# EVALUATING TARGET EVENT SEQUENCE GENERATION FOR ANDROID APPLICATIONS

Authors:

Casper S. Jensen
Aarhus University, Denmark
semadk@cs.au.dk

Mukul R. Prasad
Fujitsu Laboratories
of America, USA
mukul@us.fujitsu.com

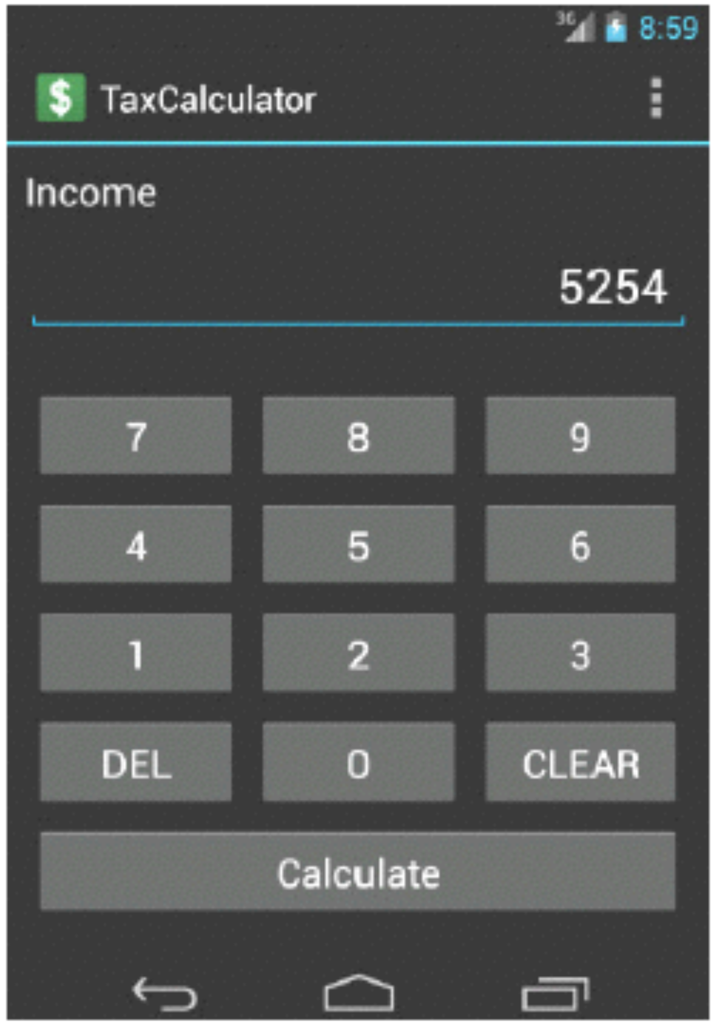
Anders Møller
Aarhus University, Denmark
amoeller@cs.au.dk

*—as a part of completion for Group Paper Presentation for CMPSCI 621—*

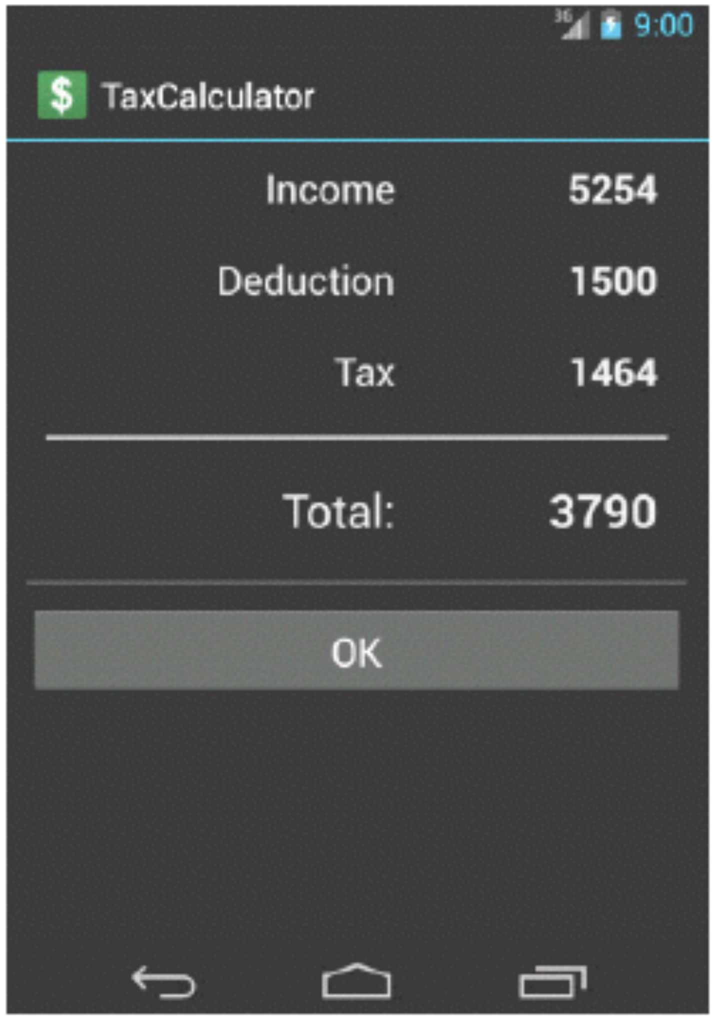
slide author names omitted for FERPA compliance

## INTRODUCTION

# MOTIVATING EXAMPLE : TAXCALCULATOR



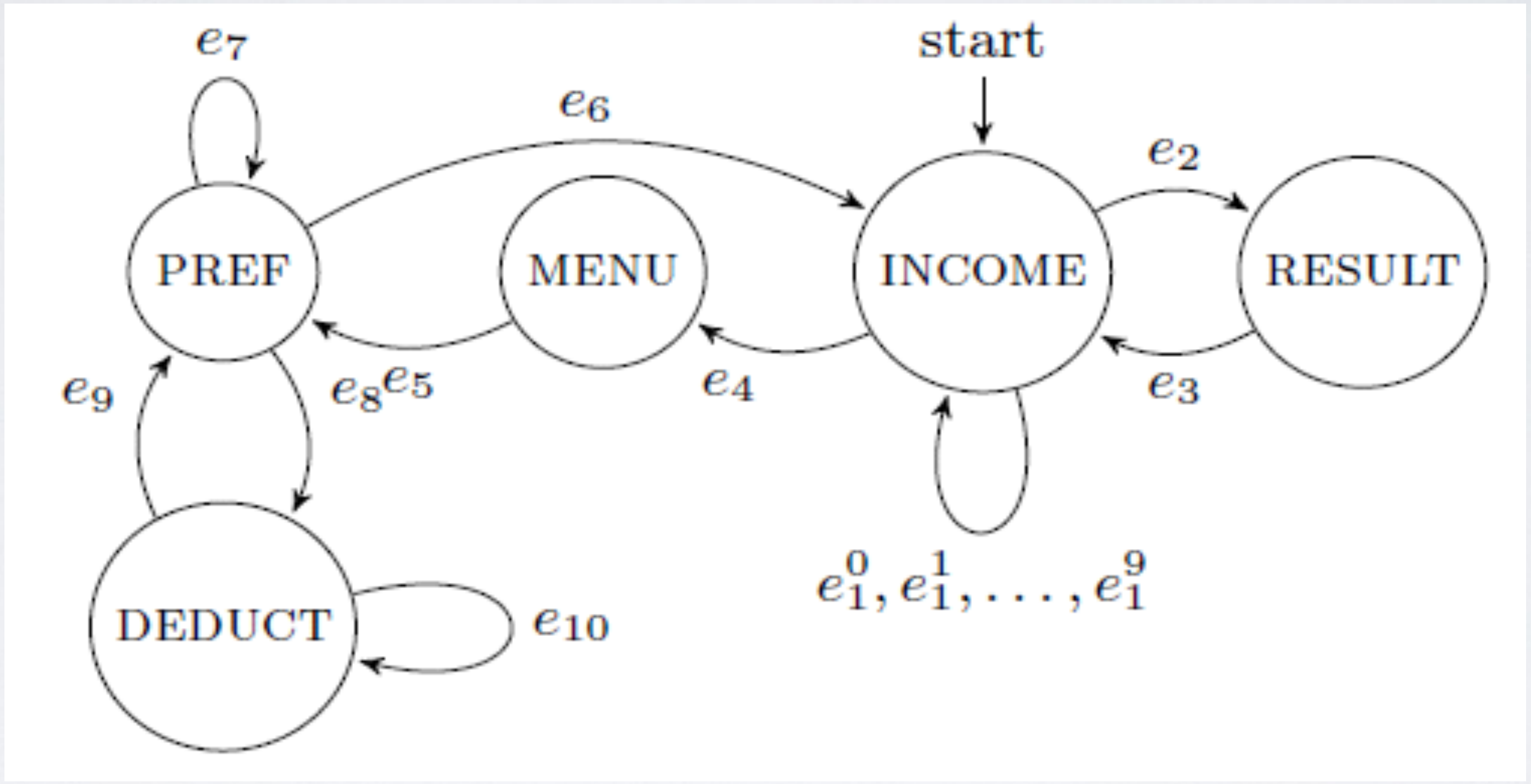
(a) INCOME



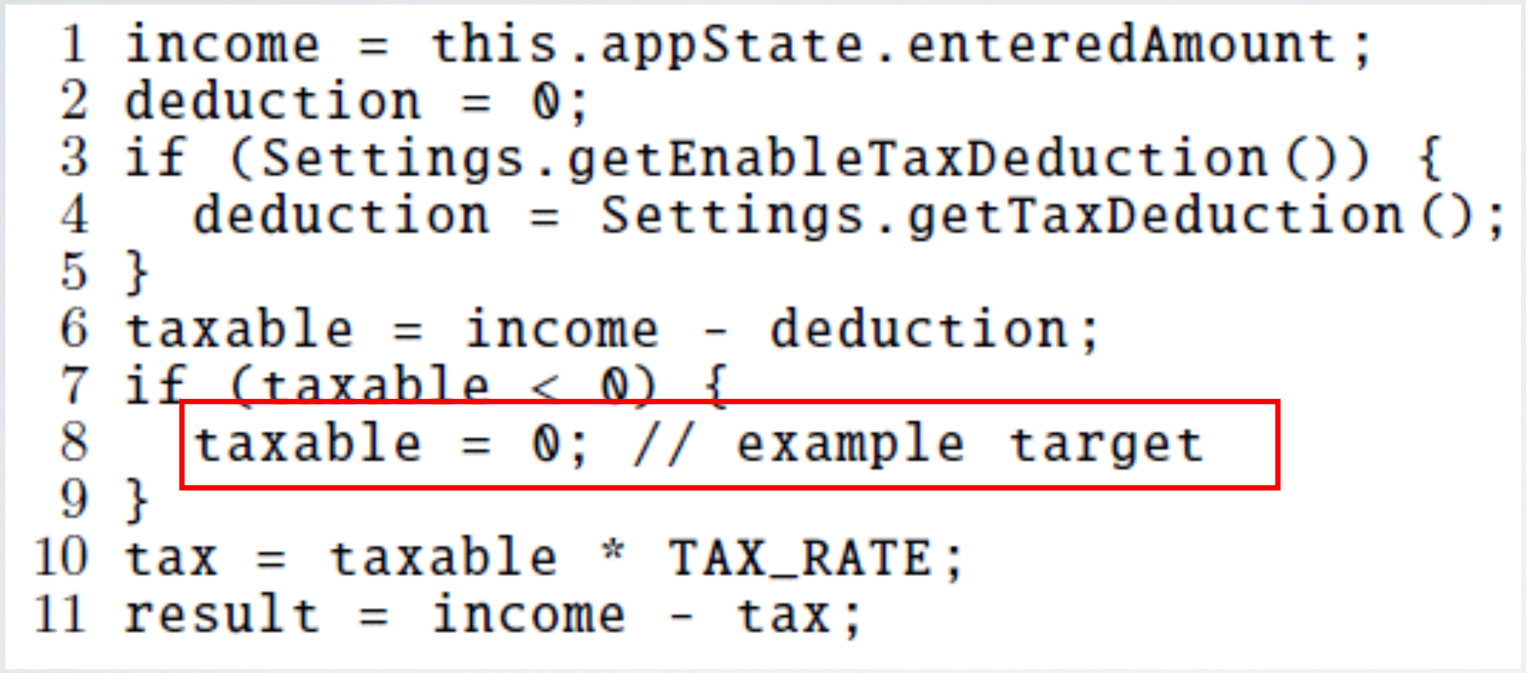
|  |  |
|---|---|
| Income | 5254 |
| Deduction | 1500 |
| Tax | 1464 |
| Total: | 3790 |

