Theoretical computer science

Tutorial - week 14

April 22, 2021

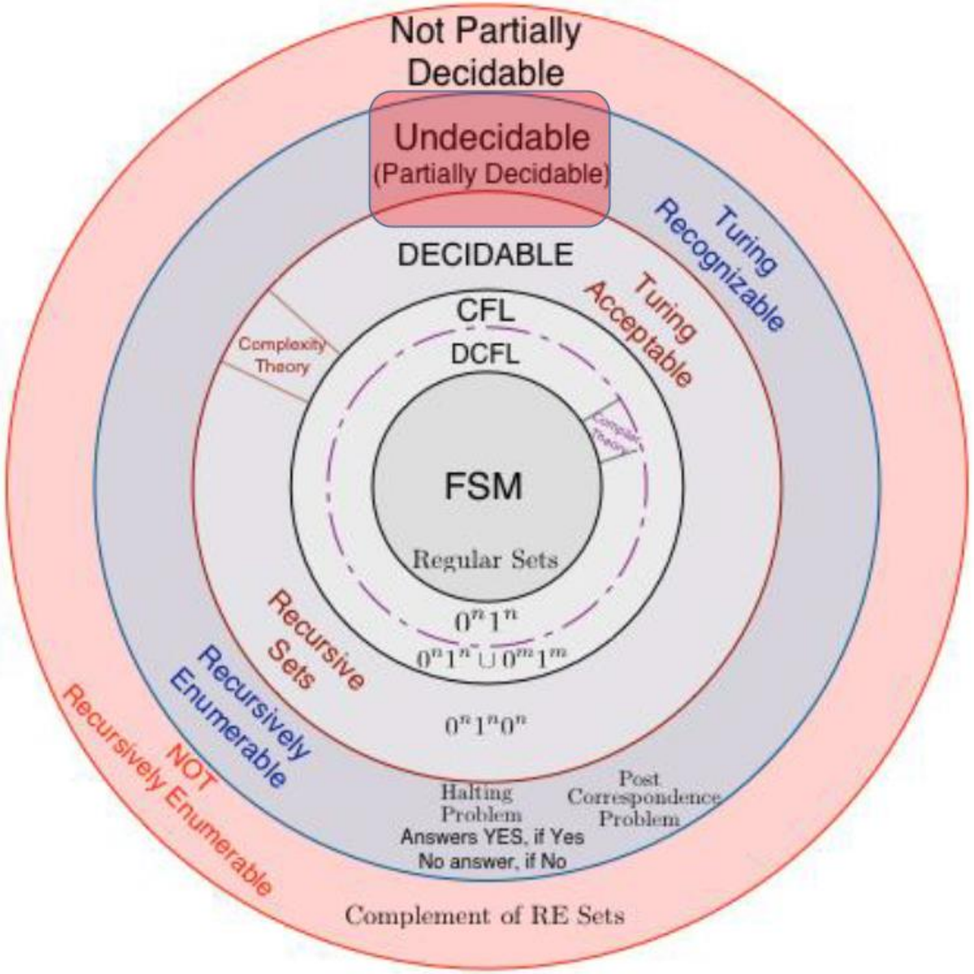
INNOPOLIS
UNIVERSITY

Agenda

- Announcements
- Rice theorem implication
- Static Analysis

Recap from the lecture

Our long journey in the outer space...



[Complexity Zoology: Active Inclusion Graph (ucdavis.edu)](https://)

Common questions about programs

- Termination
- Memory requirements
- Null pointer dereference, division by zero, overflow
- Initialization before access
- …

Rice's theorem

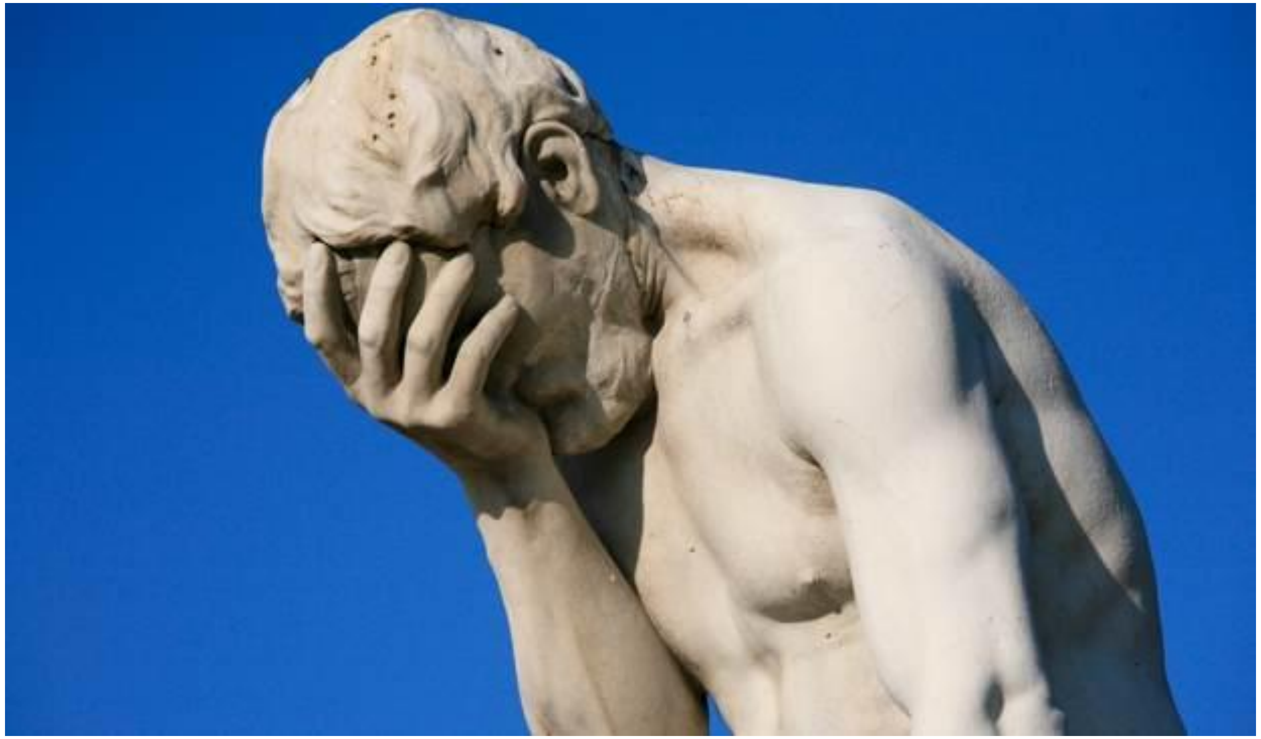
Any non-trivial property of the behavior of programs (<u>semantic</u>) in a Turing-complete language is undecidable!

