Shape Analysis via 3-Valued Logic

Mooly Sagiv

Tel Aviv University

http://www.cs.tau.ac.il/~msagiv/toplas02.pdf

www.cs.tau.ac.il/~tvla

Plan

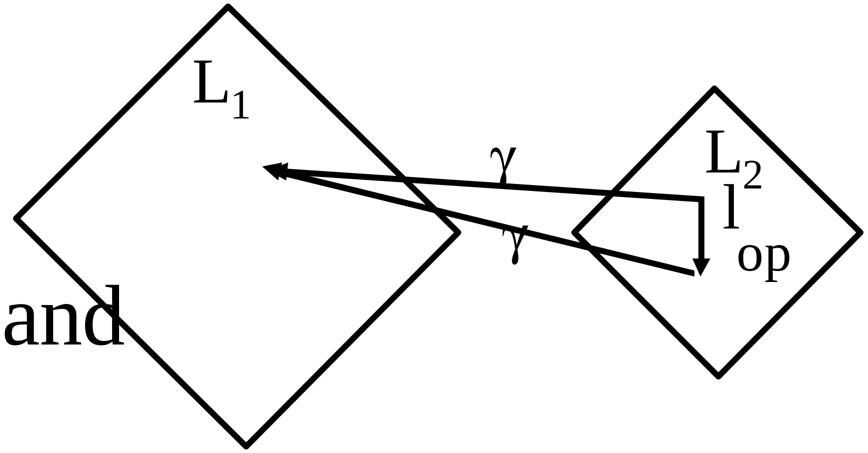
- Questions & Answers
- The TVLA system
- “Realistic” applications

Abstract (Conservative) interpretation

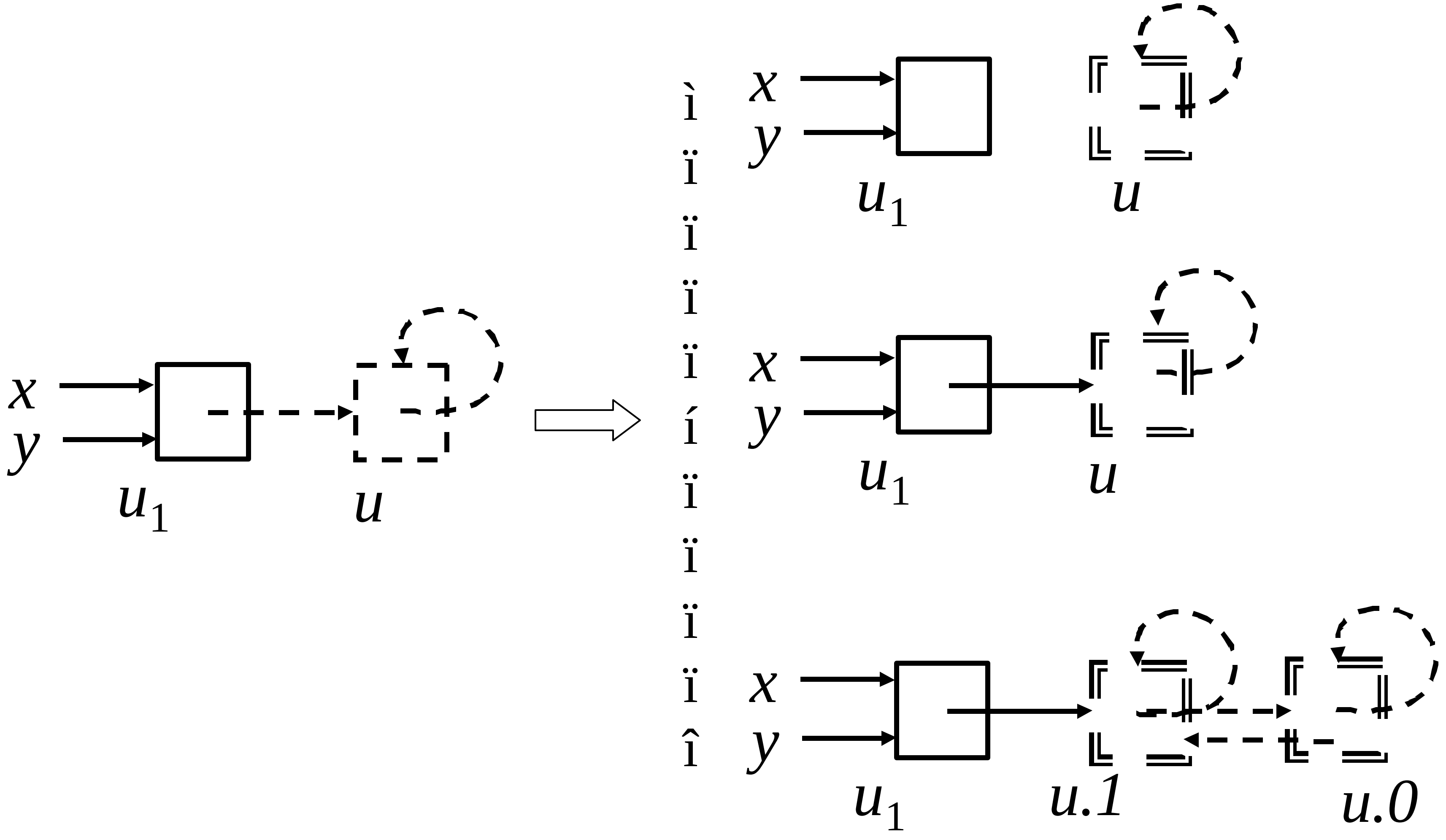
$$\begin{tikzcd}
\boxed{\text{Set of states}} \arrow[r, "\text{Operational semantics}" above, "\text{statement } s"] \arrow[d, dashed, "\alpha"] & \boxed{\text{Set of states}} \arrow[dr, dashed, "\alpha"] & \\
\boxed{\text{abstract representation}} \arrow[r, "\text{statement } s", "\text{Abstract semantics}"'] & \boxed{\text{abstract representation}} \arrow[r, phantom, "\sqsupseteq"] & \boxed{\text{abstract representation}}
\end{tikzcd}$$

Semantic Reduction

- Improve the precision by recovering properties of the program semantics
- A Galois insertion $(L_1, \alpha, \gamma, L_2)$
- An operation $op: L_2 \rightarrow L_2$ is a semantic reduction
  - $\forall l \in L_2 \; op(l) \sqsubseteq l$
  - $\gamma(op(l)) = \gamma(l)$
- Can be applied before and after basic operations

