E0:227, Program Analysis and Verification

3:1, August - December 2021

http://www.csa.iisc.ac.in/~raghavan/pav2021/

K. V. Raghavan and Deepak D'Souza

Software defects

- Defects are very common, and are a major bane of the software industry.
- Some defects are ongoing irritants, while some are disastrous.
- Defects are hard to detect and fix
  - Not enough good and usable tools for programmers
  - Often get detected only after release
  - When a program crashes or gives wrong answer, hard to identify the root cause.

Software defects

- Defects are very common, and are a major bane of the software industry.
- Some defects are ongoing irritants, while some are disastrous.
- Defects are hard to detect and fix
  - Not enough good and usable tools for programmers
  - Often get detected only after release
  - When a program crashes or gives wrong answer, hard to identify the root cause.

Testing, finding and fixing bugs (i.e., Quality Assurance) consumes 50% of total cost and time of software development.

Kinds of defects

- Low-level errors
  - Null pointers, uninitialized values
  - Array index out of bounds, buffer overrun
  - Memory leaks
  - Misuse of pointers and buffers (in languages like C)
  - Unreachable code

Kinds of defects

- Low-level errors
  - Null pointers, uninitialized values
  - Array index out of bounds, buffer overrun
  - Memory leaks
  - Misuse of pointers and buffers (in languages like C)
  - Unreachable code
- High-level errors
  - Does not satisfy user's requirements
  - Algorithmic/design errors
  - Does not interact with other software or libraries correctly
  - Performs poorly

Ariane 5 explosion

Ariane 5 explosion report

On 4 June 1996, the maiden flight of the Ariane 5 launcher ended in a failure about 40 seconds after launch...

The failure of the Ariane 501 was caused by the complete loss of guidance and attitude information ... This loss of information was due to specification and design errors in the software of the inertial reference system.

.

The internal SRI* software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer.

A common approach to software validation: Testing

- A test suite (set of test cases) is created, and executed for each version.
- **Black box** testing: Test cases are created manually by user, or generated randomly.
- **White box** testing: Test cases are generated by an analysis of the program code to increase code coverage.
  - Typically needs tool support.
- What's good about testing? *All bugs found are real bugs.*
- What's bad about testing?

A common approach to software validation: Testing

- A test suite (set of test cases) is created, and executed for each version.
- **Black box** testing: Test cases are created manually by user, or generated randomly.
- **White box** testing: Test cases are generated by an analysis of the program code to increase code coverage.
  - Typically needs tool support.
- What's good about testing? *All bugs found are real bugs.*
- What's bad about testing?
  - 100% coverage of the program's behavior is impossible.
  - Therefore, cannot find all bugs or prove the absence of bugs.
- Very hard to test the portion inside the "if" statement!

```
input x
if (hash(x) == 10) {
  ...
}
```

Program verification

The algorithmic discovery of properties of a program by inspection of the source text.

– *Manna and Pneuli, "Algorithmic Verification"*

Program verification

The algorithmic discovery of properties of a program by inspection of the source text.

