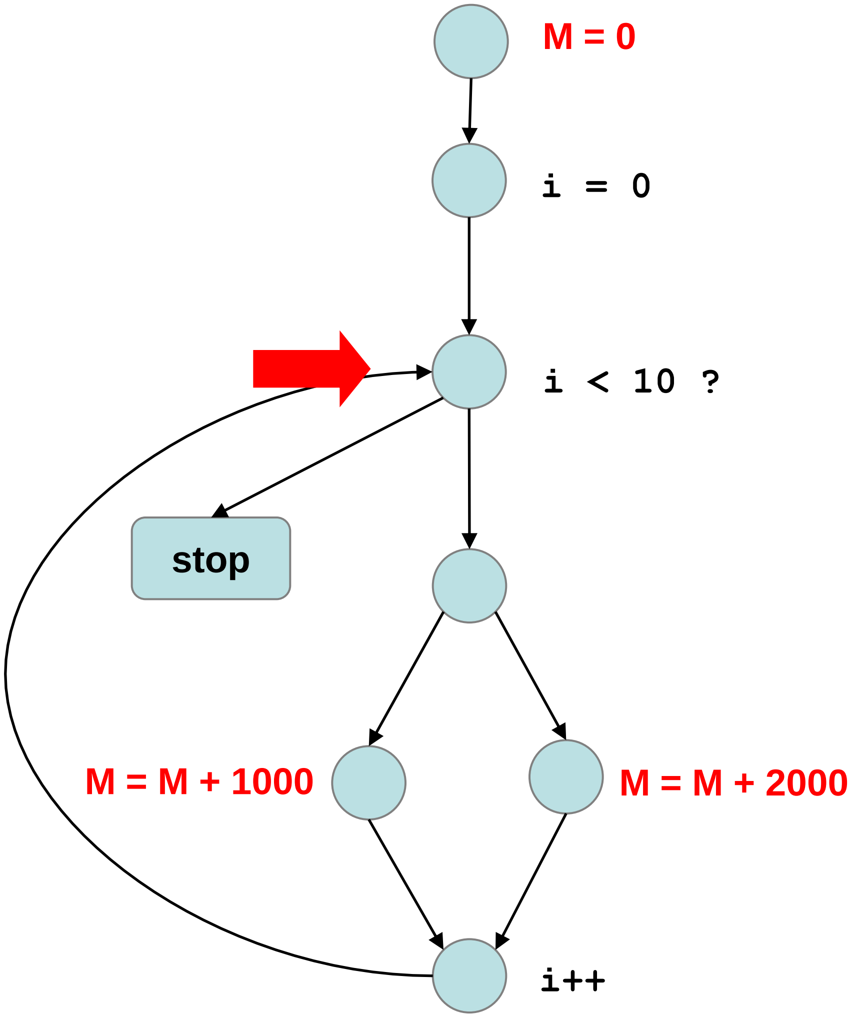


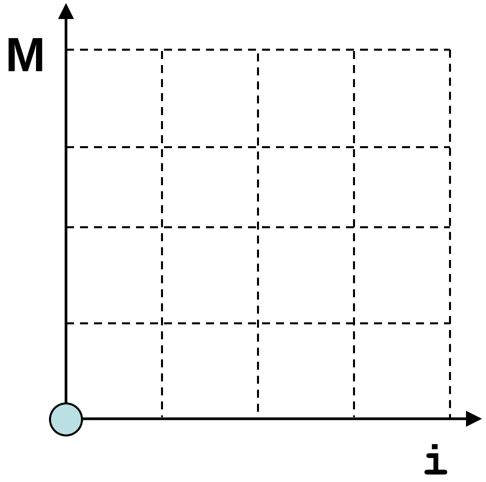
Loop initialization



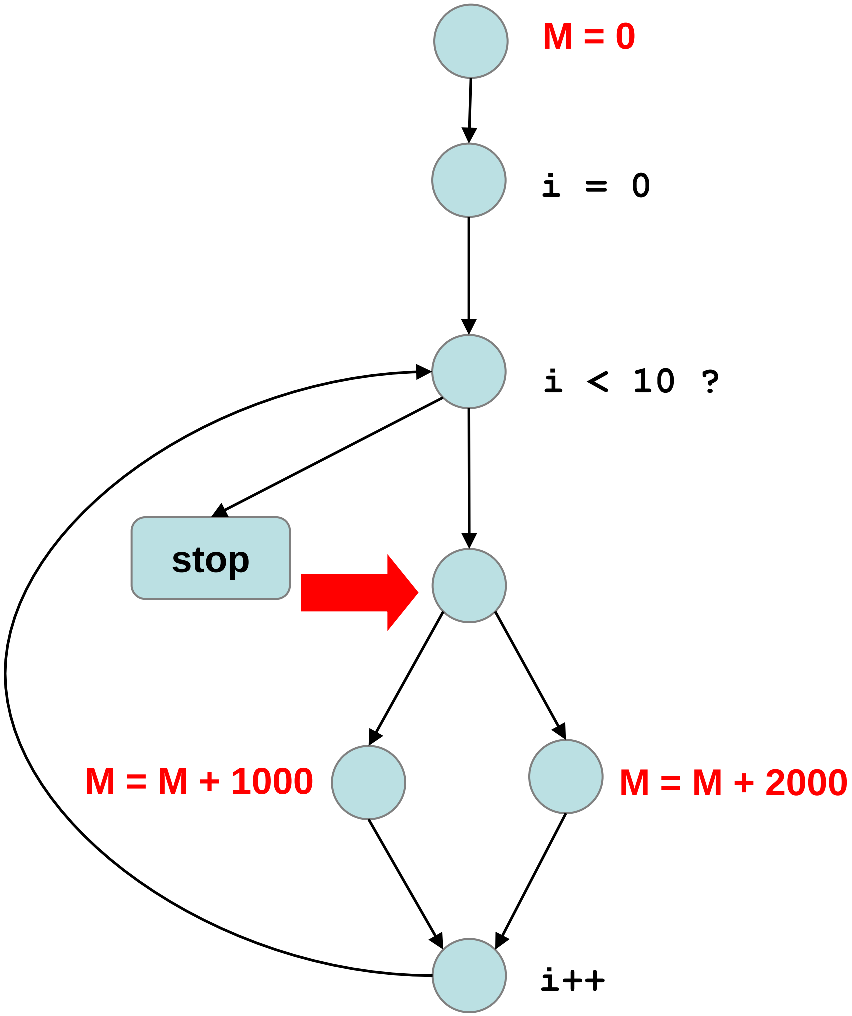


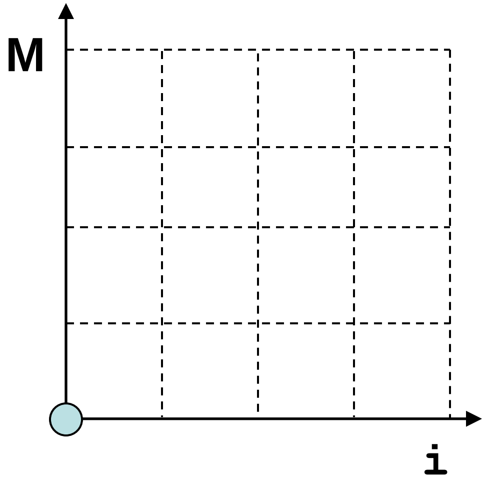
Loop entry



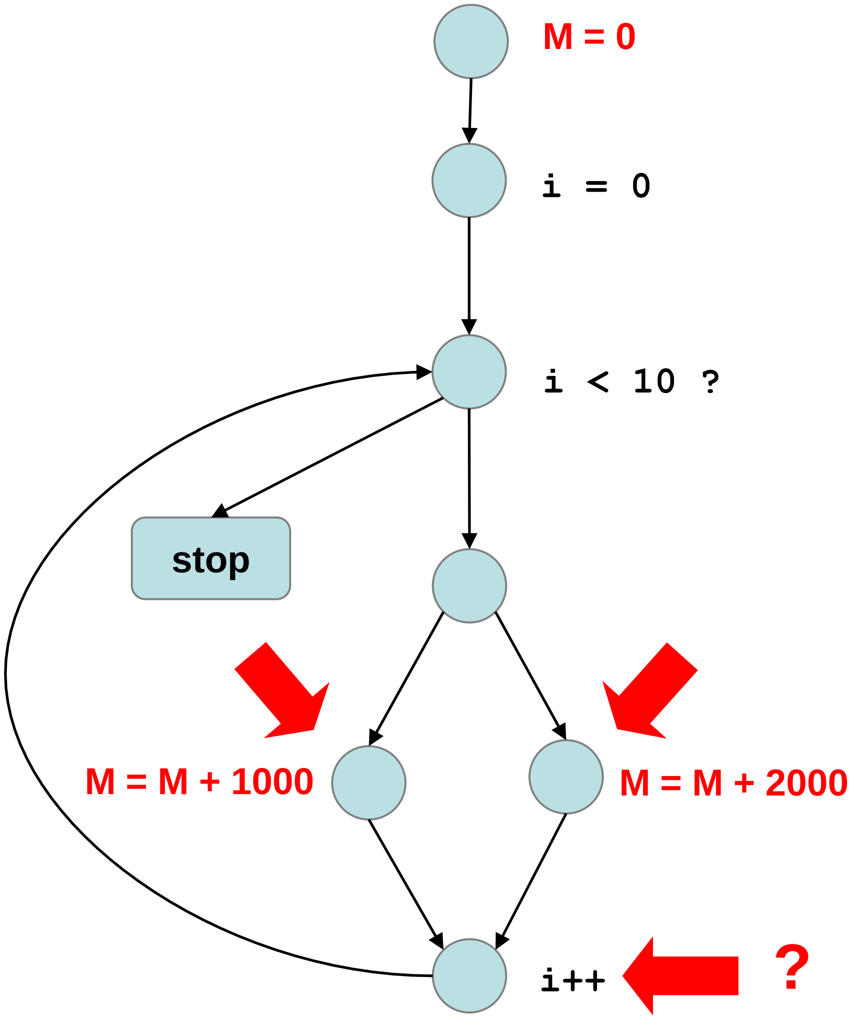


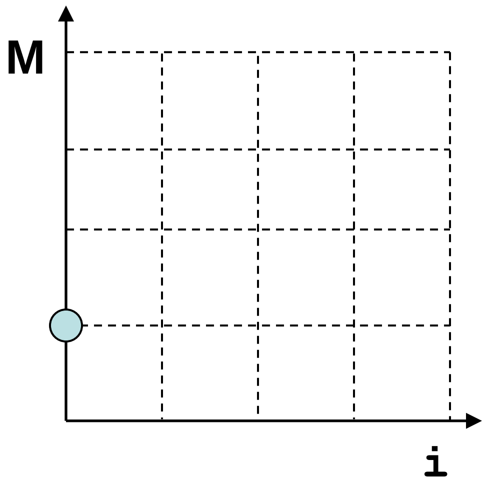
Analyzing a branching (1)