(b) RESULT

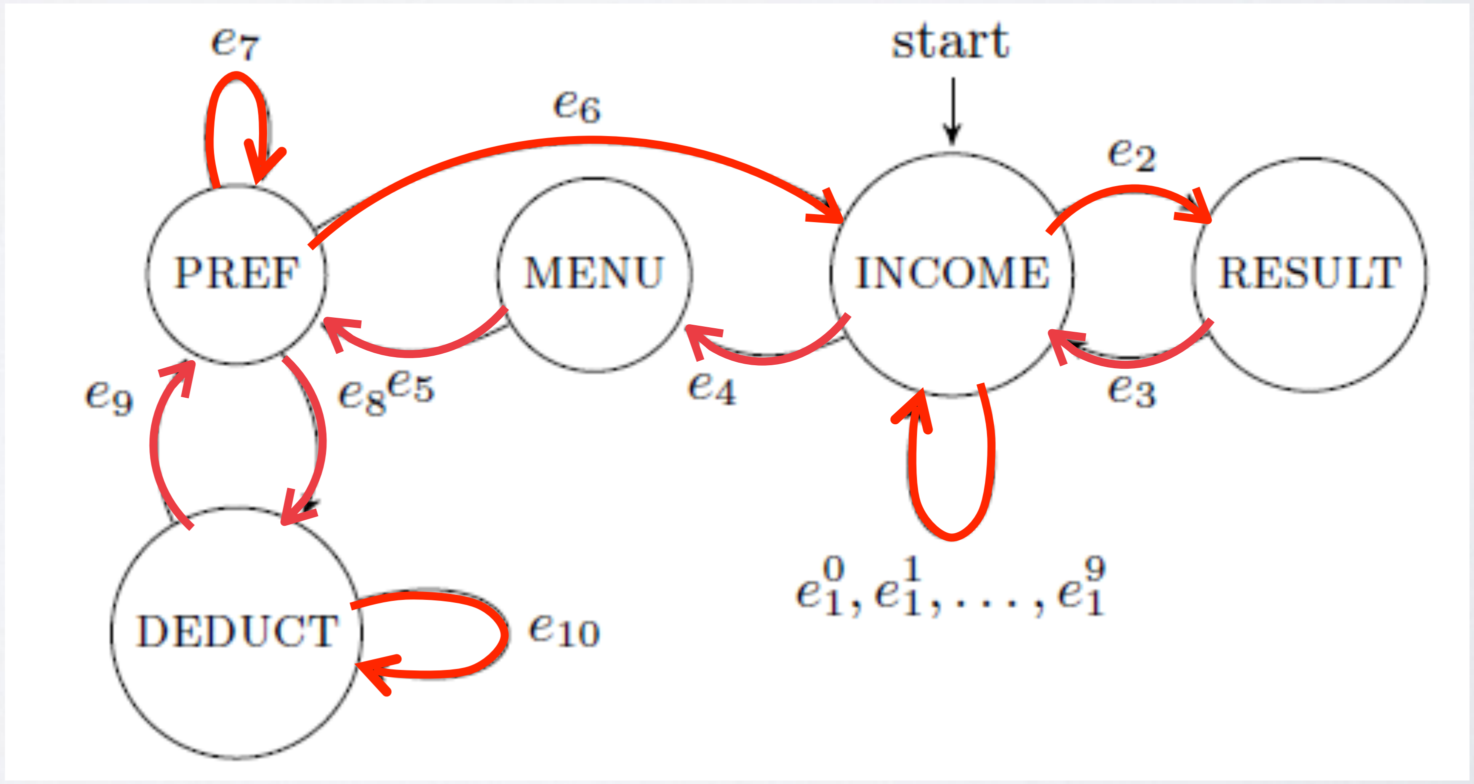
# UI MODEL OF A PART OF TAXCALCULATOR



# MOTIVATING EXAMPLE - CONTINUED



```
income = this.appState.enteredAmount;
deduction = 0;
if (Settings.getEnableTaxDeduction()) {
  deduction = Settings.getTaxDeduction();
}
taxable = income - deduction;
if (taxable < 0) {
  taxable = 0; // example target
}
tax = taxable * TAX_RATE;
result = income - tax;
```



## WHAT IS THE PROBLEM?

- Automated generation of event sequence for event-driven applications for higher code coverage

**Previous Works:**

- Random Black-box testing [1]
- Techniques involving Symbolic Execution [2]

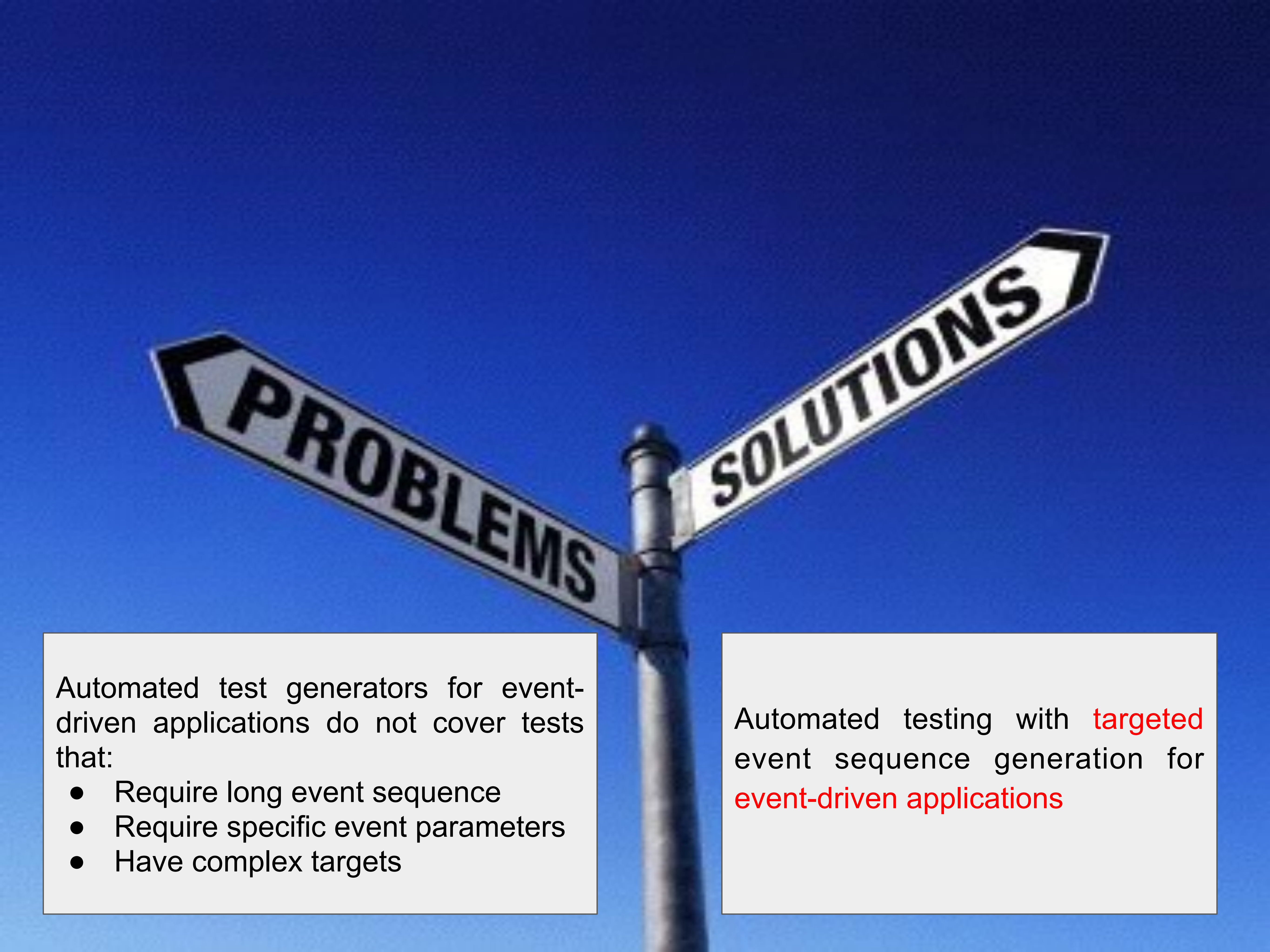
[1] C Hu et al Automating UI Testing for Android Applications in International workshop on Automation of Software Test

[2] Mirzaei et al Testing Android apps through Symbolic execution in ACM SIGSOFT Software Testing noted

## RESEARCH QUESTION

Is it possible to generate event sequences for complex targets that require long event sequence and highly constrained event parameters?

## KEY IDEA & APPROACH OVERVIEW



Automated test generators for event-driven applications do not cover tests that:

- Require long event sequence
- Require specific event parameters
- Have complex targets

Automated testing with targeted event sequence generation for event-driven applications

## Solution

# Collider

(A Targeted Event

Sequence Generation Tool)

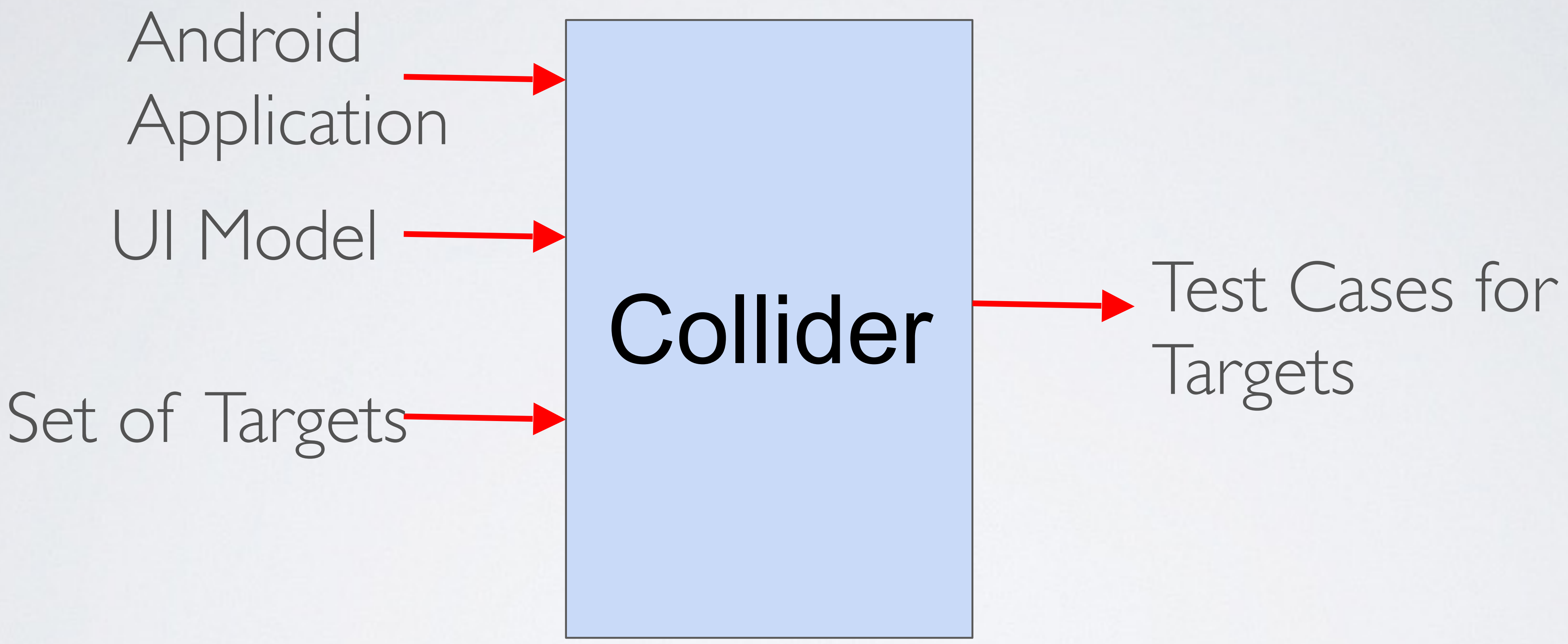
Inspirations:

- Line reachability problem for C programs [3]
- A Model-based testing technique [4]

[3] K. Ma et al, Directed Symbolic Execution, In Proc. 18th International Static Analysis Symposium, 2011

[4] S. Arlt et al, Lightweight Static Analysis or GUI Testing, In Proc. 23rd IEEE International Symposium on Software Reliability Engineering, 2011

# HIGH-LEVEL VIEW OF COLLIDER



## UI MODELS FOR EVENT-DRIVEN APPLICATIONS

UI Model of an event-driven application is a collection of event-handler methods that attach to GUI widgets

Event-handler registration
=
(GUI Widget Object, Event Kind, Event-handler Method)

# UI MODELS FOR EVENT-DRIVEN APPLICATIONS (CONT.)

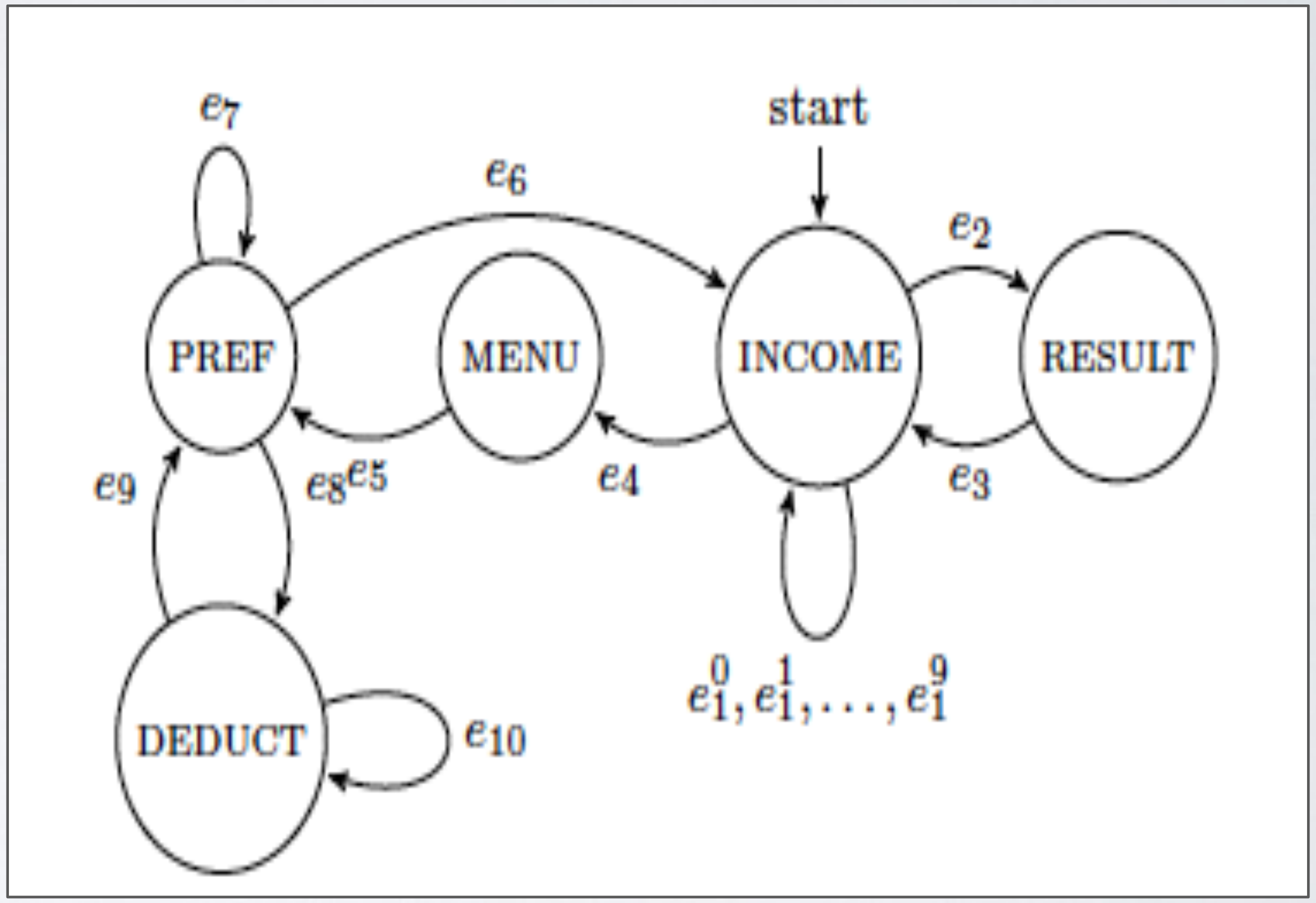
transition relation

initial state

- $M = (S, s_0, E, T)$

finite set of abstract states

finite set of event-handler registrations



# APPROACH OVERVIEW

1. Symbolic Summarization Phase (Symbolic analysis)
    1. **Input**: executable android application, event-handlers
    2. **Output**: event-handler summary

2. Sequence Generation Phase (Backward exploration)
    1. **Input**: event-handler summary, UI model
    2. **Output**: test case

## THE PROCESS

## 1. SYMBOLIC SUMMARIZATION PHASE

- Goal: to produce event handler summary for each event handler
  - Event handler summary = a set of path summaries for all the execution paths within the event handler code

- The summary is computed by performing concolic testing [5]

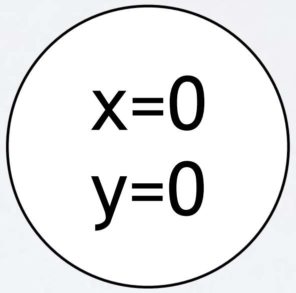
[5] Wikipedia: Concolic testing, https://en.wikipedia.org/wiki/Concolic_testing

# I. SYMBOLIC SUMMARIZATION PHASE

- Goal: to produce event handler summary for each event handler
  - Event handler summary = a set of path summaries for all the execution paths within the event handler code
- The summary is computed by performing concolic testing

```c
void f(int x, int y) {
    int z = 2*y;
    if ( x  == 100 ) {
        if (x < z) {
            assert(0); /* error */
        }
    }
}
```

< Example code written in C >

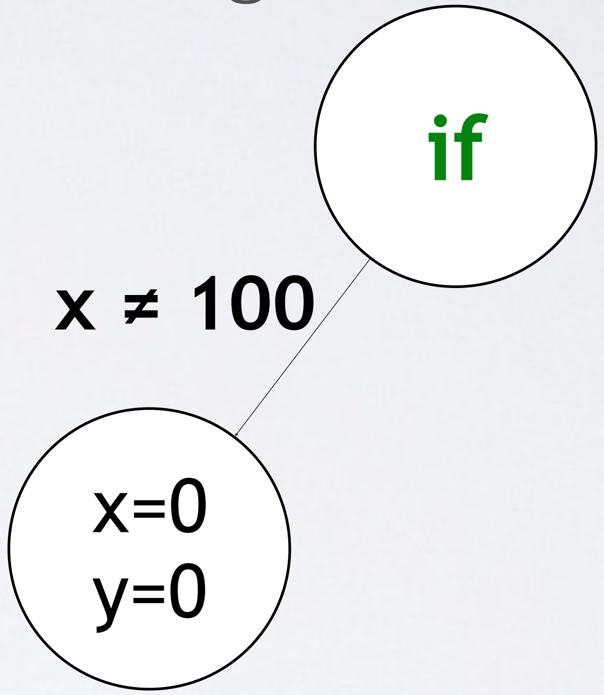


# I. SYMBOLIC SUMMARIZATION PHASE

- Goal: to produce event handler summary for each event handler
  - Event handler summary = a set of path summaries for all the execution paths within the event handler code
- The summary is computed by performing concolic testing

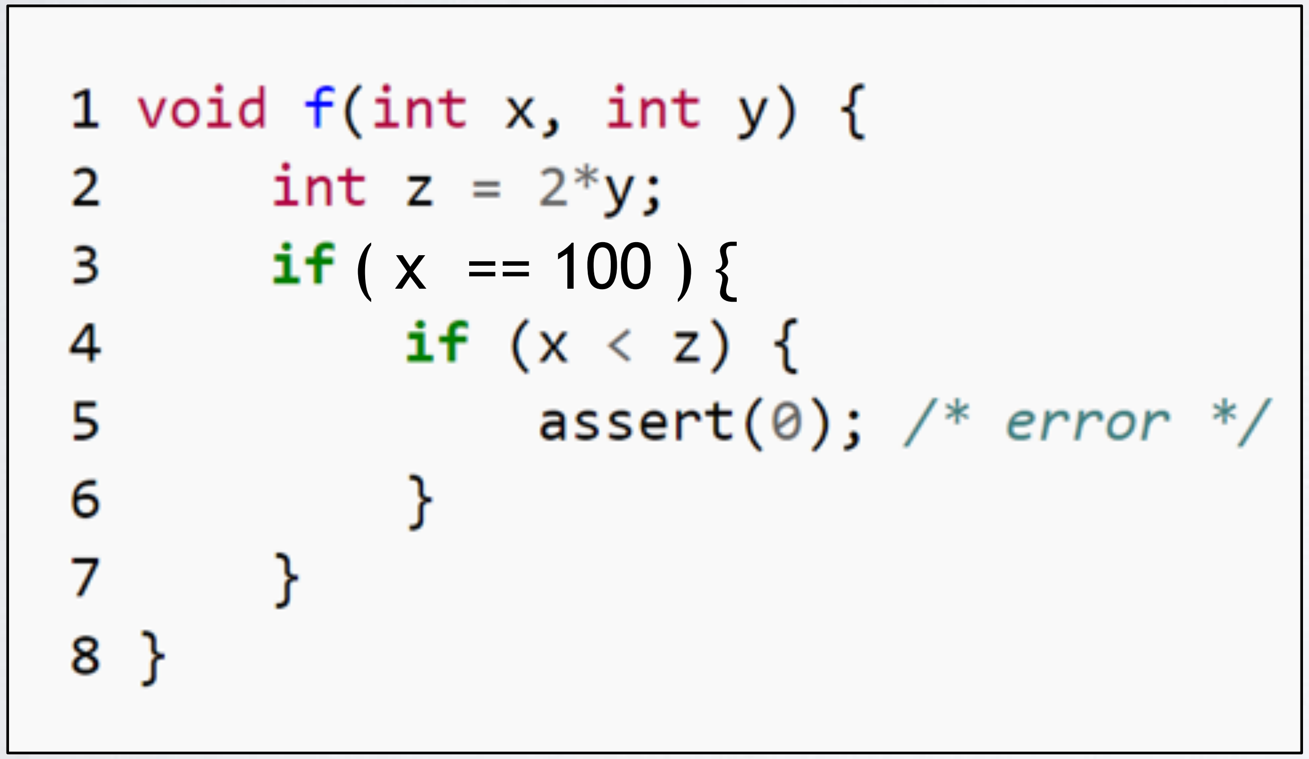
```c
void f(int x, int y) {
    int z = 2*y;
    if ( x  == 100 ) {
        if (x < z) {
            assert(0); /* error */
        }
    }
}
```