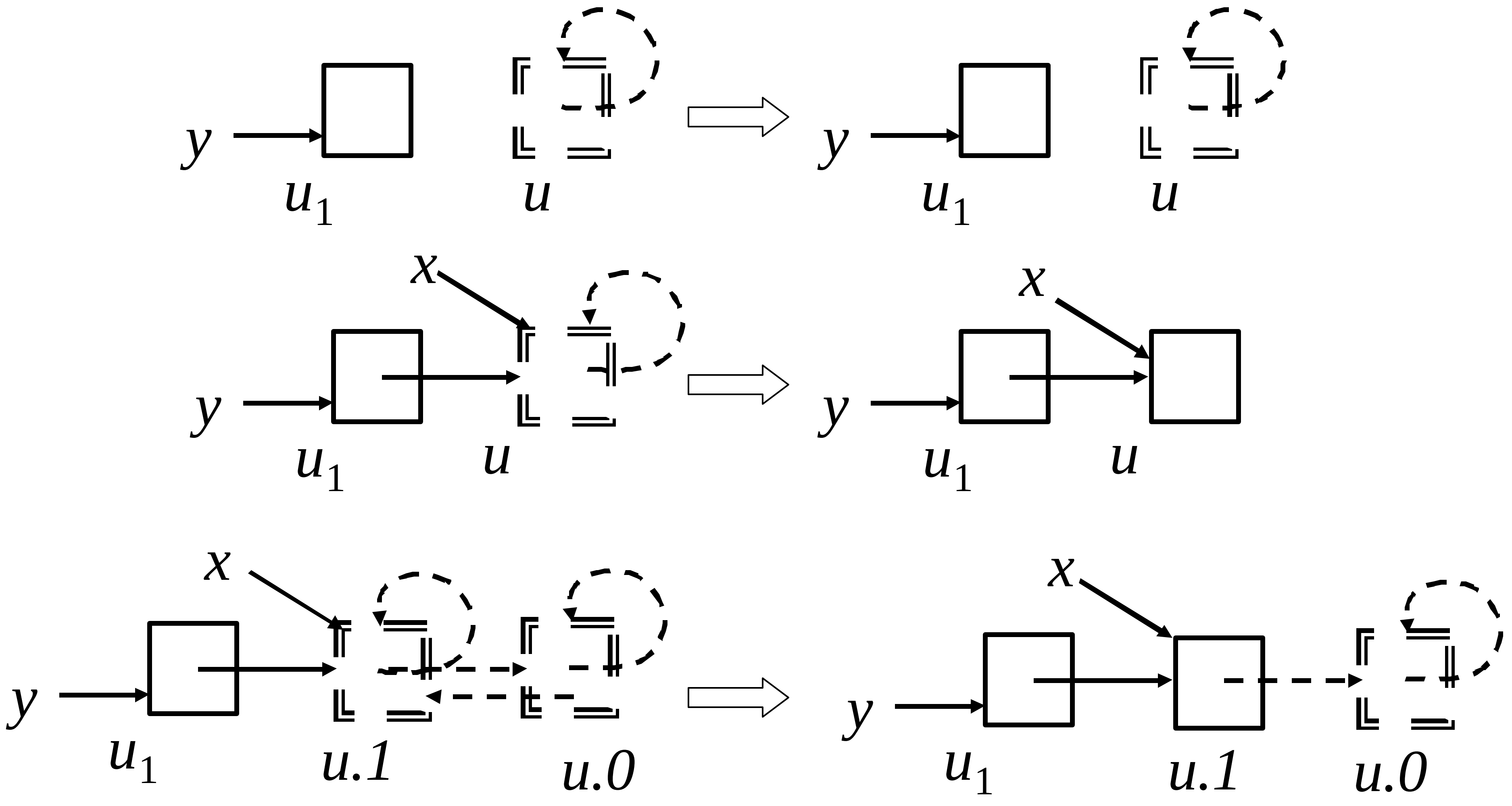


(1) Focus on $\exists\, v_1 : x(v_1) \wedge n(v_1, v)$



Why is Focus a semantic reduction?

(3) Apply Constraint Solver



Why is Coerce a semantic reduction?

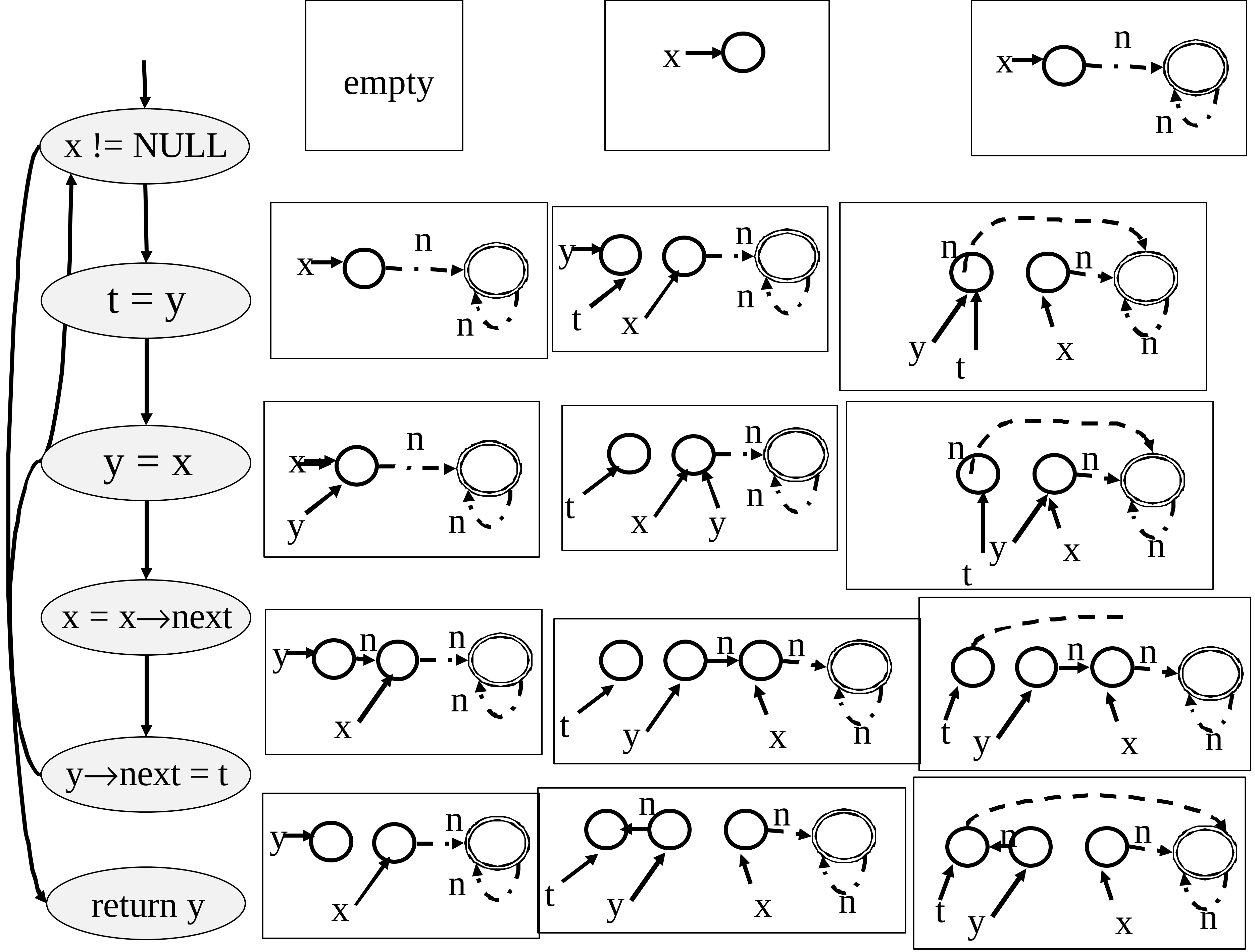
- Assume that the integrity constraints hold in the concrete semantics
- Restrict constraints to:
  - formula $\rightarrow$ $p^B(v1, v2, ..., vk)$
  - Preserved by canonical abstraction

