CS 357 D

Fundamental Principles and Techniques in
Program Analysis

Zohar Manna and Henny Sipma

Spring 2007

Gates 498

TTh 1:15 - 2:30

Organizational Matters

- **Instructors:**
  - Henny Sipma, sipma@cs.stanford.edu
  - Zohar Manna, manna@cs.stanford.edu
- **Lectures:**
  - Mostly based on course notes
  - Some guest lectures
- **Handouts:**
  - Copies of slides
  - Course notes
  - Research papers

Organizational Matters

- **Grading:**
  - No homeworks
  - No exams
  - Letter grade: project
  - Pass/no credit: attendance
- **Project options:**
  - 1-hour lecture in class on related topic
  - survey paper on related topic
  - implementation of program analysis method
  - implementation of decision procedure

Organizational Matters

- **Textbooks (optional)**
  - Zohar Manna, Amir Pnueli, Temporal Verification of Reactive Systems. Safety. Springer-Verlag, 1995.
  - Flemming Nielson, Hanne Nielson, Chris Hankin, Principles of Program Analysis, Springer-Verlag, 1999.
  - B.A. Davey, H.A. Priestley, Introduction to Lattices and Order, Cambridge University Press, 2nd ed, 2002.

Course goal

Provide good understanding of some of the fundamental principles and techniques of program analysis:

- abstract interpretation
- propagation-based analysis methods
- constraint-based analysis methods

- shape analysis
- separation logic

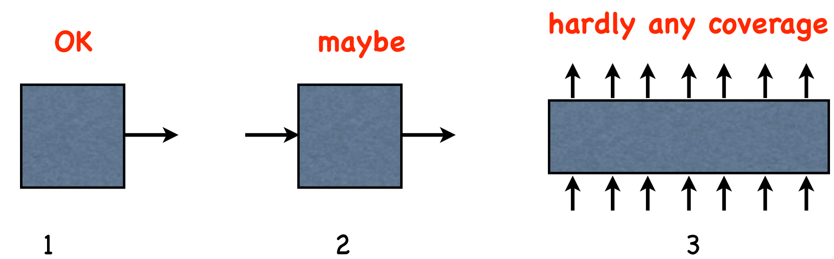
- runtime analysis

- decision procedures
- combination of decision procedures

Static program analysis: Why?

- Gain insight in program behavior based on program text
- Why not run it?

1. Fully deterministic (no input): just run it
2. Fully deterministic (with inputs): run it on different inputs
3. Concurrent program with continuing inputs: run it in different environments



Static Program Analysis: Why?

The next few decades will see a rapid growth in our software infrastructure, so that eventually we will come to rely on software in almost every interaction with our environment. Transportation, energy distribution, communications, banking, and health care will all depend on software. For end-user applications, time to market and feature count may continue to be driving forces but, in the development of our infrastructure, 'getting it right' will matter again. (from [2])

Static checking can improve software productivity because the cost of correcting an error is reduced if it is detected early (from [1])

Static Program Analysis: What?

