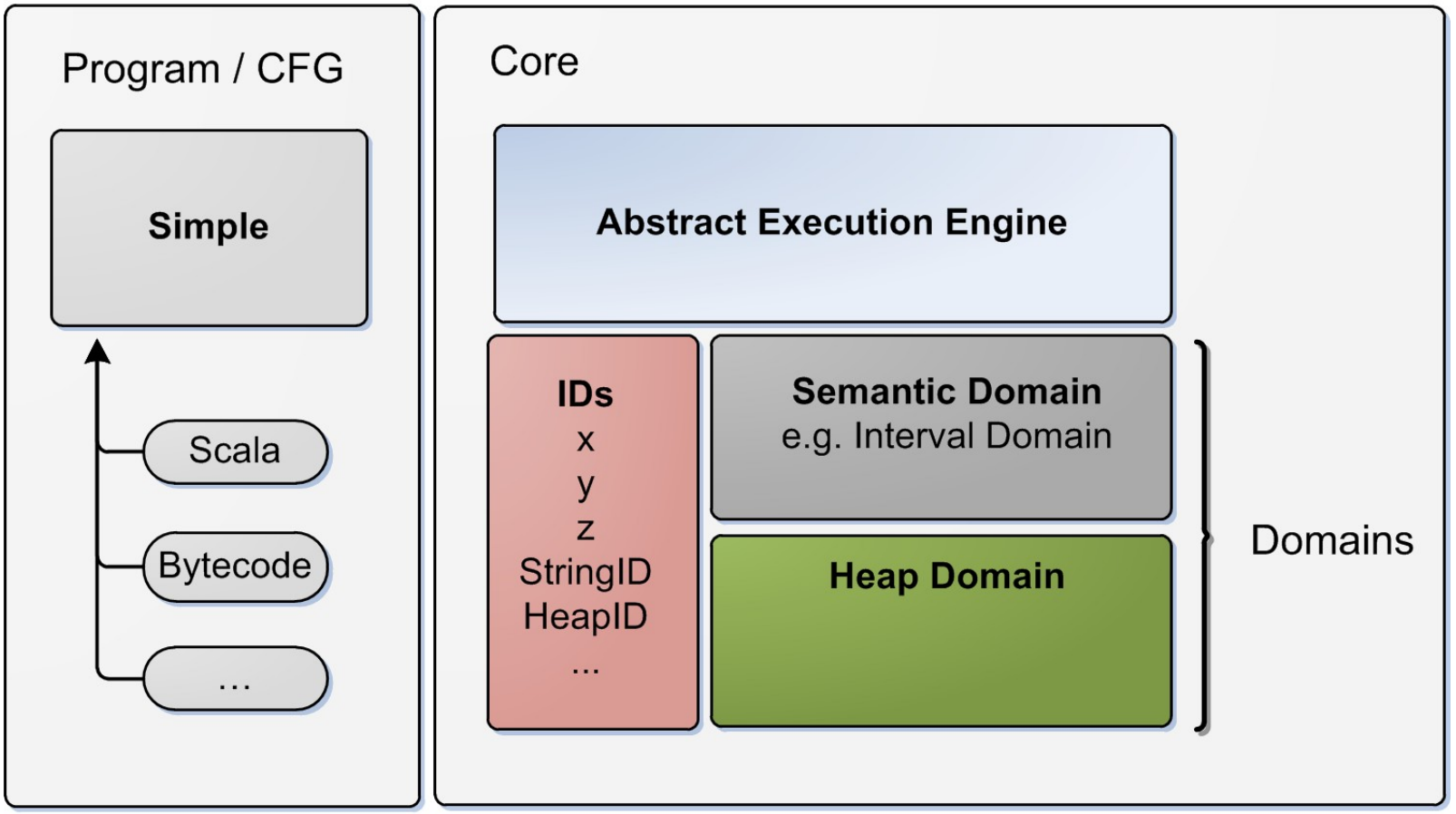
Interfacing TVLA & Sample

Raphael Fuchs

August 4, 2011

Sample: Architecture



Approach (1)