Examples:

- The class of partial computable functions that return *0* for every input, and its complement.

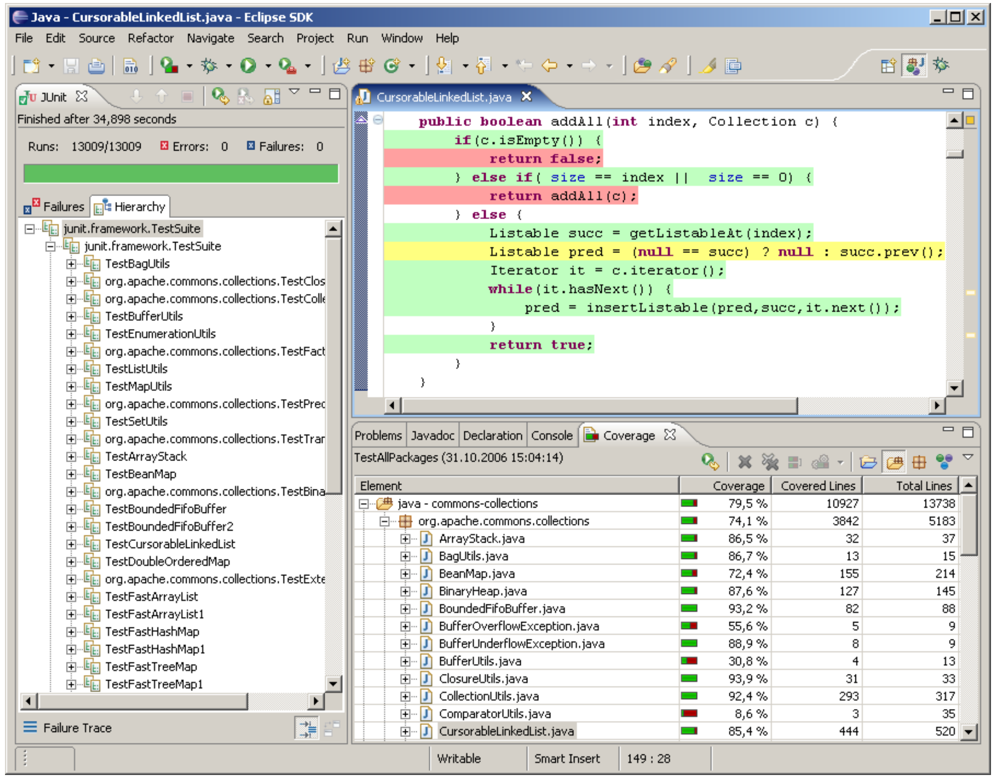
- The class of partial computable functions that return *0* for at least one input, and its complement.

- The class of partial computable functions that are constant, and its complement.



Testing

- Spots bugs with failing tests
- Check if software **runs** as **expected**
- Effective at finding
  - software regression
  - security vulnerabilities (fuzzing)
- Provide rough idea about quality of software with coverage information



Finished after 34,898 seconds

Runs: 13009/13009 Errors: 0 Failures: 0

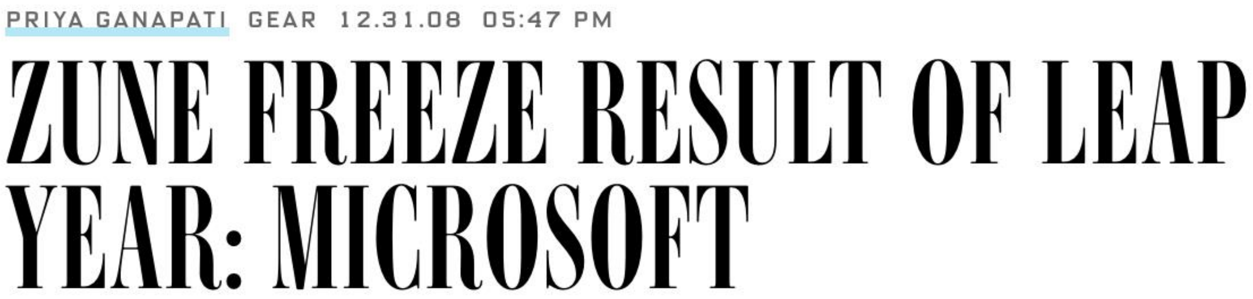
- junit.framework.TestSuite
  - junit.framework.TestSuite
    - TestBagUtils
    - org.apache.commons.collections.TestClos
    - org.apache.commons.collections.TestColle
    - TestBufferUtils
    - TestEnumerationUtils
    - org.apache.commons.collections.TestFact
    - TestListUtils
    - TestMapUtils
    - org.apache.commons.collections.TestPrec
    - TestSetUtils
    - org.apache.commons.collections.TestTrar
    - TestArrayStack
    - TestBeanMap
    - org.apache.commons.collections.TestBina
    - TestBoundedFifoBuffer
    - TestBoundedFifoBuffer2
    - TestCursorableLinkedList
    - TestDoubleOrderedMap
    - org.apache.commons.collections.TestExte
    - TestFastArrayList
    - TestFastArrayList1
    - TestFastHashMap
    - TestFastHashMap1
    - TestFastTreeMap
    - TestFastTreeMap1

```
public boolean addAll(int index, Collection c) {
    if(c.isEmpty()) {
        return false;
    } else if( size == index ||  size == 0) {
        return addAll(c);
    } else {
        Listable succ = getListableAt(index);
        Listable pred = (null == succ) ? null : succ.prev();
        Iterator it = c.iterator();
        while(it.hasNext()) {
            pred = insertListable(pred,succ,it.next());
        }
        return true;
    }
}
```