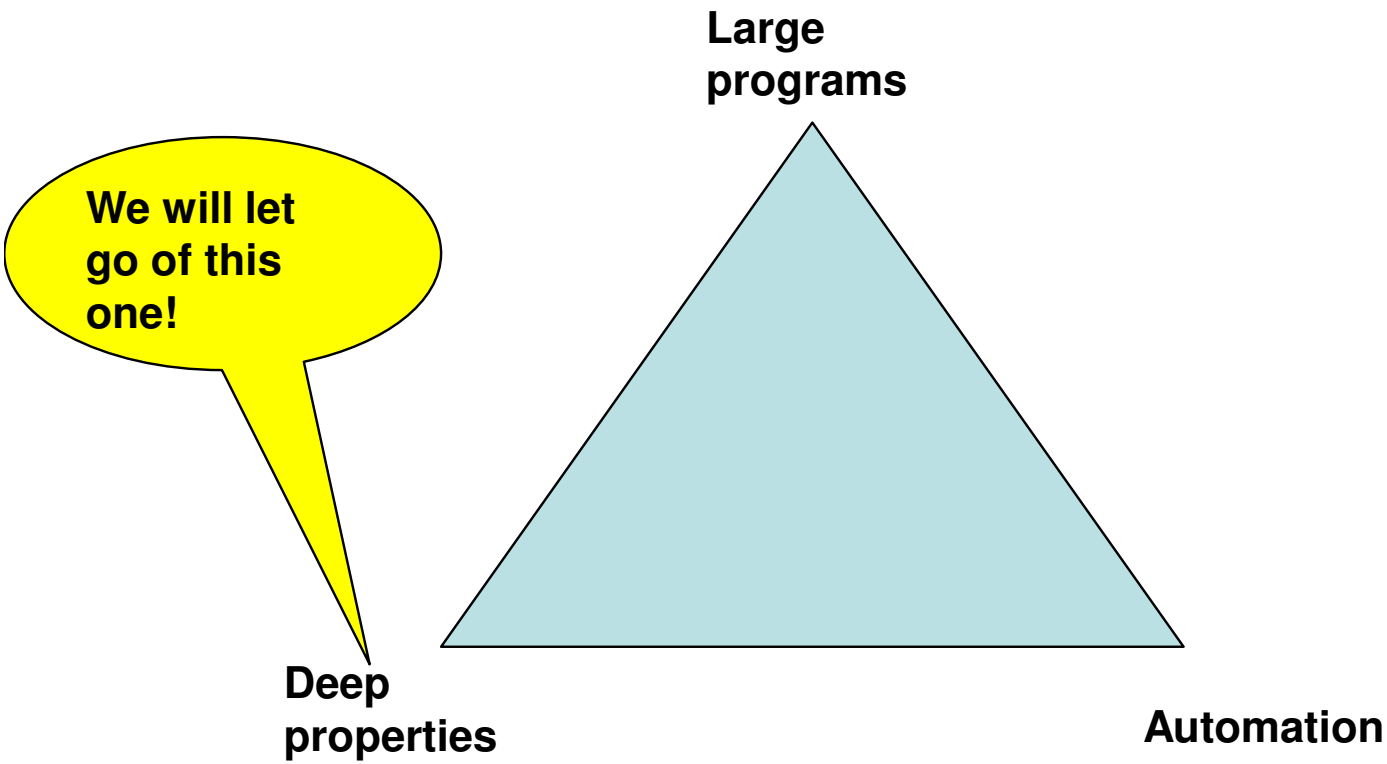
– *Manna and Pneuli, "Algorithmic Verification"*

Also known as: static analysis, static program analysis, formal methods, ...

Difficulty of program verification

- What will we prove?
  - "Deep" specifications of complex software are as complex as the software itself
  - Are difficult to prove
  - State of the art tools and automation are not good enough
- We will focus on "shallow" properties
  - That is, we will prove "partial correctness", or absence of certain classes of low-level errors (e.g., null pointer dereferences)

Elusive triangle



Credit: Sriram Rajamani, Microsoft Research India

Example: Determining whether variables are odd (o) or even (e)

p = oddNatInput()	(p,*o*)	
q = evenNatInput()	(p,*o*)	(q,*e*)
if (p > q)	(p,*o*)	(q,*e*)
p = p*2 + q	(p,*e*)	(q,*e*)
write(p)	(p,*oe*)	(q,*e*)
if (p <= q)	(p,*o*)	(q,*e*)
p = p+1	(p,*e*)	(q,*e*)
write(p)	(p,*e*)	(q,*e*)
q = q+p	(p,*e*)	(q,*e*)

A verification approach: abstract interpretation

- A kind of program execution in which variables store *abstract* values from bounded domains, not concrete values
- Input values are also from the abstract domains
- Program statement semantics are modified to work on abstract variable values
- We execute the program on *all* (abstract) inputs and observe the program properties from these runs