- **Full program verification?**

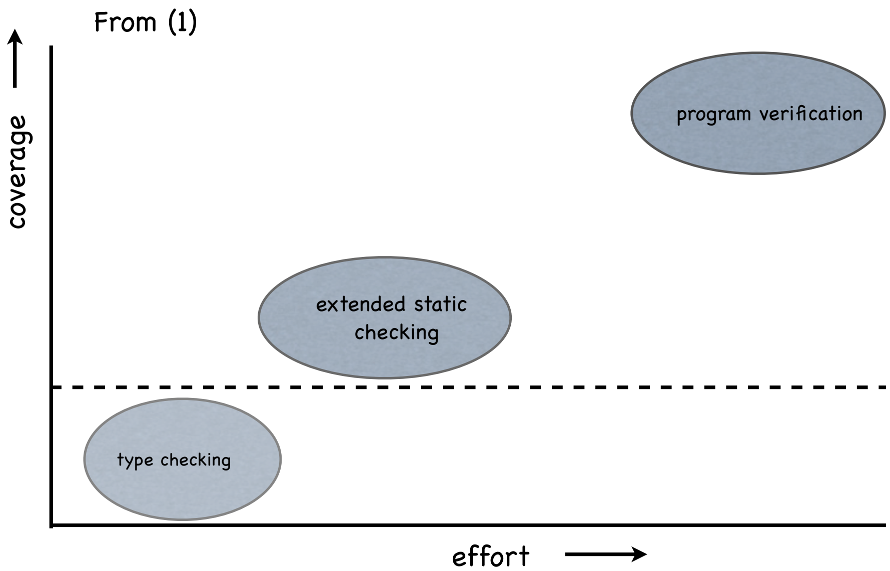
$$P \models \varphi$$

All program behaviors satisfy temporal specification φ

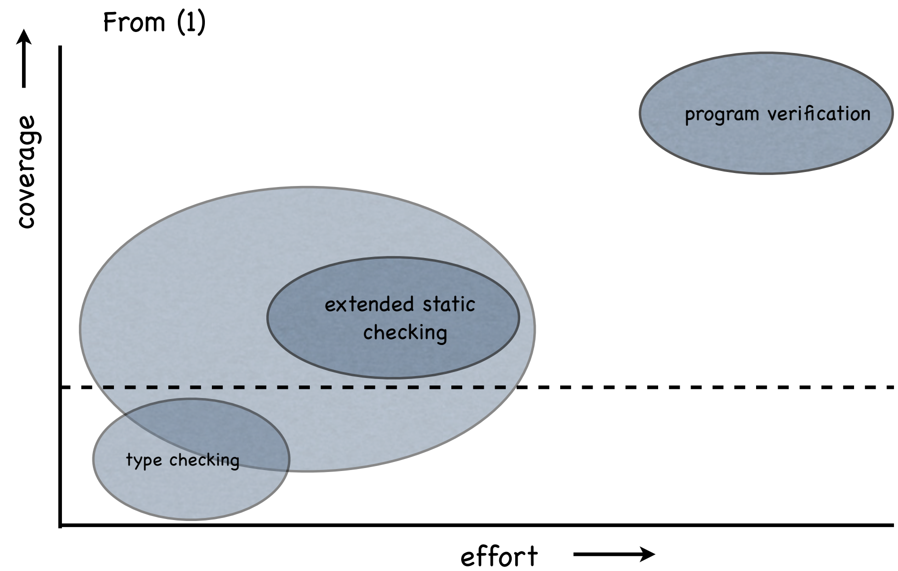
(CS 256)

- **What is the specification?**
- **Usually too hard**

Static Program Analysis: What?



Static Program Analysis: What?



Static Program Analysis: What?

- **Answer questions about program behaviors:**
  - does the program always terminate?
  - does the program ever reach this (bad) state?
  - what is the range of values of this variable at this location?
  - is there a possibility of out-of-bound array access?
  - is there a possibility of division by zero?
  - do these variables point at the same location in the heap?
  - what is the maximum amount of memory required?
  - synchronization errors (deadlocks, data races)?

Problem: undecidability

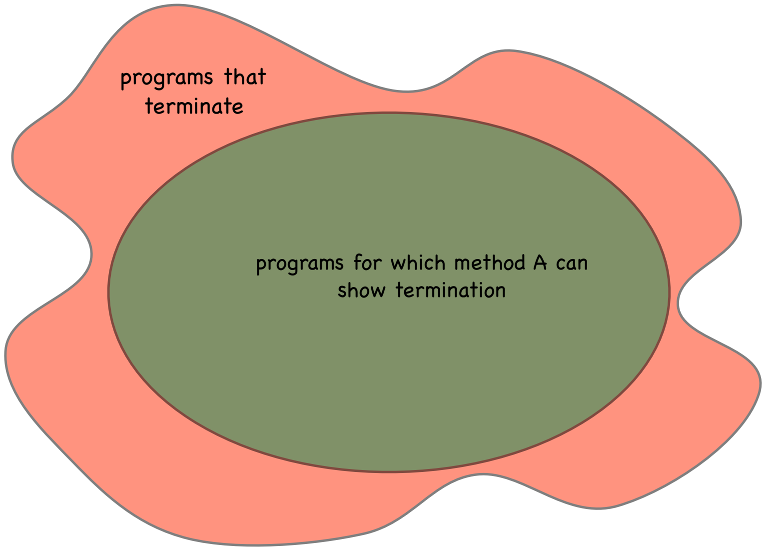
Rice's Theorem (1953)

Any nontrivial property about the language recognized by a Turing machine is undecidable.