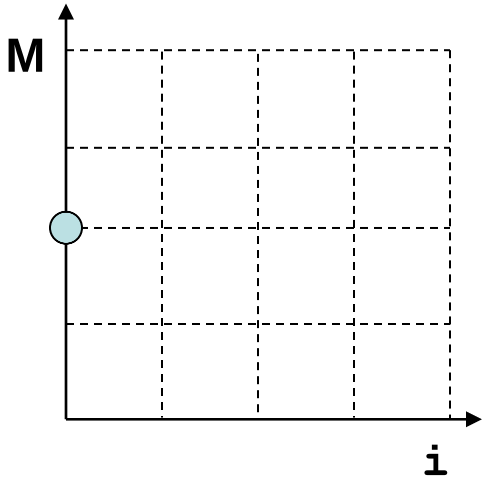




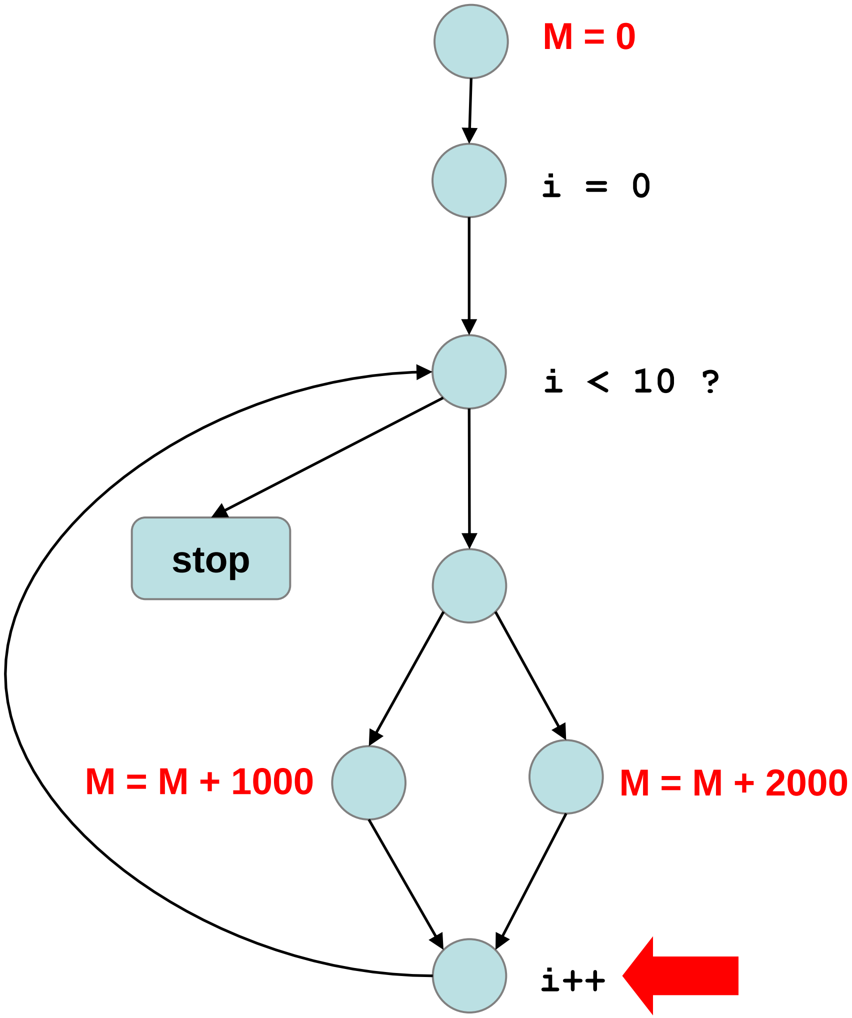
Analyzing a branching (2)

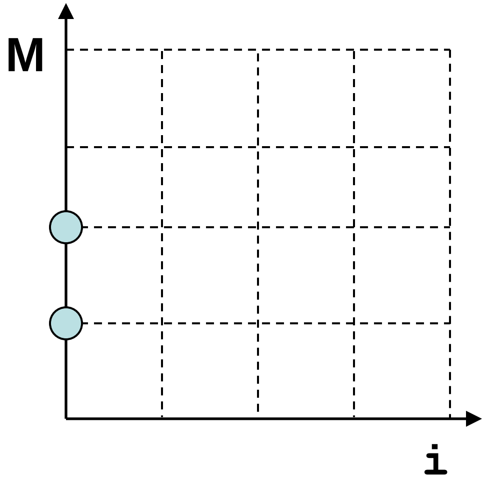






Accumulating all possible values

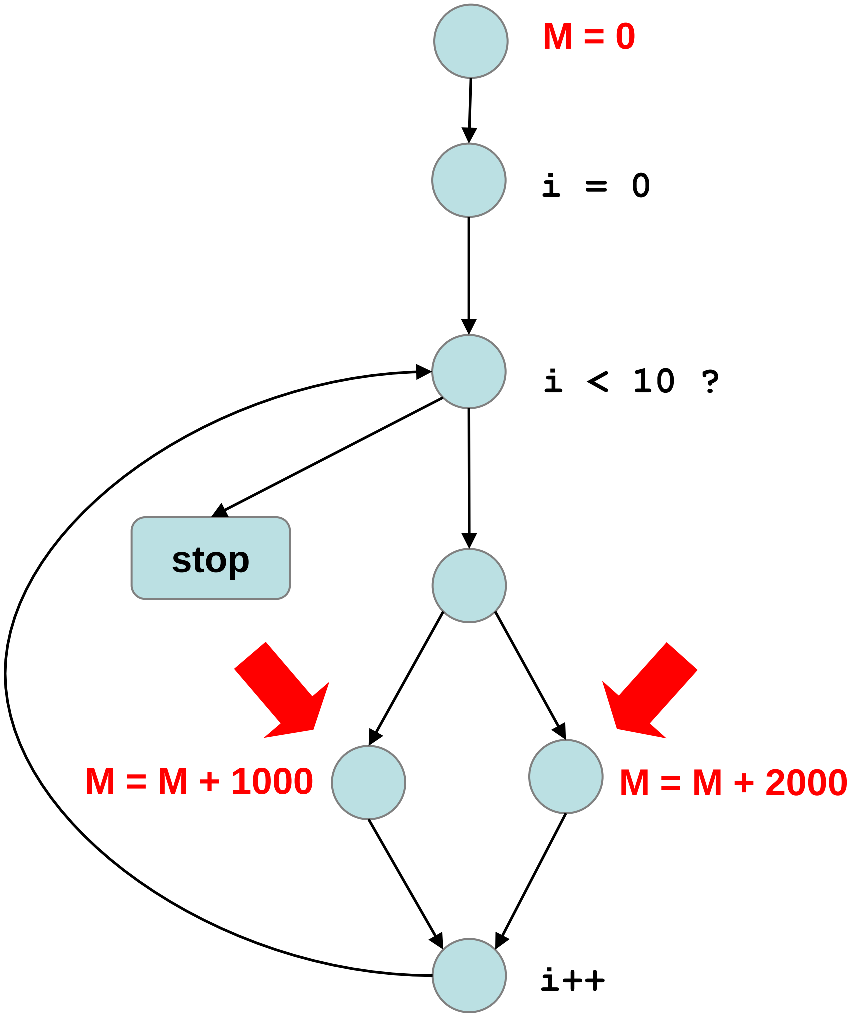


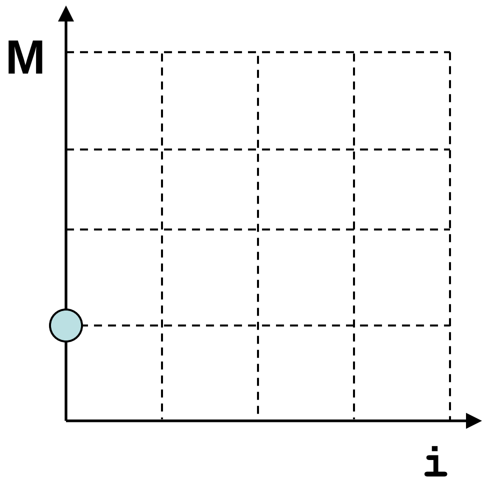


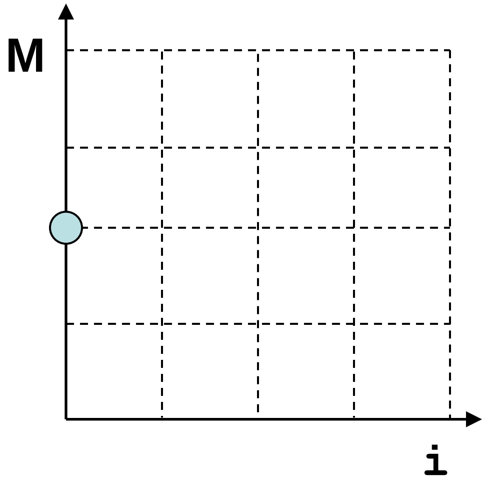
Abstraction of point clouds

- We want the analysis to terminate in reasonable time
- We need a tractable representation of point clouds in arbitrary dimensions
- Abstract Interpretation offers a broad choice of such representations
- Example: convex polyhedra
  – Compute the convex hull of a point cloud