< Example code written in C >



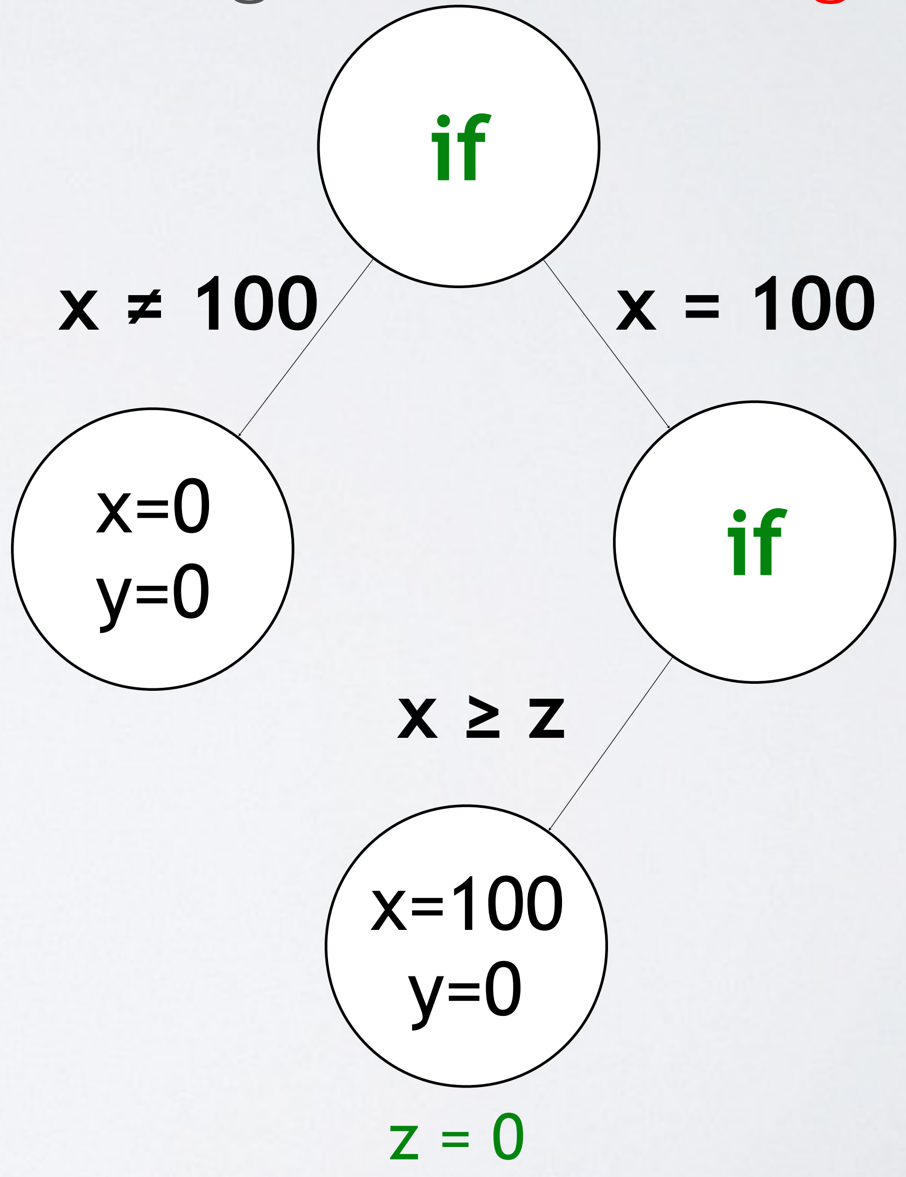
## I. SYMBOLIC SUMMARIZATION PHASE

- Goal: to produce event handler summary for each event handler
  - Event handler summary = a set of path summaries for all the execution paths within the event handler code
- The summary is computed by performing concolic testing



```
void f(int x, int y) {
    int z = 2*y;
    if ( x  == 100 ) {
        if (x < z) {
            assert(0); /* error */
        }
    }
}
```

< Example code written in C >

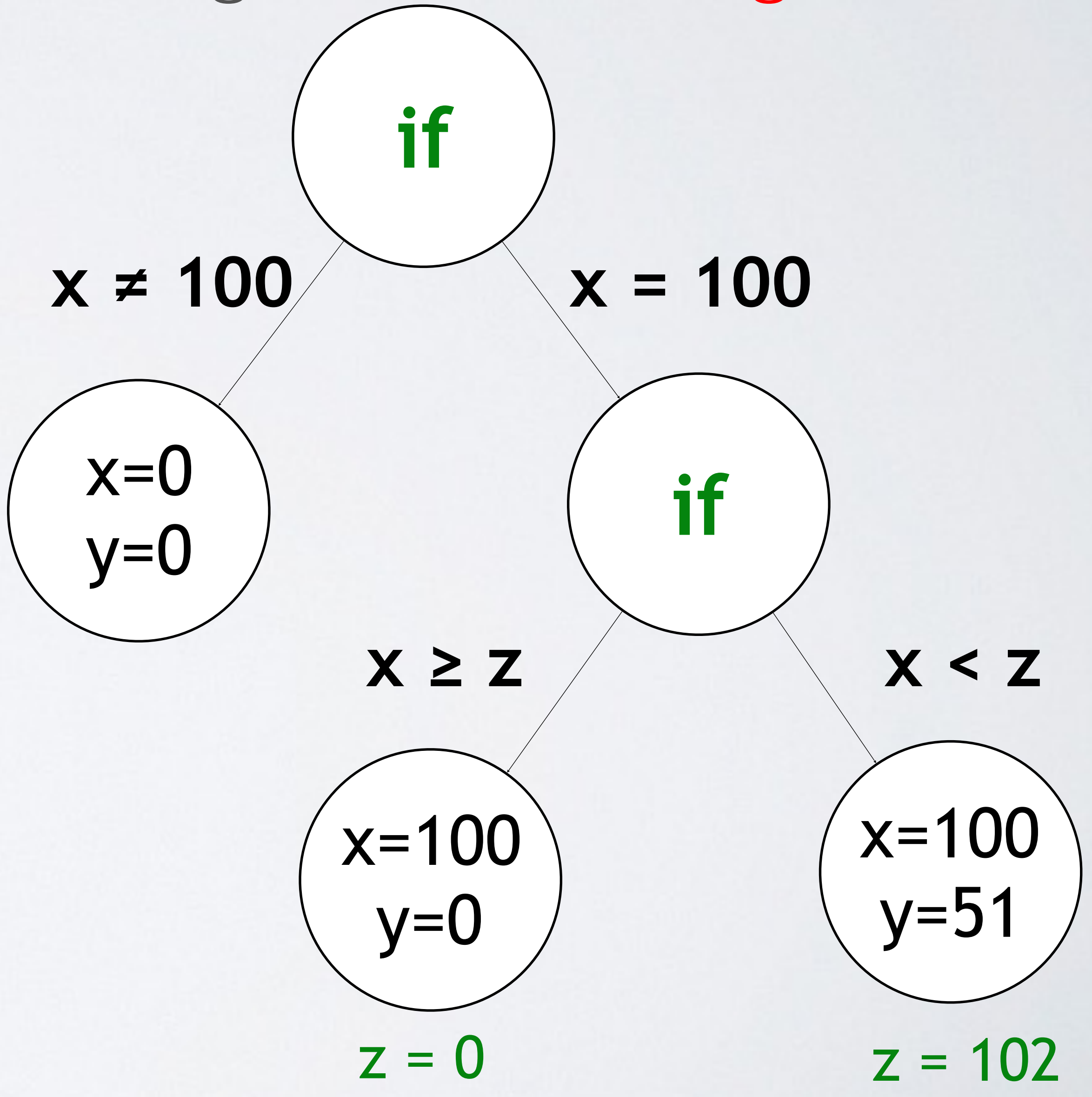


## I. SYMBOLIC SUMMARIZATION PHASE

- Goal: to produce event handler summary for each event handler
  - Event handler summary = a set of path summaries for all the execution paths within the event handler code
- The summary is computed by performing concolic testing

```c
void f(int x, int y) {
    int z = 2*y;
    if ( x  == 100 ) {
        if (x < z) {
            assert(0); /* error */
        }
    }
}
```

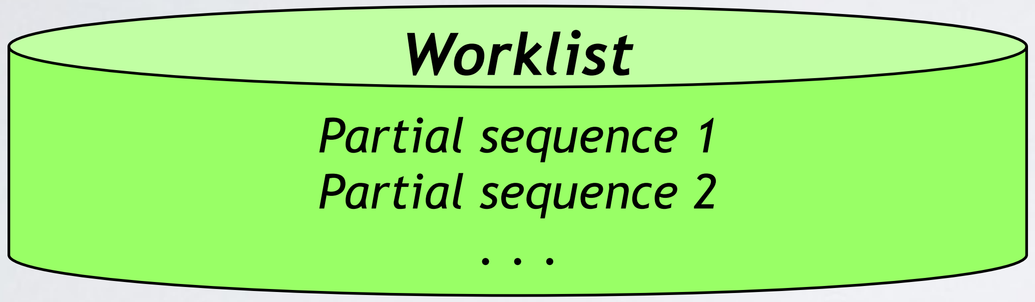
< Example code written in C >



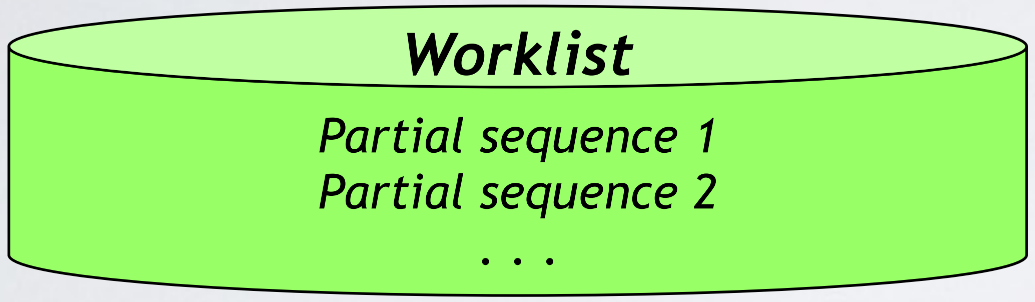
## 2. SEQUENCE GENERATION PHASE

- Goal: to generate a test case for each given ***target***

# 2. SEQUENCE GENERATION PHASE

- Goal: to generate a test case for each given ***target***
- *Partial sequence* = (a path summary + an abstract state)
- 

# 2. SEQUENCE GENERATION PHASE

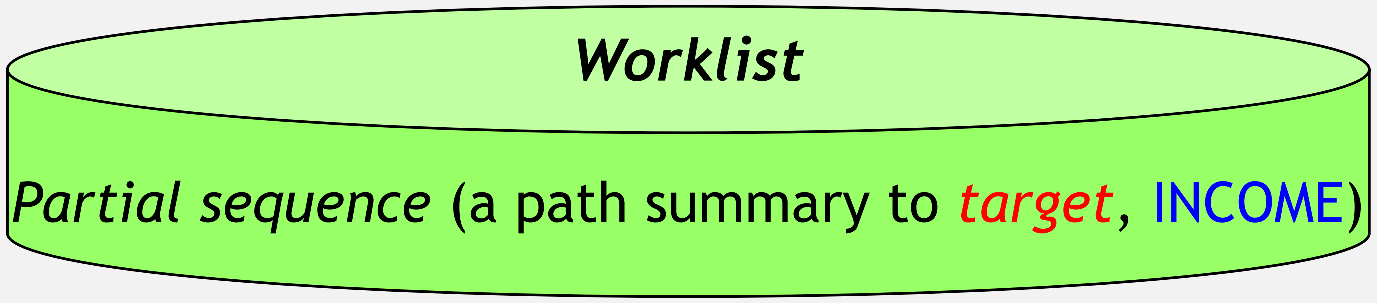
- Goal: to generate a test case for each given ***target***
- *Partial sequence* = (a path summary + an abstract state)
- 