How does the analysis behave in loops

- Rather precise
- Usually cheap
- But sometimes expensive for programs with potentially many “aliasing patterns”
- Improving scalability
  - Liveness analysis helps
    - Local abstractions
  - Partial relational analysis

Example: In-Situ List Reversal

```
typedef struct list_cell {
    int val;
    struct list_cell *next;
} *List;

List reverse (List x) {
    List y, t;
    y = NULL;
    while (x != NULL) {
        t = y;
        y = x;
        x = x → next;
        y → next = t;
    }
    return y;
}
```


x != NULL
t = y
y = x
x = x→next
y→next = t
return y
empty
x
n
y
t

Three Valued Logic Analysis (TVLA) T. Lev-Ami & R. Manevich

- Input (FO^{TC})
  - Concrete interpretation rules
  - Definition of instrumentation predicates
  - Definition of safety properties
  - First Order Transition System (TVP)
- Output
  - Warnings (text)
  - The 3-valued structure at every node (invariants)

Null Dereferences

```
typedef struct element
{
    int value;
    struct element *n;
} Element
```

```
bool search(  int value,
              Element *x)
{
Element * c = x
   while (  x != NULL )
   {
        if (c→ val == value)
                return TRUE;
        c = c → n;
   }
   return FALSE; }
```

Demo

TVLA inputs

TVP - **T**hree **V**alued **P**rogram

- Predicate declaration
- Action definitions SOS
- Control flow graph

Program independent

- TVS - **T**hree **V**alued **S**tructure

Demo

Challenge 1

- Write a C procedure on which TVLA reports false null dereference

Proving Correctness of Sorting Implementations (Lev-Ami, Reps, S, Wilhelm ISSTA 2000)

- Partial correctness
  - The elements are sorted
  - The list is a permutation of the original list
- Termination
  - At every loop iterations the set of elements reachable from the head is decreased

Example: InsertSort

```
typedef struct list_cell {
    int data;
    struct list_cell *n;
} *List;
```

pred.tvp

actions.tvp

Run Demo

```
List InsertSort(List x) {
List r, pr, rn, l, pl; r = x; pr = NULL;
  while  (r != NULL) {
    l = x; rn = r → n; pl = NULL;
    while  (l != r) {
      if  (l → data > r → data) {
        pr → n = rn; r → n = l;
        if  (pl == NULL) x = r;
        else pl → n = r;
        r = pr;
        break;
      }
      pl = l; l = l → n;
    }
    pr = r; r = rn;
  }
  return x;
}
```

Example: InsertSort

```
typedef struct list_cell {
   int data;
   struct list_cell *n;
} *List;
```

```
List InsertSort(List x) {
 if (x == NULL)   return NULL
 pr = x;  r = x->n;
 while (r != NULL) {
         pl = x; rn = r->n; l = x->n;
         while (l != r) {
                   pr->n = rn ;
                   r->n = l;
                   pl->n = r;
                   r = pr;
                   break;
                       }
                  pl = l;
                  l = l->n;
                  }
         pr = r;
         r = rn;
          }
```

Run Demo

Example: Reverse

```
typedef struct list_cell {
    int data;
    struct list_cell *n;
} *List;
```

```
List reverse (List x) {
    List y, t;
    y = NULL;
    while (x != NULL) {
        t = y;
        y = x;
        x = x → next;
        y → next = t;
    }
    return y;
}
```

Run Demo

Challenge 2

- Write a sorting C procedure on which TVLA fails to prove sortedness or permutation

Example: Mark and Sweep

```
void Mark(Node root) {
  if (root != NULL) {
    pending = $\emptyset$
    pending = pending $\cup$ {root}
    marked = $\emptyset$
    while (pending $\neq \emptyset$) {
      x = SelectAndRemove(pending)
      marked = marked $\cup$ {x}
      t = x $\rightarrow$ left
      if (t $\neq$ NULL)
        if (t $\notin$ marked)
          pending = pending $\cup$ {t}
      t = x $\rightarrow$ right
      if (t $\neq$ NULL)
        if (t $\notin$ marked)
          pending = pending $\cup$ {t}
    }
  }
  assert(marked == Reachset(root))
}
```

```
void Sweep() {
  unexplored = Universe
  collected = $\emptyset$
  while (unexplored $\neq \emptyset$) {
    x = SelectAndRemove(unexplored)
    if (x $\notin$ marked)
      collected = collected $\cup$ {x}
  }
  assert(collected ==
           Universe – Reachset(root)
        )
}
```

pred.tvp

Run Demo

Challenge 3

- Use TVLA to show termination of markAndSweep

“Realistic” Applications

Heap & Concurrency [Yahav POPL'01]

- Concurrency with the heap is evil…
- Java threads are just heap allocated objects
- Data and control are strongly related
  - Thread-scheduling info may require understanding of heap structure (e.g., scheduling queue)
  - Heap analysis requires information about thread scheduling

```
Thread t1 = new Thread();
Thread t2 = new Thread();
…
t = t1;
…
t.start();
```