TestAllPackages (31.10.2006 15:04:14)

Element	Coverage	Covered Lines	Total Lines
java - commons-collections	79,5 %	10927	13738
org.apache.commons.collections	74,1 %	3842	5183
ArrayStack.java	86,5 %	32	37
BagUtils.java	86,7 %	13	15
BeanMap.java	72,4 %	155	214
BinaryHeap.java	87,6 %	127	145
BoundedFifoBuffer.java	93,2 %	82	88
BufferOverflowException.java	55,6 %	5	9
BufferUnderflowException.java	88,9 %	8	9
BufferUtils.java	30,8 %	4	13
ClosureUtils.java	93,9 %	31	33
CollectionUtils.java	92,4 %	293	317
ComparatorUtils.java	8,6 %	3	35
CursorableLinkedList.java	85,4 %	444	520

Software quality matters (Embedded SW)



PRIYA GANAPATI GEAR 12.31.08 05:47 PM

ZUNE FREEZE RESULT OF LEAP YEAR: MICROSOFT

Software quality matters (ZUNE)

```
while (year > 365) {
  if (IsLeapYear(year)) {
    if (days > 366) {
      days -= 366;
      year += 1;
    }
  } else {
    days -= 365;
    year += 1;
  }
}
```

December 31, 2008

Suggested solution: wait for tomorrow

Edsger W. Dijkstra

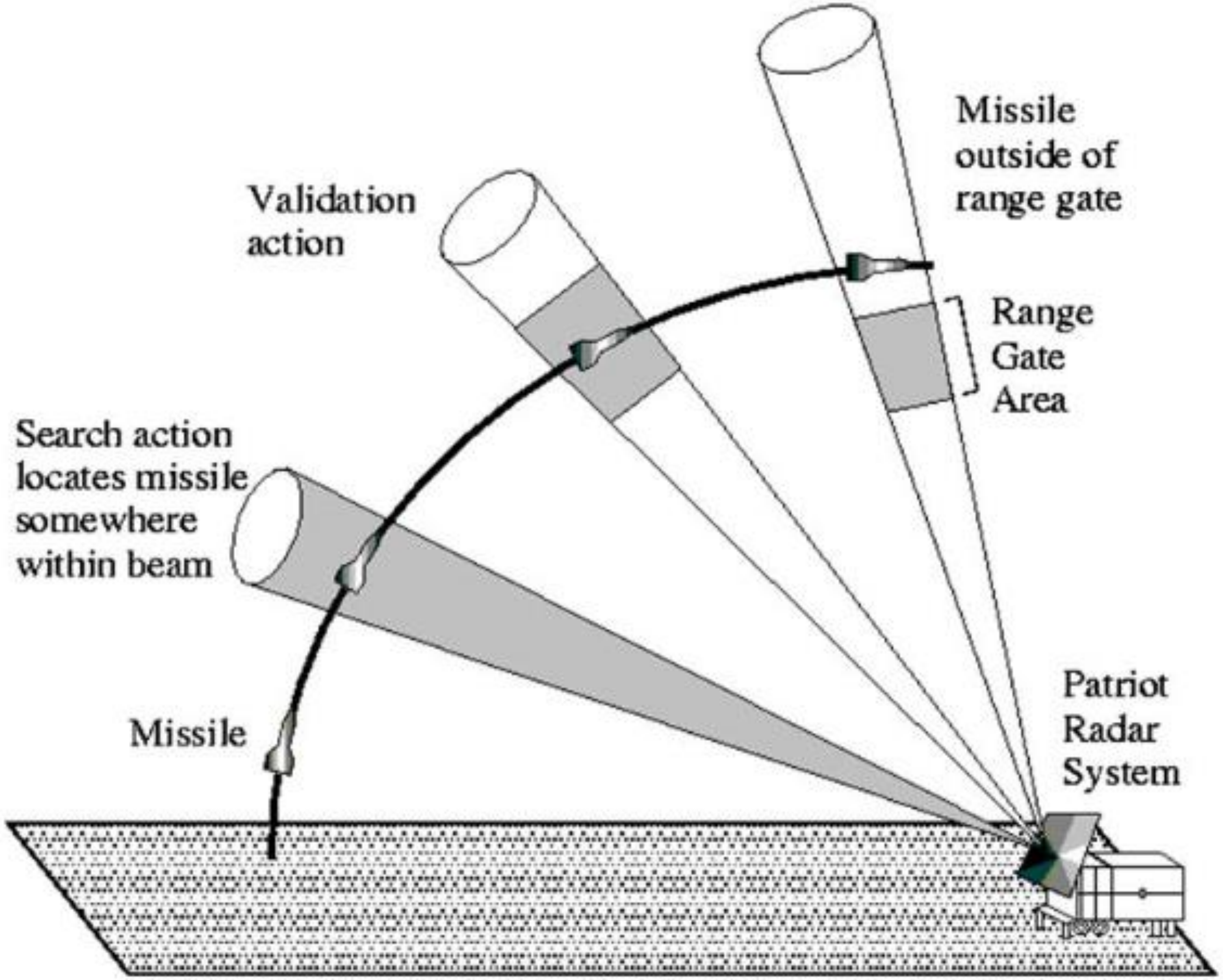
“Testing can only prove the presence of bugs, not their absence.”

Software quality matters (Safety critical)

On the night of the 25th of February, 1991, a Patriot missile system operating in Dhahran, Saudi Arabia, failed to track and intercept an incoming Scud. The Iraqi missile impacted into an army barracks, killing 28 U.S. soldiers and injuring another 98.