-

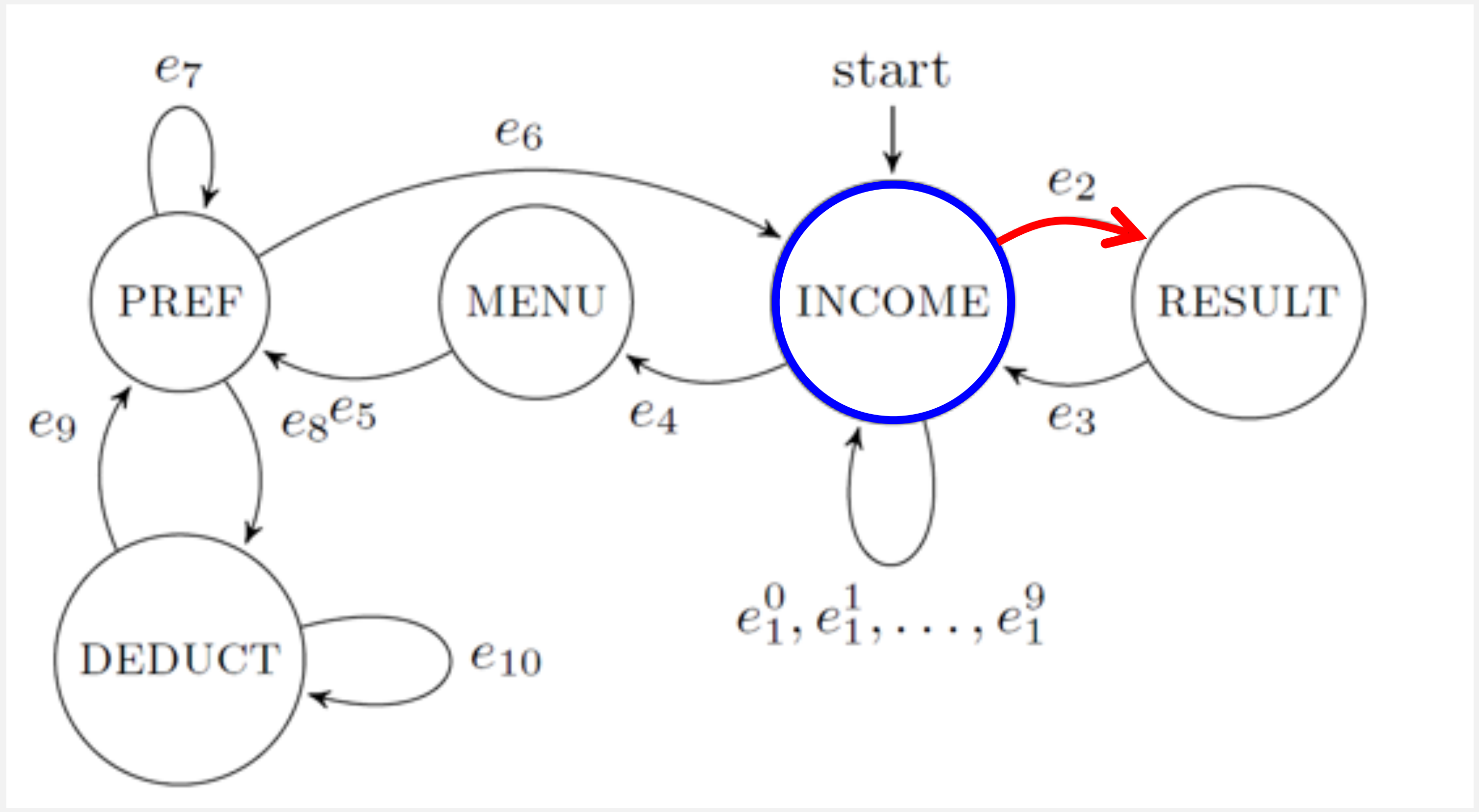
```
Initialize the worklist
while (worklist is not empty
    identify anchors
    for each anchor,
        identify connector sequences
        construct a new partial sequence
        if (new partial sequence == test case)
            return new partial sequence
        else
            add new partial sequence to Worklist
        Reprioritize Worklist
return false
```

# INITIALIZE THE WORKLIST

- *path summary that triggers the **target***
- *abstract state which has an outgoing transition to the path summary*



*target = event handler for $e_2$*

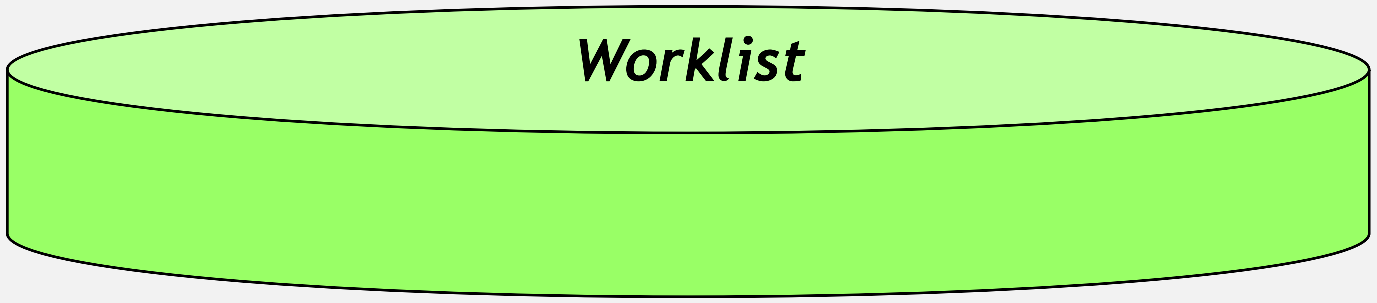


# IDENTIFY ANCHORS

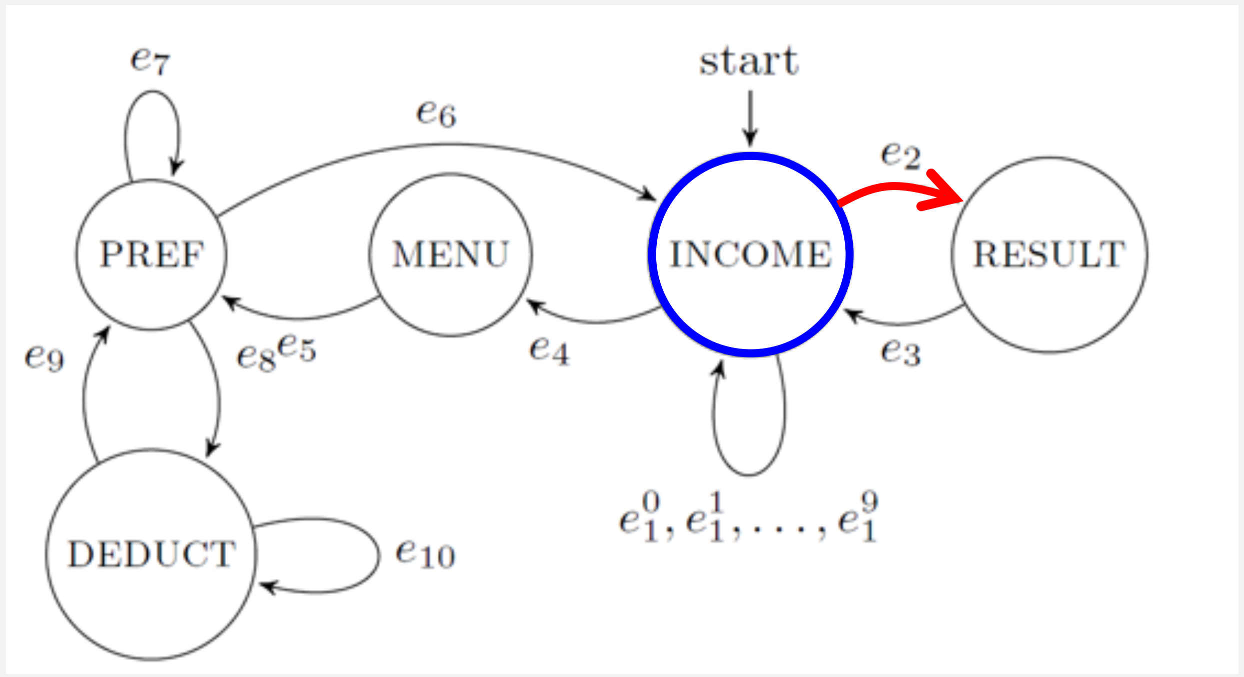
- path summaries that affect the path condition of the partial sequence are identified as anchors.

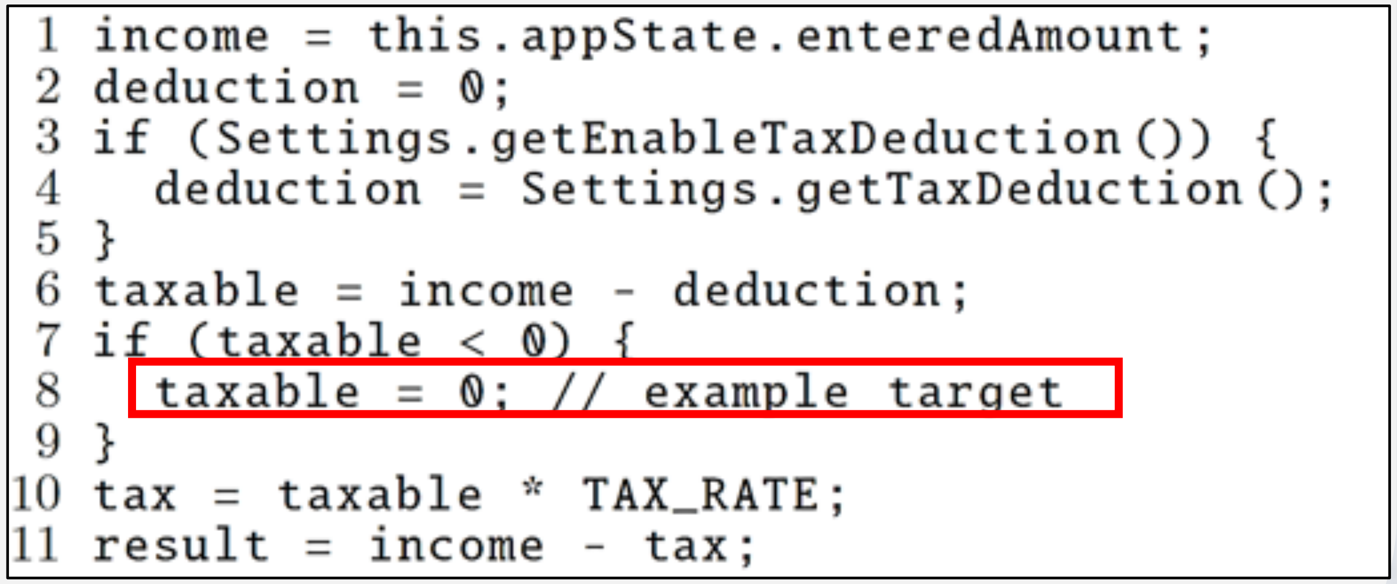
# IDENTIFY ANCHORS

- path summaries that affect the path condition of the partial sequence are identified as anchors.



