Frequency-Aware Escape Analysis

Roberto Castañeda Lozano, Daniel Lundén, Saranya Natarajan, Antón Seoane Ampudia, Daniel Skantz, Jesper Wilhelmsson

2025-10-20

Outline

Control-Flow Sensitivity

Frequency-Aware Escape Analysis

Example

Challenges

Control-Flow Sensitivity

Frequency-Aware Escape Analysis

Example

Challenges

What Does “Control-Flow Sensitive” Really Mean?

Escape partiality?

```
1  Foo foo = new Foo();
2  if (...) {
3      return null;
4  } else {
5      return foo;
6  }
```


Escape partiality?

```
1  Foo foo = new Foo();
2  if (...) {
3      return null;
4  } else {
5      return foo;
6  }
```

- Without: `foo` escapes

Escape partiality?

```
1  Foo foo = new Foo();
2  if (...) {
3      return null;
4  } else {
5      return foo;
6  }
```

- Without: `foo` escapes
- With: `foo` escapes **at line 5**

Allocation Sinkability?

```
1  Foo foo = new Foo();
2  if (...) {
3      return null;
4  } else {
5
6      return foo;
7  }
```


Allocation Sinkability?

```
1  Foo foo = new Foo();
2  if (...) {
3      return null;
4  } else {
5
6      return foo;
7  }
```

- Without: allocation is fixed

Allocation Sinkability?

```
1  Foo foo = new Foo();
2  if (...) {
3      return null;
4  } else {
5
6      return foo;
7  }
```

- Without: allocation is fixed
- With: allocation can be sunk

Allocation Sinkability?

```
1
2  if (...) {
3      return null;
4  } else {
5      Foo foo = new Foo();
6      return foo;
7  }
```

- Without: allocation is fixed
- With: allocation can be sunk

Scalar Replacement Across Merges?

```
1
2   if (...) {
3       dog = new Dog();
4       dog.age = 25;
5   } else {
6       if (...) {
7           dog = new Dog();
8           dog.age = 12;
9       } else {
10          dog = new Dog();
11          dog.age = 3;
12      }
13      dog.name = ”Bobby”;
14  }
15  return dog.name;
```


Scalar Replacement Across Merges?

```
1
2   if (...) {
3       dog = new Dog();
4       dog.age = 25;
5   } else {
6       if (...) {
7           dog = new Dog();
8           dog.age = 12;
9       } else {
10          dog = new Dog();
11          dog.age = 3;
12      }
13      dog.name = ”Bobby”;
14  }
15  return dog.name;
```

- Without: two-level allocation merges, cannot scalar replace

Scalar Replacement Across Merges?

```
1
2   if (...) {
3       dog = new Dog();
4       dog.age = 25;
5   } else {
6       if (...) {
7           dog = new Dog();
8           dog.age = 12;
9       } else {
10          dog = new Dog();
11          dog.age = 3;
12      }
13      dog.name = ”Bobby”;
14  }
15  return dog.name;
```

- Without: two-level allocation merges, cannot scalar replace
- With: can scalar replace through arbitrary allocation merges

Scalar Replacement Across Merges?

```
1   String dogName;
2   if (...) {
3
4
5   } else {
6       dogName = ”Bobby”;
7
8
9
10
11
12
13
14  }
15  return dogName;
```

- Without: two-level allocation merges, cannot scalar replace
- With: can scalar replace through arbitrary allocation merges

Tracing of Changes in Object Connectivity?

```
1  Foo foo = new Foo();
2  Bar bar = new Bar();
3  foo.a = bar;
4  ...
5  foo.a = null;
6  return foo;
```


Tracing of Changes in Object Connectivity?

```
1  Foo foo = new Foo();
2  Bar bar = new Bar();
3  foo.a = bar;
4  ...
5  foo.a = null;
6  return foo;
```

- Without: both `foo` and `bar` escape

Tracing of Changes in Object Connectivity?

```
1  Foo foo = new Foo();
2  Bar bar = new Bar();
3  foo.a = bar;
4  ...
5  foo.a = null;
6  return foo;
```

- Without: both `foo` and `bar` escape
- With: only `foo` escapes

Control-Flow Sensitivity

Frequency-Aware Escape Analysis

Example

Challenges

Frequency-Aware Escape Analysis

- Goal: leverage the most profitable features of “classic” partial EA

Freque

Partial Escape Analysis and Scalar Replacement for Java

► G

DISSERTATION

submitted in partial fulfillment of the requirements
for the academic degree

Doktor der technischen Wissenschaften

in the Doctoral Program in Engineering Sciences

Submitted by

Dipl.-Ing. Lukas Stadler

Frequency-Aware Escape Analysis

- Goal: leverage the most profitable features of “classic” partial EA

Frequency-Aware Escape Analysis

- Goal: leverage the most profitable features of “classic” partial EA
  - with the least amount of complexity

Frequency-Aware Escape Analysis

- Goal: leverage the most profitable features of “classic” partial EA
  - with the least amount of complexity
  - in a manner that fits C2’s design as much as possible

Frequency-Aware Escape Analysis

- Goal: leverage the most profitable features of “classic” partial EA
  - with the least amount of complexity
  - in a manner that fits C2’s design as much as possible
- Features
  - partial escape analysis

Frequency-Aware Escape Analysis

- Goal: leverage the most profitable features of “classic” partial EA
  - with the least amount of complexity
  - in a manner that fits C2’s design as much as possible
- Features
  - partial escape analysis
  - allocation sinking

Frequency-Aware Escape Analysis

- Goal: leverage the most profitable features of “classic” partial EA
  - with the least amount of complexity
  - in a manner that fits C2’s design as much as possible
- Features
  - partial escape analysis
  - allocation sinking
  - allocation merge-agnostic

Frequency-Aware Escape Analysis

- Goal: leverage the most profitable features of “classic” partial EA
  - with the least amount of complexity
  - in a manner that fits C2’s design as much as possible
- Features
  - partial escape analysis
  - allocation sinking
  - allocation merge-agnostic
- Key design decision: assume static (conservative) object connectivity