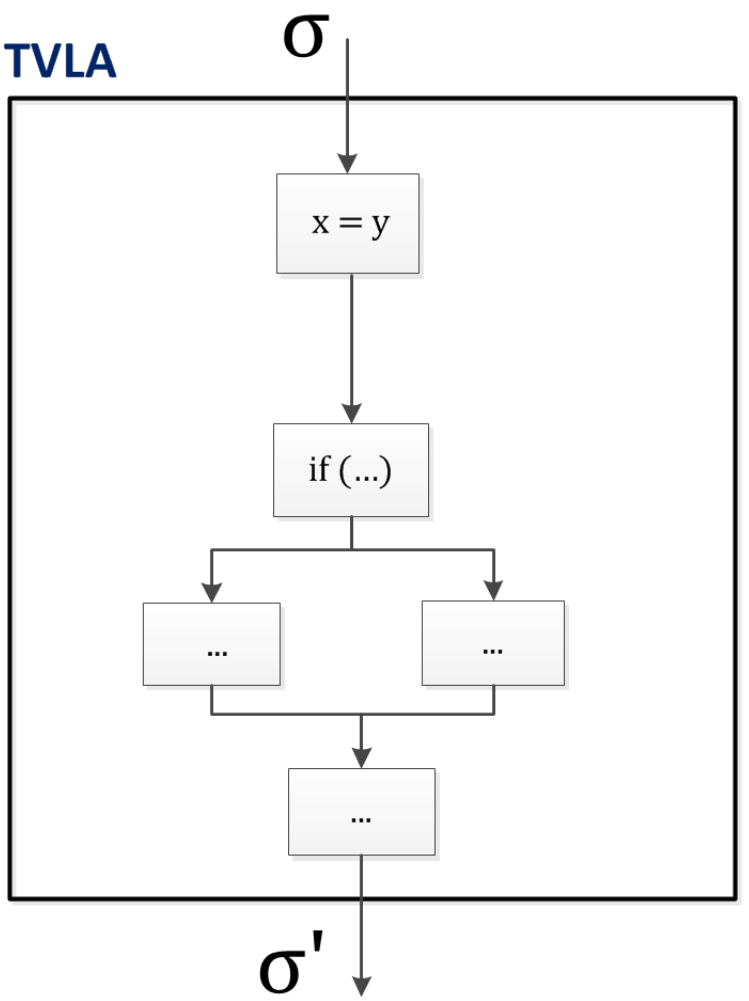
- Simple statements result in calls on heap domain
- For every heap operation, invoke TVLA
- E.g.