Example: An abstraction

- Abstract value domain V_1 for a single variable: $\{o, e, oe\}$.
- Abstract domain:

  $$L_1 = Var \rightarrow V_1$$

  where *Var* is the set of variables in the program.
- Modified operator semantics:

$+$	o	e	oe
o	e	o	oe
e	o	e	oe
oe	oe	oe	oe

$*$	o	e	oe
o	o	e	oe
e	e	e	e
oe	oe	e	oe

- From the operator semantics, we can construct an abstract transfer function, with signature $L_1 \rightarrow L_1$, for each possible statement in the language.

Example: The abstract interpretation

Abstract interpretation, using domain L_1

	<(p,*oe*), (q,*oe*)>
p=oddNatInput()	
q=evenNatInput()	
if (p > q)	
p = p*2 + q	
write(p)	
if (p <= q)	
p = p+1	
write(p)	
q = q+p	

Example: The abstract interpretation

Abstract interpretation, using domain L_1

	<(p,*oe*), (q,*oe*)>
p=oddNatInput()	<(p,*o*), (q,*oe*)>
q=evenNatInput()	
if (p > q)	
p = p*2 + q	
write(p)	
if (p <= q)	
p = p+1	
write(p)	
q = q+p	

Example: The abstract interpretation

Abstract interpretation, using domain L_1

	<(p,*oe*), (q,*oe*)>
p=oddNatInput()	<(p,*o*), (q,*oe*)>
q=evenNatInput()	<(p,*o*), (q,*e*)>
if (p > q)	
p = p*2 + q	
write(p)	
if (p <= q)	
p = p+1	
write(p)	
q = q+p	

Example: The abstract interpretation

Abstract interpretation, using domain L_1

	<(p,*oe*), (q,*oe*)>
p=oddNatInput()	<(p,*o*), (q,*oe*)>
q=evenNatInput()	<(p,*o*), (q,*e*)>
if (p > q)	<(p,*o*), (q,*e*)>
p = p*2 + q	
write(p)	
if (p <= q)	
p = p+1	
write(p)	
q = q+p	

Example: The abstract interpretation

Abstract interpretation, using domain L_1

	<(p,*oe*), (q,*oe*)>	
p=oddNatInput()	<(p,*o*), (q,*oe*)>	
q=evenNatInput()	<(p,*o*), (q,*e*)>	
if (p > q)	<(p,*o*), (q,*e*)>	
p = p*2 + q		<(p,*e*), (q,*e*)>
write(p)		
if (p <= q)		
p = p+1		
write(p)		
q = q+p		

Example: The abstract interpretation

Abstract interpretation, using domain L_1

	<(p,*oe*), (q,*oe*)>	
p=oddNatInput()	<(p,*o*), (q,*oe*)>	
q=evenNatInput()	<(p,*o*), (q,*e*)>	
if (p > q)	<(p,*o*), (q,*e*)>	
p = p*2 + q		<(p,*e*), (q,*e*)>
write(p)	<(p,*oe*), (q,*e*)>	
if (p <= q)		
p = p+1		
write(p)		
q = q+p		

Example: The abstract interpretation

Abstract interpretation, using domain L_1

	<(p,*oe*), (q,*oe*)>	
p=oddNatInput()	<(p,*o*), (q,*oe*)>	
q=evenNatInput()	<(p,*o*), (q,*e*)>	
if (p > q)	<(p,*o*), (q,*e*)>	
p = p*2 + q		<(p,*e*), (q,*e*)>
write(p)	<(p,*oe*), (q,*e*)>	
if (p <= q)	<(p,*oe*), (q,*e*)>	
p = p+1		
write(p)		
q = q+p		