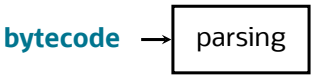
Frequency-Aware Escape Analysis

- Goal: leverage the most profitable features of “classic” partial EA
  - with the least amount of complexity
  - in a manner that fits C2’s design as much as possible
- Features
  - partial escape analysis
  - allocation sinking
  - allocation merge-agnostic
- Key design decision: assume static (conservative) object connectivity
  - theoretical analysis precision loss but seemingly low practical impact

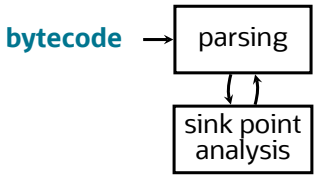
Frequency-Aware Escape Analysis

- Goal: leverage the most profitable features of “classic” partial EA
  - with the least amount of complexity
  - in a manner that fits C2’s design as much as possible
- Features
  - partial escape analysis
  - allocation sinking
  - allocation merge-agnostic
- Key design decision: assume static (conservative) object connectivity
  - theoretical analysis precision loss but seemingly low practical impact
  - not overall worse, this simplification has significant benefits (more later)

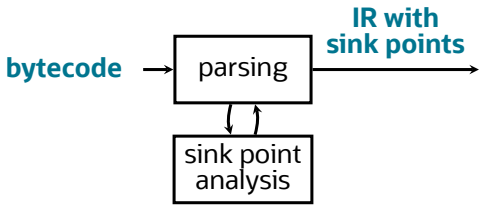
Frequency-Aware Escape Analysis: Architecture



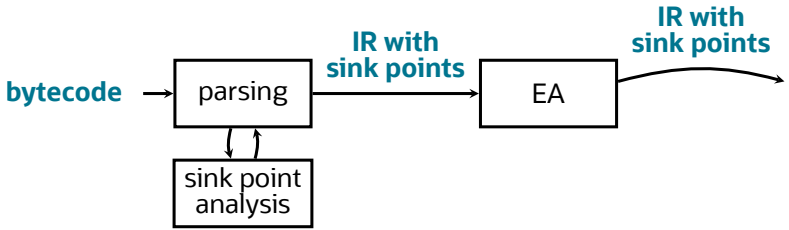
Frequency-Aware Escape Analysis: Architecture



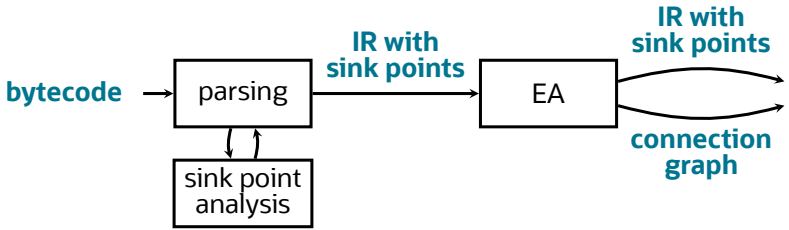
Frequency-Aware Escape Analysis: Architecture



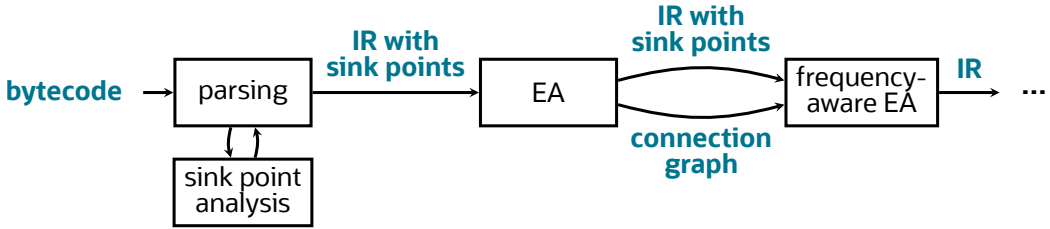
Frequency-Aware Escape Analysis: Architecture



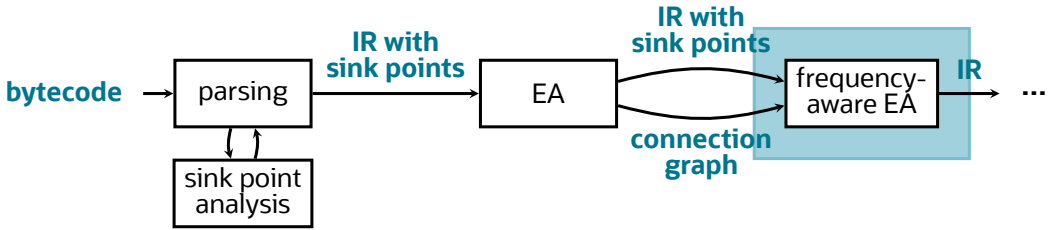
Frequency-Aware Escape Analysis: Architecture



Frequency-Aware Escape Analysis: Architecture



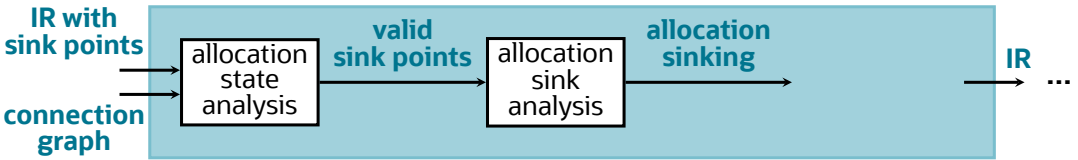
Frequency-Aware Escape Analysis: Architecture



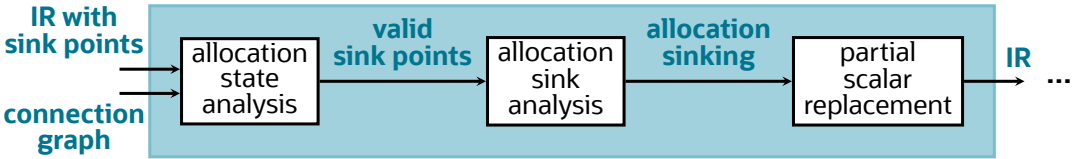
Frequency-Aware Escape Analysis: Architecture