***Anchor*** *= ?*

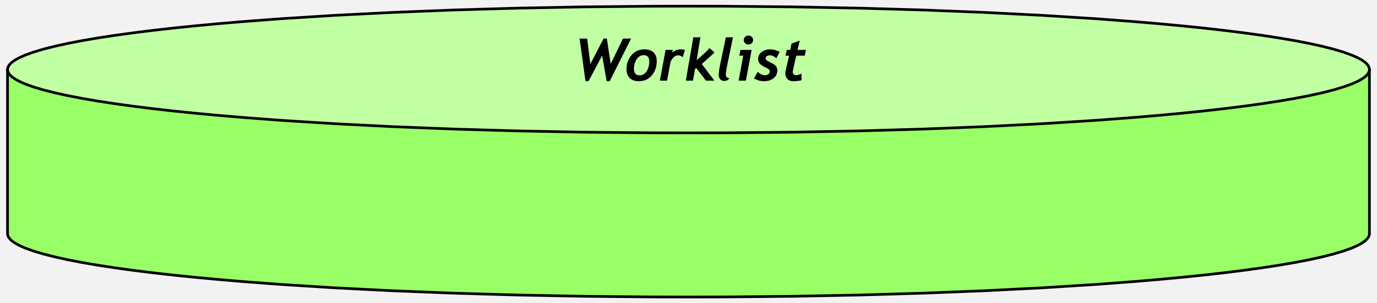




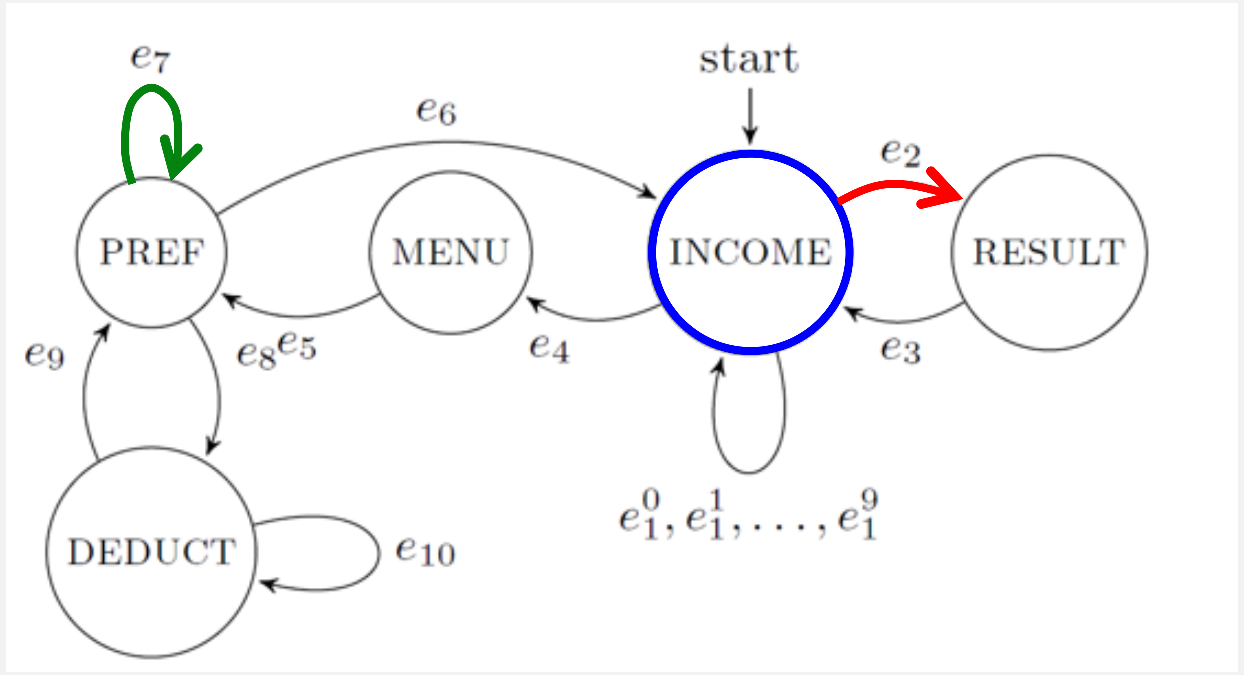
```
income = this.appState.enteredAmount;
deduction = 0;
if (Settings.getEnableTaxDeduction()) {
  deduction = Settings.getTaxDeduction();
}
taxable = income - deduction;
if (taxable < 0) {
  taxable = 0; // example target
}
tax = taxable * TAX_RATE;
result = income - tax;
```

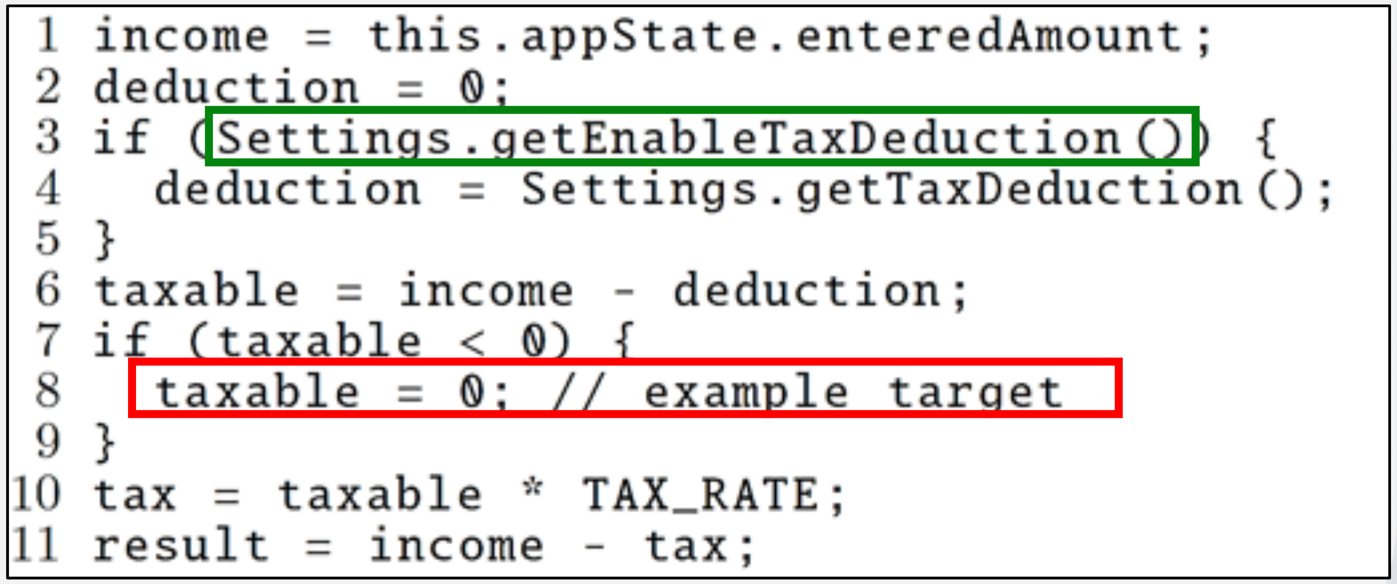
# IDENTIFY ANCHORS

- path summaries that affect the path condition of the partial sequence are identified as anchors.



***Anchor*** *= ?*

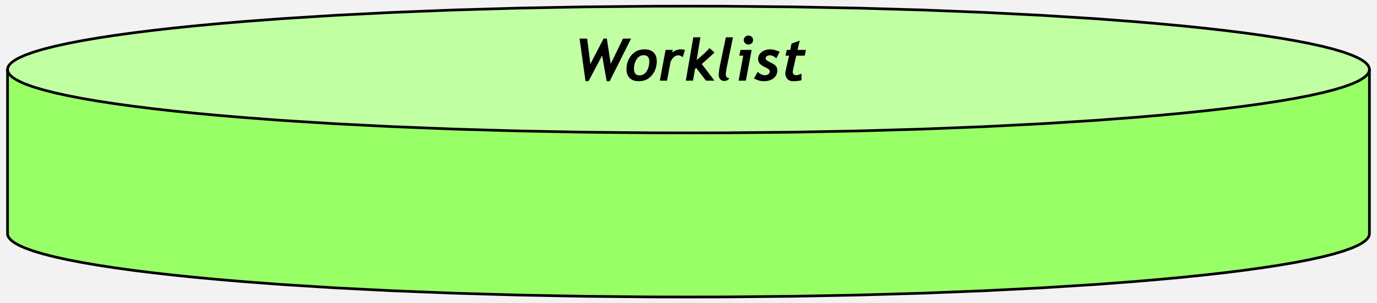




```
income = this.appState.enteredAmount;
deduction = 0;
if (Settings.getEnableTaxDeduction()) {
  deduction = Settings.getTaxDeduction();
}
taxable = income - deduction;
if (taxable < 0) {
  taxable = 0; // example target
}
tax = taxable * TAX_RATE;
result = income - tax;
```