Example: The abstract interpretation

Abstract interpretation, using domain L_1

	<(p,*oe*), (q,*oe*)>	
p=oddNatInput()	<(p,*o*), (q,*oe*)>	
q=evenNatInput()	<(p,*o*), (q,*e*)>	
if (p > q)	<(p,*o*), (q,*e*)>	
p = p*2 + q		<(p,*e*), (q,*e*)>
write(p)	<(p,*oe*), (q,*e*)>	
if (p <= q)	<(p,*oe*), (q,*e*)>	
p = p+1		<(p,*oe*), (q,*e*)>
write(p)		
q = q+p		

Example: The abstract interpretation

Abstract interpretation, using domain L_1

	<(p,*oe*), (q,*oe*)>	
p=oddNatInput()	<(p,*o*), (q,*oe*)>	
q=evenNatInput()	<(p,*o*), (q,*e*)>	
if (p > q)	<(p,*o*), (q,*e*)>	
p = p*2 + q		<(p,*e*), (q,*e*)>
write(p)	<(p,*oe*), (q,*e*)>	
if (p <= q)	<(p,*oe*), (q,*e*)>	
p = p+1		<(p,*oe*), (q,*e*)>
write(p)	<(p,*oe*), (q,*e*)>	
q = q+p		

Example: The abstract interpretation

Abstract interpretation, using domain L_1

	<(p,*oe*), (q,*oe*)>	
p=oddNatInput()	<(p,*o*), (q,*oe*)>	
q=evenNatInput()	<(p,*o*), (q,*e*)>	
if (p > q)	<(p,*o*), (q,*e*)>	
p = p*2 + q		<(p,*e*), (q,*e*)>
write(p)	<(p,*oe*), (q,*e*)>	
if (p <= q)	<(p,*oe*), (q,*e*)>	
p = p+1		<(p,*oe*), (q,*e*)>
write(p)	<(p,*oe*), (q,*e*)>	
q = q+p	<(p,*oe*), (q,*oe*)>	

Example: The abstract interpretation

	Abstract interpretation, using domain L_1		Ideal results	
	<(p,*oe*), (q,*oe*)>			
p=oddNatInput()	<(p,*o*), (q,*oe*)>		(p,*o*)	
q=evenNatInput()	<(p,*o*), (q,*e*)>		(p,*o*)	(q,*e*)
if (p > q)	<(p,*o*), (q,*e*)>		(p,*o*)	(q,*e*)
p = p*2 + q		<(p,*e*), (q,*e*)>	(p,*e*)	(q,*e*)
write(p)	<(p,*oe*), (q,*e*)>		(p,*oe*)	(q,*e*)
if (p <= q)	<(p,*oe*), (q,*e*)>		(p,*o*)	(q,*e*)
p = p+1		<(p,*oe*), (q,*e*)>	(p,*e*)	(q,*e*)
write(p)	<(p,*oe*), (q,*e*)>		(p,*e*)	(q,*e*)
q = q+p	<(p,*oe*), (q,*oe*)>		(p,*e*)	(q,*e*)

Example: The abstract interpretation

Abstract interpretation, using domain L_1

	<(p,*oe*), (q,*oe*)>	
p=oddNatInput()	<(p,*o*), (q,*oe*)>	
q=evenNatInput()	<(p,*o*), (q,*e*)>	
if (p > q)	<(p,*o*), (q,*e*)>	
p = p*2 + q		<(p,*e*), (q,*e*)>
write(p)	<(p,*oe*), (q,*e*)>	
if (p <= q)	**X** <(p,*oe*), (q,*e*)>	
p = p+1		**X** <(p,*oe*), (q,*e*)>
write(p)	**X** <(p,*oe*), (q,*e*)>	
q = q+p	**X** <(p,*oe*), (q,*oe*)>	

Ideal results

(p,*o*)	
(p,*o*)	(q,*e*)
(p,*o*)	(q,*e*)
(p,*e*)	(q,*e*)
(p,*oe*)	(q,*e*)
(p,*o*)	(q,*e*)
(p,*e*)	(q,*e*)
(p,*e*)	(q,*e*)
(p,*e*)	(q,*e*)

Example: Another abstraction

- Abstract value domain V_2 for a single variable: $\{o, e\}$.
- The alternative domain:

  $$L_2 = 2^{Var \to V_2}$$

  where *Var* is the set of variables in the program.
- Same operator tables as before.
- From the operator semantics, we can construct an abstract transfer function, $L_2 \to L_2$, for each possible statement in the language.