Frequency-Aware Escape Analysis: Architecture

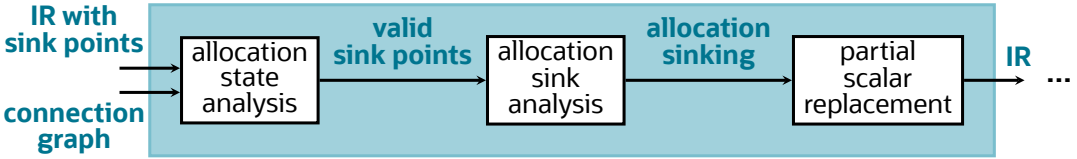
Frequency-Aware Escape Analysis: Architecture



Frequency-Aware Escape Analysis: Architecture

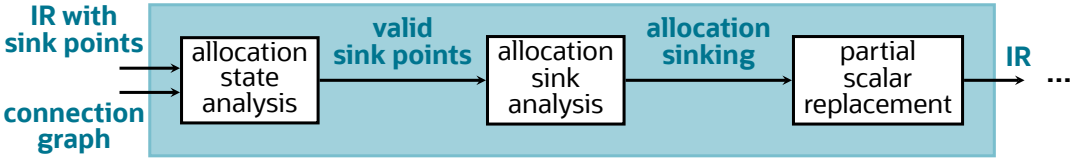


Frequency-Aware Escape Analysis: Architecture



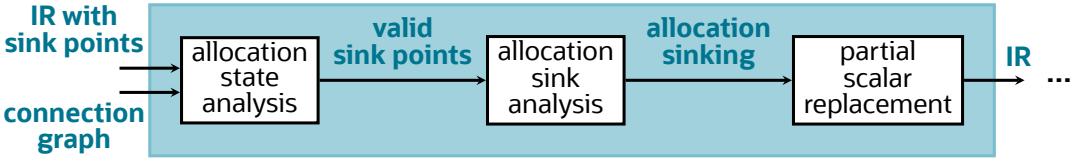
- Important difference with Stadler: decoupled analysis and transformation

Frequency-Aware Escape Analysis: Architecture



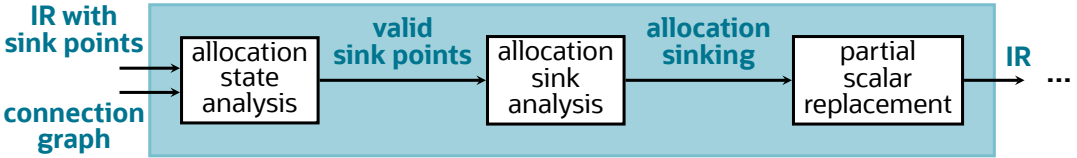
- Important difference with Stadler: decoupled analysis and transformation
  - less local and greedy, potentially better decisions

Frequency-Aware Escape Analysis: Architecture



- Important difference with Stadler: decoupled analysis and transformation
  - less local and greedy, potentially better decisions
  - more modular and maintainable

Frequency-Aware Escape Analysis: Architecture



- Important difference with Stadler: decoupled analysis and transformation
  - less local and greedy, potentially better decisions
  - more modular and maintainable
  - enabled by static object connectivity model

Control-Flow Sensitivity

Frequency-Aware Escape Analysis

Example

Challenges

Example

```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23
24          person.walk();
25      }
26
27  }
```


After Sink Point Analysis

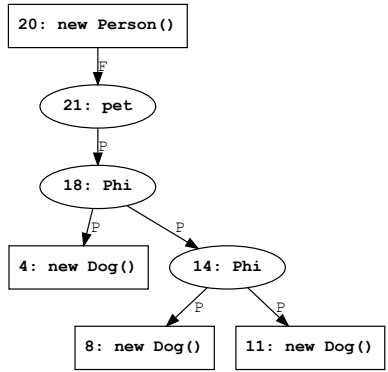
```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```


After Sink Point Analysis

```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```

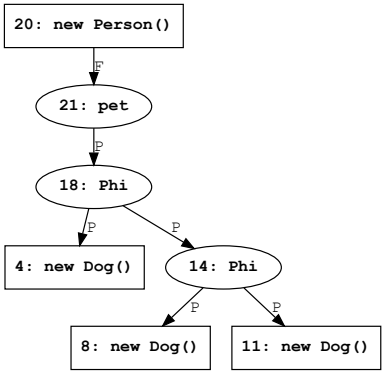

After EA (Input to Frequency-Aware EA)

```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



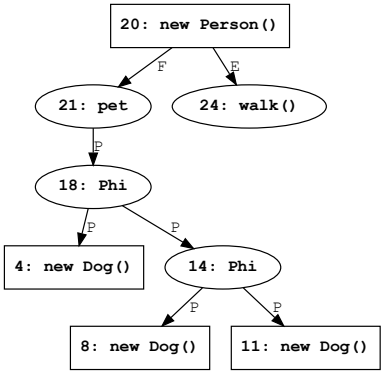
Allocation State Analysis: Assigning Allocations to Escape Sites

```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



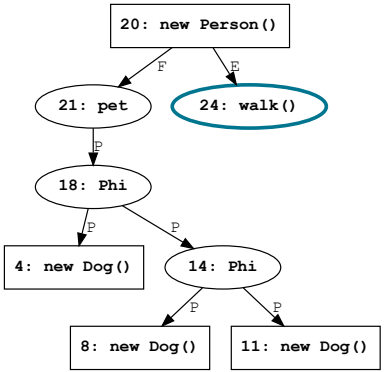
Allocation State Analysis: Assigning Allocations to Escape Sites

```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



Allocation State Analysis: Assigning Allocations to Escape Sites

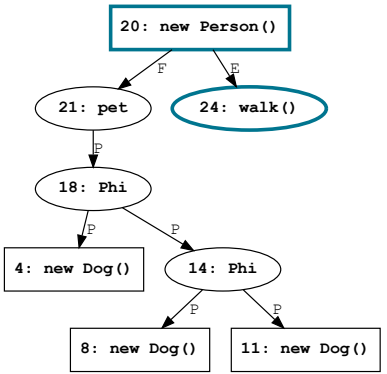
```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