February 25, 1991

Software quality matters (Safety critical)



Software quality matters (Patriot)

- Time measured in 1/10 seconds
- Binary expansion of 1/10: 0.0001100110011001100110011001100....
- 24-bit register

  0.00011001100110011001100
- error of
  - 0.0000000000000000000000011001100... binary, or ~0.000000095 decimal
- After 100 hours of operation error is 0.000000095×100×3600×10=0.34
- A Scud travels at about 1,676 meters per second, and so travels more than half a kilometer in this time

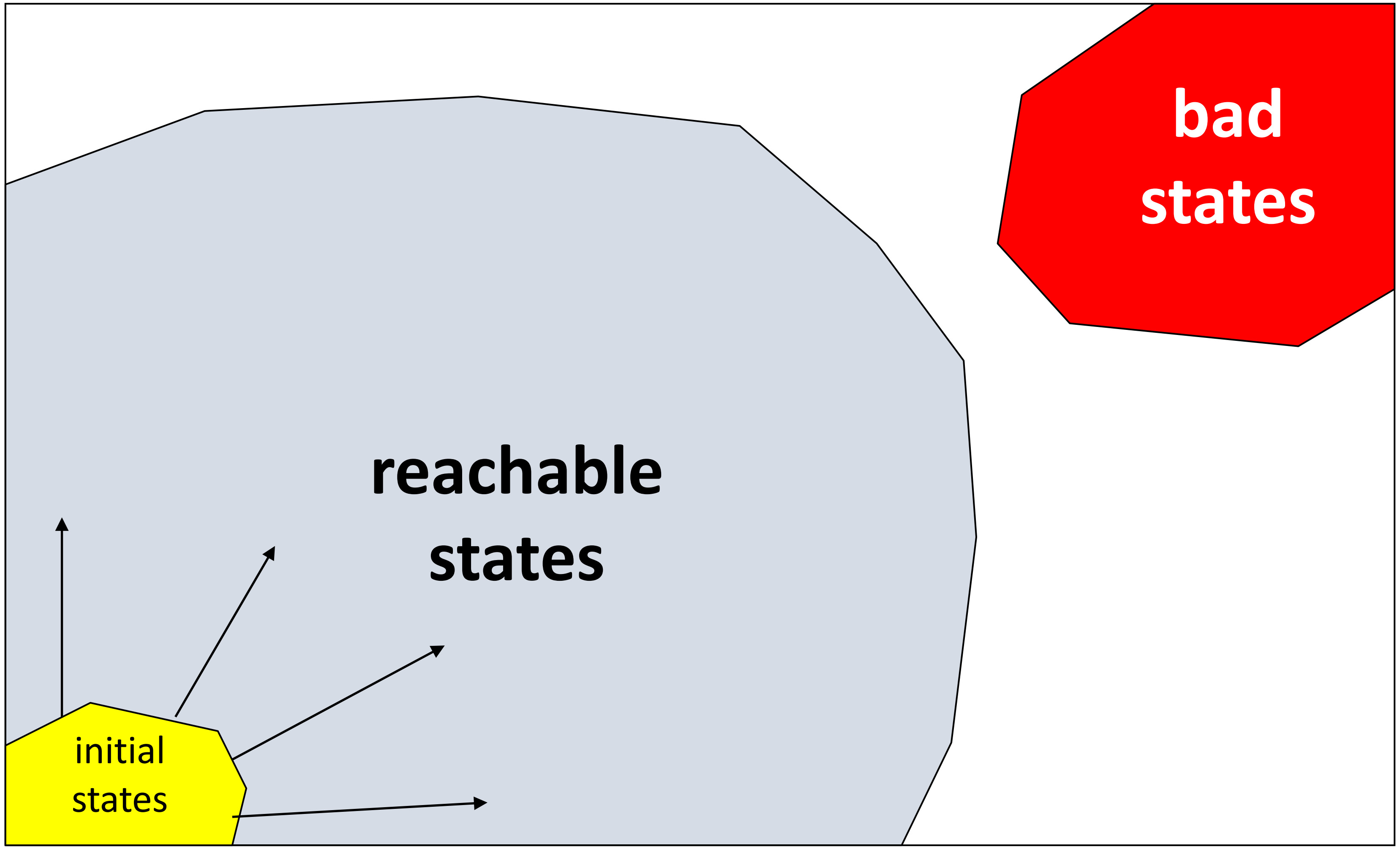
Suggested solution: reboot every 10 hours

What is Static Analysis?