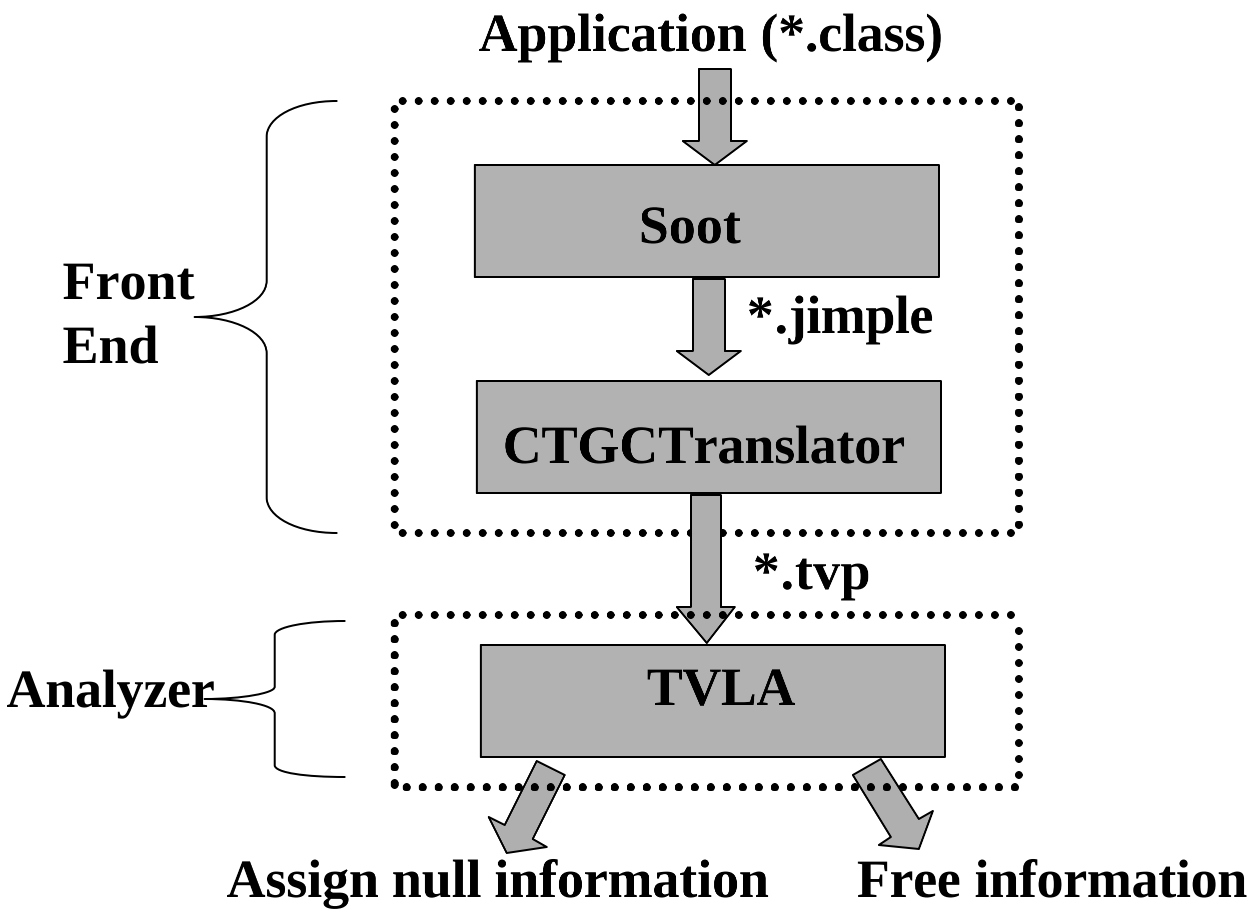
Examples Verified

Program	**Property**
twoLock Q	No interference No memory leaks Partial correctness
Producer/consumer	No interference No memory leaks
Apprentice Challenge	Counter increasing
Dining philosophers with resource ordering	Absence of deadlock
Mutex	Mutual exclusion
Web Server	No interference

Compile-Time GC for Java (Ran Shaham, SAS’03, SCP)

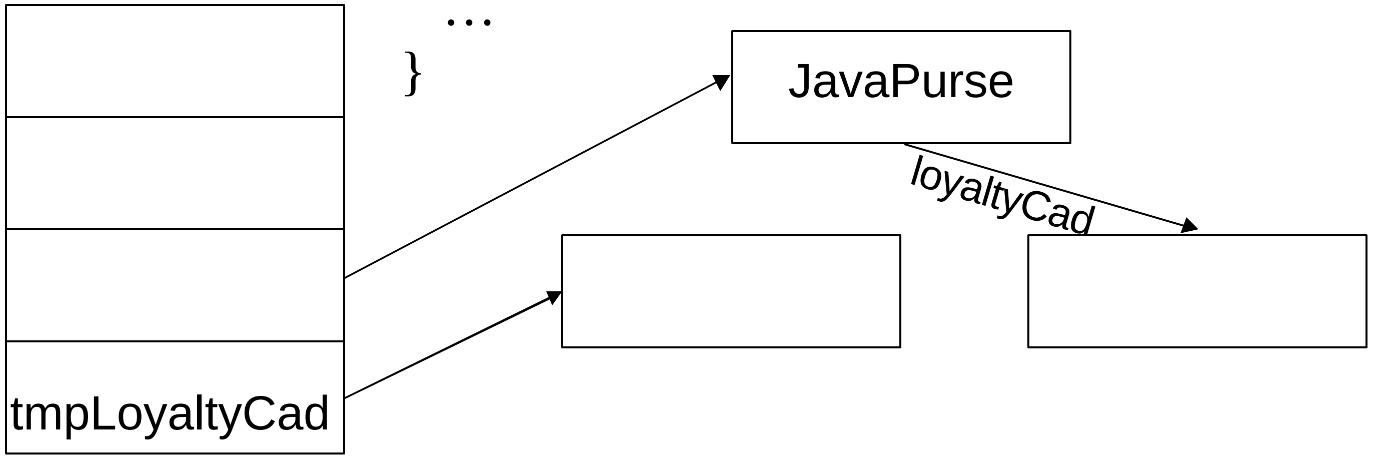
- The compiler can issue free when objects are no longer needed
- Analysis of Java/JavaCard programs
- Requires forward information
- Maintained via history automata
  - Provides instrumentation predicates
- More automatic analysis (G. Arnold)