escaping-allocations(24) =

Allocation State Analysis: Assigning Allocations to Escape Sites

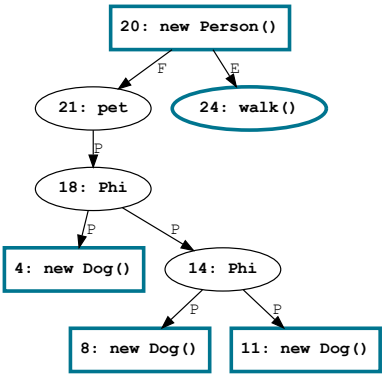
```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



$$\text{escaping-allocations}(24) = \{20\} \cup$$

Allocation State Analysis: Assigning Allocations to Escape Sites

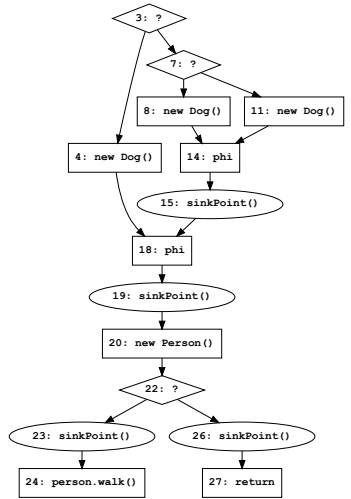
```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = "Bobby";
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



$$\text{escaping-allocations}(24) = \{20\} \cup \{4, 8, 11\}$$

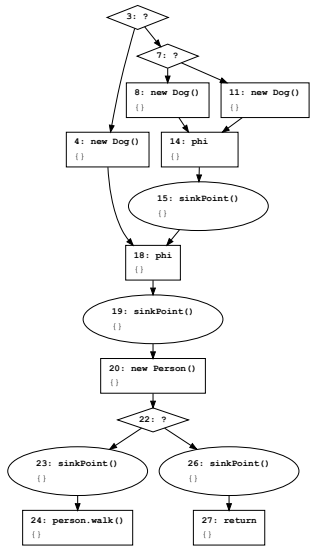
Allocation State Analysis: Projecting a CFG

```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



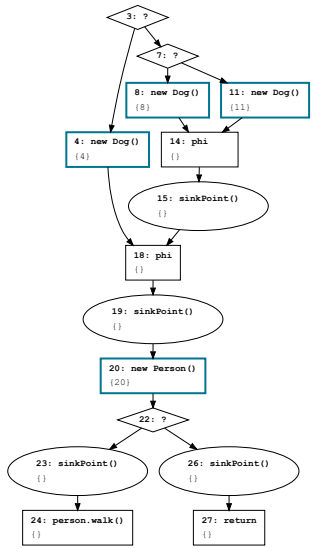
Allocation State Analysis: Propagating Virtual State

```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



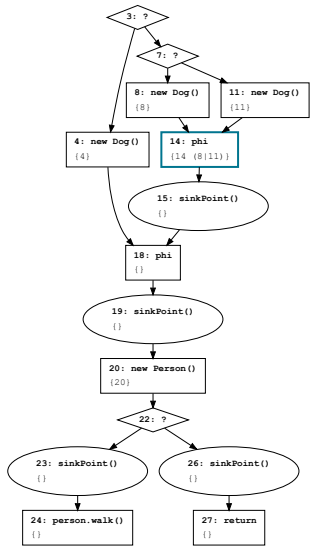
Allocation State Analysis: Propagating Virtual State

```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



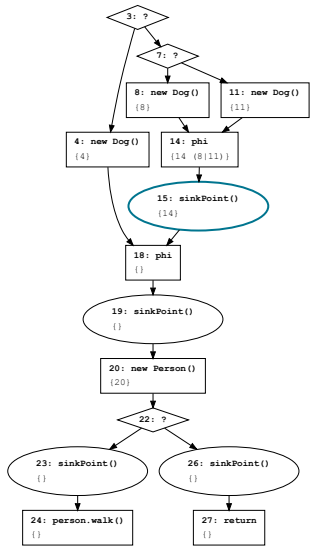
Allocation State Analysis: Propagating Virtual State

```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



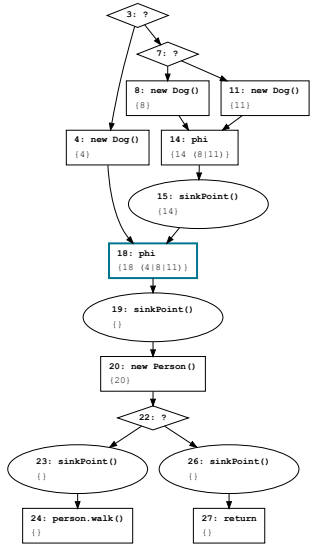
Allocation State Analysis: Propagating Virtual State

```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



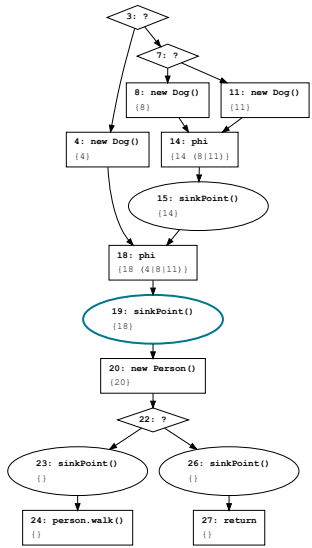
Allocation State Analysis: Propagating Virtual State

```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



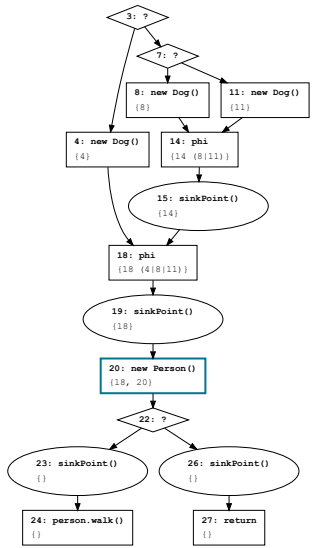
Allocation State Analysis: Propagating Virtual State

```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



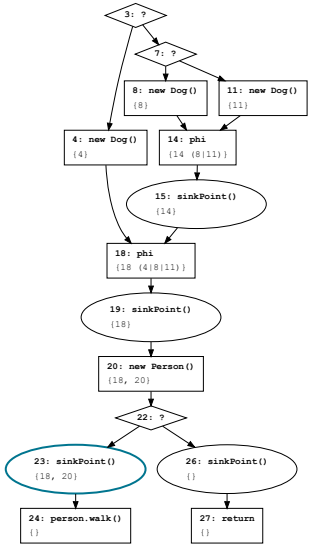
Allocation State Analysis: Propagating Virtual State