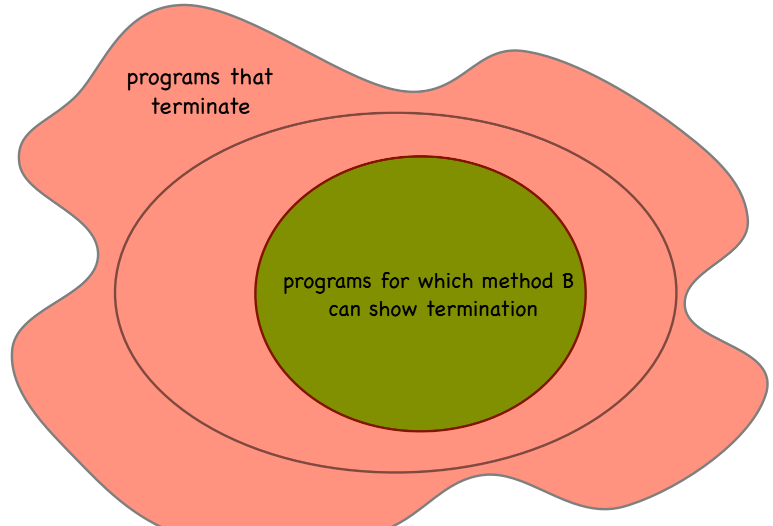
Informally:

Any interesting program property is undecidable

Problem: undecidability

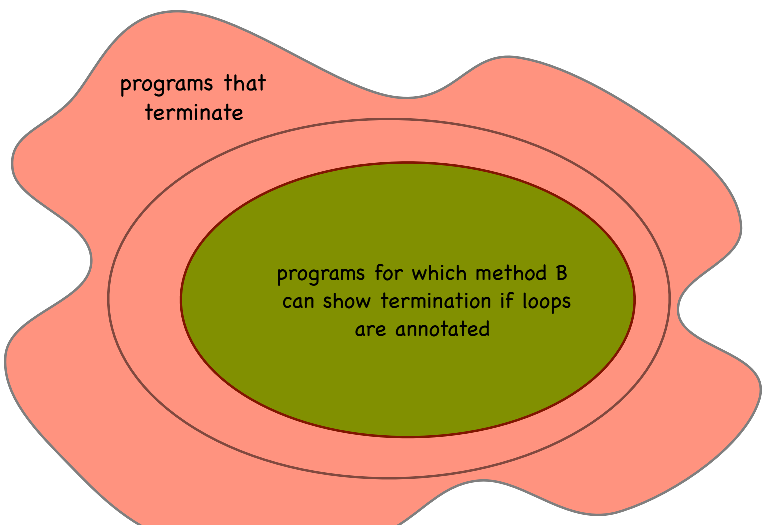


Problem: undecidability



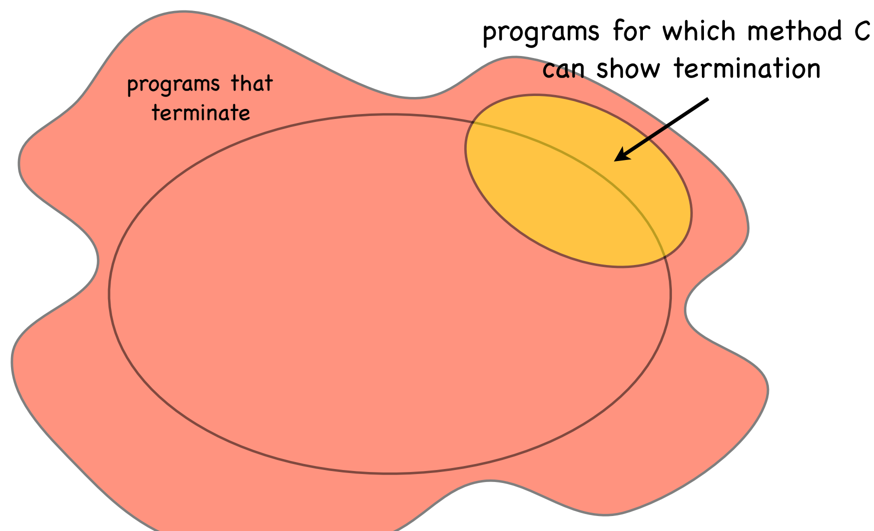
Method B is weaker than method A

Problem: undecidability



Method B can make use of user-supplied annotations

Problem: undecidability



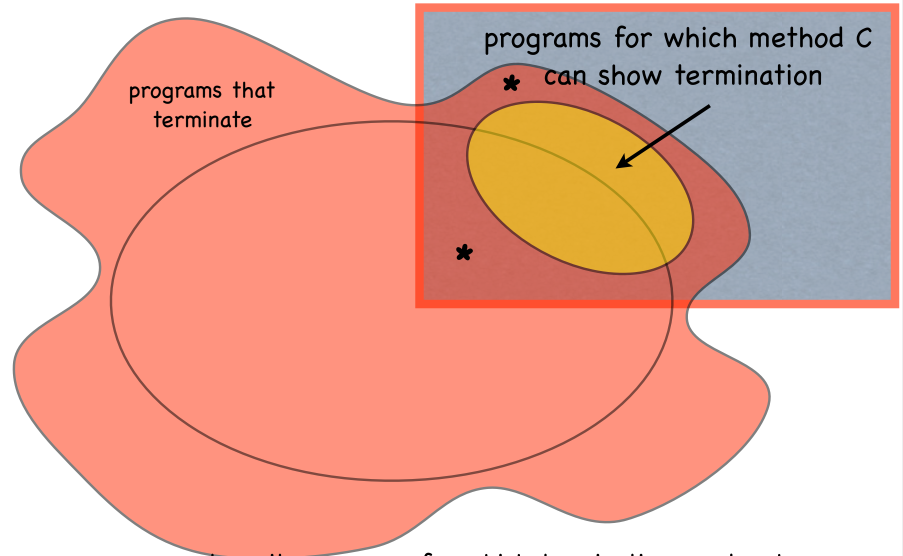
Method C is specialized for a particular class of programs of interest

Soundness and Completeness

- The ideal static checker is
  - **sound**: if the program has an error, the checker will report it
  - **complete:** if the checker reports an error, it is a genuine error

- Most practical static checkers are neither

Problem: undecidability



Force programmers to write programs for which termination can be shown

Soundness and Completeness

- The ideal static checker is
  - **sound**: if the program has an error, the checker will report it
  - **complete:** if the checker reports an error, it is a genuine error

- Most practical static checkers are neither
- The real issue is: **accuracy**

Problem: Complexity

In general, accuracy is expensive

- exponential in the size of the program
- exponential in the number of variables

Most of the research in program analysis is focused on this trade-off between performance and precision

Real Problem: Lack of formal semantics

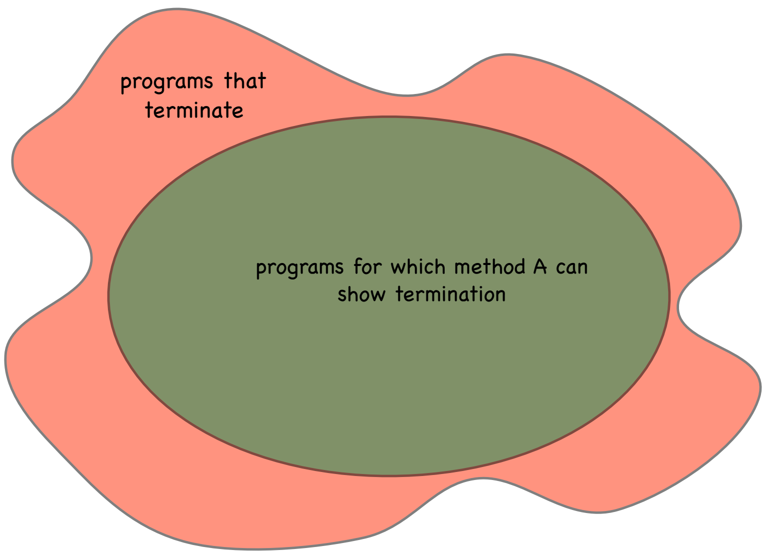
Goal: obtain information about all program behaviors

- (CS 256) SPL : Simple Programming Language
  - First-order model: well-defined semantics in terms of transition systems and program behaviors as sequences of states
- (Real life) C++ program
  - Semantics of the program is defined by the compiler