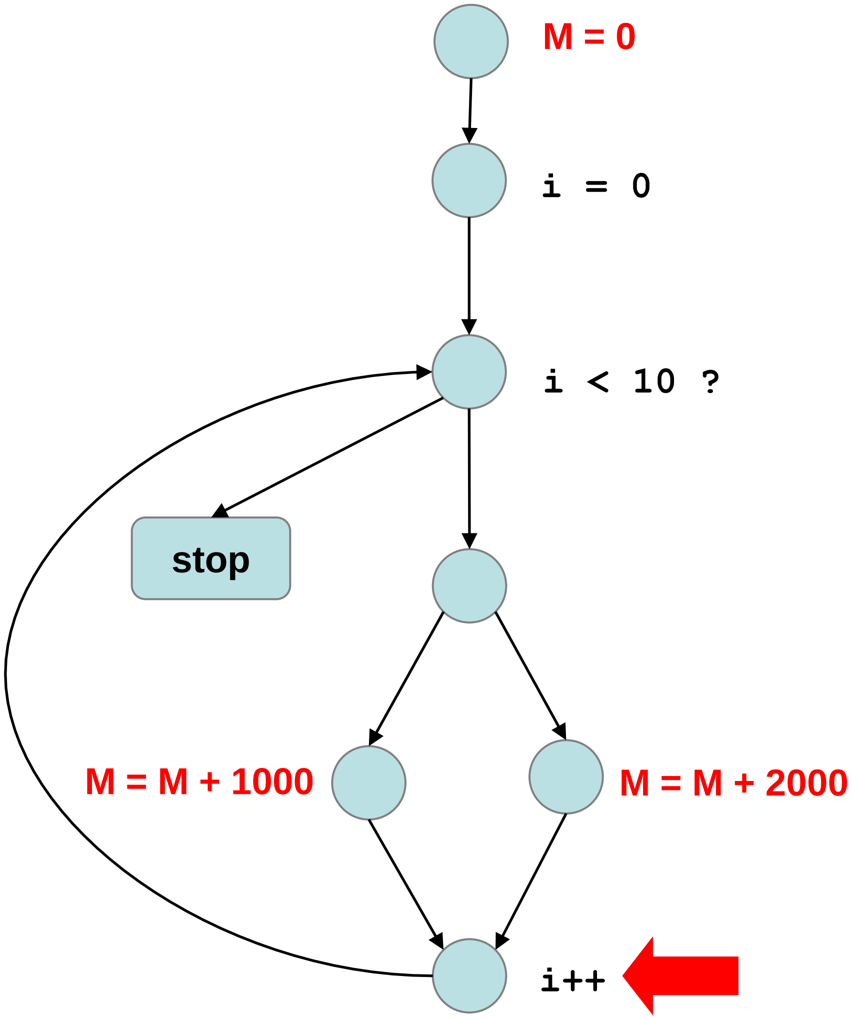
Analyzing a branching

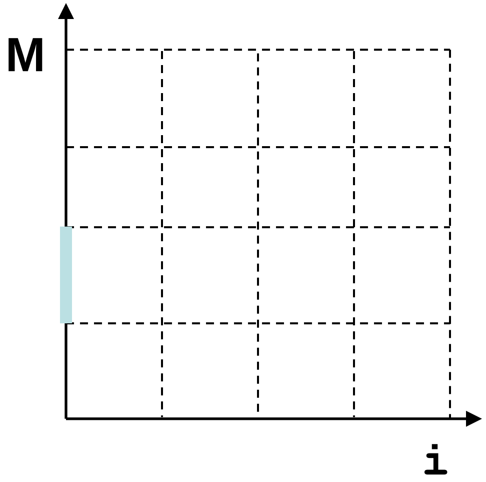




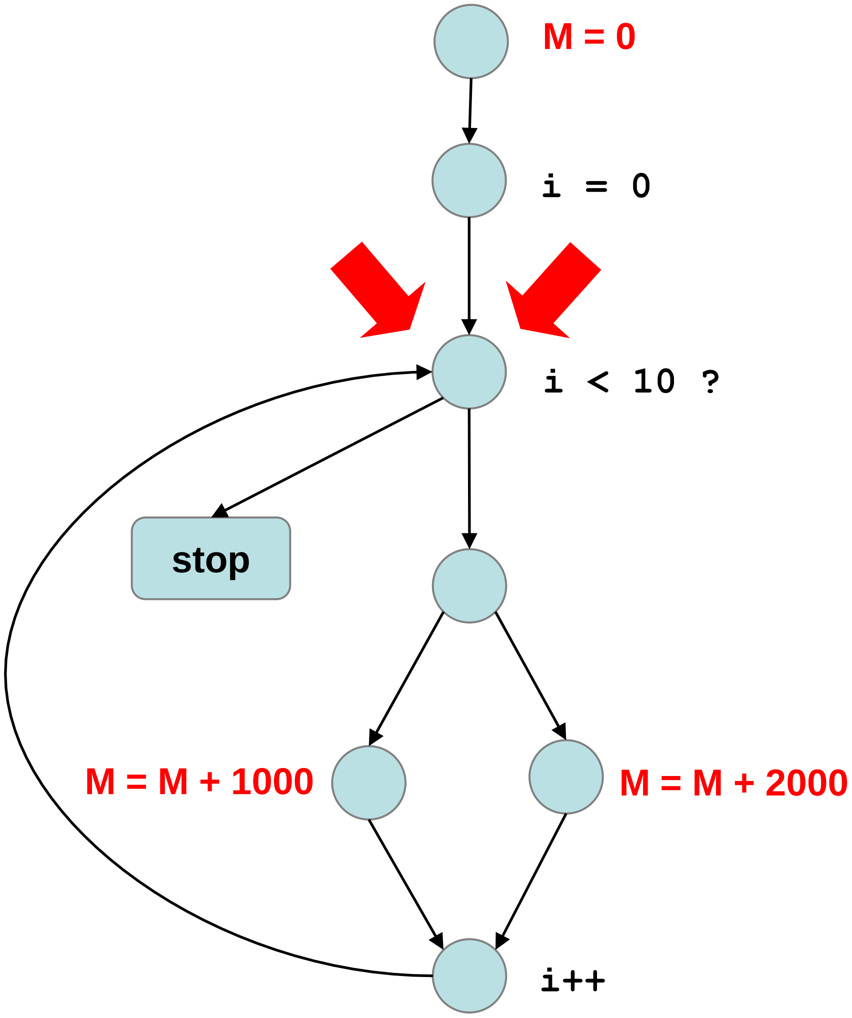


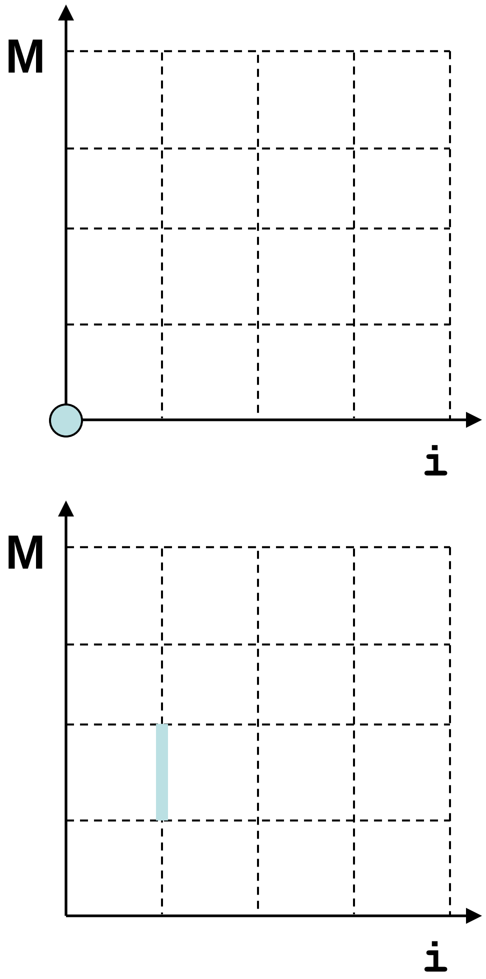
Convex hull



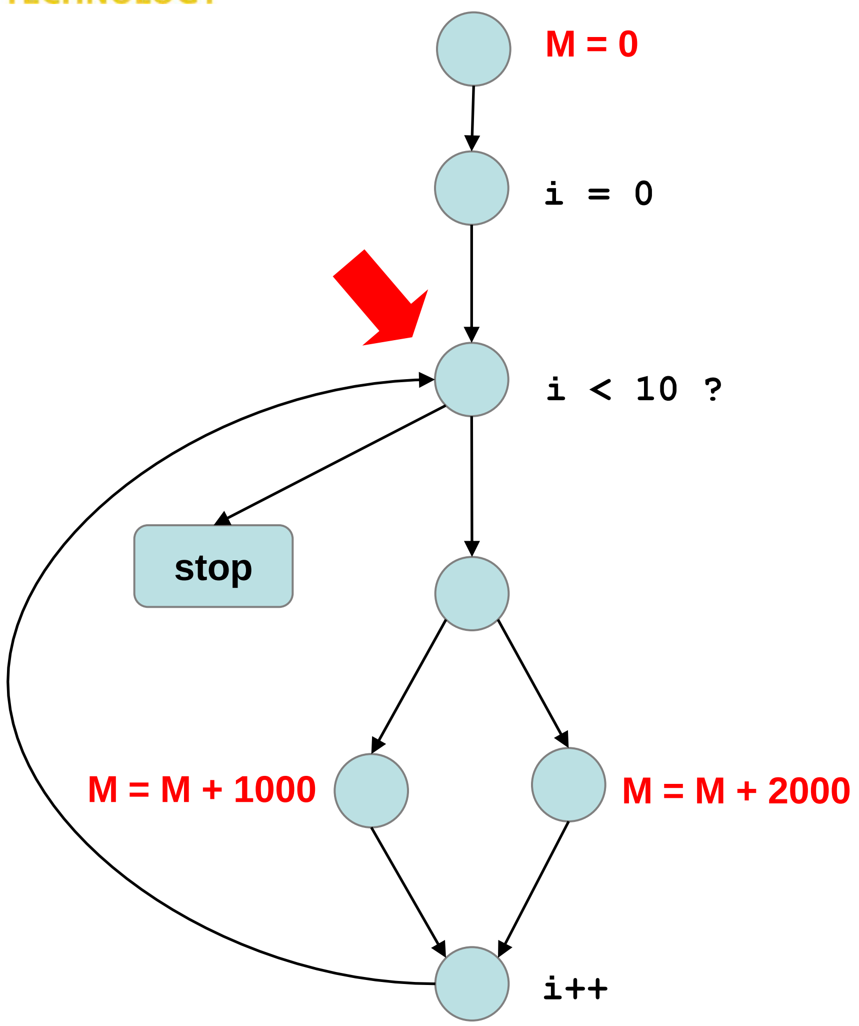


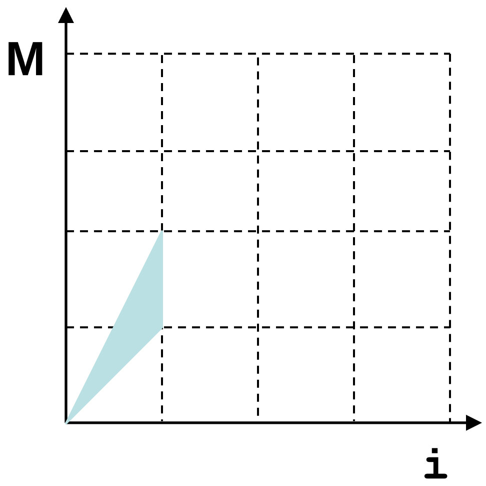
Iterating the loop analysis



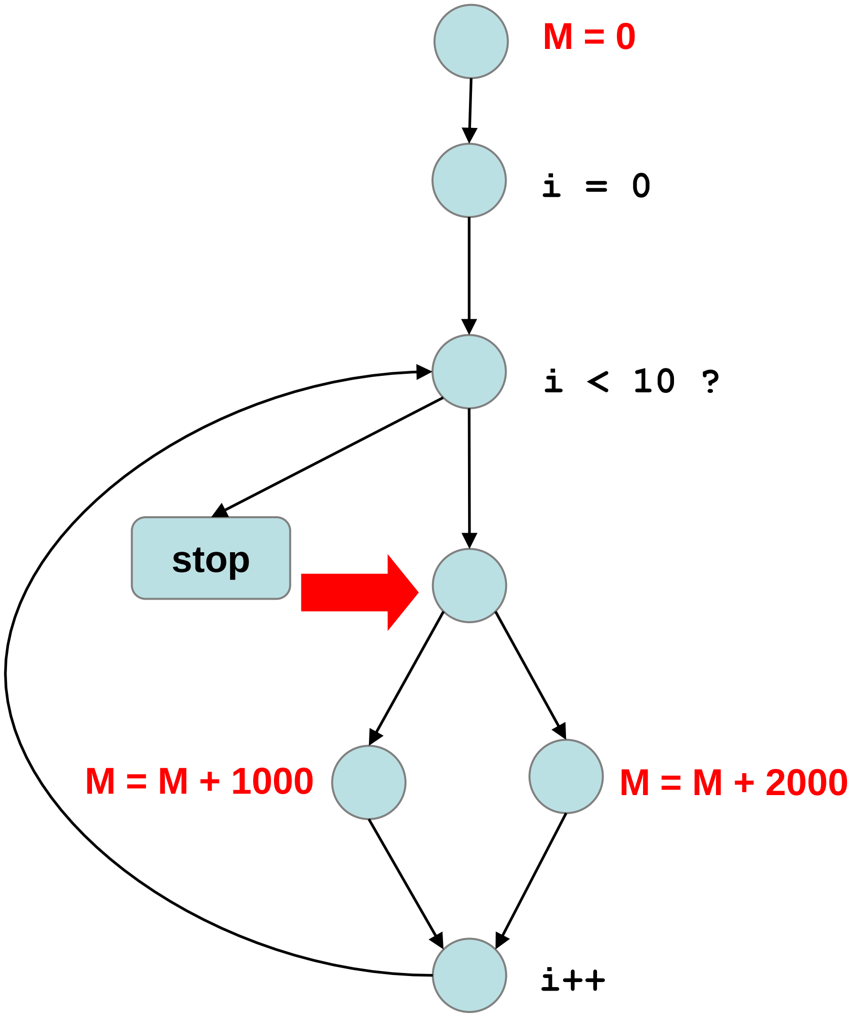


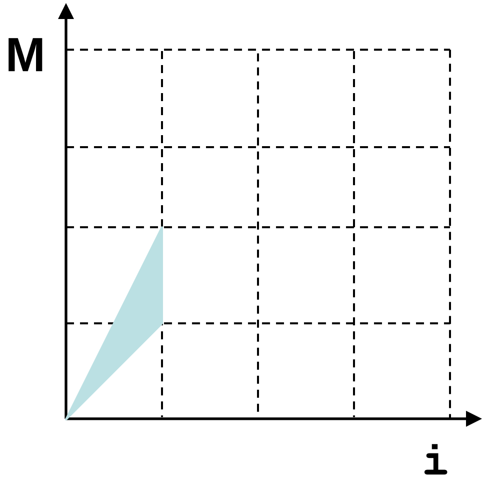
Building the loop invariant



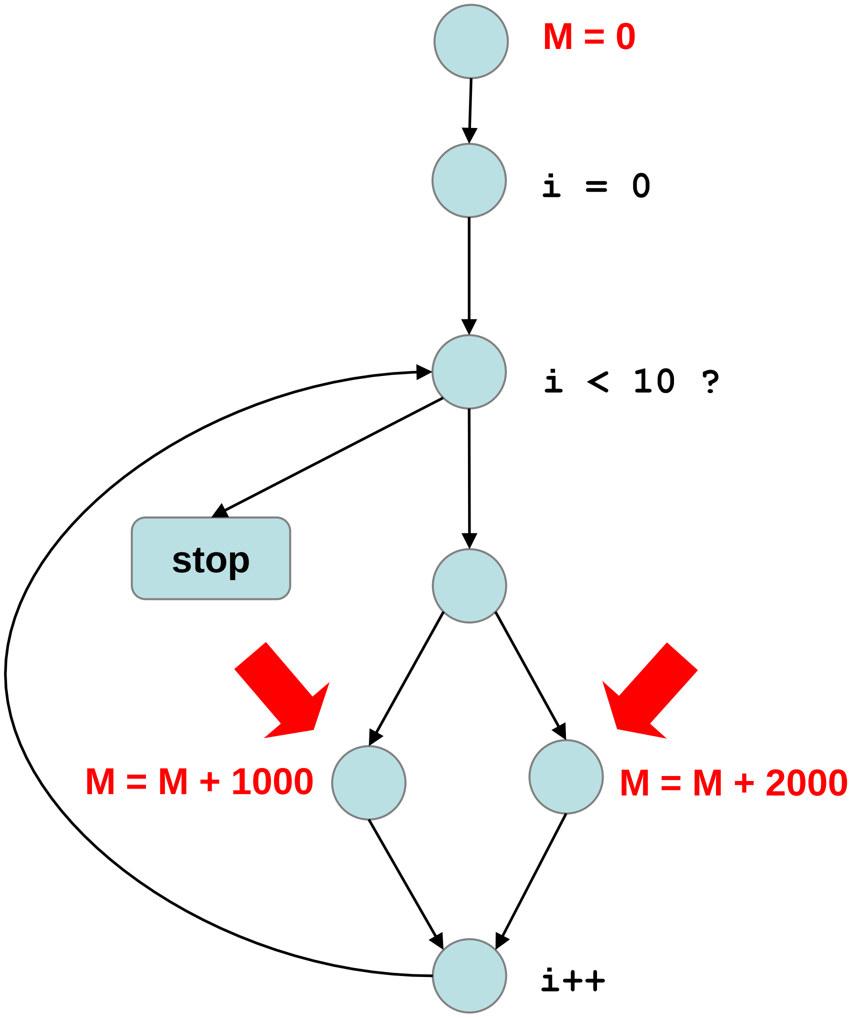


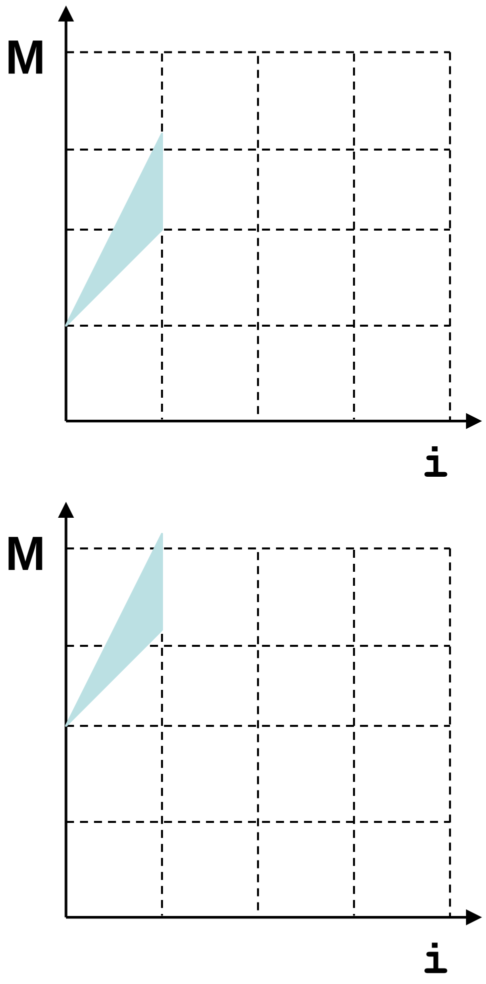
Analyzing a branching



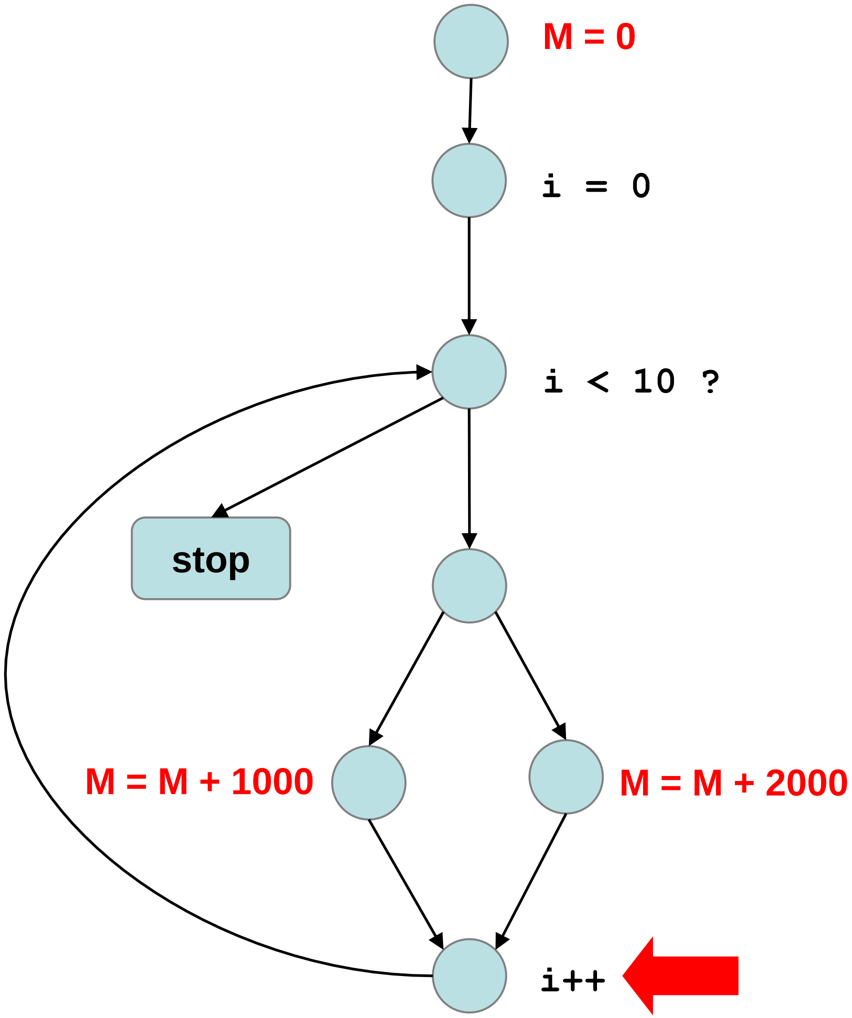


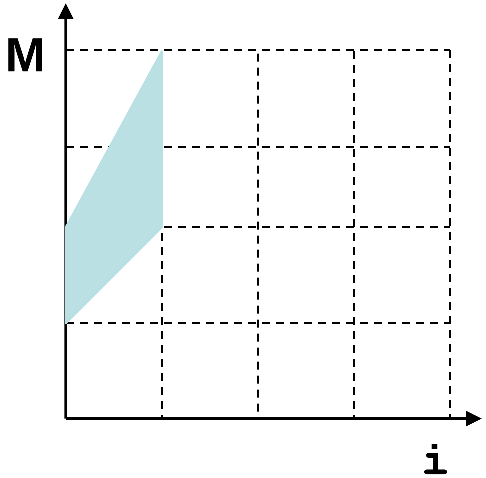