x = y 

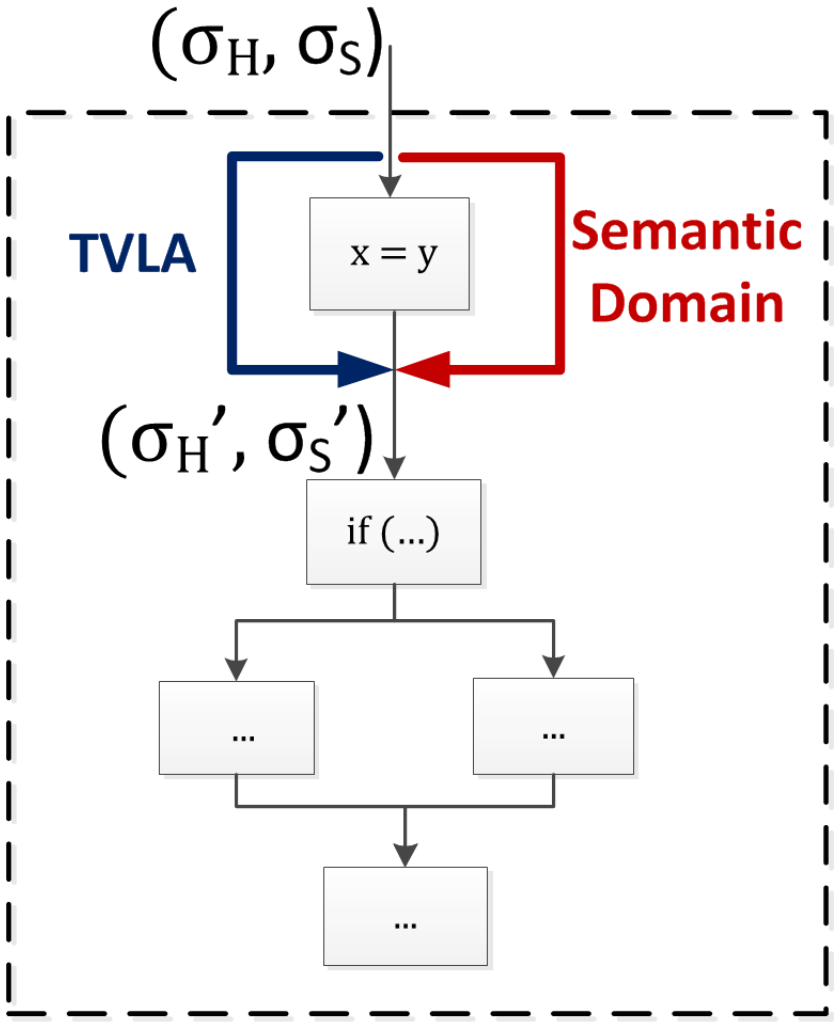
```
assignVariable(x,y)


%action
assignVariable(target,source) {
   %f { source(v) }
   {  target(v) = source(v) }
}
```

Approach (2)



Approach (2)



Differences in Abstraction

Simple

- High-level language
- Object-oriented
- Lacks distinction: expressions vs. Statements
- Types

TVP (TVLA)

- Quite low-level
- Usually C-like programs encoded
- Semantics encoded using FOL with TC
- Untyped

Translation not straightforward...

Heap Operations

TVP Actions for:

- Object Creation `new Foo()`
- Variable assignment (reference copy) `x = y`
- Field assignment `x.n = y`
- Field access `x.n`
- Assumptions `if (x != null) ... else ...`

- Least upper bound on heap states

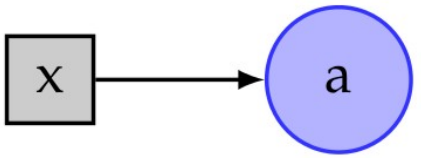
Heap State

State = Set of 3-valued logical structures

Each structure defined by:

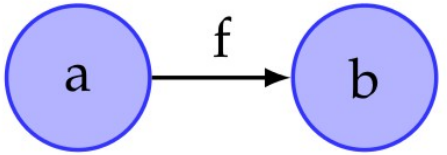
- Individuals: Blocks of allocated memory
- Unique unary predicates for program variables, e.g.

`x(a) = true`



- Binary function predicates for fields, e.g.

`f(a,b) = true`



Instrumentation Predicates

- Our analysis supports lists precisely
- Standard set of predicates
  - For every field f
    - `t[f](v1,v2)`
      Transitive reflexive reachability: Can v2 be reached from v1 by following f-fields?
    - `is[f](v)`
      Shared-ness: Is there more than one object whose f-field references v?
  - For every field f and variable x
    - `r[f,x](v)`
      Reachability: Can v be reached from x along f-fields?