CTGC architecture



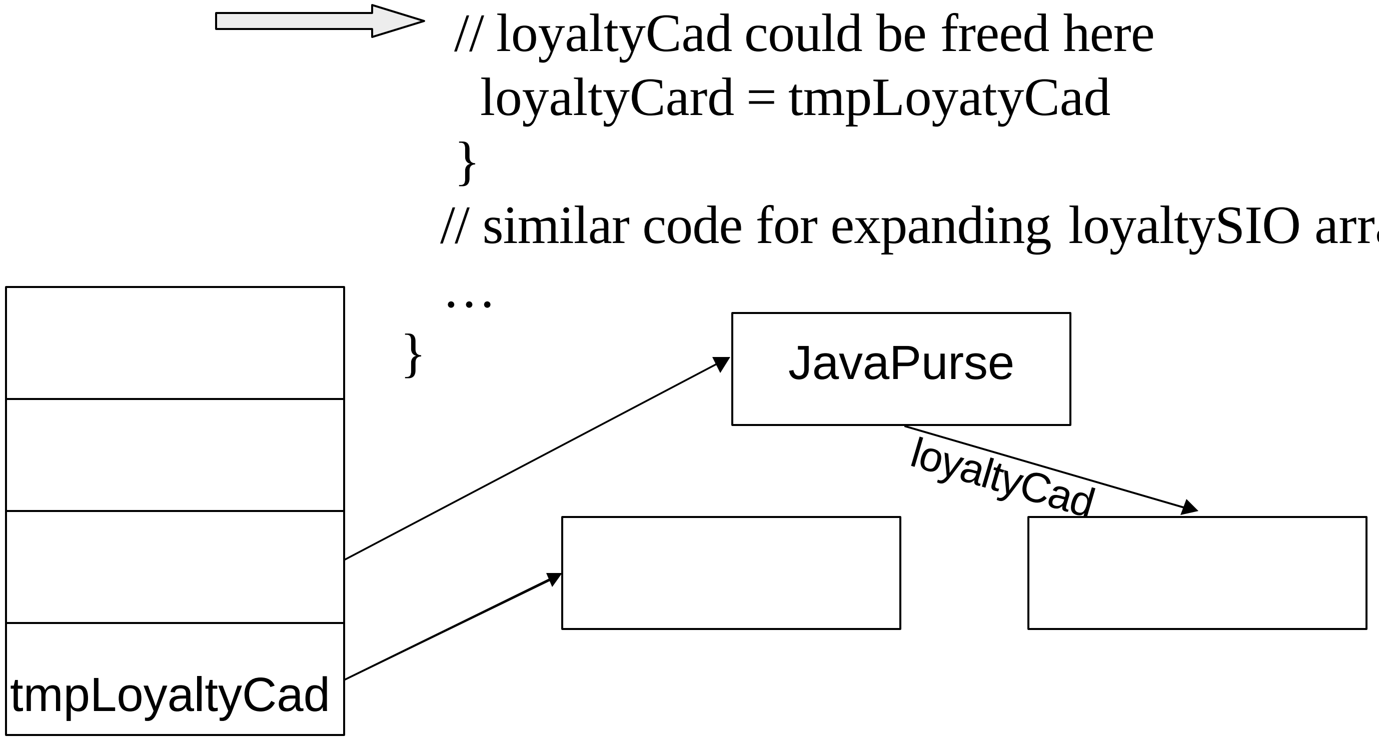
Usage of CTGC output (1)

```
private void expandLoyaltyProgramIfNeeded() {
    currLoyatyCount++;
    if (currLoyaltyCount > loyaltyCount.length) {
      tmpLoyaltyCad = new short[loyaltyCount.length * 2];
      // The array is currently copied using a for loop
      Util.arrayCopyNonAtomic(loyaltyCad, 0, tmpLoyatyCad, …),
    // loyaltyCad could be freed here
      loyaltyCard = tmpLoyatyCad
    }
   // similar code for expanding loyaltySIO array
   …
}
```



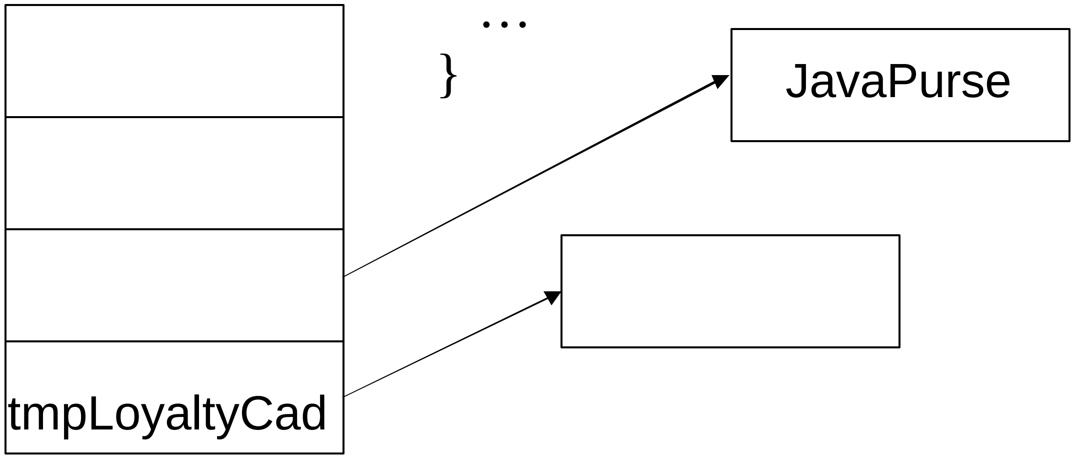
Usage of CTGC output (1)

```
private void expandLoyaltyProgramIfNeeded() {
    currLoyatyCount++;
    if (currLoyaltyCount > loyaltyCount.length) {
      tmpLoyaltyCad = new short[loyaltyCount.length * 2];
      // The array is currently copied using a for loop
      Util.arrayCopyNonAtomic(loyaltyCad, 0, tmpLoyatyCad, …),
    // loyaltyCad could be freed here
      loyaltyCard = tmpLoyatyCad
    }
   // similar code for expanding loyaltySIO array
   …
}
```



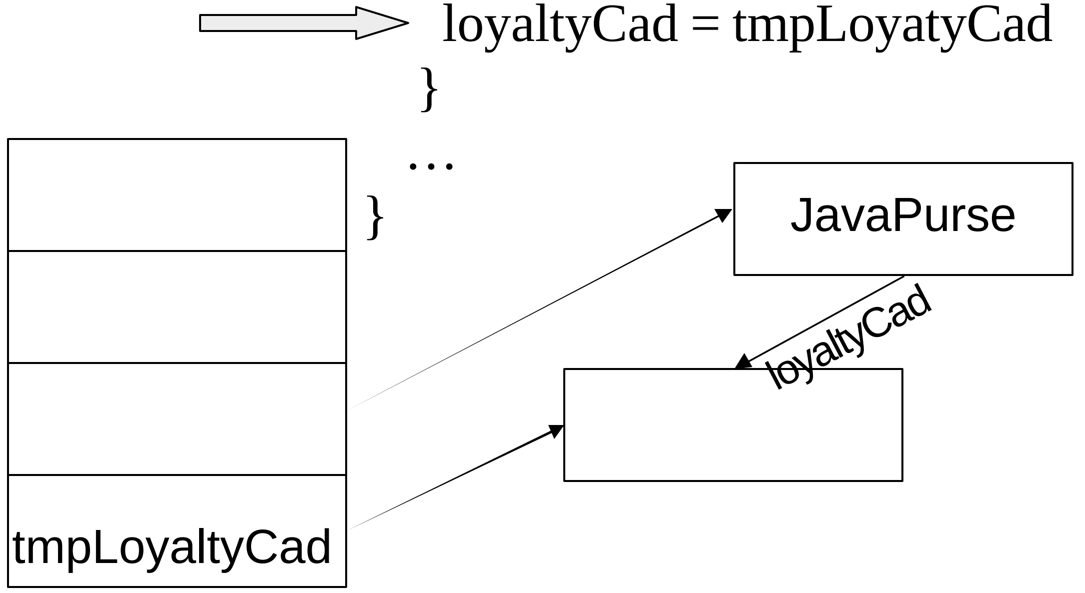
Usage of CTGC output (1)

```
private void expandLoyaltyProgramIfNeeded() {
    currLoyatyCount++;
    if (currLoyaltyCount > loyaltyCount.length) {
      tmpLoyaltyCad = new short[loyaltyCount.length * 2];
      // The array is currently copied using a for loop
      Util.arrayCopyNonAtomic(loyaltyCad, 0, tmpLoyatyCad, …
    // loyaltyCad could be freed here
      loyaltyCad = tmpLoyatyCad
    }
   …
}
```