Analyzing a branching



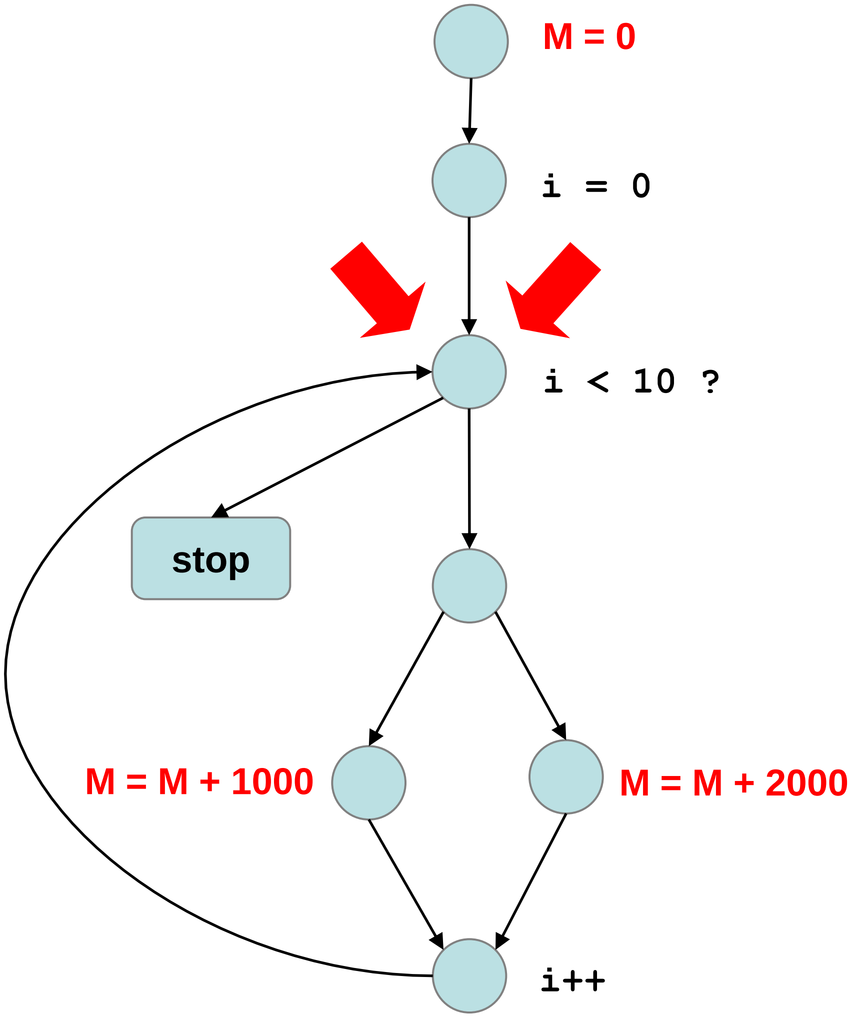


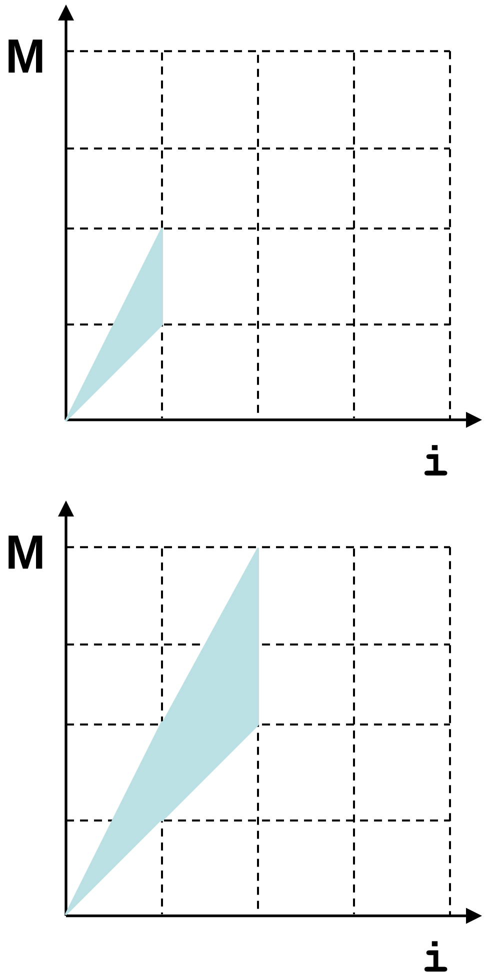
Convex hull



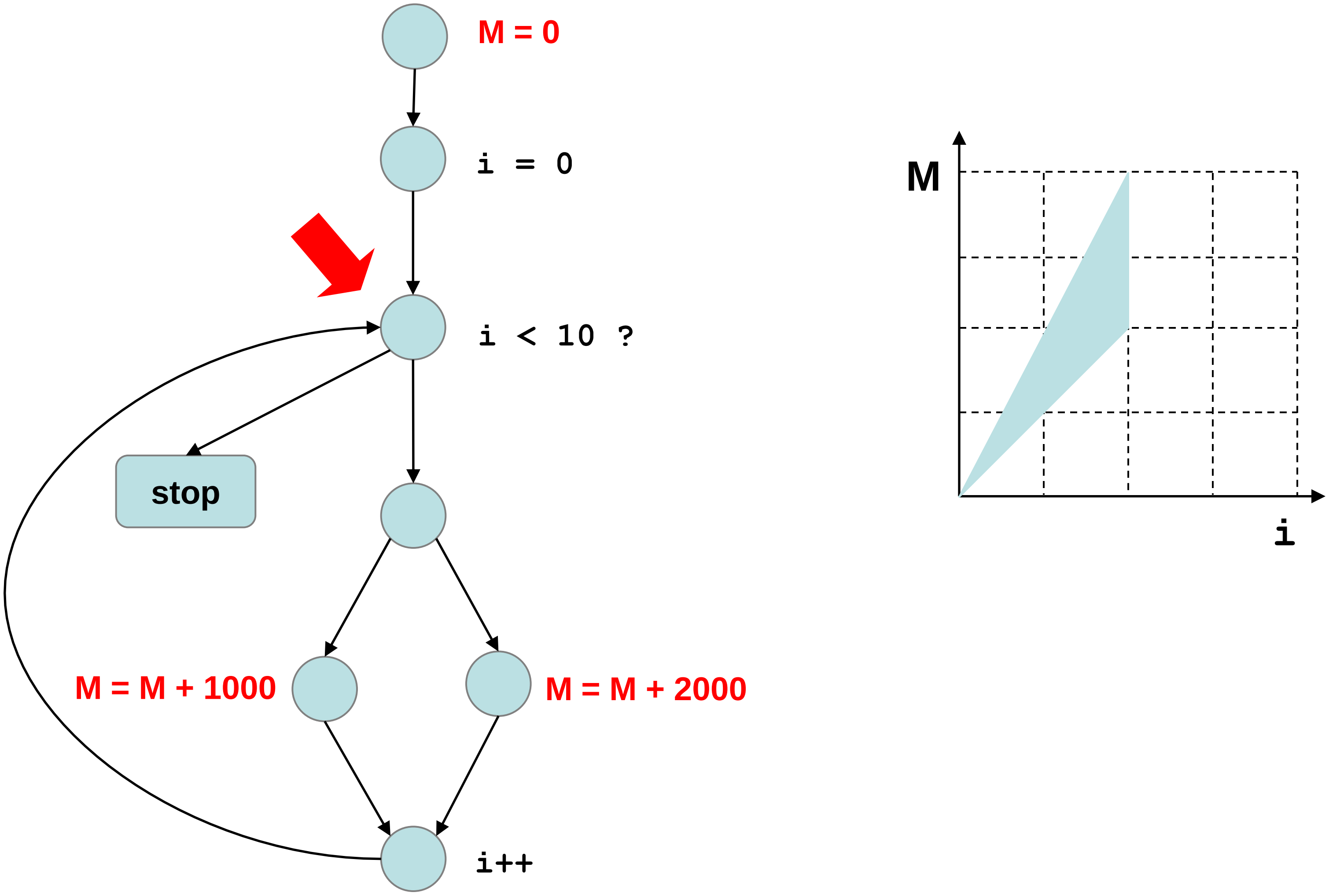


Building the loop invariant





Keeping iterating…



Passing to the limit

- We want this iterative process to end at some point
- We need to converge when analyzing loops
- After some iteration steps, we use a *widening* operation at loop entry to enforce convergence

Widening ∇

- Let a_1, a_2, …a_n, … be a sequence of polyhedra, then the sequence
  - $w_1 = a_1$
  - $w_{n+1} = w_n \nabla a_{n+1}$

  is ultimately stationary