Numerical Information

- Only heap (shape) analyzed, so far
- What about numerical information?
  - E.g. Integer fields
  - TVLA does not support this
- Idea:
  - Let every heap node be an identifier and associate some numerical info with it
  - More generally, any information from a semantic domain, for instance strings and access permissions

Heap Identifiers (1)

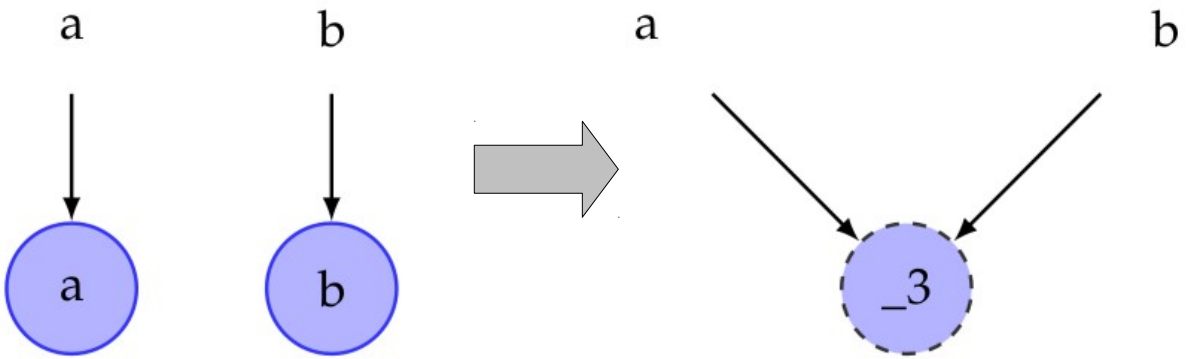
- Heap identifiers (names of nodes) are *not* static. We have to consider
  - Summarization: Due to abstraction, nodes are merged into a summary node.
  - Materialization: Nodes can be extracted again from a summary node.
  - Merging due to heap isomorphism
- Update ID space according to these events

Heap Identifiers (2)

Problem: TVLA drops all names of individuals

- We do not obtain any information what happened
- We cannot update ID space and semantic domain

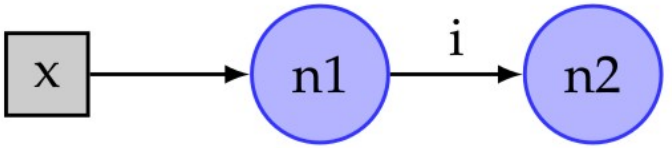
Trick: Use unary non-abstraction predicates to track identifiers (names of nodes)



Example: Intervals

```
1  def iAssign(unknown:Boolean) = {
2    val x = new IntNode
3
4    if (unknown) {
5      x.i = 1
6    } else {
7      x.i = 2
8    }
9  }
```

Interval domain end-state:



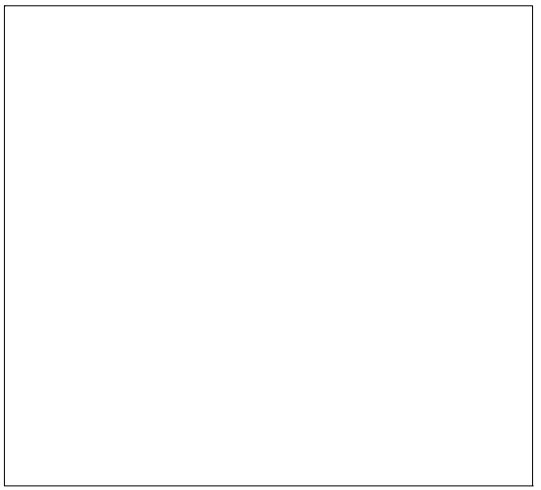
Interval domain end-state:

```
n1 -> T
n2 -> [1..2]
```

Example: Initializing a List

```
1  def createZeroList(n: Int) = {
2    var h: IntNode = null
3    var cur :IntNode = null
4    var i = 0
5
6    while (i < n) {
7      cur = new IntNode
8      cur.i = 0
9      cur.n = h
10     h = cur
11     i += 1
12   }
13 }
```