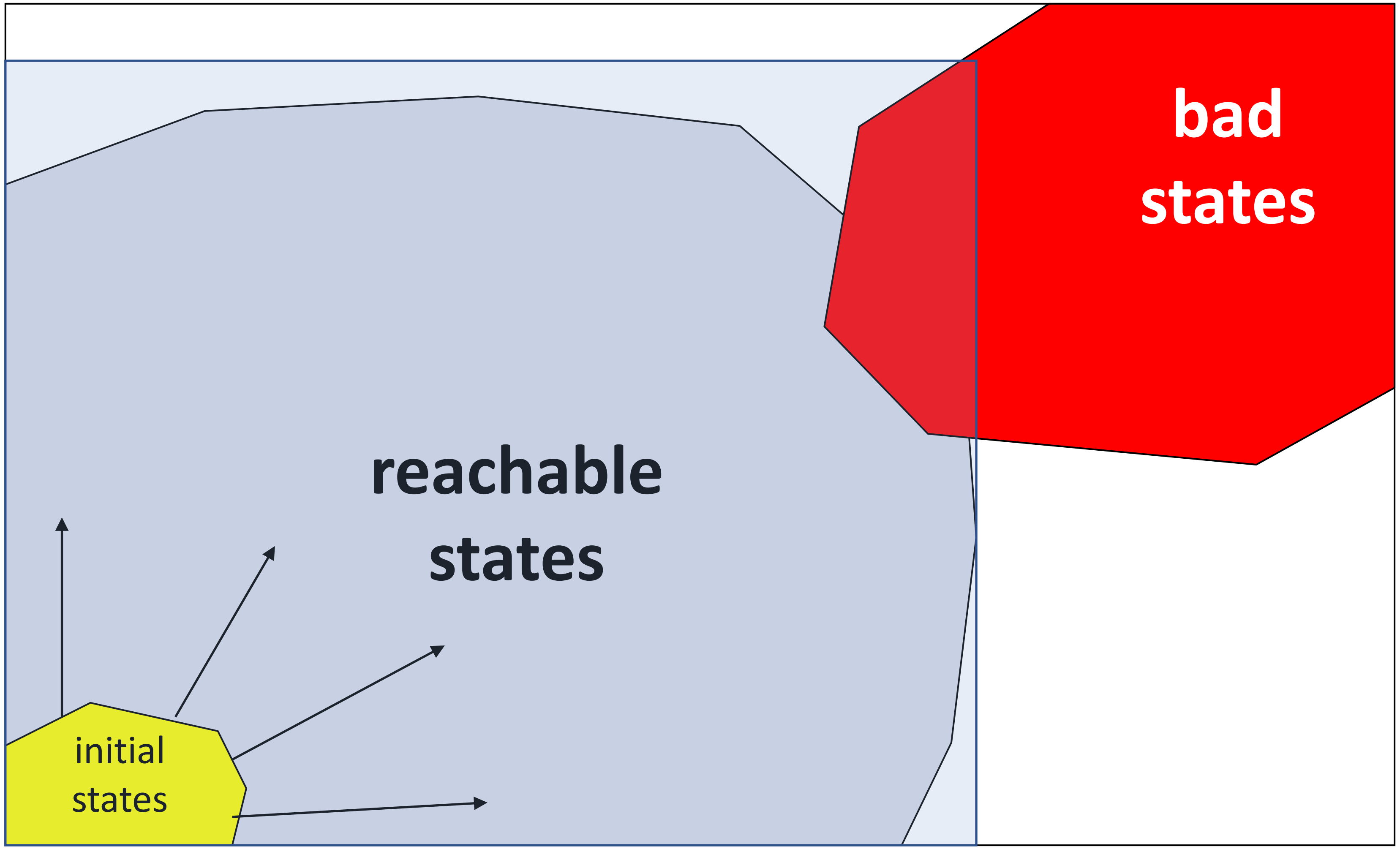
Program as a Data

- Discovers properties of a program (or a program point) by reasoning statically the program (without running the program)
- Discovered properties can be:
  - the program terminates.
  - arrays are always accessed within their bounds.
  - a variable **p** at line 10 cannot have a null pointer.
  - a variable **p** at line 10 cannot be a dangling reference.