```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



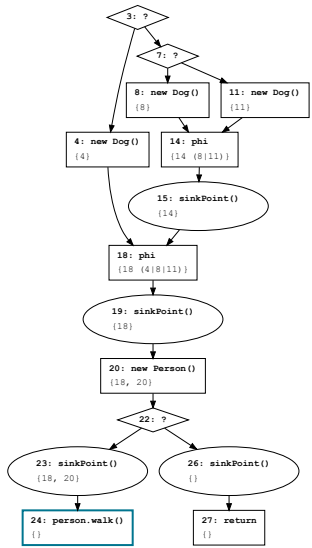
Allocation State Analysis: Propagating Virtual State

```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



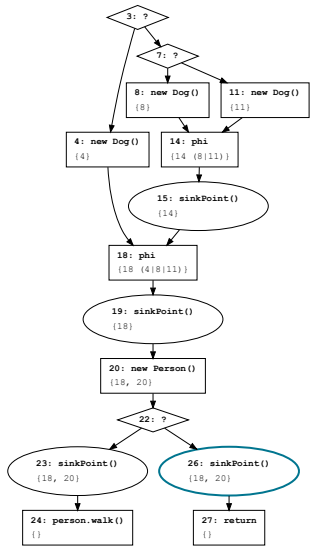
Allocation State Analysis: Propagating Virtual State

```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



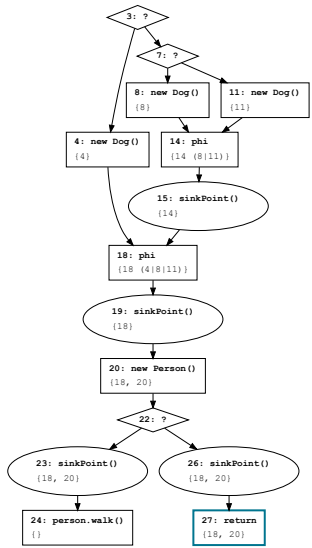
Allocation State Analysis: Propagating Virtual State

```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



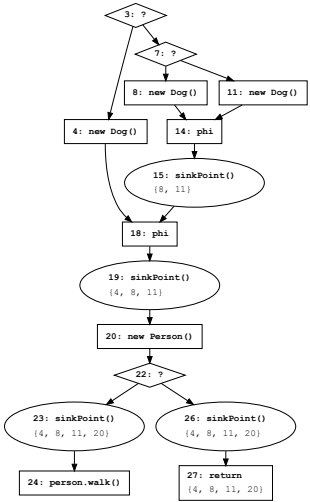
Allocation State Analysis: Propagating Virtual State

```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



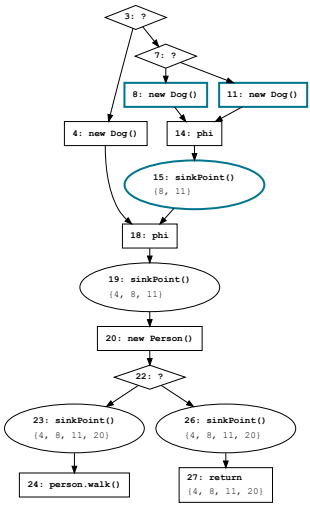
Allocation State Analysis: Result

```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



Allocation State Analysis: Result

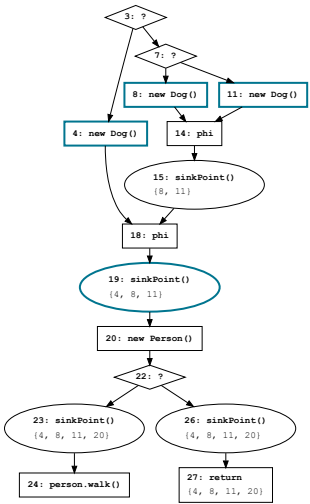
```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



- $\{8, 11\}$ can be sunk to 15

Allocation State Analysis: Result

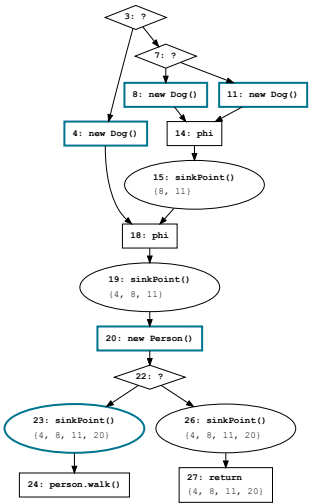
```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



- $\{8, 11\}$ can be sunk to 15
- $\{4, 8, 11\}$ can be sunk to 19

Allocation State Analysis: Result

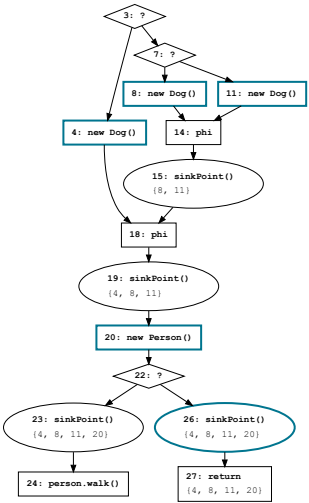
```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



- $\{8, 11\}$ can be sunk to 15
- $\{4, 8, 11\}$ can be sunk to 19
- $\{4, 8, 11, 20\}$ can be sunk to 23