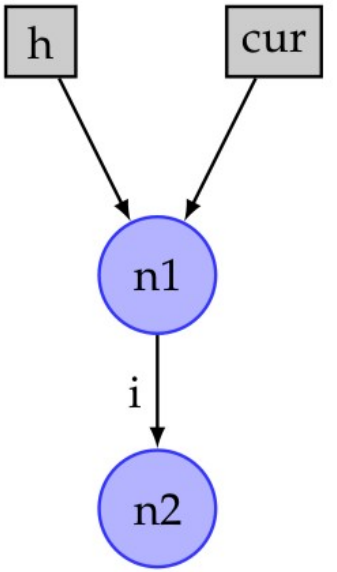
Heap Entry-State



Example: Initializing a List

```
1  def createZeroList(n: Int) = {
2    var h: IntNode = null
3    var cur :IntNode = null
4    var i = 0
5
6    while (i < n) {
7      cur = new IntNode
8      cur.i = 0
9      cur.n = h
10     h = cur
11     i += 1
12   }
13 }
```

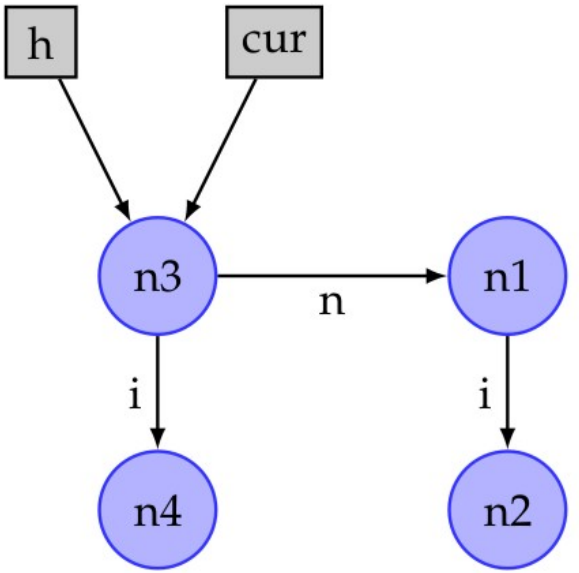
After first iteration:



Example: Initializing a List

```
1  def createZeroList(n: Int) = {
2    var h: IntNode = null
3    var cur :IntNode = null
4    var i = 0
5
6    while (i < n) {
7      cur = new IntNode
8      cur.i = 0
9      cur.n = h
10     h = cur
11     i += 1
12   }
13 }
```

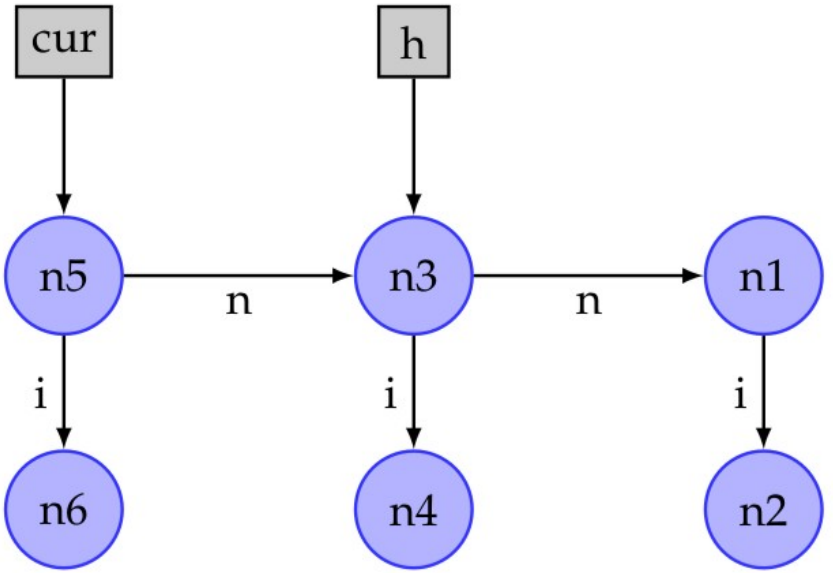
After second iteration:



Example: Initializing a List

```
1  def createZeroList(n: Int) = {
2    var h: IntNode = null
3    var cur :IntNode = null
4    var i = 0
5
6    while (i < n) {
7      cur = new IntNode
8      cur.i = 0
9      cur.n = h
10     h = cur
11     i += 1
12   }
13 }
```