- The widening is a *join* operation i.e.,

$$a \subseteq a \nabla b \ \& \ b \subseteq a \nabla b$$

Widening for intervals

- $[a, b] \nabla [c, d] =$

  [**if** c < a **then** -∞ **else** a, **if** b < d **then** +∞ **else** b]

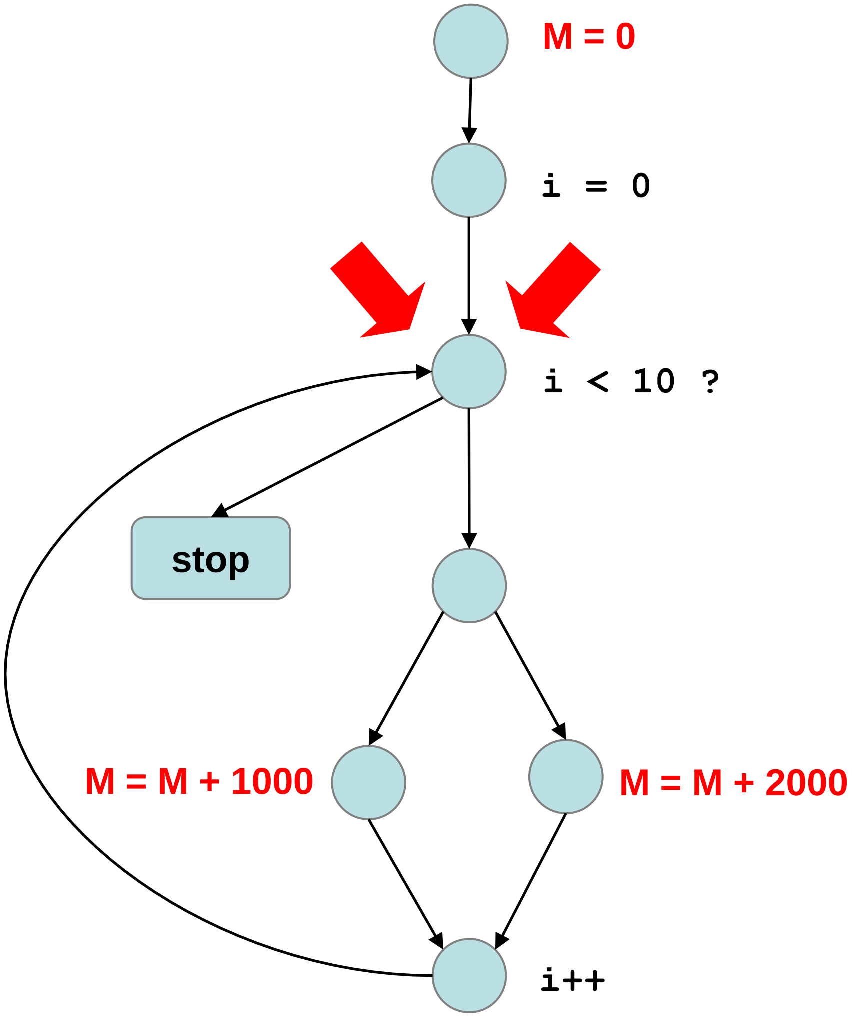
- Example:

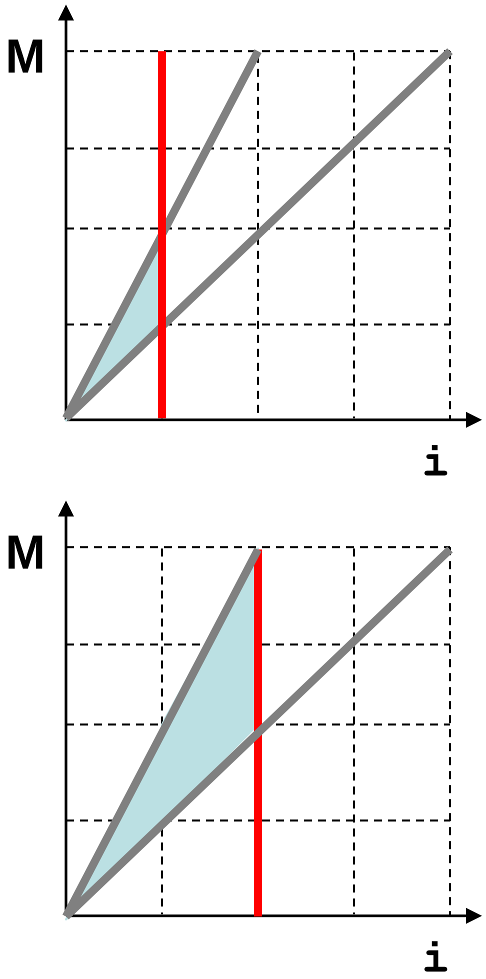
  $$[10, 20] \nabla [11, 30] = [10, +\infty]$$

Widening for polyhedra

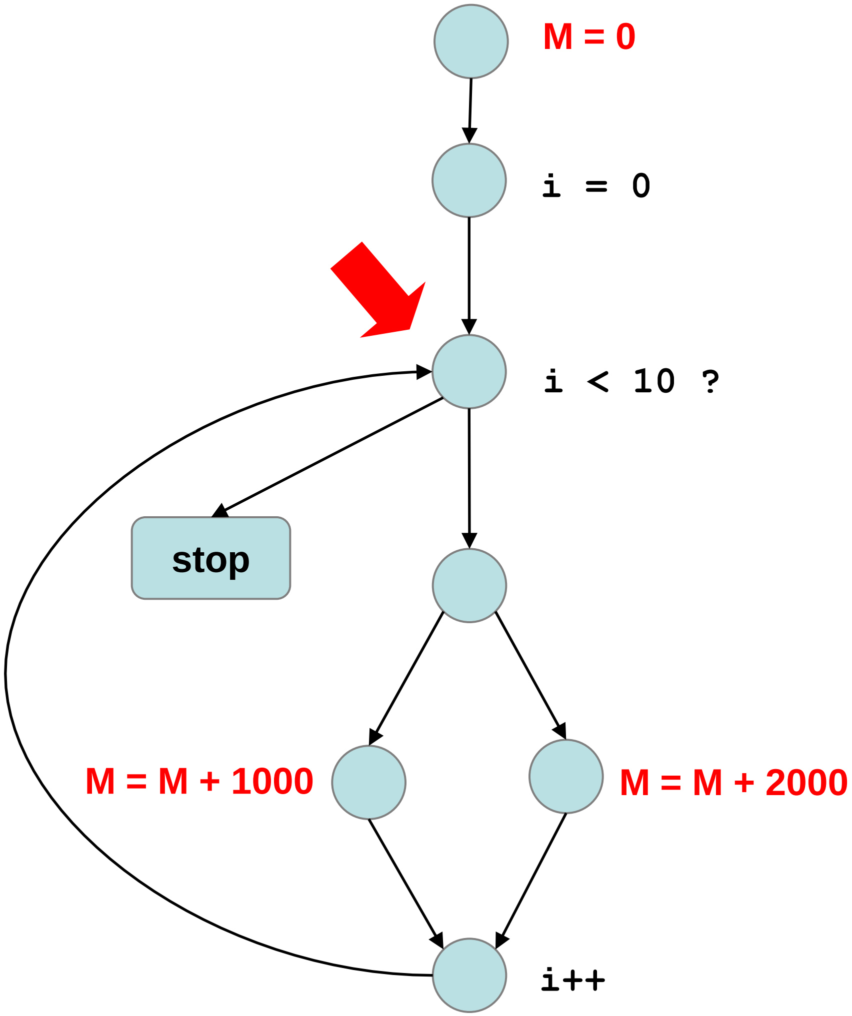
- We eliminate the faces of the computed convex envelope that are not stable
- Convergence is reached in at most N steps where N is the number of faces of the polyhedron at loop entry