Allocation State Analysis: Result

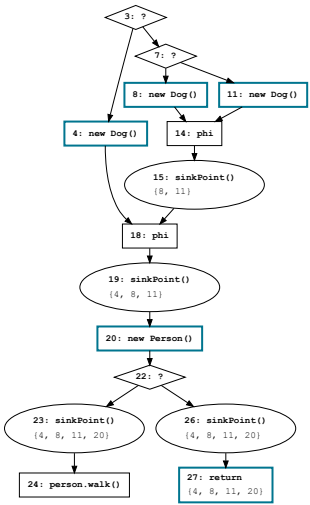
```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = ”Bobby”;
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



- $\{8, 11\}$ can be sunk to 15
- $\{4, 8, 11\}$ can be sunk to 19
- $\{4, 8, 11, 20\}$ can be sunk to 23
- $\{4, 8, 11, 20\}$ can be sunk to 26

Allocation State Analysis: Result

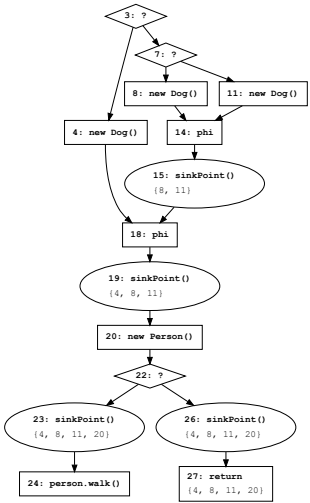
```
1   static void run(int dogLifeStage) {
2       Dog dog;
3       if (dogLifeStage == 2) {
4           dog = new Dog();
5           dog.age = 25;
6       } else {
7           if (dogLifeStage == 1) {
8               dog = new Dog();
9               dog.age = 12;
10          } else {
11              dog = new Dog();
12              dog.age = 3;
13          }
14          // dog <- phi (...)
15          sinkPoint();
16          dog.name = "Bobby";
17      }
18      // dog <- phi (...)
19      sinkPoint();
20      Person person = new Person();
21      person.pet = dog;
22      if (dogLifeStage < 2) {
23          sinkPoint();
24          person.walk();
25      }
26      sinkPoint();
27  }
```



- $\{8, 11\}$ can be sunk to 15
- $\{4, 8, 11\}$ can be sunk to 19
- $\{4, 8, 11, 20\}$ can be sunk to 23
- $\{4, 8, 11, 20\}$ can be sunk to 26
- $\{4, 8, 11, 20\}$ can be sunk **below** 27

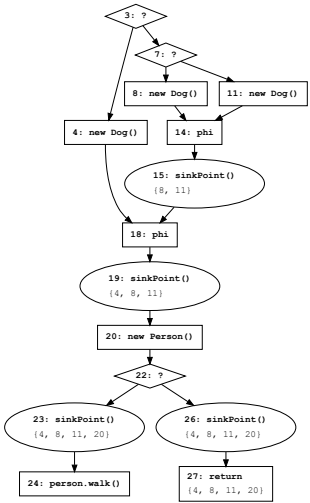
Allocation Sink Analysis: Feasible Sinkings

- Given
  - a projected CFG



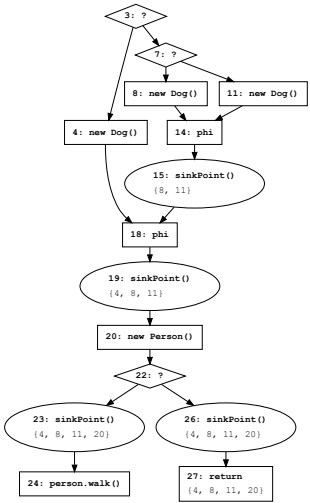
Allocation Sink Analysis: Feasible Sinkings

- Given
  - a projected CFG
  - a set of valid sink points for each allocation



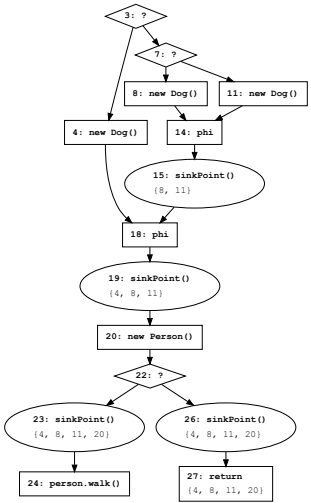
Allocation Sink Analysis: Feasible Sinkings

- Given
  - a projected CFG
  - a set of valid sink points for each allocation
- Find a minimum-cost “sinking” of all allocations



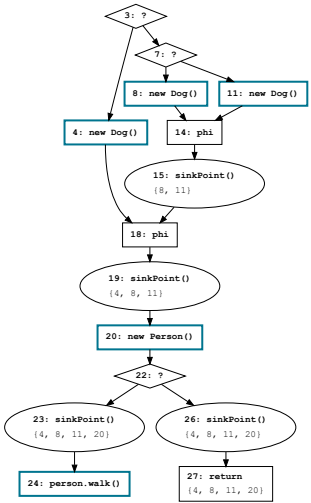
Allocation Sink Analysis: Feasible Sinkings

- Given
  - a projected CFG
  - a set of valid sink points for each allocation
- Find a minimum-cost “sinking” of all allocations
- Subject to:
  - allocations dominate their escape points



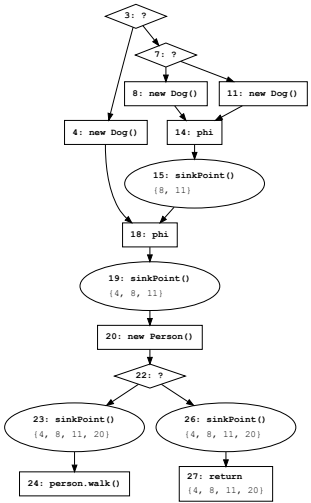
Allocation Sink Analysis: Feasible Sinkings

- Given
  - a projected CFG
  - a set of valid sink points for each allocation
- Find a minimum-cost “sinking” of all allocations
- Subject to:
  - allocations dominate their escape points
    $\{4, 8, 11, 20\}$ dominate 24



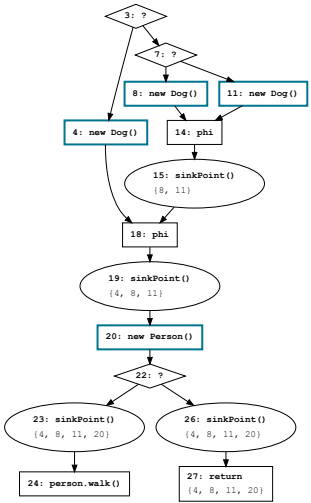
Allocation Sink Analysis: Feasible Sinkings

- Given
  - a projected CFG
  - a set of valid sink points for each allocation
- Find a minimum-cost “sinking” of all allocations
- Subject to:
  - allocations dominate their escape points
    $\{4, 8, 11, 20\}$ dominate 24
  - allocations are dominated by their dependencies