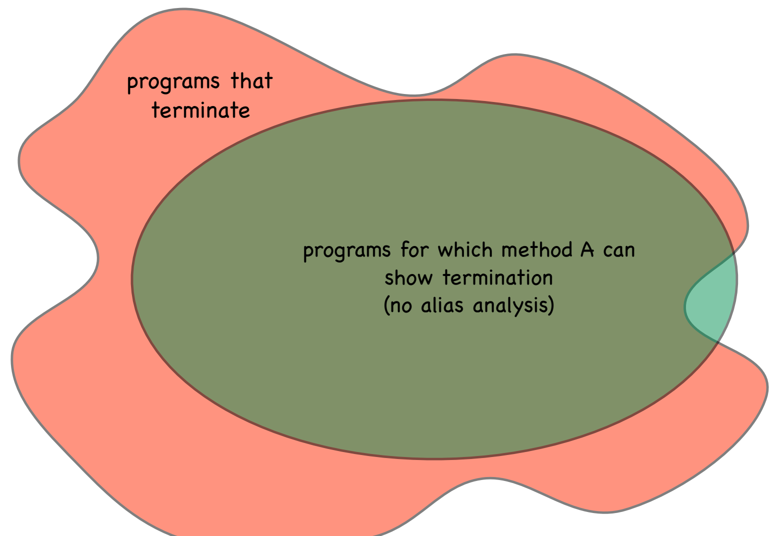
Many problems

- Procedures
- Pointers
- Aliasing
- Optimizing compilers
- Data structures
- Object orientation
- ....

Termination analysis



Termination analysis



Not sound and not complete

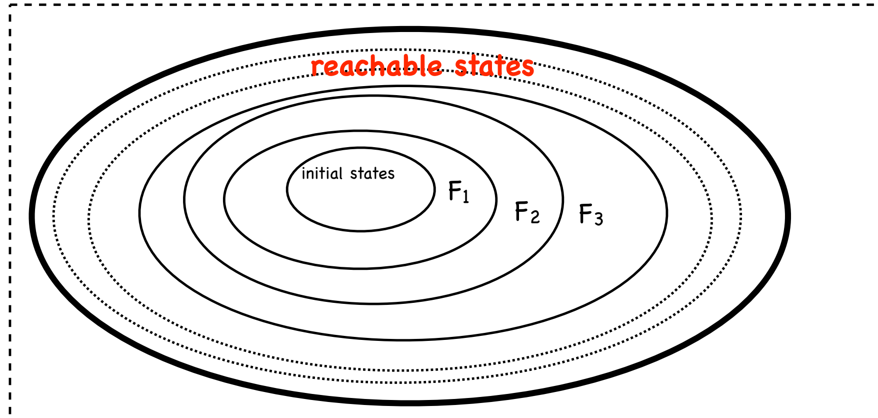
Course preview: Abstract interpretation

Framework for symbolic execution of programs

- Cousot&Cousot, 1977
- Used to approximate the reachable state space
- Approach: Perform forward propagation in an abstract domain
- Domains considered:
  - Linear inequalities (polyhedra)
  - Linear equalities
  - Intervals
  - Octagons, Octahedra
  - Template constraint matrices

Course preview: Forward propagation

Symbolic forward simulation to obtain an overapproximation of the reachable state space (invariants)



F_n : set of reachable states after n program steps

Forward propagation

$$\mathcal{F}_0 = \Theta$$

$$\mathcal{F}_1 = \mathcal{F}_0 \vee (\bigvee_{\tau \in T} \; post(\tau, \mathcal{F}_0))$$

$$\mathcal{F}_2 = \mathcal{F}_1 \vee (\bigvee_{\tau \in T} \; post(\tau, \mathcal{F}_1))$$

until

$$\mathcal{F}_{n+1} \rightarrow \mathcal{F}_n$$

with $post(\tau, \varphi) = \exists V_0 \; (\varphi(V_0) \wedge \rho_\tau(V_0, V))$

Forward propagation: two problems

1. May not converge in finite time

Example:

```
integer i where i = 0
while (true) do i = i+1
```

$\mathcal{F}_0$: $i = 0$

$\mathcal{F}_1$: $i = 0 \vee i = 1$

$\mathcal{F}_2$: $i = 0 \vee i = 1 \vee i = 2$

⋮

We never reach the fixed point: $i \geq 0$

Forward propagation: two problems

2. We may not be able to detect convergence:

checking validity of

$$\mathcal{F}_{n+1} \rightarrow \mathcal{F}_n$$

may not be decidable

Solution to the second problem: Abstract Interpretation

Perform the symbolic simulation in an abstract domain

Domain	Converges?	Reference
Linear equalities	yes	Karr, 76 Muller-Olm,Seidl, '04 Gulwani, Necula, '03
Linear inequalities	no	Cousot,Halbwachs, '79
Intervals	no	Cousot,Cousot, '76
Octagons	no	Mine, '01
Octahedrons	no	Clarisso, Cortadella, '04
TCM	no	SSM, '04