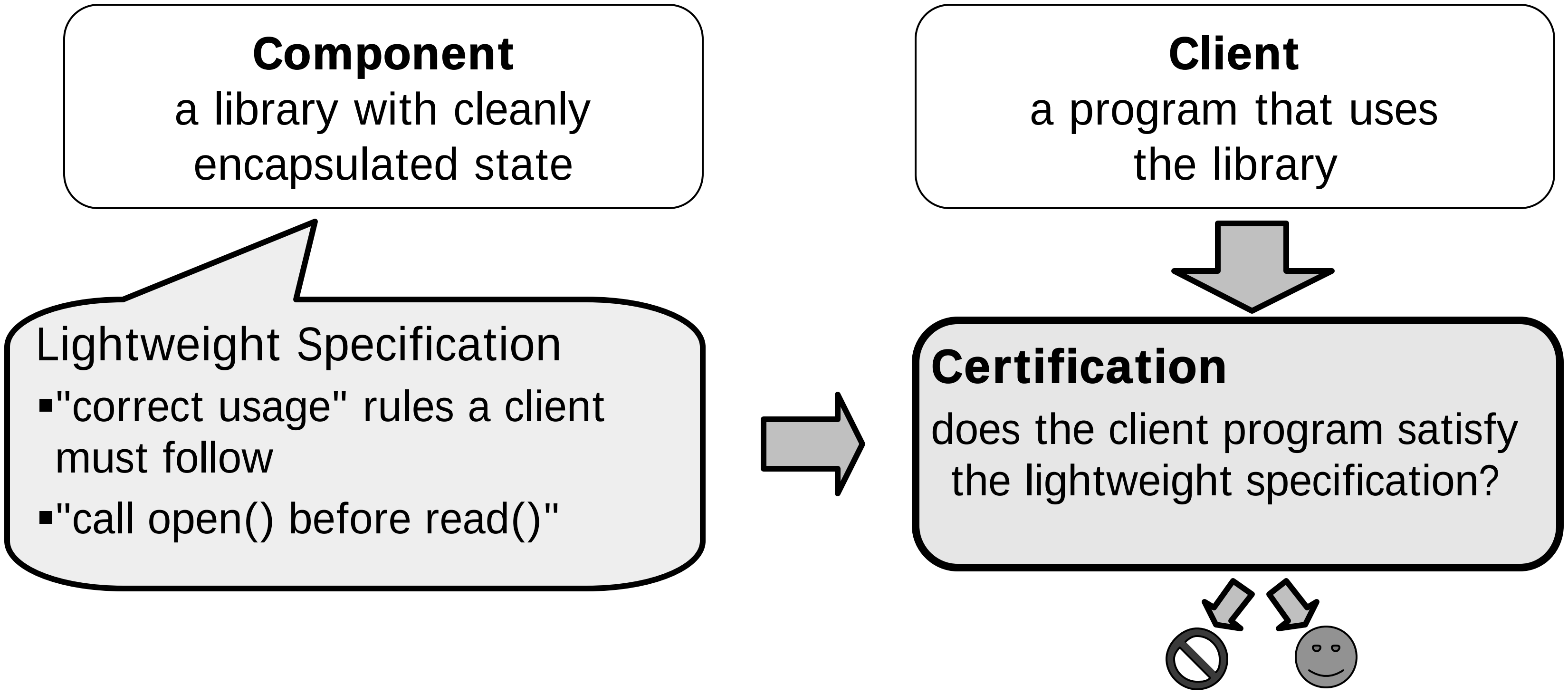
Usage of CTGC output (1)

```
private void expandLoyaltyProgramIfNeeded() {
    currLoyatyCount++;
    if (currLoyaltyCount > loyaltyCount.length) {
      tmpLoyaltyCad = new short[loyaltyCount.length * 2];
      // The array is currently copied using a for loop
      Util.arrayCopyNonAtomic(loyaltyCad, 0, tmpLoyatyCad, …);
    // loyaltyCad could be freed here
      loyaltyCad = tmpLoyatyCad
    }
   …
}
```



Verification of Safety Properties (PLDI'02, 04)

The *Canvas* Project (with IBM Watson) (***C***omponent ***A***nnotation, ***V***erification ***a***nd ***S***tuff)