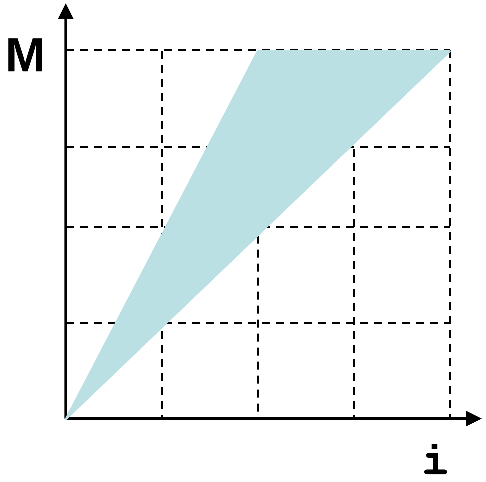
Widening





After widening





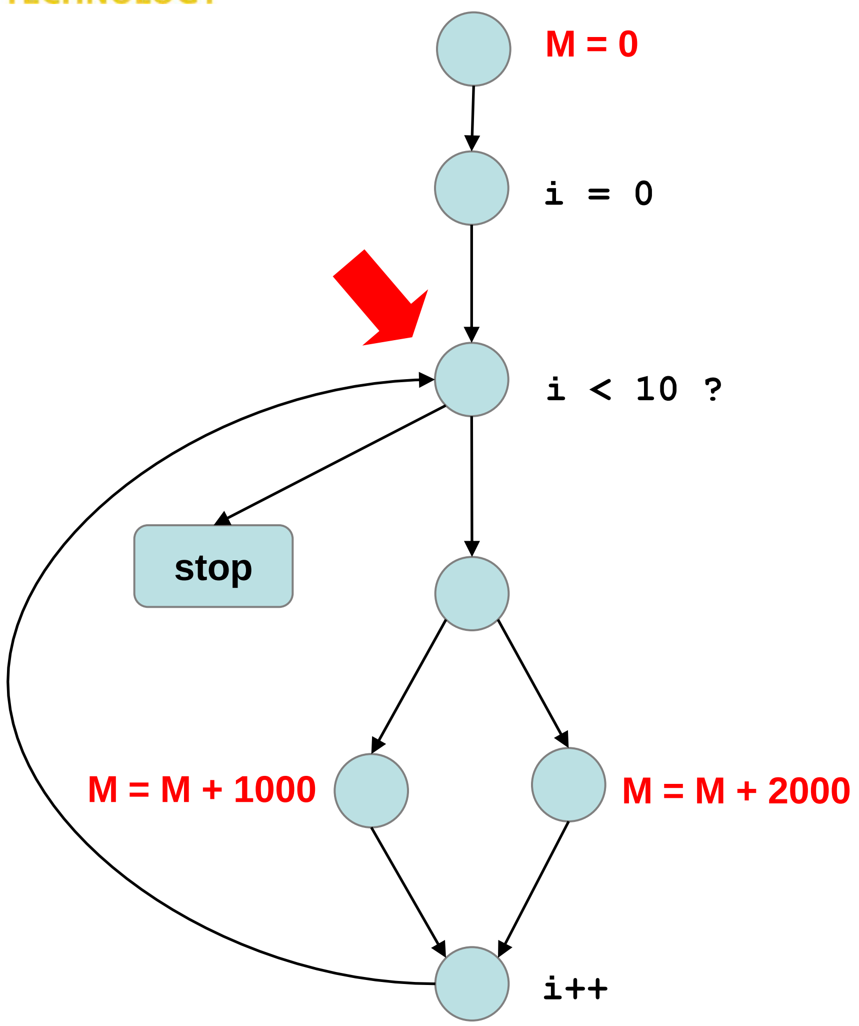
Detecting convergence

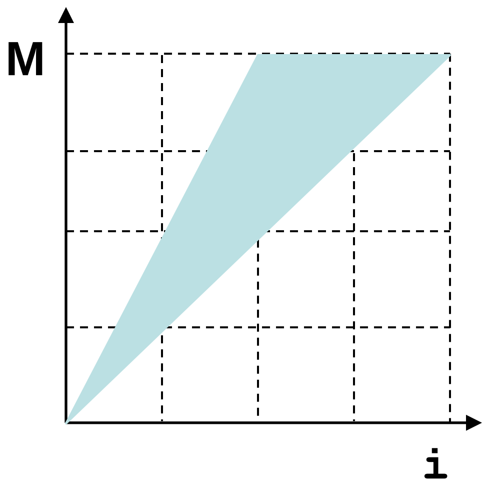
- Abstract iteration sequence
  - $F_1 = P$ (initial polyhedron)
  - $F_{n+1} = F_n$ if $\mathbf{S}(F_n) \subseteq F_n$

    $F_n \nabla \mathbf{S}(F_n)$ otherwise

  where **S** is the semantic transformer associated to the loop body
- **<u>Theorem:</u>** if there exists N such that $F_{N+1} \subseteq F_N$, then $F_n = F_N$ for $n > N$.

Convergence





The computation has converged

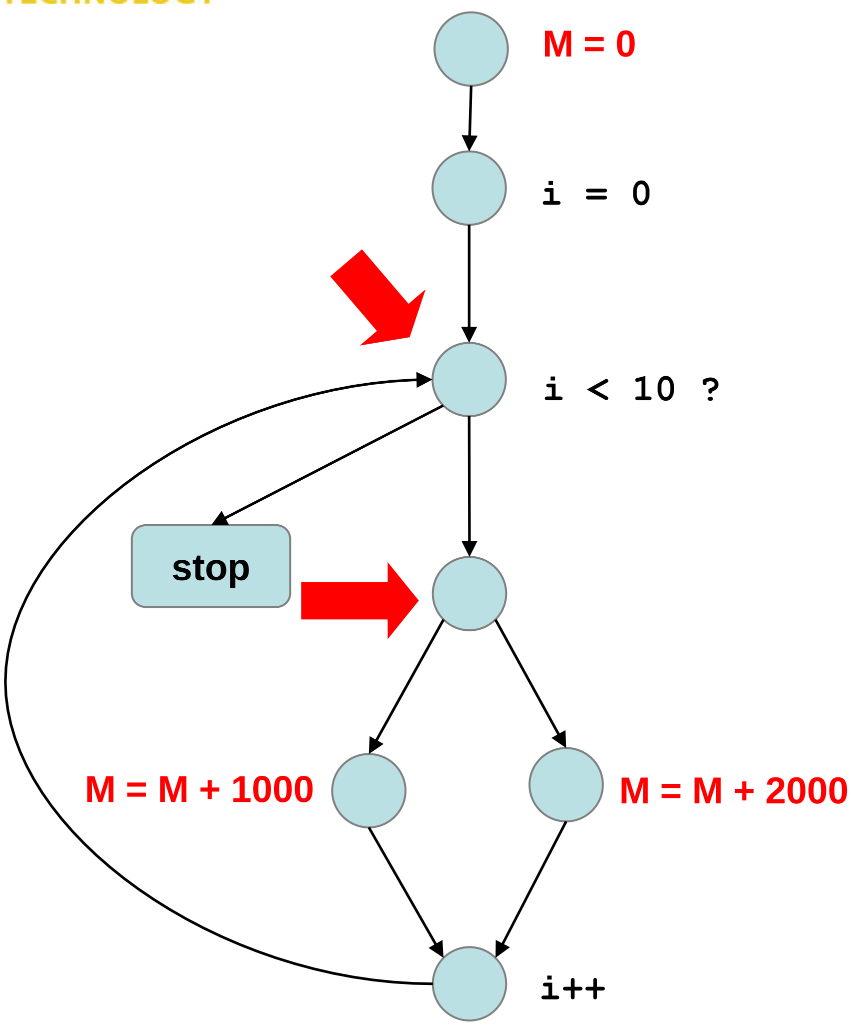
We are not done yet…

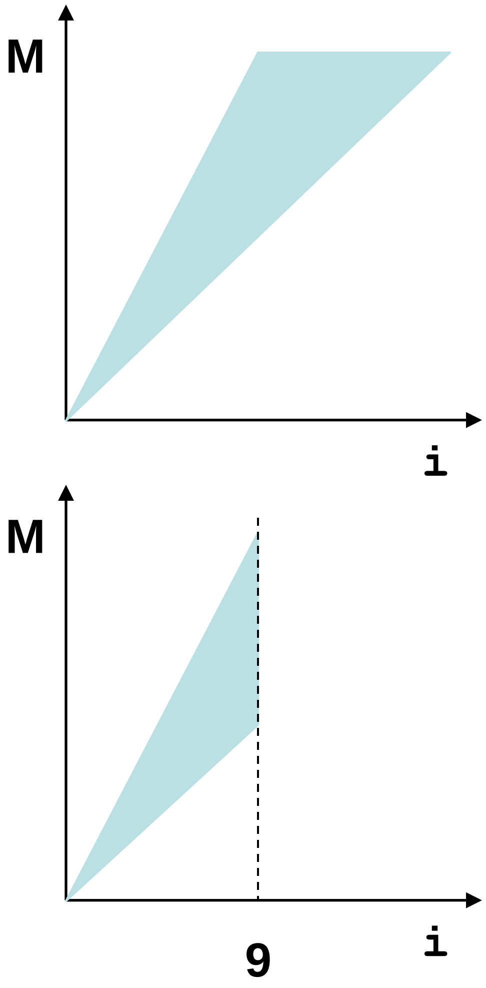
- The analyzer has just proven that

$$1000 * \mathtt{i} \le M \le 2000 * \mathtt{i}$$

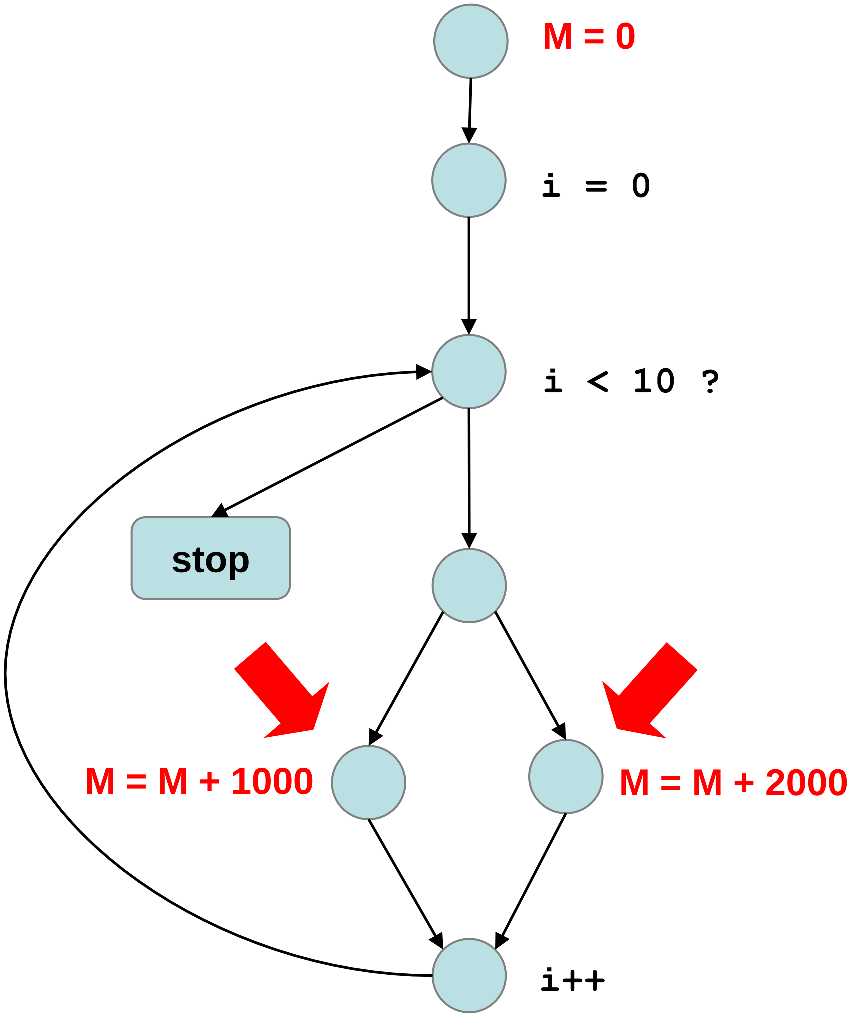
- But we have lost all information about the termination condition $0 \le \mathtt{i} \le 10$
- Since we have obtained an envelope of all possible values of the variables, if we run the computation again we still get such an envelope
- The point is that this new envelope can be smaller
- This refinement step is called *narrowing*