Checking for convergence is decidable in all these domains

Forward propagation: Example

integer i,j **where** i=2 $\wedge$ j=0
while true do
[i := i + 4
or
(i,j) := (i + 2,j + 1)]

Abstract domain: Linear inequalities over the reals

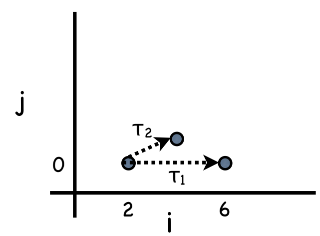
Forward propagation: iteration 1

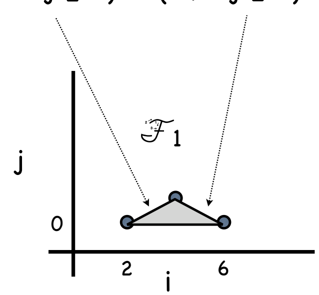
$\mathcal{F}_0$: $(j = 0) \wedge (i = 2)$

post$(\tau_1, \mathcal{F}_0)$: $(j = 0) \wedge (i = 6)$

post$(\tau_2, \mathcal{F}_0)$: $(j = 1) \wedge (i = 4)$

$\mathcal{F}_1$: $(0 \leq j \leq 1) \wedge (i - 2j \geq 2) \wedge (i + 2j \leq 6)$





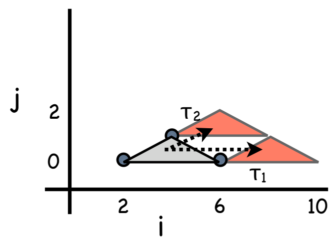
Forward propagation: iteration 2

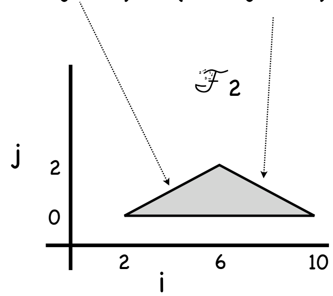
$\mathcal{F}_1$: $(0 \le j \le 1) \wedge (i - 2j \ge 2) \wedge (i + 2j \le 6)$

post(τ_1, $\mathcal{F}_1$) : $(0 \le j \le 1) \wedge (i - 2j \ge 6) \wedge (i + 2j \le 10)$

post(τ_2, $\mathcal{F}_1$) : $(1 \le j \le 2) \wedge (i - 2j \ge 2) \wedge (i + 2j \le 10)$

$\mathcal{F}_2$: $(0 \le j \le 2) \wedge (i - 2j \ge 2) \wedge (i + 2j \le 10)$





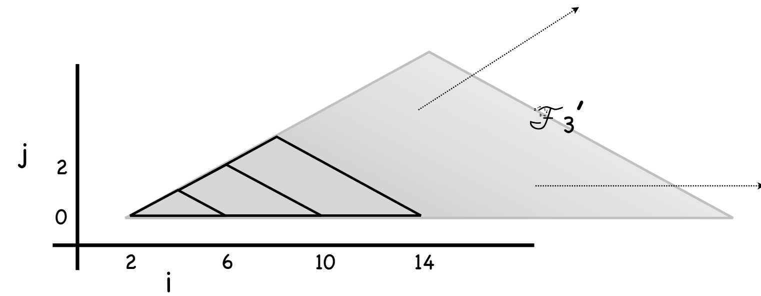
Forward propagation: widening after iteration 3

$\mathcal{F}_1$: $(0 \le j \le 1) \wedge (i - 2j \ge 2) \wedge (i + 2j \le 6)$

$\mathcal{F}_2$: $(0 \le j \le 2) \wedge (i - 2j \ge 2) \wedge (i + 2j \le 10)$

$\mathcal{F}_3$: $(0 \le j \le 3) \wedge (i - 2j \ge 2) \wedge (i + 2j \le 14)$

$\mathcal{F}_3' = \mathcal{F}_2 \nabla \mathcal{F}_3$: $(0 \le j) \wedge (i - 2j \ge 2)$