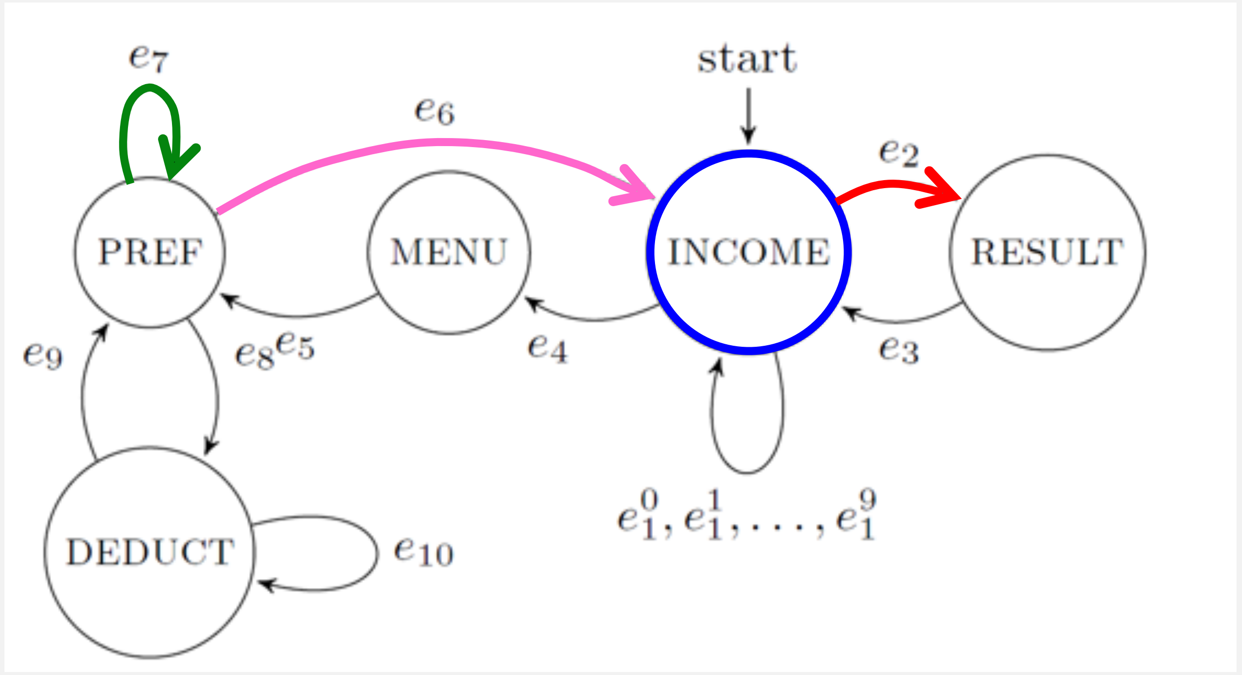
# IDENTIFY CONNECTOR SEQUENCES

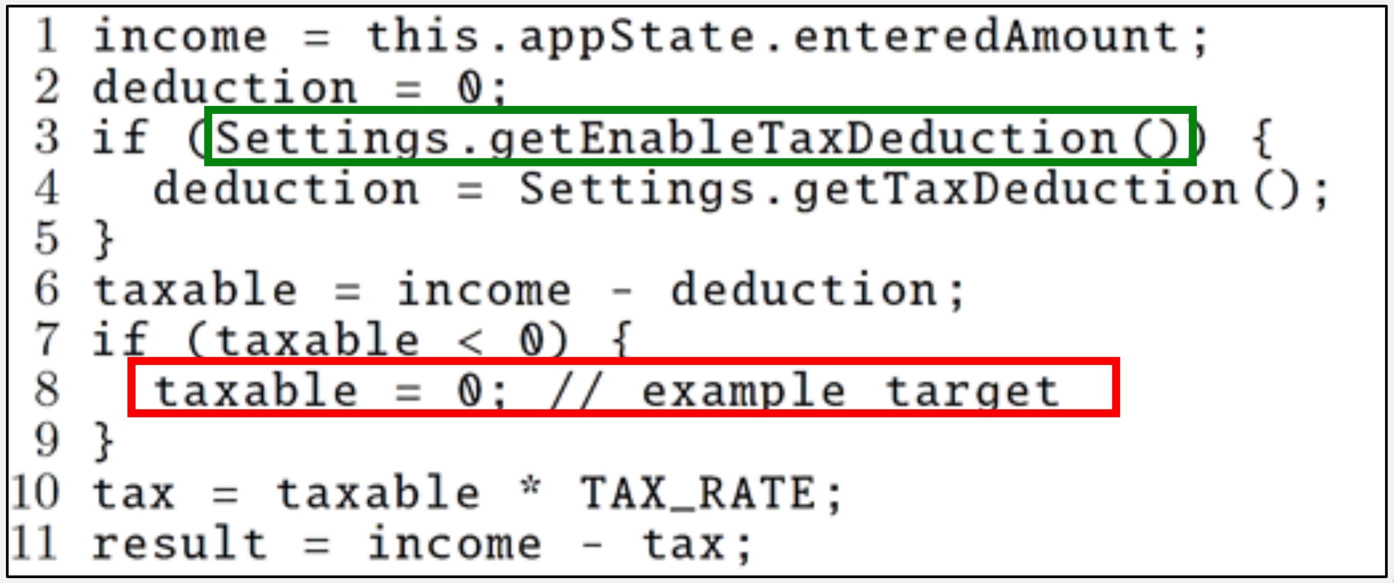
- For each given anchor, a set of connector sequences that lead from ***the anchor*** to the ***partial sequence*** is extracted by using the UI model



***Anchor*** *= path summaries for* $e_7$

***Connector*** *= path summaries for* $e_6$

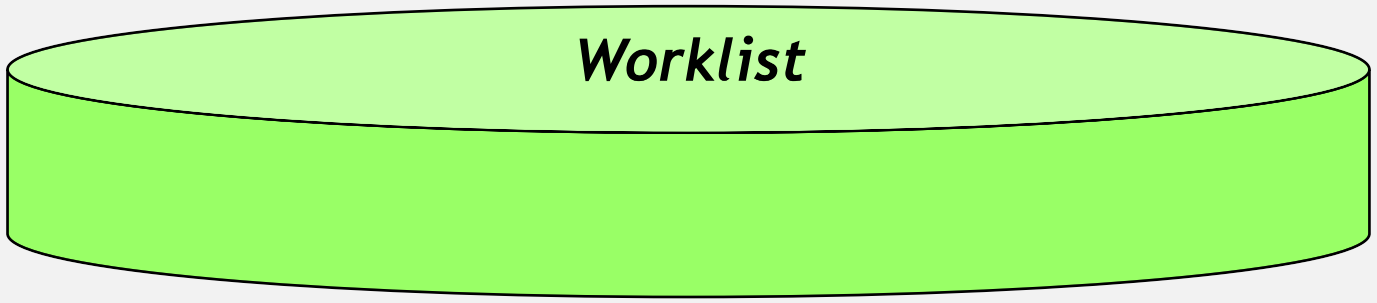




```
income = this.appState.enteredAmount;
deduction = 0;
if (Settings.getEnableTaxDeduction()) {
  deduction = Settings.getTaxDeduction();
}
taxable = income - deduction;
if (taxable < 0) {
  taxable = 0; // example target
}
tax = taxable * TAX_RATE;
result = income - tax;
```

# IDENTIFY CONNECTOR SEQUENCES

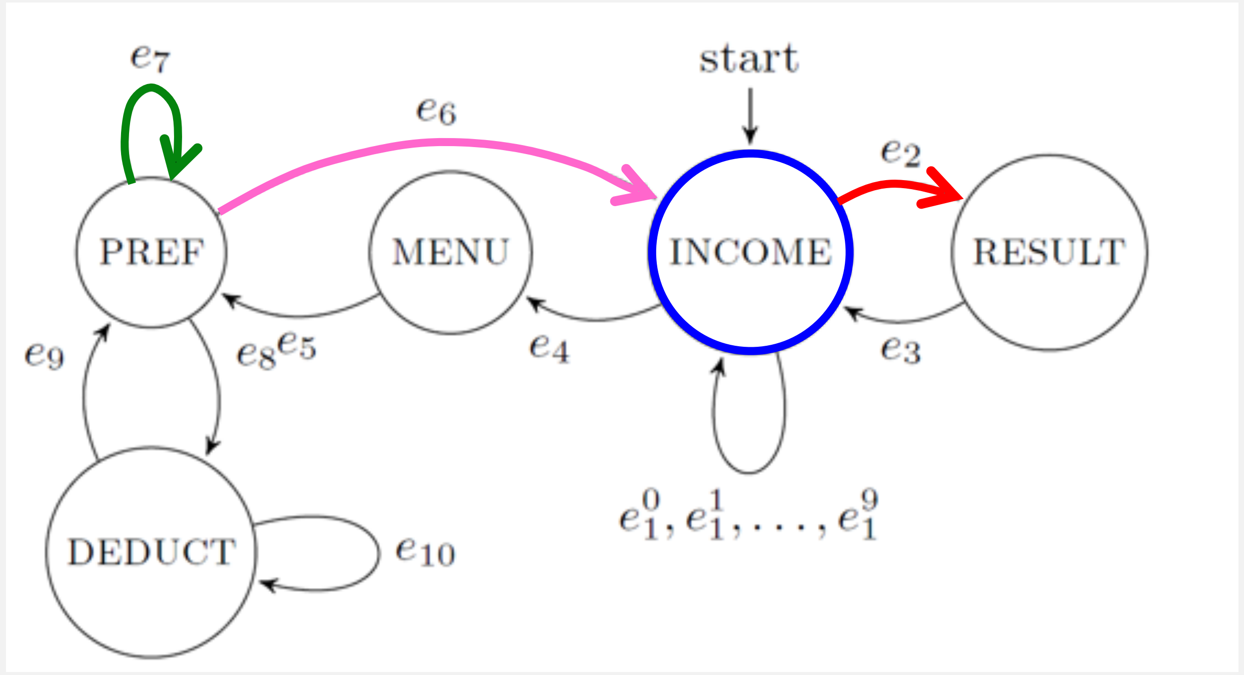
- For each given anchor, a set of connector sequences that lead from ***the anchor*** to the ***partial sequence*** is extracted by using the UI model



***Anchor*** *= path summaries for $e_7$*

***Connector*** *= path summaries for $e_6$*

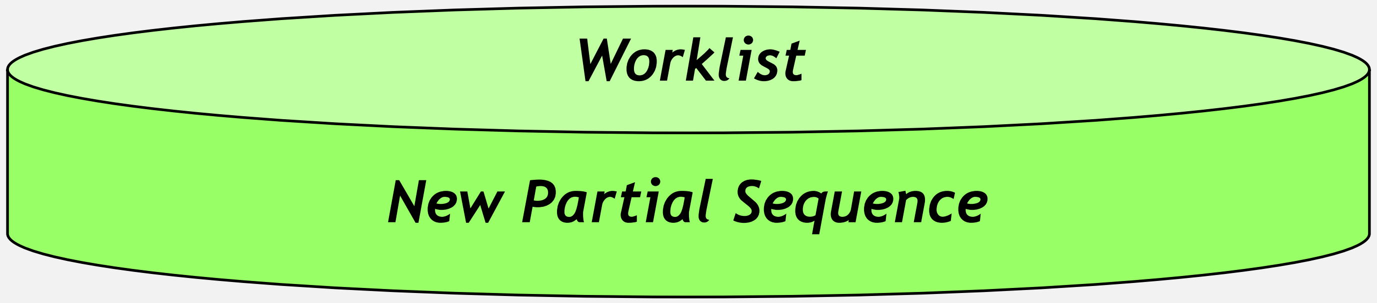
***New Partial Sequence*** =
(*Anchor*, *Connector*, *original partial sequence*)



```
income = this.appState.enteredAmount;
deduction = 0;
if (Settings.getEnableTaxDeduction()) {
  deduction = Settings.getTaxDeduction();
}
taxable = income - deduction;
if (taxable < 0) {
  taxable = 0; // example target
}
tax = taxable * TAX_RATE;
result = income - tax;
```

# IDENTIFY CONNECTOR SEQUENCES

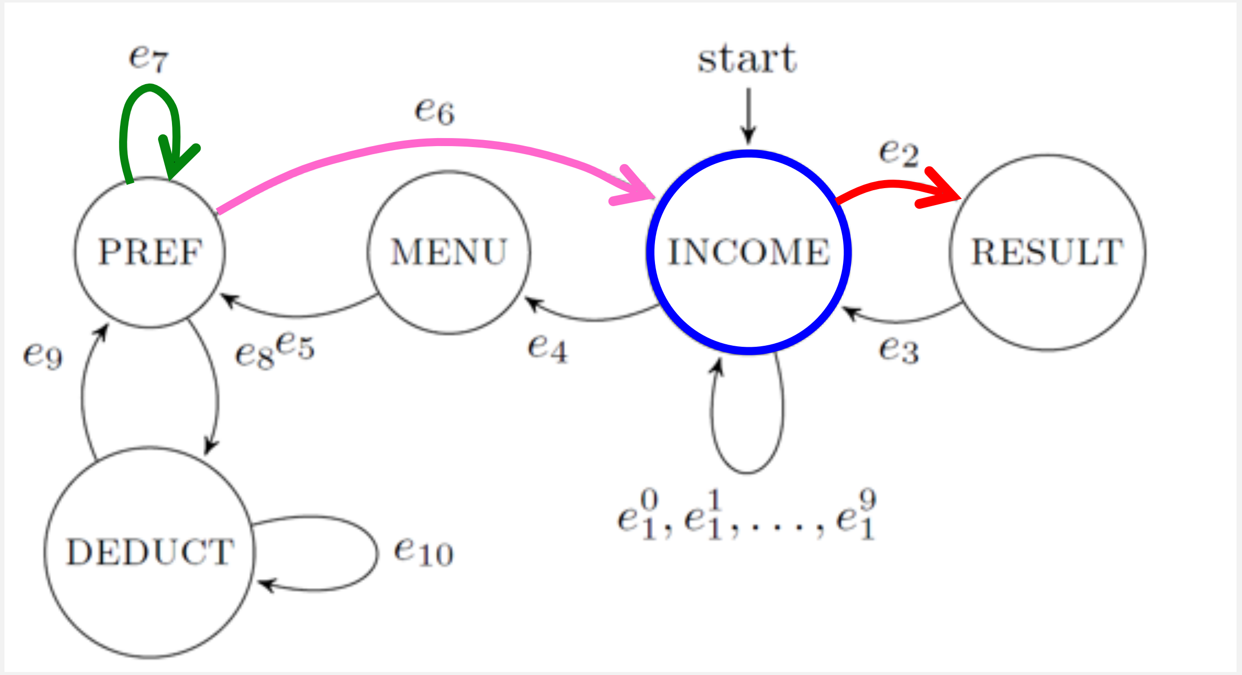
- For each given anchor, a set of connector sequences that lead from ***the anchor*** to the ***partial sequence*** is extracted by using the UI model



***Anchor*** = *path summaries for* $e_7$

***Connector*** = *path summaries for* $e_6$

***New Partial Sequence*** =
(*Anchor*, *Connector*, *original partial sequence*)



```
income = this.appState.enteredAmount;
deduction = 0;
if (Settings.getEnableTaxDeduction()) {
  deduction = Settings.getTaxDeduction();
}
taxable = income - deduction;
if (taxable < 0) {
  taxable = 0; // example target
}
tax = taxable * TAX_RATE;
result = income - tax;
```