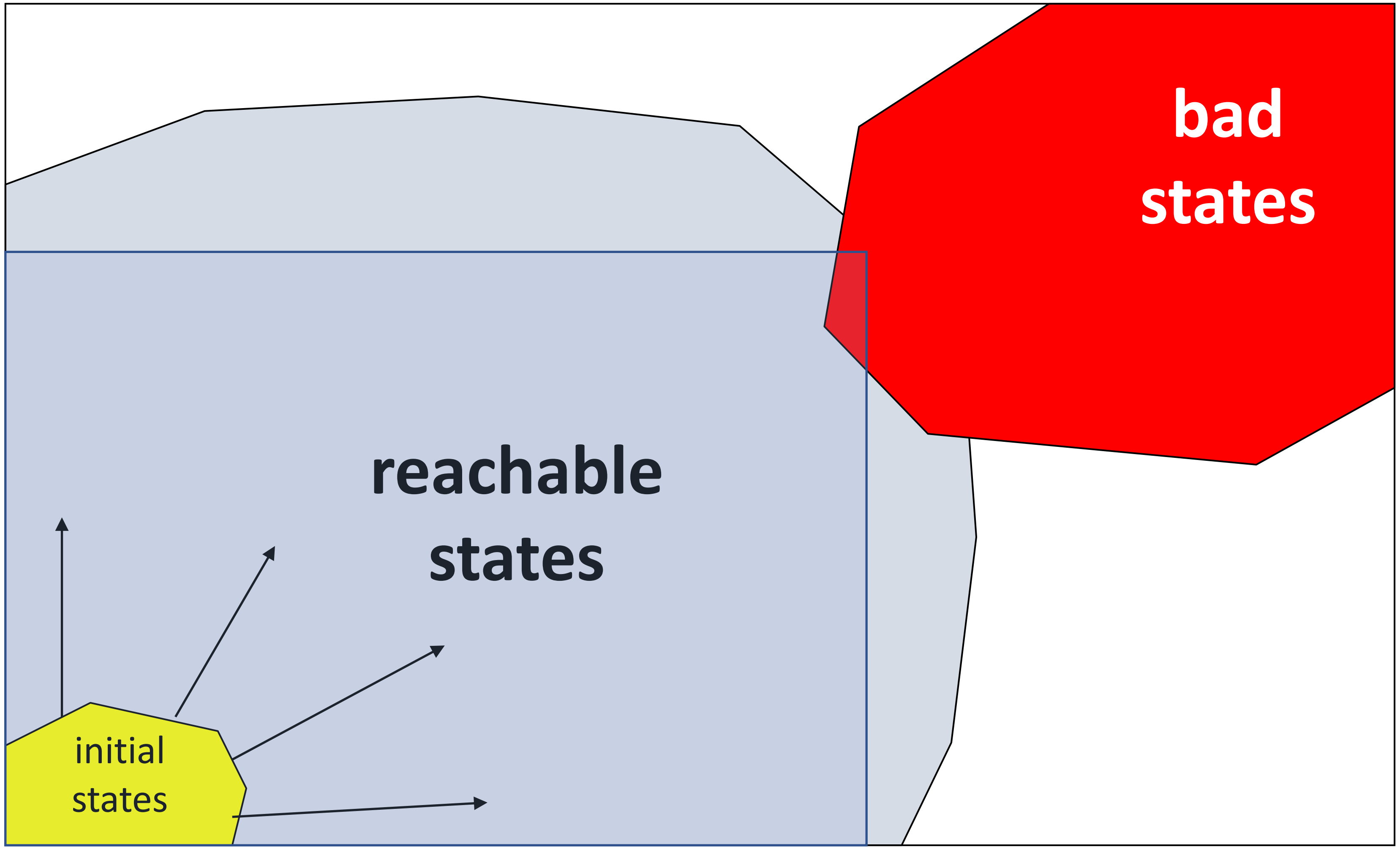
Goal: exploring program states



Goal: exploring abstract states



Goal: exploring abstract states



Soundness & Completeness

- Soundness: Catch all violations
  - Over-approximate behavior
  - May result in false positives
- Completeness: Only catch actual violations
  - Under-approximate behavior
  - May result in false negatives

Static Analysis

Program as a Data

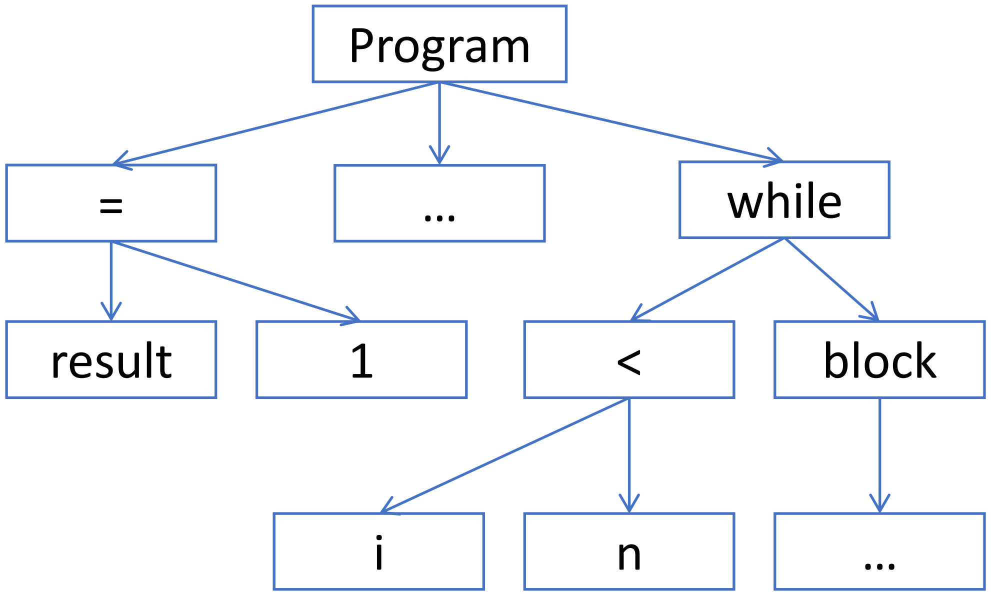
- Discovers properties of a program (or a program point) by reasoning statically the program (without running the program)
- Discovered properties can be:
  - the program terminates.
  - arrays are always accessed within their bounds.
  - a variable **p** at line 10 cannot have a null pointer.
  - a variable **p** at line 10 cannot be a dangling reference.

Abstract Syntax Trees

- Tree representation of the syntactic structure of source code.
  - Parsers convert concrete syntax into abstract syntax, and deal with resulting ambiguities.
- Records only the semantically relevant information.
  - Abstract: doesn’t represent every detail (like parentheses); these can be inferred from the structure.

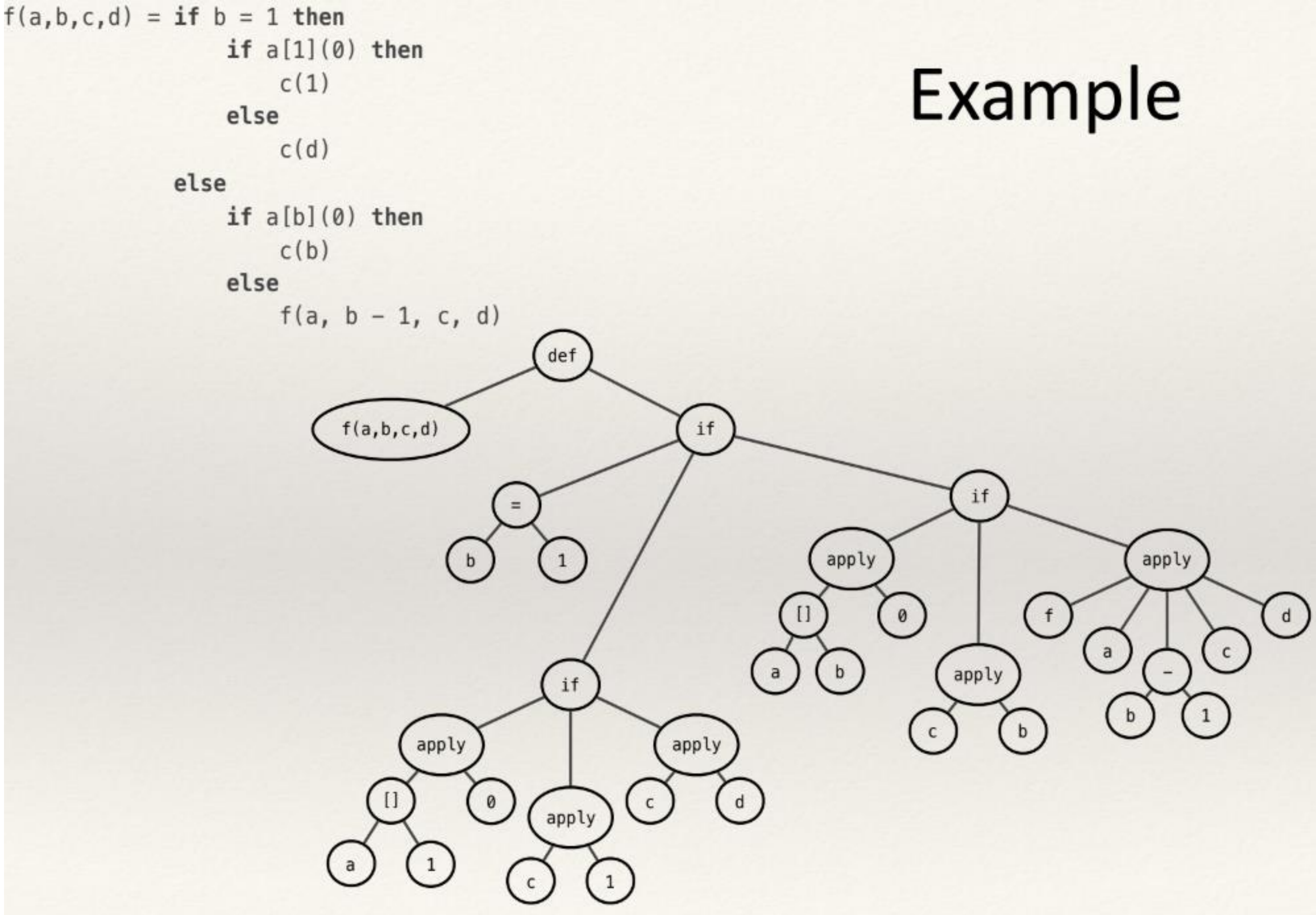
Abstract Syntax Trees

```
int result = 1;
int i = 2;
while (i < n) {
  result *= i;
  i++;
}
return result;
```