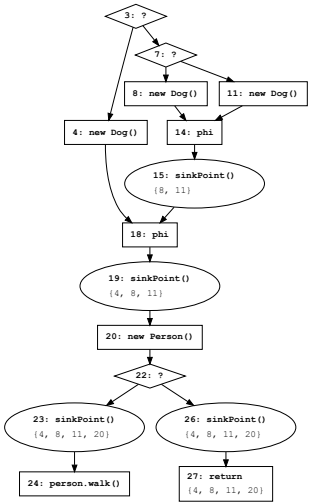
Allocation Sink Analysis: Feasible Sinkings

- Given
  - a projected CFG
  - a set of valid sink points for each allocation
- Find a minimum-cost “sinking” of all allocations
- Subject to:
  - allocations dominate their escape points
    $\{4, 8, 11, 20\}$ dominate 24
  - allocations are dominated by their dependencies
    $\{20\}$ is dominated by $\{4, 8, 11\}$



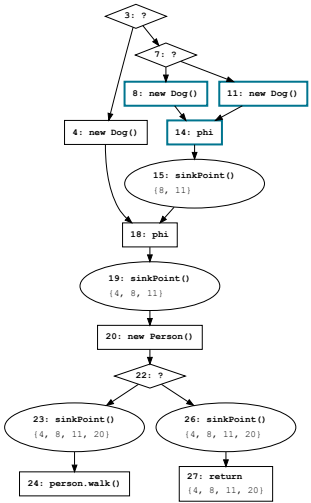
Allocation Sink Analysis: Feasible Sinkings

- Given
  - a projected CFG
  - a set of valid sink points for each allocation
- Find a minimum-cost “sinking” of all allocations
- Subject to:
  - allocations dominate their escape points
    $\{4, 8, 11, 20\}$ dominate 24
  - allocations are dominated by their dependencies
    $\{20\}$ is dominated by $\{4, 8, 11\}$
  - ϕ-related allocations are sunk and merged together



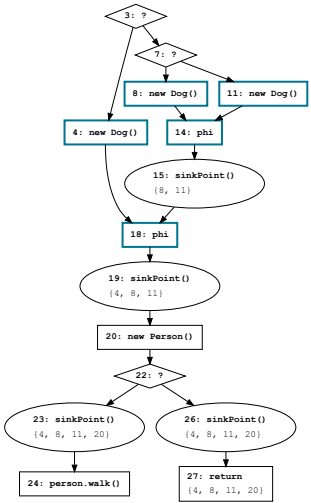
Allocation Sink Analysis: Feasible Sinkings

- Given
  - a projected CFG
  - a set of valid sink points for each allocation
- Find a minimum-cost "sinking" of all allocations
- Subject to:
  - allocations dominate their escape points
    $\{4, 8, 11, 20\}$ dominate 24
  - allocations are dominated by their dependencies
    $\{20\}$ is dominated by $\{4, 8, 11\}$
  - ϕ-related allocations are sunk and merged together
    if 8 is sunk below 14, so is 11 (and vice versa)



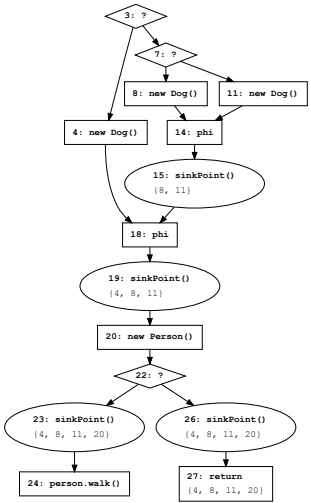
Allocation Sink Analysis: Feasible Sinkings

- Given
  - a projected CFG
  - a set of valid sink points for each allocation
- Find a minimum-cost "sinking" of all allocations
- Subject to:
  - allocations dominate their escape points
    $\{4, 8, 11, 20\}$ dominate 24
  - allocations are dominated by their dependencies
    $\{20\}$ is dominated by $\{4, 8, 11\}$
  - ϕ-related allocations are sunk and merged together
    if 8 is sunk below 14, so is 11 (and vice versa)
    if 4 is sunk below 18, so is $\{8, 11\}$ (and vice versa)



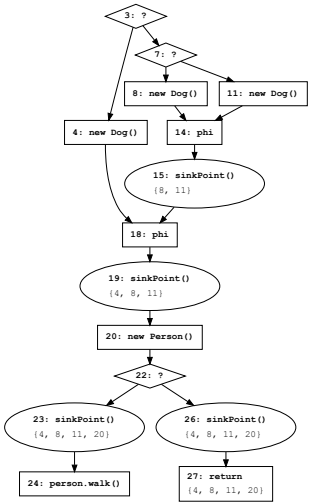
Allocation Sink Analysis: Feasible Sinkings

- Given
  - a projected CFG
  - a set of valid sink points for each allocation
- Find a minimum-cost “sinking” of all allocations
- Subject to:
  - allocations dominate their escape points
    $\{4, 8, 11, 20\}$ dominate 24
  - allocations are dominated by their dependencies
    $\{20\}$ is dominated by $\{4, 8, 11\}$
  - ϕ-related allocations are sunk and merged together
    if 8 is sunk below 14, so is 11 (and vice versa)
    if 4 is sunk below 18, so is $\{8, 11\}$ (and vice versa)
  - identity semantics are preserved



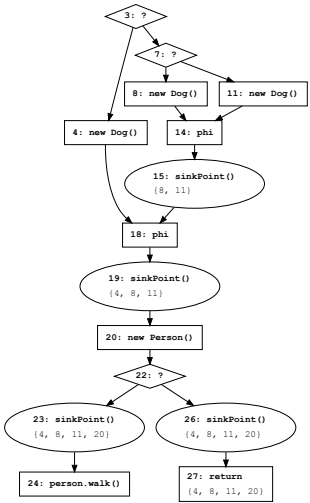
Allocation Sink Analysis: Feasible Sinkings

- Given
  - a projected CFG
  - a set of valid sink points for each allocation