Component
a library with cleanly encapsulated state

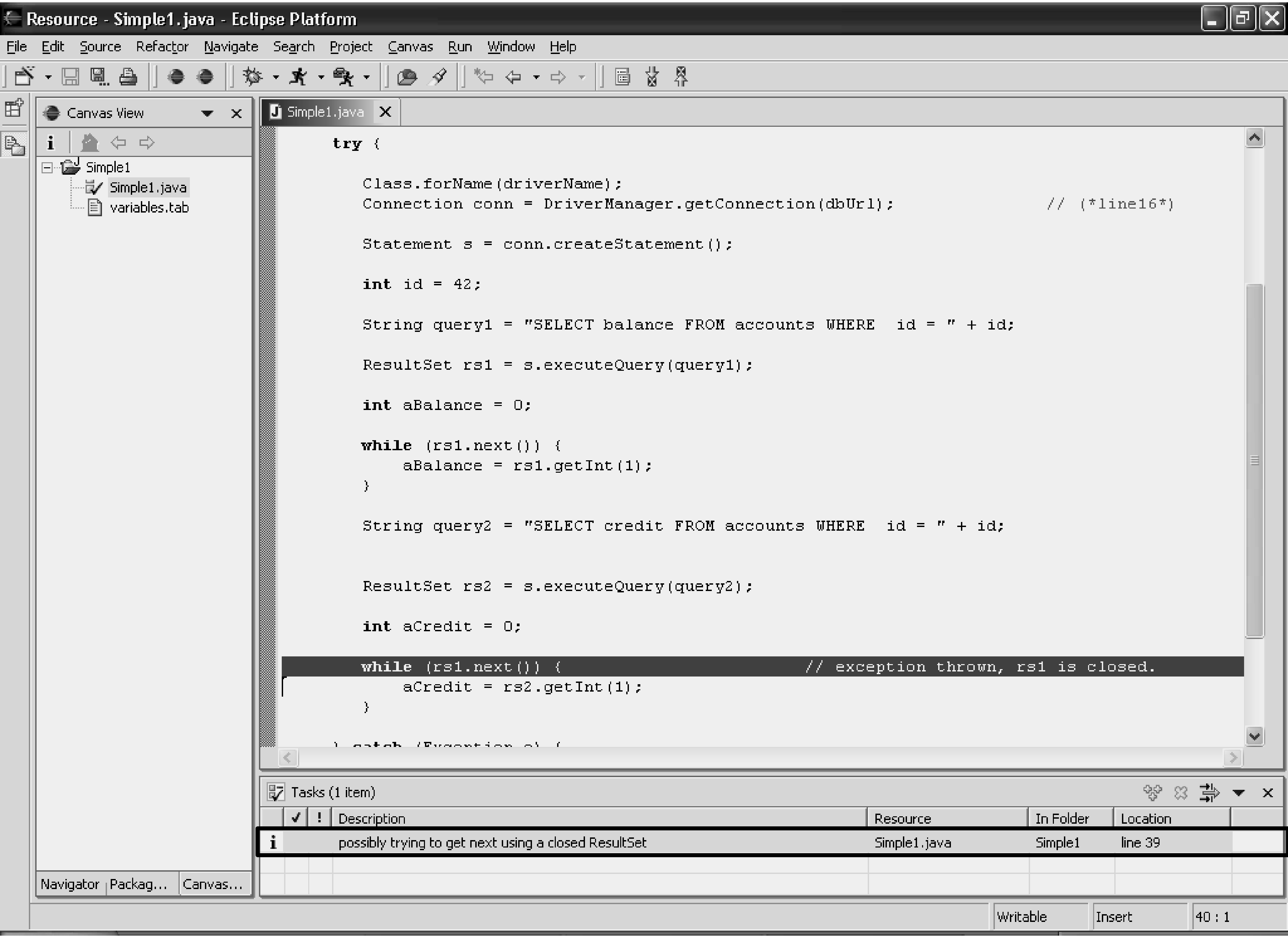
Client
a program that uses the library

Lightweight Specification
- "correct usage" rules a client must follow
- "call open() before read()"

Certification
does the client program satisfy the lightweight specification?

Prototype Implementation

- Applied to several example programs
  - Up to 5000 lines of Java
- Used to verify
  - Absence of concurrent modification exception
  - JDBC API conformance
  - IOStreams API conformance

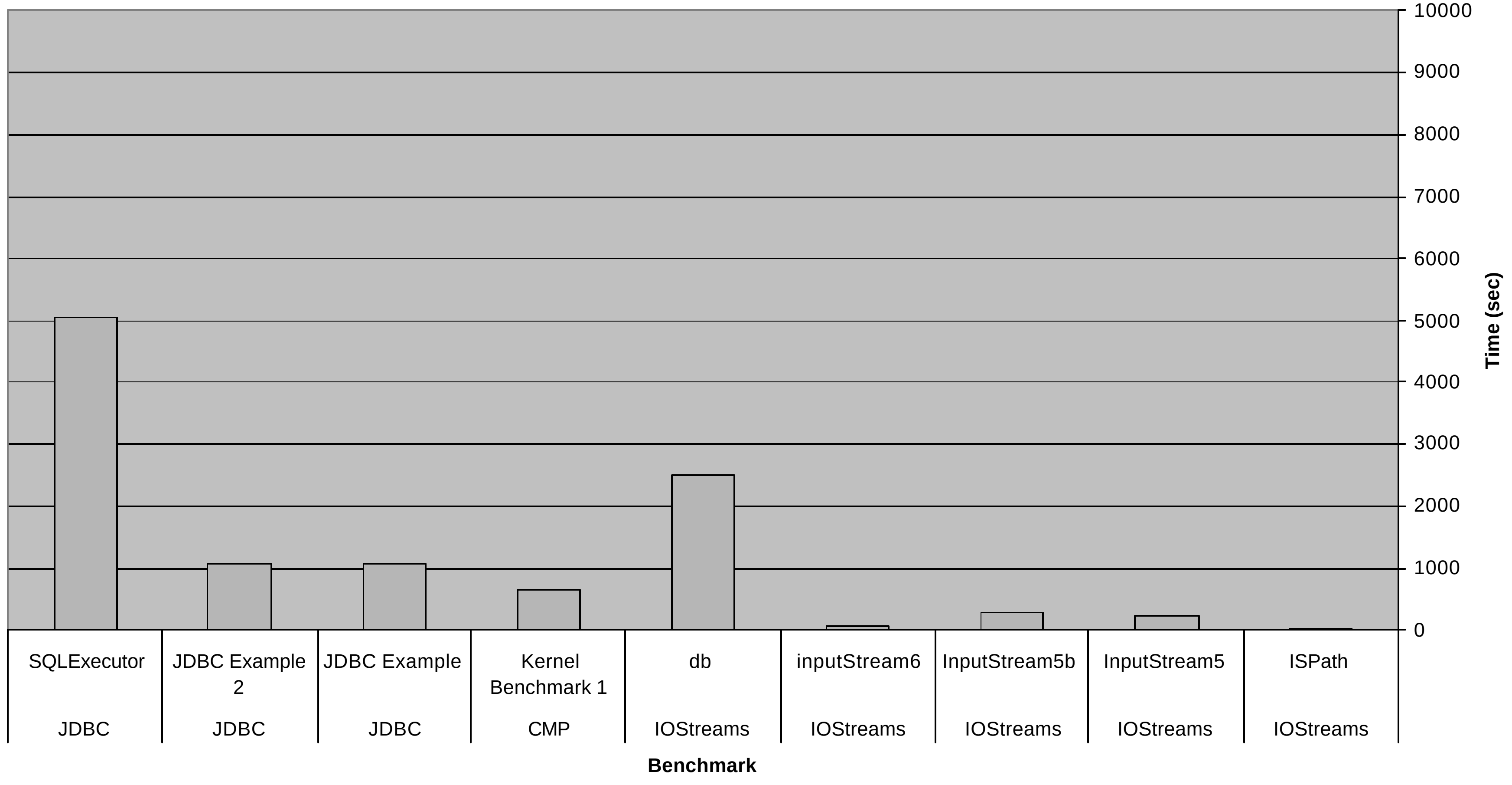

Resource - Simple1.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Canvas Run Window Help
Canvas View
Simple1
Simple1.java
variables.tab
Navigator Packag... Canvas...
Simple1.java
Writable
Insert
40 : 1

```
    try {

        Class.forName(driverName);
        Connection conn = DriverManager.getConnection(dbUrl);                // (*line16*)

        Statement s = conn.createStatement();

        int id = 42;

        String query1 = "SELECT balance FROM accounts WHERE  id = " + id;

        ResultSet rs1 = s.executeQuery(query1);

        int aBalance = 0;

        while (rs1.next()) {
            aBalance = rs1.getInt(1);
        }

        String query2 = "SELECT credit FROM accounts WHERE  id = " + id;


        ResultSet rs2 = s.executeQuery(query2);

        int aCredit = 0;

        while (rs1.next()) {                        // exception thrown, rs1 is closed.
            aCredit = rs2.getInt(1);
        }

    } catch (Exception e) {
```