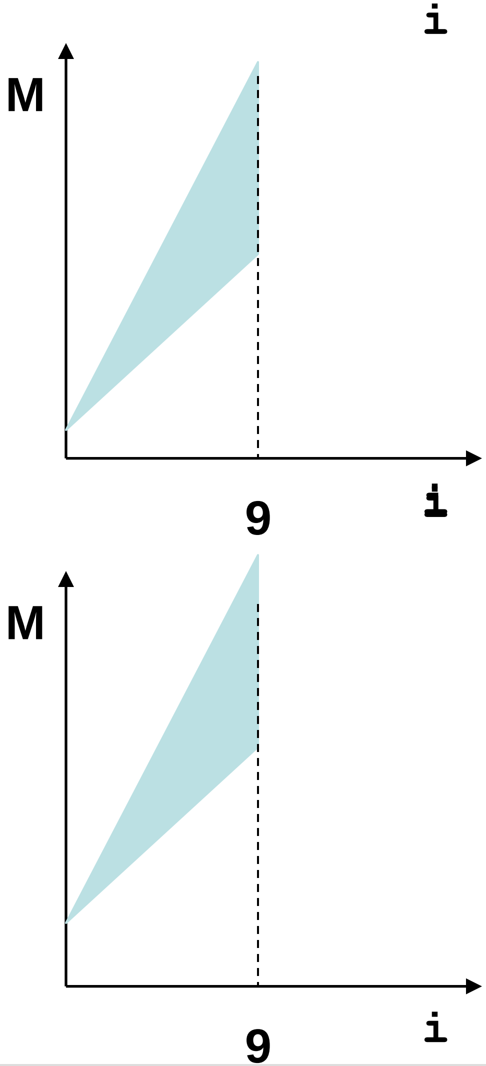
Refinement



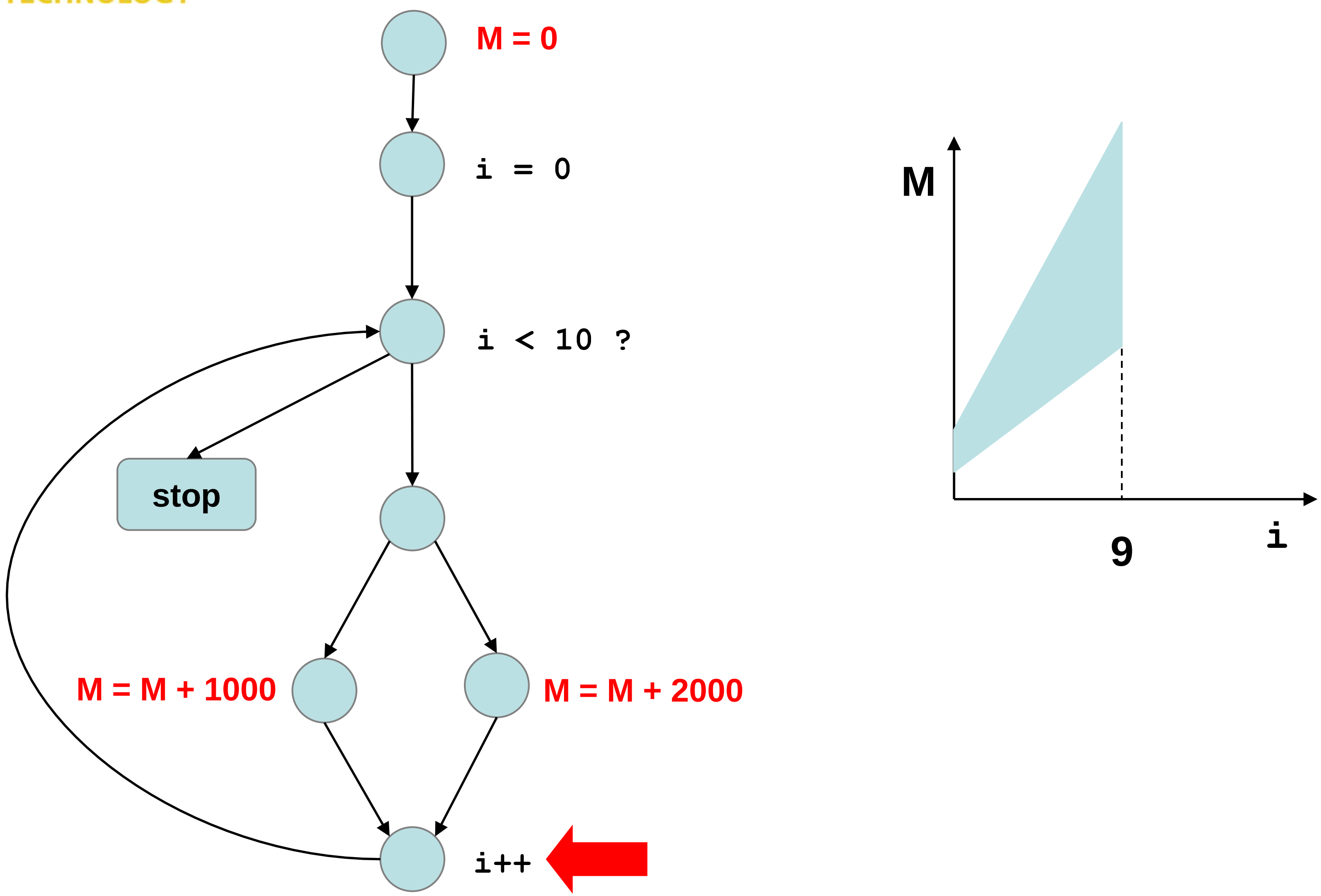


Analyzing a branching

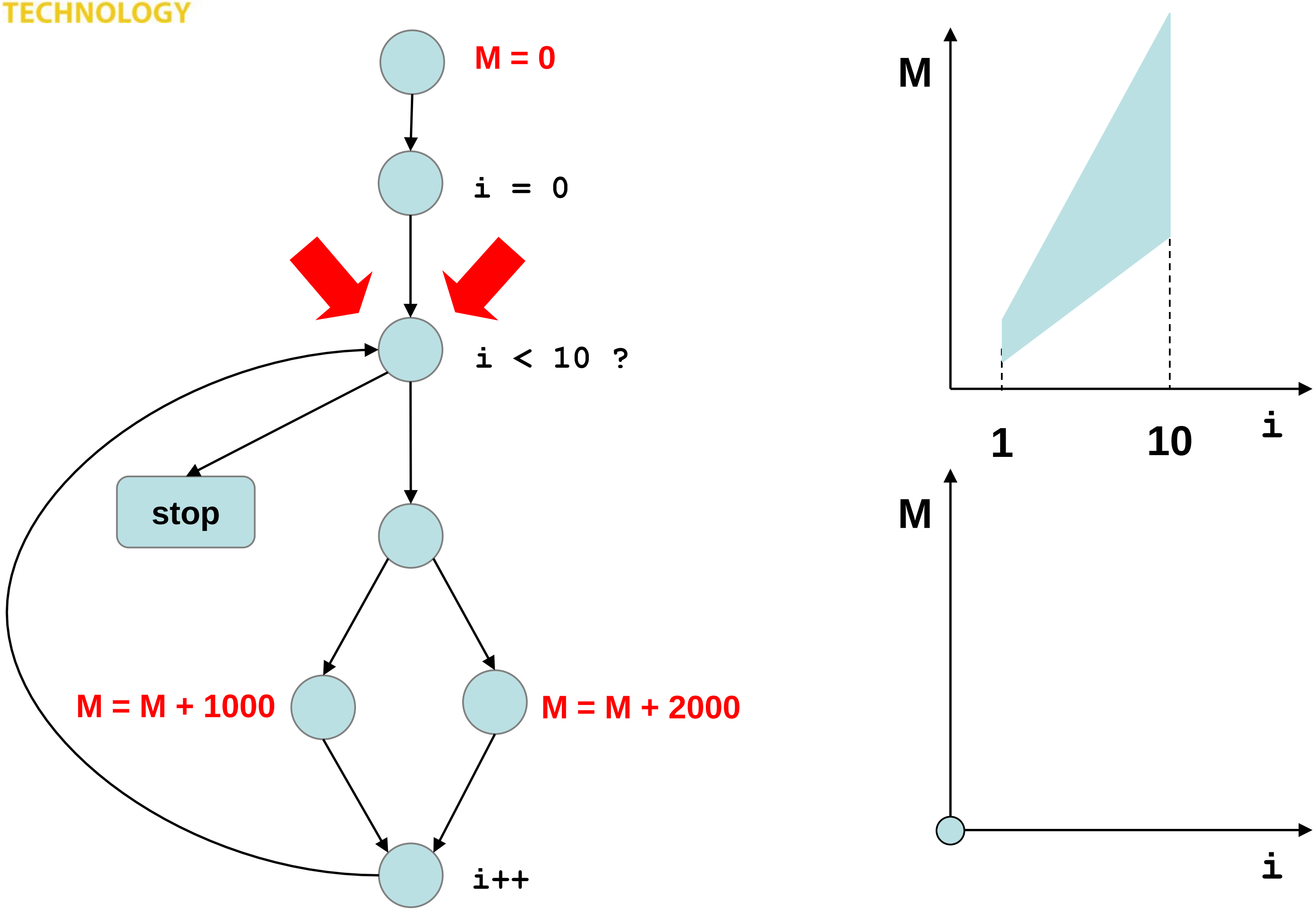




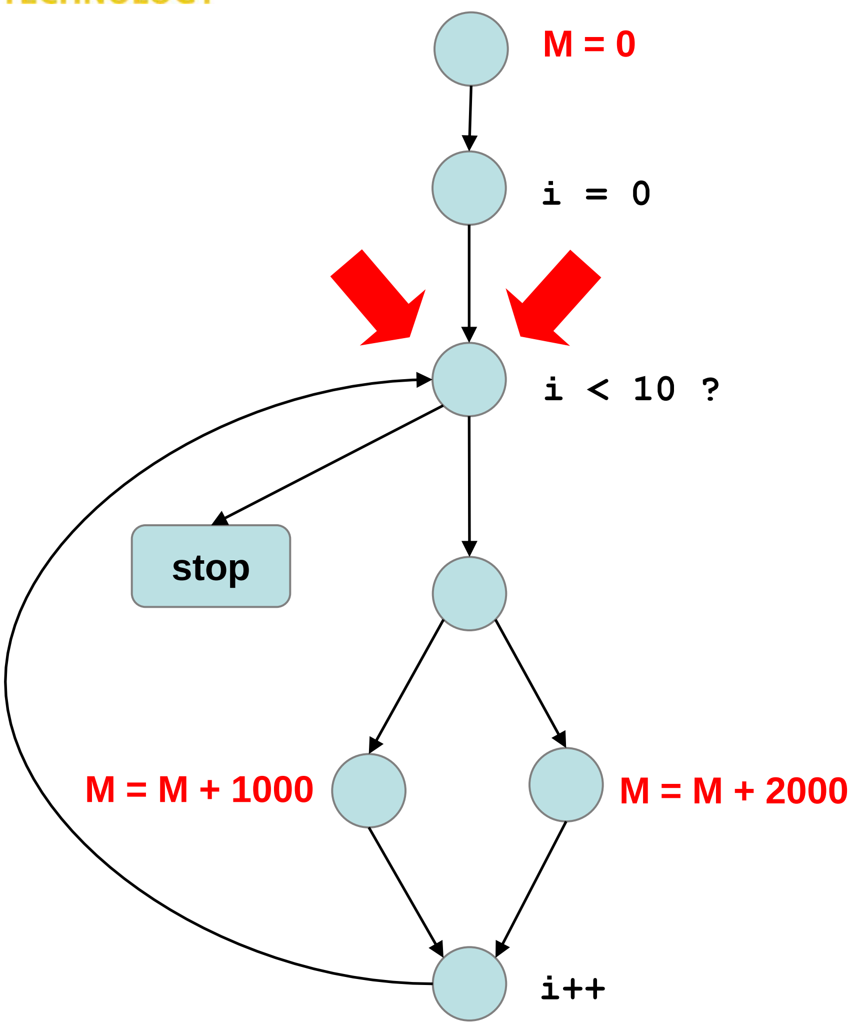
Convex hull

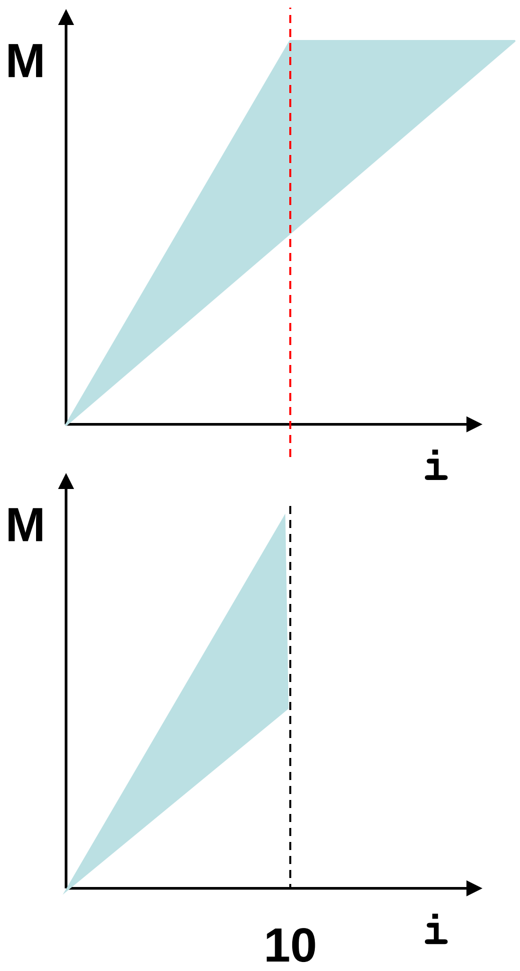


Back to loop entry

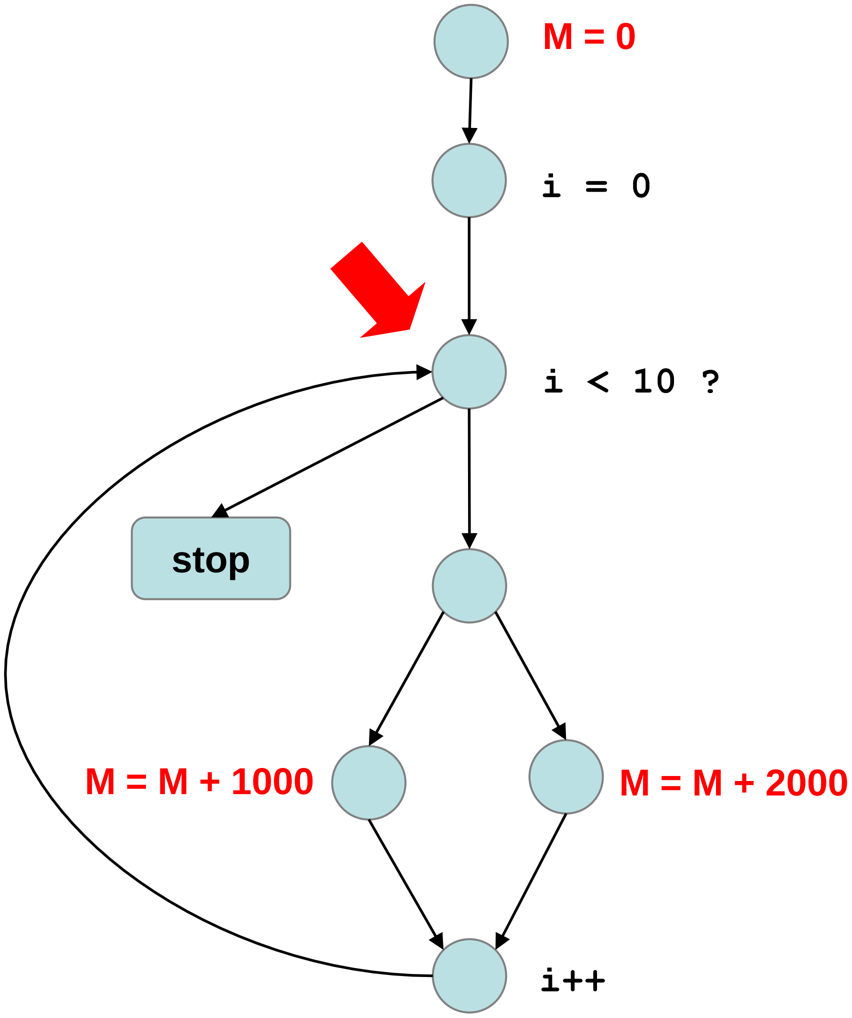


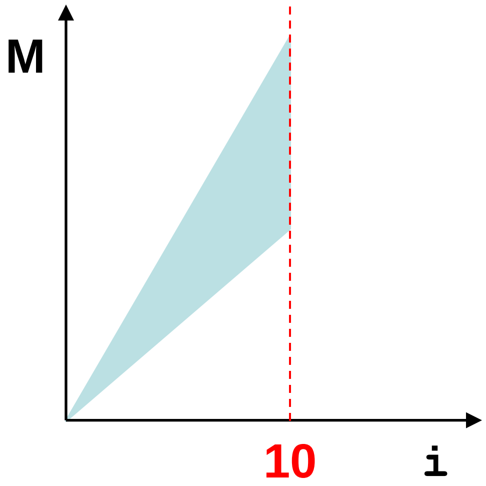
Narrowing



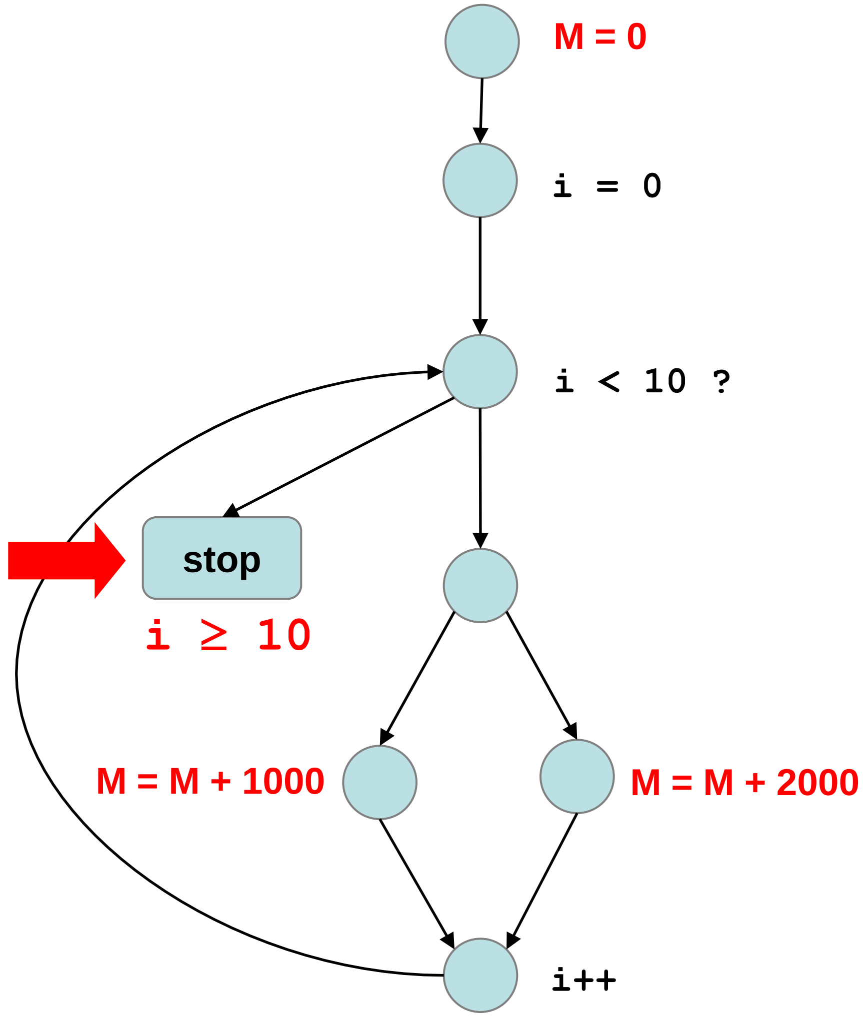


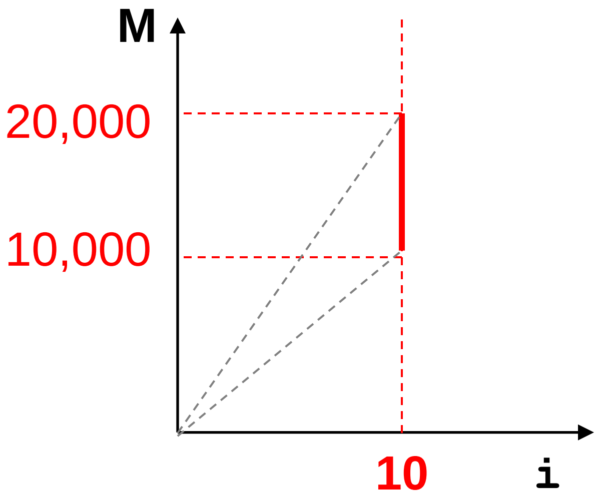
Refined loop invariant





Invariant at loop exit





Interpretation of the results

Space allocated in the buffer

Buffer size: 5,000 10,000 15,000 20,000 25,000

Certain buffer overflow

Possible buffer overflow

No buffer overflow

Achievements and challenges

Assured static analyzers for NASA

- **C Global Surveyor**: verified array bound compliance for NASA mission-critical software
  - Mars Exploration Rovers: 550K LOC
  - Deep Space 1: 280K LOC
  - Mars Path Finder: 140K LOC
- **Pointer analysis**:
  - International Space Station payload software (major bug found)

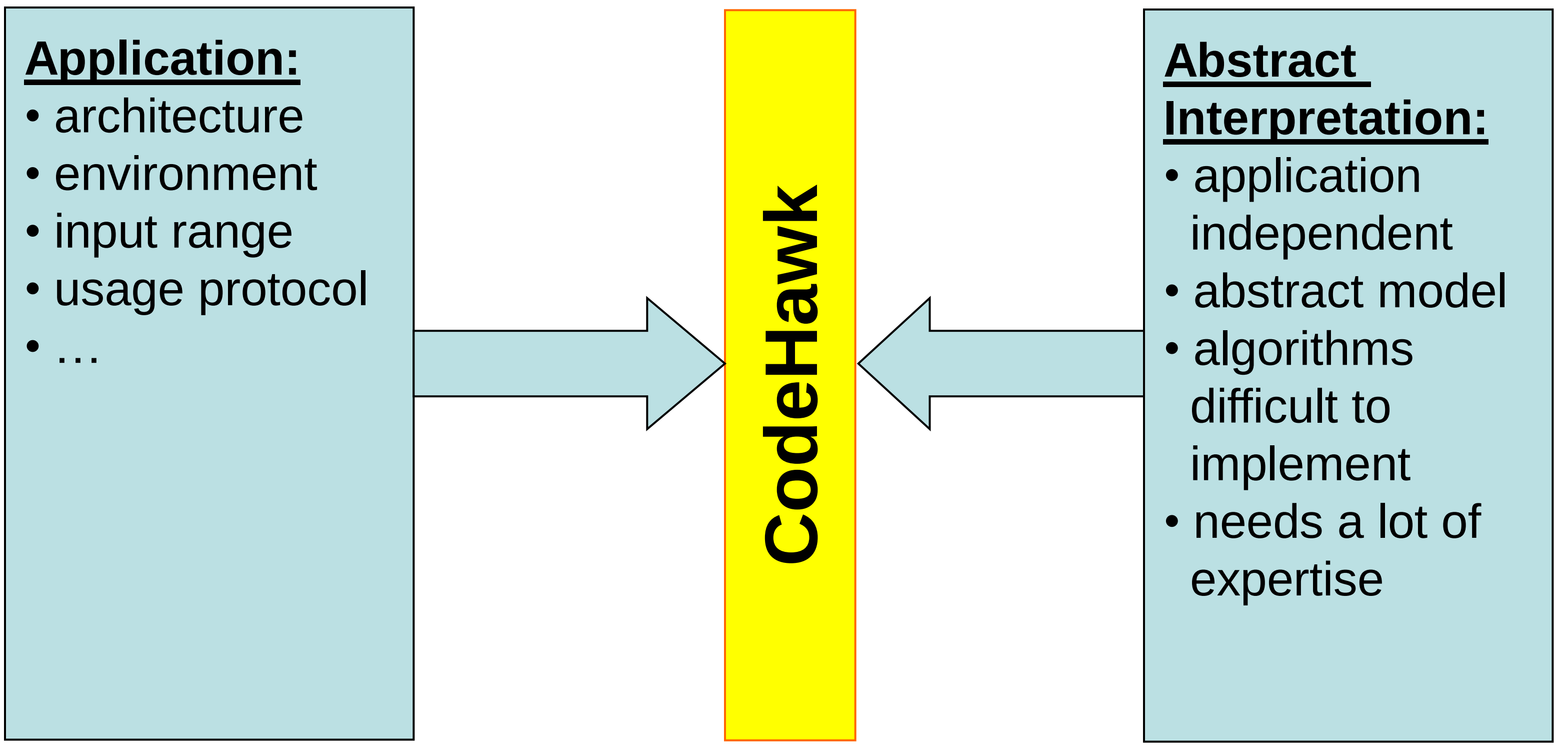
What we observe

- No scalable and precise general-purpose abstract interpreter
- PolySpace:
  - Handles all kinds of runtime errors
  - Decent precision (<20% false positives)
  - Doesn't scale (topped out at ≅ 40K LOC)
- Customization is the key
  - Specialized for a property or a class of applications
  - Manually crafted by experts

CodeHawk™

- Abstract Interpretation development platform/ static analyzer generator
- Automated generation of customized static analyzers
  - Leverage from pre-built analyzers
  - Directly tunable by the end-user

Where CodeHawk stands



Recent Applications

- **Malware detection**
  - Customized analyzers for specific kinds of malware
  - Naturally resistant to complex obfuscating transformations
  - Evaluated on NSA test case
- **Library/Component analysis**
  - Proof of absence of buffer overflow in OpenSSH's dynamic buffer library
- **Shared variables**
  - Protection policies for shared variables
  - Evaluated on a Lockheed Martin/Maritime code

Conclusions

- Promising and proven technology
  - key distinction for assurance: no false negatives
  - can verify application properties as well as detect defects
  - can be tailored for various domains (e.g., malware)
- Not a silver bullet
  - bullet generator; but each modeled domain offers leverage
  - required expertise still high outside of turnkey libraries