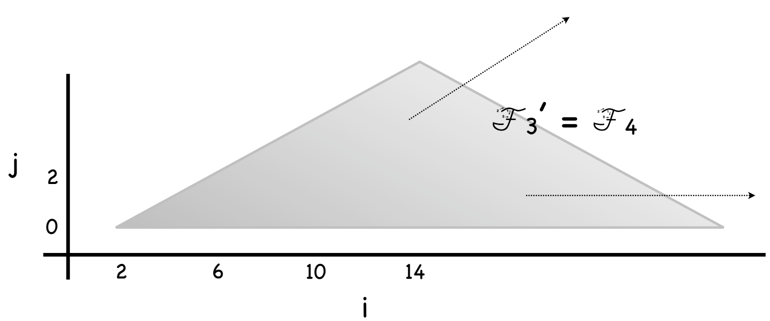


Forward propagation: convergence at iteration 4

post(τ_1, $\mathcal{F}_3'$) : $(0 \le j) \wedge (i - 2j \ge 2)$

post(τ_2, $\mathcal{F}_3'$) : $(0 \le j) \wedge (i - 2j \ge 2)$

$\mathcal{F}_4 = \mathcal{F}_3' \cup$ post($\{\tau_1, \tau_2\}$, $\mathcal{F}_3'$) : $(0 \le j) \wedge (i - 2j \ge 2)$



Course preview: Constraint-based analysis

- Set-based analysis: derive constraints on the set of values that variables may have at given program locations

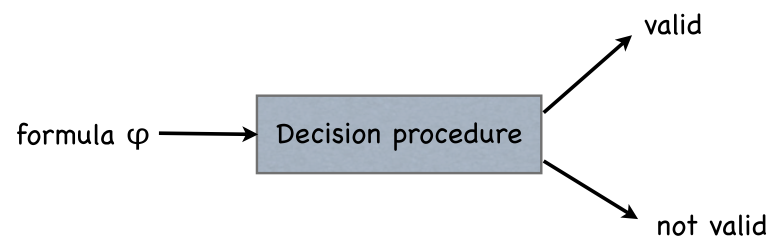
- Property-based analysis:
  1. Define template property: fix type and shape of the property
  2. Encode the conditions for the property to hold as a system of constraints
  3. Solve the constraints
  4. Every solution is a property of the given type and shape

Constraint-based analysis

- Application to
  - Invariant generation
  - Generation of ranking functions
  - Generation of temporal (safety) properties

Course preview: Decision procedures

for a theory T



always terminates

Example of use of decision procedures

```
y = 5;
if (x > 5) {
    y = 0;
}
if (x < 3) {
    z = x/y;
}
```

Possibility of division by zero?

➡Use decision procedure to show that

$$x > 5 \ \wedge \ x < 3$$

is unsatisfiable

Course preview: Decision procedures

- Single theory:
  - Propositional logic
  - Linear arithmetic
  - Recursive data structures (term algebras)
  - Sets, multisets

- Combination of decision procedures:
  - Nelson-Oppen
  - Sets, multisets with cardinality
  - Recursive data structures with cardinality
  - Queues with cardinality

Course preview: other topics

- Shape analysis (Reps et al.)
- Separation logic (Reynolds et al.)
- Static analysis tools (FindBugs, Pugh et al.)
- Dynamic program analysis

Approximation

- In practice there is a trade-off between
  - missed errors (unsoundness)
  - spurious warnings (incompleteness)
  - performance (complexity)
  - annotation overhead
- Balance between cost and performance
- Theory can help to get better approximations at lower cost

Summary

- Start with well-defined first-order program execution model
  - Abstract interpretation
  - Forward propagation
  - Constraint-based analysis
- Decision procedures
  - useful in any program analysis context
- Techniques for analysis of real-life programming languages
  - shape analysis
  - separation logic

References

(1) Cormac Flanagan, K. Rustan Leino, Mark Lillibridge, Greg Nelson, James B. Saxe, Raymie Stata, Extended Static Checking for Java, PLDI 2002.

(2) Daniel Jackson and Martin Rinard, Software Analysis: A Roadmap, in The Future of Software Engineering, ACM Press, 2000.