## RE-PRIORITIZE PARTIAL SEQUENCES

- Some of partial sequences are less important than the others
- E.g. multiple anchors that affect the same path condition, which result in multiple new partial sequences in the wordlist
- We can decrease the priority of any redundant partial sequences

## EVALUATION & RESULTS

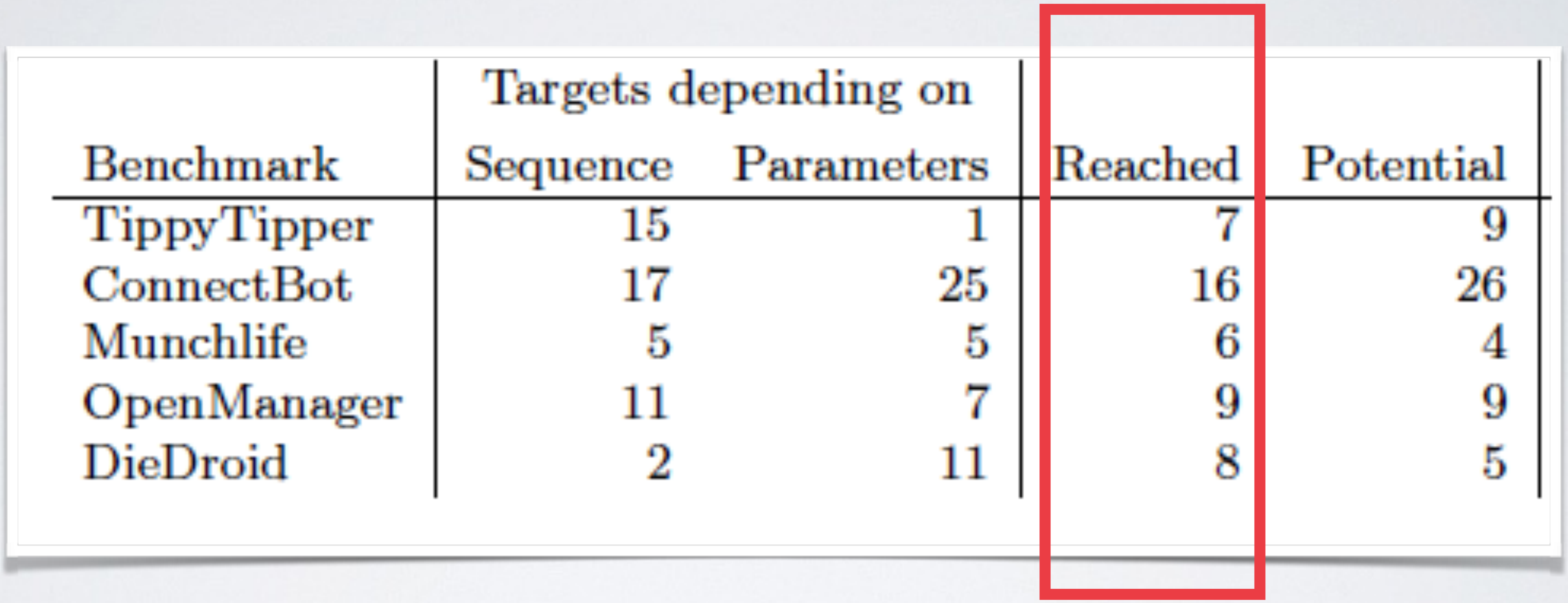
# EXPERIMENTAL SETUP:

Collider — symbolic execution → Symbolic Java PathFinder

Collider — concolic execution → Android emulator by Android SDK

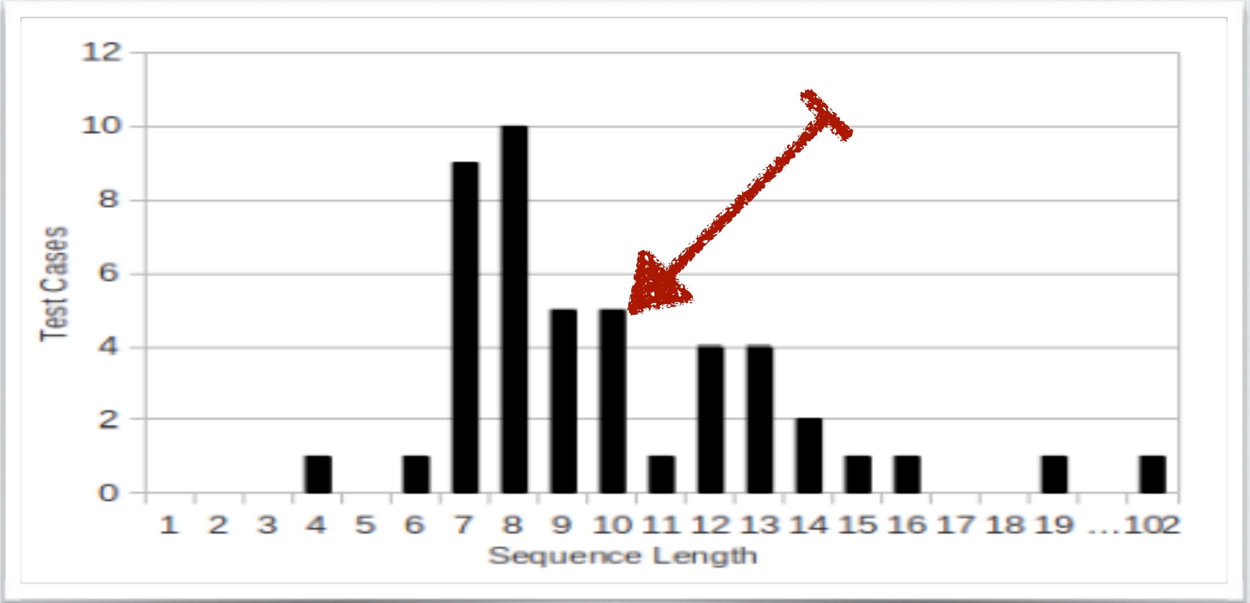
| Baseline tools |
|---|
| Simple Crawler |
| Monkey provided by Android SDK |

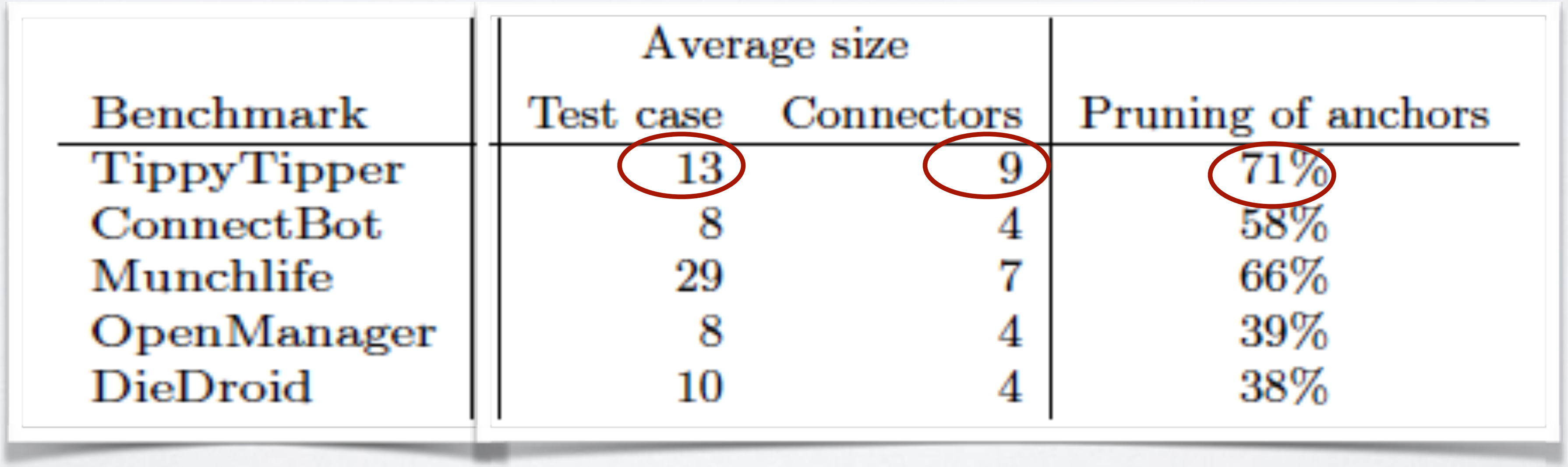
## Q1: Was the algorithm able to reach challenging targets?



| Benchmark | Targets depending on: Sequence | Targets depending on: Parameters | Reached | Potential |
|---|---|---|---|---|
| TippyTipper | 15 | 1 | 7 | 9 |
| ConnectBot | 17 | 25 | 16 | 26 |
| Munchlife | 5 | 5 | 6 | 4 |
| OpenManager | 11 | 7 | 9 | 9 |
| DieDroid | 2 | 11 | 8 | 5 |

## Q2: Does use of anchors & connectors have an effect on ability to reach targets, when compared to simple backward Breadth-First-Search technique?





| Benchmark | Average size: Test case | Average size: Connectors | Pruning of anchors |
|---|---|---|---|
| TippyTipper | 13 | 9 | 71% |
| ConnectBot | 8 | 4 | 58% |
| Munchlife | 29 | 7 | 66% |
| OpenManager | 8 | 4 | 39% |
| DieDroid | 10 | 4 | 38% |

## Q3: Does prioritization and ignoring of less important event sequences have any effect at all?

| | WITH PRIORITIZATION | WITHOUT PRIORITIZATION |
|---|---|---|
| TARGETS REACHED | 46 | 25 |
| RUNNING TIME FOR SEQUENCE GENERATION | 45 seconds | 2.5 hours |

## DISCUSSION QUESTIONS

## QUESTION 1

We see **some** of the challenging targets were reached. What happens to the *missed targets*? Why were they missed? What are the necessary steps?

**Possible Answer:** The algorithm supports symbolic reasoning of numeric values and booleans, resulting in imprecise treatment of, for example, strings and object. Improvisationn in. this field might help us reach the missed targets

## QUESTION 2

Authors use *handwritten UI models* for each application for symbolic generation. **Automated UI generation** models weren't present at the time. Would human effort mean presence of errors? This also reduces the feasibility of scaling the evaluation to a larger number of benchmarks.

**Possible Answer:** One possible alternative would be using automated UI-model generation technique**[6].** Would that be effective enough?

**[6]: Wei Yang et al., A Grey-Box Approach for Automated GUI-Model Generation of Mobile Applications, FASE'13 Proceedings of the 16th international conference on Fundamental Approaches to Software Engineering**

# QUESTION 3

Can we use ***Collider*** for other types of event-driven applications?

**Possible Answer:** the authors believe that their approach might also be applicable for other types of event-driven programs such as JavaScript web applications and desktop GUI applications in a case that event-handlers are smaller in web or GUI applications

## QUESTION 4

Can ***Collider*** be used to generate test cases for all types of targets?

**Possible Answer:** In a case that target is simple, collider's execution time is greater than existing approaches

## QUESTIONS 5

**Does this technique work well with all Android Applications?**

**Possible Answer:** There was one application ("TippyTipper") which was evaluated by the authors in the paper which did not respond very well. It took 30 minutes for the event sequence generation phase to complete for this application while for other four took few seconds. This was possibly because of lot of connectors between events. Such applications might take a lot of time.

## QUESTION 6

**Can we apply this technique to other mobile application platforms such as IOS and windows?**

**Possible Answer:** the authors have mentioned that their approach works well with small number of long event sequences which is the case in mobile applications. Hence ,any mobile application that satisfies this condition, ex:- iOs/windows should work well with the algorithm is our assumption.