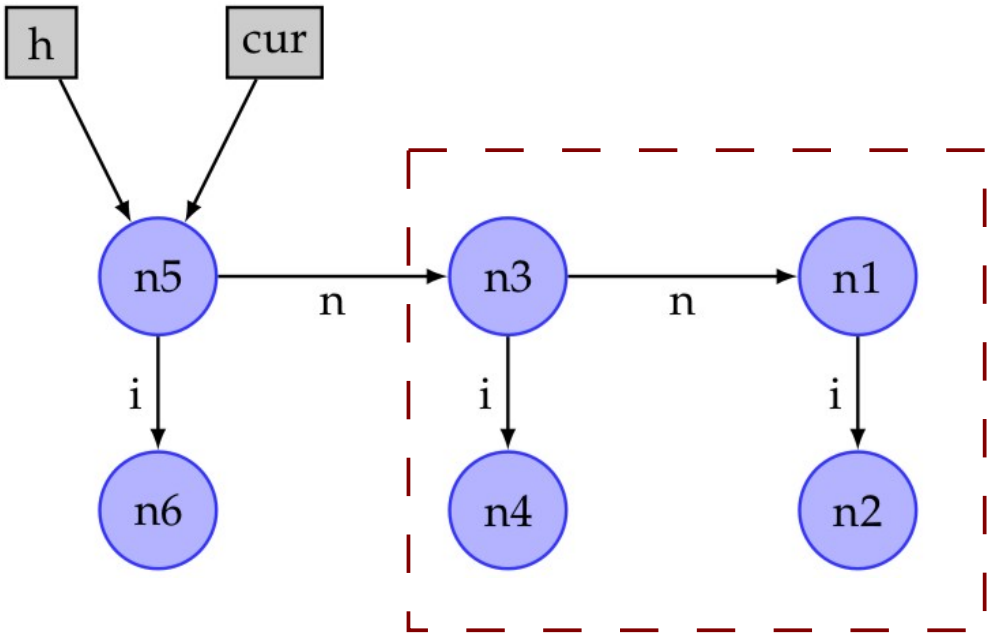
Before `h = cur`
(third iteration)



Example: Initializing a List

```
1  def createZeroList(n: Int) = {
2    var h: IntNode = null
3    var cur :IntNode = null
4    var i = 0
5
6    while (i < n) {
7      cur = new IntNode
8      cur.i = 0
9      cur.n = h
10     h = cur
11     i += 1
12   }
13 }
```

After `h = cur`
(third iteration)

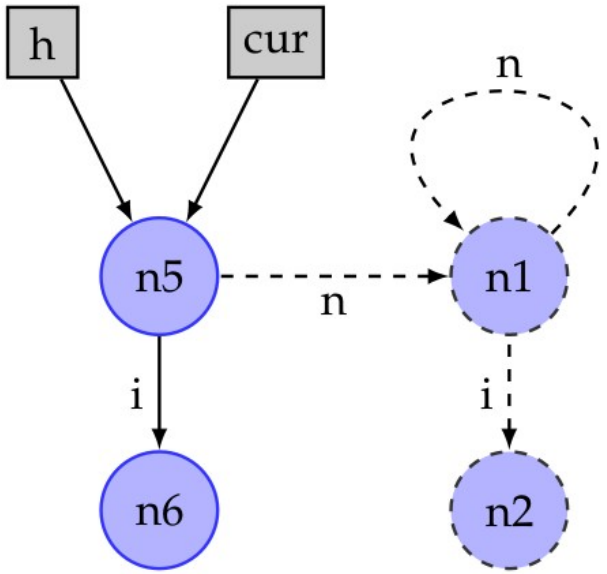


n1 and n3
n2 and n4

look the same to TVLA!