Example: The abstract interpretation

Abstract interpretation, using domain L_2

	$\{<(p,o),(q,o)>, <(p,o),(q,e)>$ $<(p,e),(q,o)>, <(p,e),(q,e)>\}$
p=oddNatInput()	,
q=evenNatInput()	
if (p > q)	
p = p*2 + q	
write(p)	
if (p <= q)	
p = p+1	
write(p)	
q = q+p	

Example: The abstract interpretation

Abstract interpretation, using domain L_2

	{<(p,*o*), (q,*o*)>, <(p,*o*), (q,*e*)> <(p,*e*), (q,*o*)>, <(p,*e*), (q,*e*)>}
p=oddNatInput()	{<(p,*o*), (q,*o*)>, <(p,*o*), (q,*e*)>}
q=evenNatInput()	
if (p > q)	
p = p*2 + q	
write(p)	
if (p <= q)	
p = p+1	
write(p)	
q = q+p	

Example: The abstract interpretation

Abstract interpretation, using domain L_2

	{<(p,*o*), (q,*o*)>, <(p,*o*), (q,*e*)> <(p,*e*), (q,*o*)>, <(p,*e*), (q,*e*)>}
p=oddNatInput()	{<(p,*o*), (q,*o*)>, <(p,*o*), (q,*e*)>}
q=evenNatInput()	{<(p,*o*), (q,*e*)>}
if (p > q)	
p = p*2 + q	
write(p)	
if (p <= q)	
p = p+1	
write(p)	
q = q+p	

Example: The abstract interpretation

Abstract interpretation, using domain L_2

	{<(p,*o*), (q,*o*)>, <(p,*o*), (q,*e*)> <(p,*e*), (q,*o*)>, <(p,*e*), (q,*e*)>}
p=oddNatInput()	{<(p,*o*), (q,*o*)>, <(p,*o*), (q,*e*)>}
q=evenNatInput()	{<(p,*o*), (q,*e*)>}
if (p > q)	{<(p,*o*), (q,*e*)>}
p = p*2 + q	
write(p)	
if (p <= q)	
p = p+1	
write(p)	
q = q+p	

Example: The abstract interpretation

Abstract interpretation, using domain L_2

	{<(p,*o*), (q,*o*)>,	<(p,*o*), (q,*e*)>
	<(p,*e*), (q,*o*)>,	<(p,*e*), (q,*e*)>}
p=oddNatInput()	{<(p,*o*), (q,*o*)>,	<(p,*o*), (q,*e*)>}
q=evenNatInput()	{<(p,*o*), (q,*e*)>}	
if (p > q)	{<(p,*o*), (q,*e*)>}	
p = p*2 + q		{<(p,*e*), (q,*e*)>}
write(p)		
if (p <= q)		
p = p+1		
write(p)		
q = q+p		

Example: The abstract interpretation

Abstract interpretation, using domain L_2

	{<(p,*o*), (q,*o*)>,	<(p,*o*), (q,*e*)>
	<(p,*e*), (q,*o*)>,	<(p,*e*), (q,*e*)>}
p=oddNatInput()	{<(p,*o*), (q,*o*)>,	<(p,*o*), (q,*e*)>}
q=evenNatInput()	{<(p,*o*), (q,*e*)>}	
if (p > q)	{<(p,*o*), (q,*e*)>}	
p = p*2 + q		{<(p,*e*), (q,*e*)>}
write(p)	{<(p,*o*), (q,*e*)>,	
if (p <= q)		
p = p+1		
write(p)		
q = q+p		