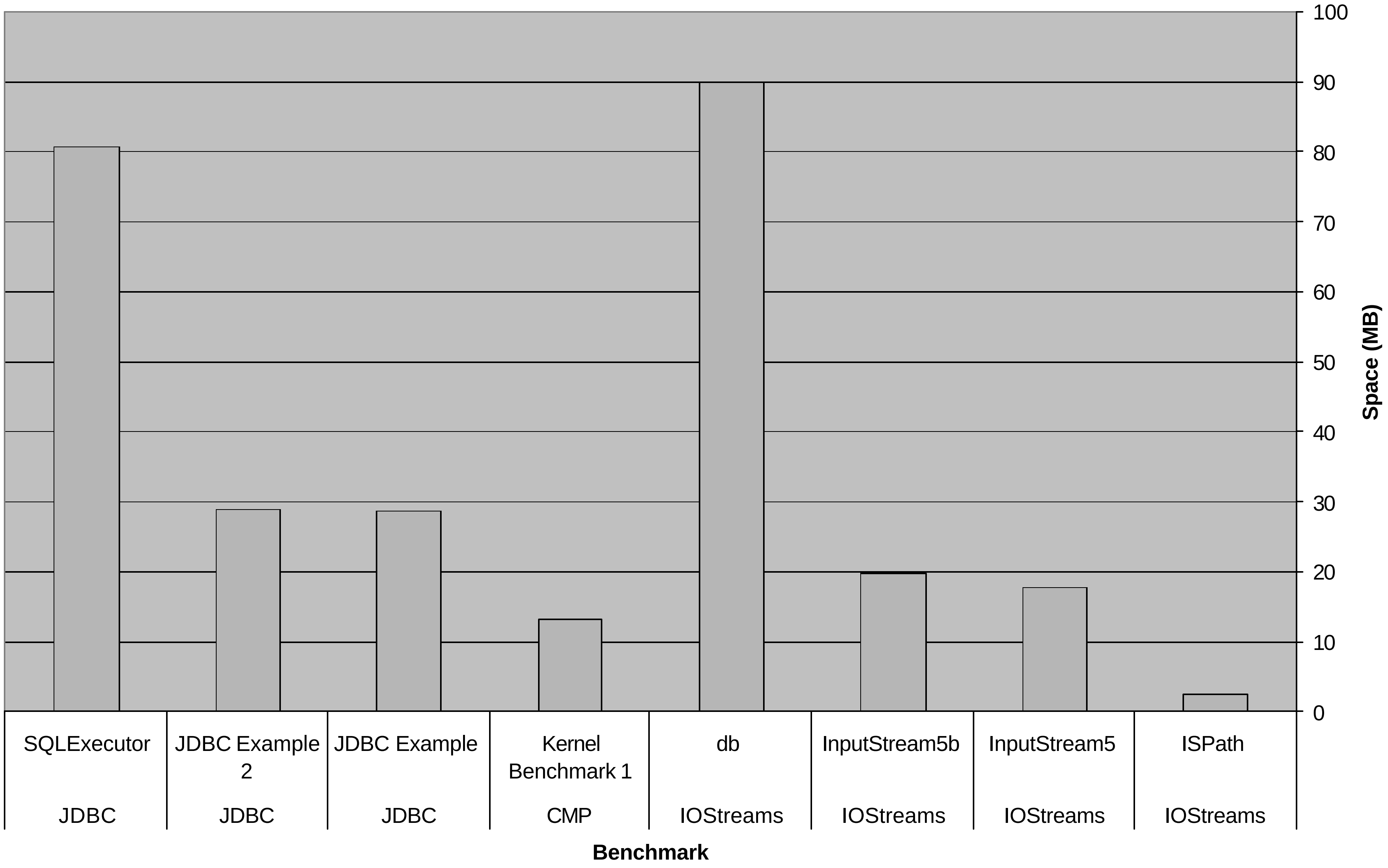
Tasks (1 item)

	✓	!	Description	Resource	In Folder	Location
i			possibly trying to get next using a closed ResultSet	Simple1.java	Simple1	line 39

Analysis Times



Space



Space (MB)
0
10
20
30
40
50
60
70
80
90
100
SQLExecutor
JDBC
JDBC Example 2
JDBC
JDBC Example
JDBC
Kernel Benchmark 1
CMP
db
IOStreams
InputStream5b
IOStreams
InputStream5
IOStreams
ISPath
IOStreams
Benchmark

Scaling

- Staged analysis
- Controlled complexity
  - More coarse abstractions [Manevich SAS’04]
- Handle libraries
  - Use procedure specifications [Yorsh, TACAS’04]
  - Decision procedures for linked data structures [Immerman, CAV’04, Lev-Ami, CADE’05]
- Handling procedures
  - Compute procedure summaries [Jeannet, SAS’04]
  - Local heaps [Rinetzky, POPL’05, SAS’05]

Why is Heap Analysis Difficult?

- Destructive updating through pointers
  - `p®next = q`
  - Produces complicated aliasing relationships
  - Track aliasing on 3-valued structures
- Dynamic storage allocation
  - No bound on the size of run-time data structures
  - Canonical abstraction ⇒ finite-sized 3-valued structures
- Data-structure invariants typically only hold at the beginning and end of operations
  - Need to verify that data-structure invariants are re-established
  - Query the 3-valued structures that arise at the exit

Summary

- Canonical abstraction is powerful
  - Intuitive
  - Adapts to the property of interest
- Used to verify interesting program properties
  - Very few false alarms
- But scaling is an issue

Summary

- Effective Abstract Interpretation
  - Always terminates
  - Precise enough
  - But still expensive
- Can model
  - Heap
  - Unbounded arrays
  - Concurrency
- More instrumentation can mean more efficient
- But canonical abstraction is limited
  - Correlation between list lengths
  - Arithmetic
  - Partial heaps