That is what your IDE and compiler are doing

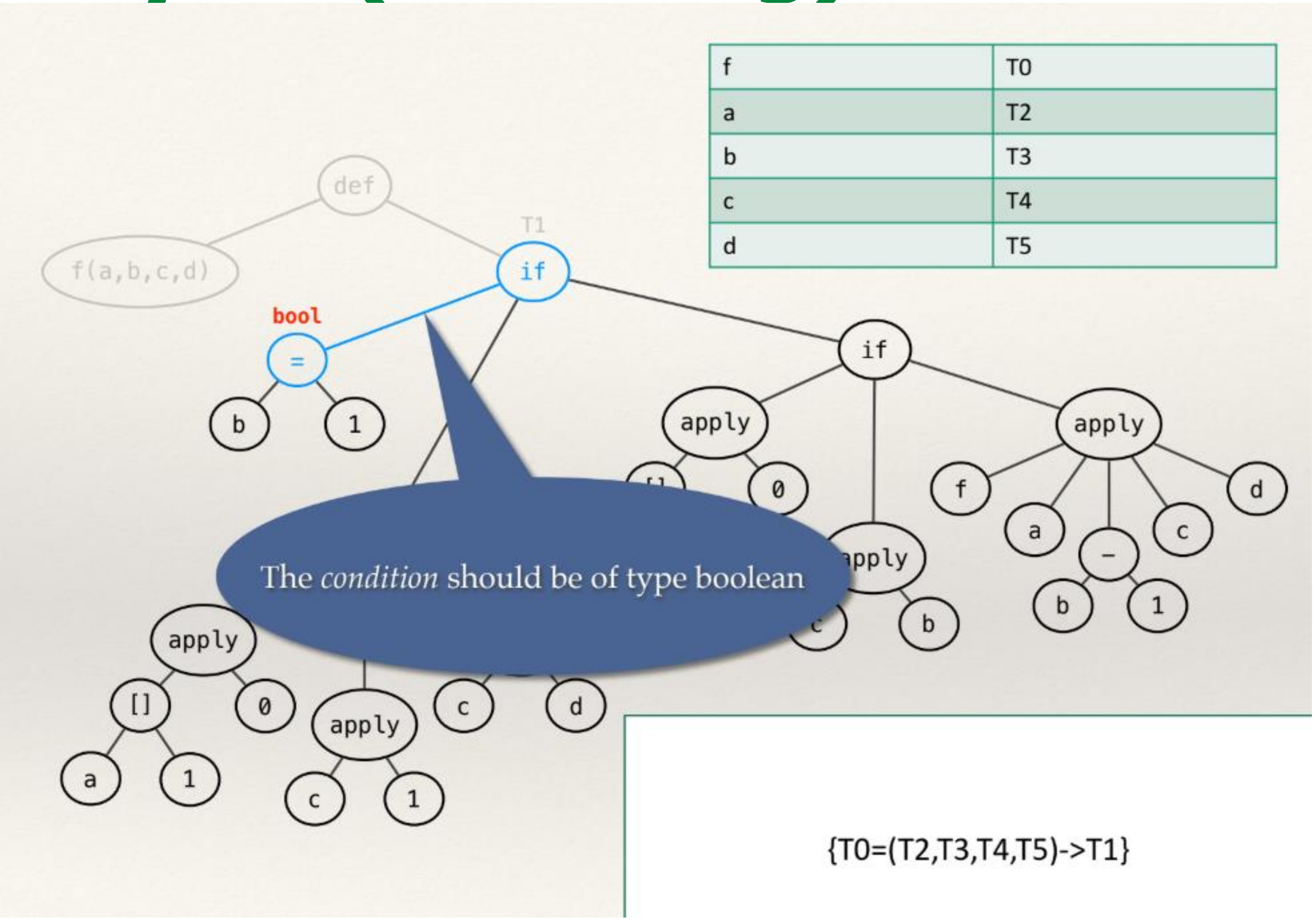
Type analysis (checking)



```
f(a,b,c,d) = if b = 1 then
                 if a[1](0) then
                     c(1)
                 else
                     c(d)
             else
                 if a[b](0) then
                     c(b)
                 else
                     f(a, b - 1, c, d)
```