Example: The abstract interpretation

Abstract interpretation, using domain L_2

	{<(p,*o*), (q,*o*)>,	<(p,*o*), (q,*e*)>
	<(p,*e*), (q,*o*)>,	<(p,*e*), (q,*e*)>}
p=oddNatInput()	{<(p,*o*), (q,*o*)>,	<(p,*o*), (q,*e*)>}
q=evenNatInput()	{<(p,*o*), (q,*e*)>}	
if (p > q)	{<(p,*o*), (q,*e*)>}	
p = p*2 + q		{<(p,*e*), (q,*e*)>}
write(p)	{<(p,*o*), (q,*e*)>,	<(p,*e*), (q,*e*)>}
if (p <= q)		
p = p+1		
write(p)		
q = q+p		

Example: The abstract interpretation

Abstract interpretation, using domain L_2

	{<(p,*o*), (q,*o*)>,	<(p,*o*), (q,*e*)>
	<(p,*e*), (q,*o*)>,	<(p,*e*), (q,*e*)>}
p=oddNatInput()	{<(p,*o*), (q,*o*)>,	<(p,*o*), (q,*e*)>}
q=evenNatInput()	{<(p,*o*), (q,*e*)>}	
if (p > q)	{<(p,*o*), (q,*e*)>}	
p = p*2 + q		{<(p,*e*), (q,*e*)>}
write(p)	{<(p,*o*), (q,*e*)>,	<(p,*e*), (q,*e*)>}
if (p <= q)	{<(p,*o*), (q,*e*)>,	<(p,*e*), (q,*e*)>}
p = p+1		
write(p)		
q = q+p		

Example: The abstract interpretation

Abstract interpretation, using domain L_2

	{<(p,*o*), (q,*o*)>,	<(p,*o*), (q,*e*)>
	<(p,*e*), (q,*o*)>,	<(p,*e*), (q,*e*)>}
p=oddNatInput()	{<(p,*o*), (q,*o*)>,	<(p,*o*), (q,*e*)>}
q=evenNatInput()	{<(p,*o*), (q,*e*)>}	
if (p > q)	{<(p,*o*), (q,*e*)>}	
p = p*2 + q		{<(p,*e*), (q,*e*)>}
write(p)	{<(p,*o*), (q,*e*)>,	<(p,*e*), (q,*e*)>}
if (p <= q)	{<(p,*o*), (q,*e*)>,	<(p,*e*), (q,*e*)>}
p = p+1	{<(p,*e*), (q,*e*)>,	<(p,*o*), (q,*e*)>}
write(p)		
q = q+p		

Example: The abstract interpretation

Abstract interpretation, using domain L_2

	{<(p,*o*), (q,*o*)>,	<(p,*o*), (q,*e*)>
	<(p,*e*), (q,*o*)>,	<(p,*e*), (q,*e*)>}
p=oddNatInput()	{<(p,*o*), (q,*o*)>,	<(p,*o*), (q,*e*)>}
q=evenNatInput()	{<(p,*o*), (q,*e*)>}	
if (p > q)	{<(p,*o*), (q,*e*)>}	
p = p*2 + q		{<(p,*e*), (q,*e*)>}
write(p)	{<(p,*o*), (q,*e*)>,	<(p,*e*), (q,*e*)>}
if (p <= q)	{<(p,*o*), (q,*e*)>,	<(p,*e*), (q,*e*)>}
p = p+1	{<(p,*e*), (q,*e*)>,	<(p,*o*), (q,*e*)>}
write(p)	{<(p,*e*), (q,*e*)>,	<(p,*o*), (q,*e*)>}
q = q+p		