Example: Initializing a List

Summarization of n1, n3 and n2, n4

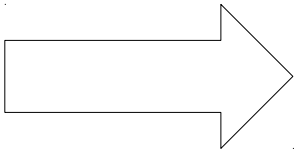


{n1, n3} -> {n1}

{n2, n4} -> {n2}

Heap ID	Semantic Domain
n2	[0..0]
n4	[0..0]
n6	[0..0]

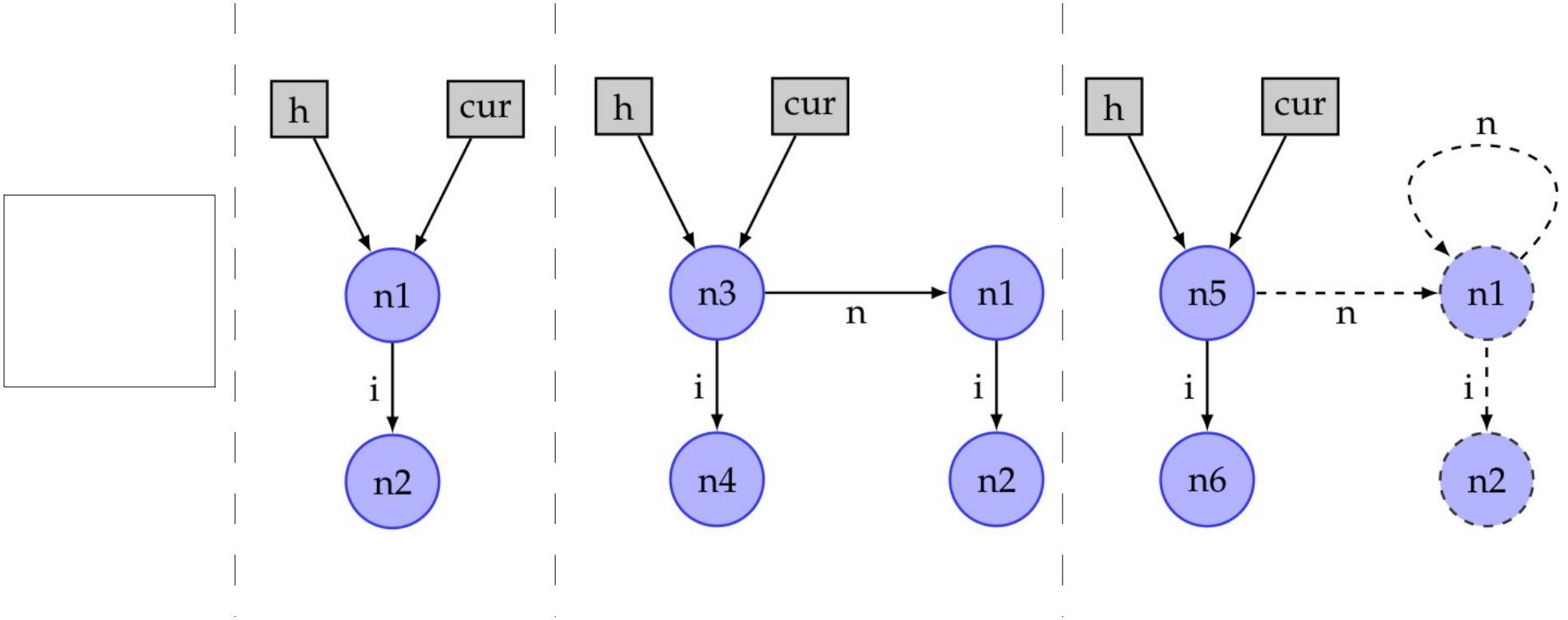
n2 ⊔ n4



Heap ID	Semantic Domain
n2	[0..0]
n6	[0..0]

Example: Initializing a List

Resulting Structures



Testing & Performance

- Developed test suite with ~ 40 tests to evaluate our domain.
- Total execution time: ~5 min (previously: 45)
- Calls to TVLA a severe performance bottleneck
  - JVM start-up time:

    => ~ 5x speed-up
  - Inefficiency of CFG iteration: Caching!

    => ~ 2x speed-up

Questions?