Example

Type analysis (checking)

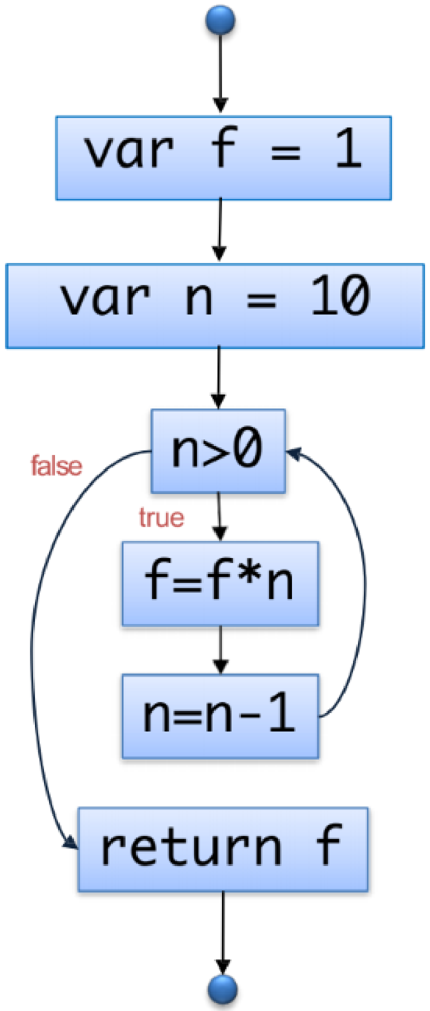


f	T0
a	T2
b	T3
c	T4
d	T5

The *condition* should be of type boolean

{T0=(T2,T3,T4,T5)->T1}

Sign analysis

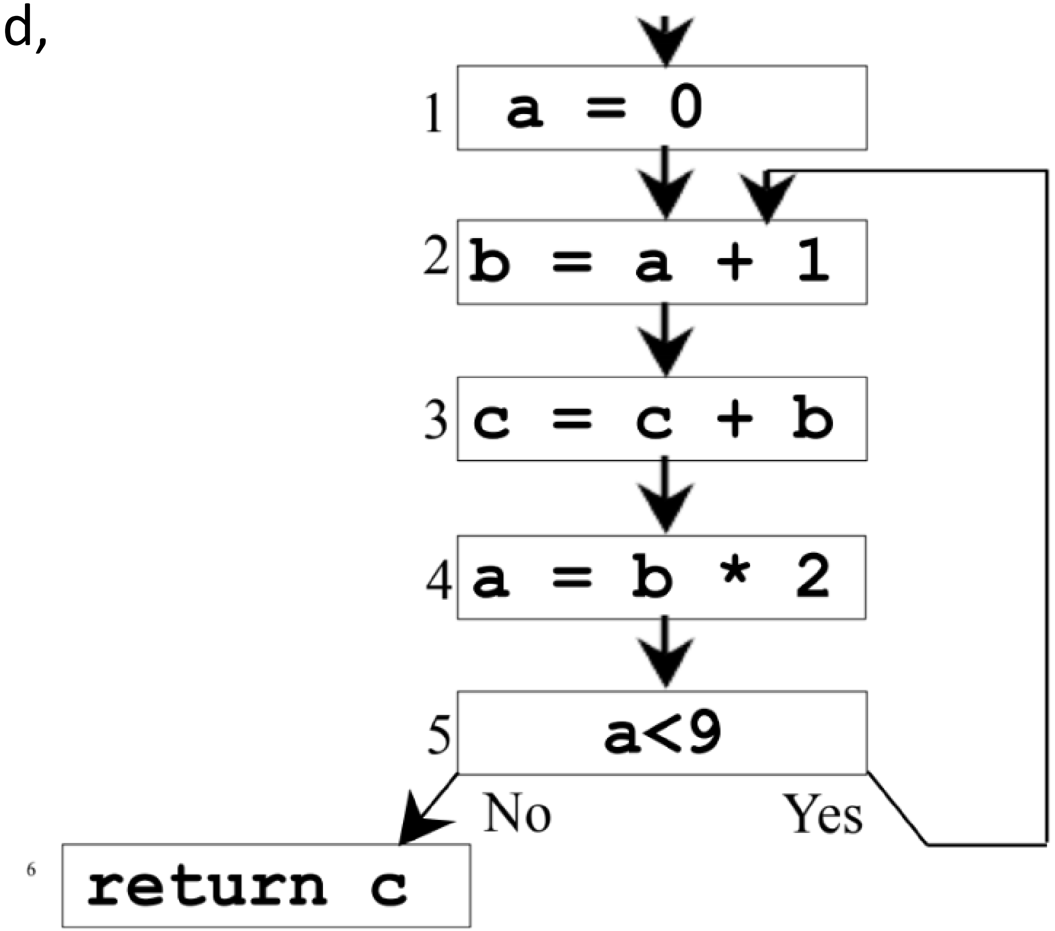


1st iteration	2nd iteration
$x_1=[f \mapsto +, n \mapsto \bot]$	$x_1=[f \mapsto +, n \mapsto \bot]$
$x_2=[f \mapsto +, n \mapsto +]$	$x_2=[f \mapsto +, n \mapsto +]$
$x_3=[f \mapsto +, n \mapsto +]$	$x_3=[f \mapsto +, n \mapsto ?]$
$x_4=[f \mapsto +, n \mapsto +]$	$x_4=[f \mapsto ?, n \mapsto ?]$
$x_5=[f \mapsto +, n \mapsto ?]$	$x_5=[f \mapsto ?, n \mapsto ?]$
$x_6=[f \mapsto +, n \mapsto +]$	$x_6=[f \mapsto +, n \mapsto ?]$

Liveness analysis

A variable is live at a particular point in the program if its value at that point will be used in the future (dead, otherwise).

- dead code elimination



Wrap up

- What have you learnt?
- What for this could be useful?