Example: The abstract interpretation

	Abstract interpretation, using domain L_2		Ideal results	
	{<(p,*o*), (q,*o*)>,	<(p,*o*), (q,*e*)>		
	<(p,*e*), (q,*o*)>,	<(p,*e*), (q,*e*)>}		
p=oddNatInput()	{<(p,*o*), (q,*o*)>,	<(p,*o*), (q,*e*)>}	(p,*o*)	
q=evenNatInput()	{<(p,*o*), (q,*e*)>}		(p,*o*)	(q,*e*)
if (p > q)	{<(p,*o*), (q,*e*)>}		(p,*o*)	(q,*e*)
p = p*2 + q		{<(p,*e*), (q,*e*)>}	(p,*e*)	(q,*e*)
write(p)	{<(p,*o*), (q,*e*)>,	<(p,*e*), (q,*e*)>}	(p,*oe*)	(q,*e*)
if (p <= q)	{<(p,*o*), (q,*e*)>,	<(p,*e*), (q,*e*)>}	(p,*o*)	(q,*e*)
p = p+1	{<(p,*e*), (q,*e*)>,	<(p,*o*), (q,*e*)>}	(p,*e*)	(q,*e*)
write(p)	{<(p,*e*), (q,*e*)>,	<(p,*o*), (q,*e*)>}	(p,*e*)	(q,*e*)
q = q+p	{<(p,*e*), (q,*e*)>,	<(p,*o*), (q,*o*)>}	(p,*e*)	(q,*e*)

Example: The abstract interpretation

	Abstract interpretation, using domain L_2		Ideal results	
	{<(p,*o*), (q,*o*)>,	<(p,*o*), (q,*e*)>		
	<(p,*e*), (q,*o*)>,	<(p,*e*), (q,*e*)>}		
p=oddNatInput()	{<(p,*o*), (q,*o*)>,	<(p,*o*), (q,*e*)>}	(p,*o*)	
q=evenNatInput()	{<(p,*o*), (q,*e*)>}		(p,*o*)	(q,*e*)
if (p > q)	{<(p,*o*), (q,*e*)>}		(p,*o*)	(q,*e*)
p = p*2 + q		{<(p,*e*), (q,*e*)>}	(p,*e*)	(q,*e*)
write(p)	{<(p,*o*), (q,*e*)>,	<(p,*e*), (q,*e*)>}	(p,*oe*)	(q,*e*)
if (p <= q)	{<(p,*o*), (q,*e*)>,	**X**<(p,*e*), (q,*e*)>}	(p,*o*)	(q,*e*)
p = p+1	{<(p,*e*), (q,*e*)>,	**X**<(p,*o*), (q,*e*)>}	(p,*e*)	(q,*e*)
write(p)	{<(p,*e*), (q,*e*)>,	**X**<(p,*o*), (q,*e*)>}	(p,*e*)	(q,*e*)
q = q+p	{<(p,*e*), (q,*e*)>,	**X**<(p,*o*), (q,*o*)>}	(p,*e*)	(q,*e*)

Example: The abstract interpretation

	Abstract interpretation, using domain L_2		Ideal results	
	{<(p,*o*), (q,*o*)>,	<(p,*o*), (q,*e*)>		
	<(p,*e*), (q,*o*)>,	<(p,*e*), (q,*e*)>}		
p=oddNatInput()	{<(p,*o*), (q,*o*)>,	<(p,*o*), (q,*e*)>}	(p,*o*)	
q=evenNatInput()	{<(p,*o*), (q,*e*)>}		(p,*o*)	(q,*e*)
if (p > q)	{<(p,*o*), (q,*e*)>}		(p,*o*)	(q,*e*)
p = p*2 + q		{<(p,*e*), (q,*e*)>}	(p,*e*)	(q,*e*)
write(p)	{<(p,*o*), (q,*e*)>,	<(p,*e*), (q,*e*)>}	(p,*oe*)	(q,*e*)
if (p <= q)	{<(p,*o*), (q,*e*)>,	**X**<(p,*e*), (q,*e*)>}	(p,*o*)	(q,*e*)
p = p+1	{<(p,*e*), (q,*e*)>,	**X**<(p,*o*), (q,*e*)>}	(p,*e*)	(q,*e*)
write(p)	{<(p,*e*), (q,*e*)>,	**X**<(p,*o*), (q,*e*)>}	(p,*e*)	(q,*e*)
q = q+p	{<(p,*e*), (q,*e*)>,	**X**<(p,*o*), (q,*o*)>}	(p,*e*)	(q,*e*)

In comparison to the L_1 domain

- L_2 is a *more precise* domain. Result at the end of the program was <(p,*oe*),(q,*oe*)> with L_1, which *over-approximates* $\{<(p,e),(q,e)>,<(p,o),(q,o)>\}$.
- However, L_1 is *more efficient*.
- Both are less precise than ideal!

Other examples of verification problems

Analysis	*Abstract domain*
Null-pointer deref.	$Var \rightarrow \{not\text{-}pointer, null, non\text{-}null\} \times 2^{Var}$
Array overruns	$Var \rightarrow IntegerRanges$
File IO	$File\text{-}handles \rightarrow Files$, $Files \rightarrow \{open, closed\}$,
Reachability	Reachability condition
Mutual exclusion	set of locks taken

Other applications of program analysis

- Compilers
  - Live variables analysis
    - Useful, e.g., for register allocation
  - Side-effect analysis of functions
    - Useful, e.g., for code motion
  - Interaction between statements
    - Useful, e.g., for separating sequential code into independent threads
- Code development tools
  - Refactoring; e.g., rename method, extract method
  - Generating code automatically from specifications
  - Automated generation of test cases

Overview of PAV course

- Introduction (1 lecture)
- Lattice theory (2)
- Theory of abstract interpretation (3)
- Implementation of abstract interpretation:
  - Intra-procedural (2)
  - Inter-procedural (9)
- Pointer analysis (2)
- Program slicing (4)
- Type systems (4)
- Assertional reasoning — a first-order predicate logic for proving facts about programs (2)

Prerequisites

- Discrete structures such as sets, relations, partially ordered sets, functions
- (Undergraduate level) algorithms
- Mathematical logic (propositional, first-order)
- General mathematical maturity: comfort with notation, understanding and writing proofs
- Familiarity with imperative languages like C or Java
- Programming project will involve Java (only basic features of Java)

What we will not cover

- Software engineering
  - How to collect requirements from customers and prioritize them
  - Planning and management of software development
  - Design, architecture, coding
  - *in* Formal Methods in Software Engineering, Jan. 2022
- Programming languages
- Analysis of parallel/concurrent programs, distributed systems

Lectures format

- Live lectures via Teams
- Last 15 minutes in most of the lectures will be reserved for working out problems in small groups.
- Lectures will be recorded for your future reference (no videos provided before lectures)
- Attendance is mandatory, and will be noted in every class
  - Missing up to 3 classes attracts no penalty
  - missing 4-6 lectures will result in one grade point penalty from final grade, missing 7-9 lectures will result in two grade points penalty, and so on.

Assignments and exams (tentative)

- 5-6 written assignments (40%)
  - For each assigment, 20% of your marks obtained will be deducted for each day of delay in submitting.
- Programming assignment: 20%
- Exams: mid-sem (15%), final (25%)

Misconduct policy

- Academic misconduct (e.g., copying) will not be tolerated
- Discussion in exams $\Rightarrow$ automatic fail grade for both students
- Assignments
  - Work individually.
  - If necessary, you can seek clarifications on basic concepts from others (preferably via chat on the class Teams forum)
  - However, you must develop the solutions to the given problems on your own (without discussions), and write the answers on your own.
  - No looking at others solutions, no showing your solution to others!
  - If you refer to general materials other than class lecture notes and text books, mention them. However, do not search on the internet for answers to assignment problems or for program fragments for the project.
- Penalties
  - For *each* instance of a violation of above policy $\Rightarrow$
    Zero for the entire assignment, *plus* one grade-point reduction in final grade (for the one who copied).
  - Grade-point reductions over multiple violations will accumulate.