- Find a minimum-cost “sinking” of all allocations
- Subject to:
  - allocations dominate their escape points
    $\{4, 8, 11, 20\}$ dominate 24
  - allocations are dominated by their dependencies
    $\{20\}$ is dominated by $\{4, 8, 11\}$
  - ϕ-related allocations are sunk and merged together
    if 8 is sunk below 14, so is 11 (and vice versa)
    if 4 is sunk below 18, so is $\{8, 11\}$ (and vice versa)
  - identity semantics are preserved
  - allocations cannot be sunk within a synchronized region
  - …



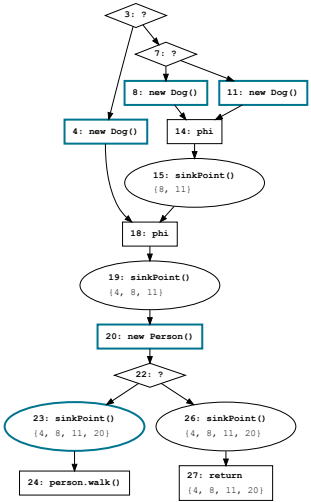
Allocation Sink Analysis: Cost Model

- Allocation cost weighted by execution frequency



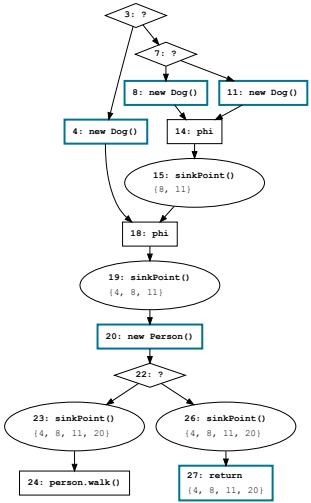
Allocation Sink Analysis: Cost Model

- Allocation cost weighted by execution frequency
  it is cheaper to sink $\{4, 8, 11, 20\}$ to 23



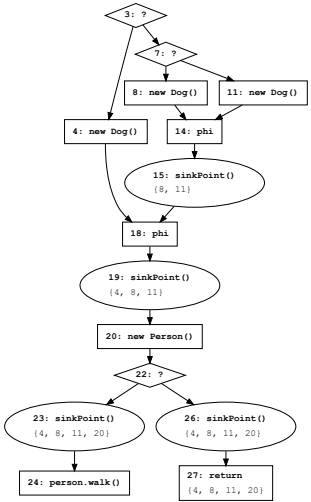
Allocation Sink Analysis: Cost Model

- Allocation cost weighted by execution frequency
  it is cheaper to sink $\{4, 8, 11, 20\}$ to 23
  it is **free** to sink $\{4, 8, 11, 20\}$ below 27!



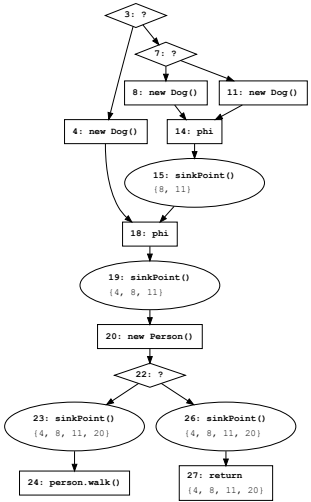
Allocation Sink Analysis: Cost Model

- Allocation cost weighted by execution frequency
  it is cheaper to sink $\{4, 8, 11, 20\}$ to 23
  it is **free** to sink $\{4, 8, 11, 20\}$ below 27!
- "GC pressure"



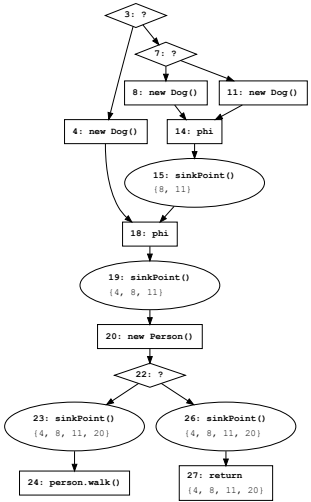
Allocation Sink Analysis: Cost Model

- Allocation cost weighted by execution frequency
  it is cheaper to sink $\{4, 8, 11, 20\}$ to 23
  it is **free** to sink $\{4, 8, 11, 20\}$ below 27!
- "GC pressure"
  weight allocation cost also by object size



Allocation Sink Analysis: Cost Model

- Allocation cost weighted by execution frequency
  it is cheaper to sink $\{4, 8, 11, 20\}$ to 23
  it is **free** to sink $\{4, 8, 11, 20\}$ below 27!
- "GC pressure"
  weight allocation cost also by object size
- Code size



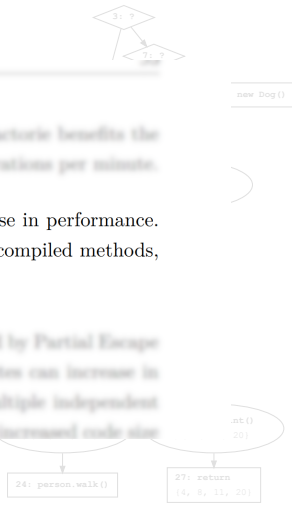
Allocation Sink Analysis: Cost Model

- Allocation cost weighted by execution frequency
- "C
- C

in performance, with many being above 10%. ScalaDaCapo factorie benefits the most in terms of performance, with a 33% improvement in iterations per minute.

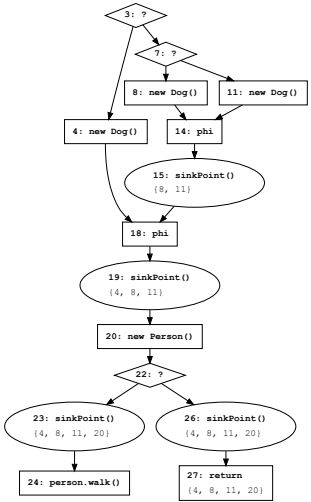
Notably, the DaCapo jython benchmark shows a 2.1% decrease in performance. Partial Escape Analysis can in rare cases increase the size of compiled methods, which has a negative influence on this benchmark.

While there will never be more dynamic allocations in code optimized by Partial Escape Analysis than in the unoptimized code, the number of allocation sites can increase in some cases, e.g., when the same virtual object is materialized in multiple independent branches. Benchmarks that are sensitive to this may suffer from the increased code size



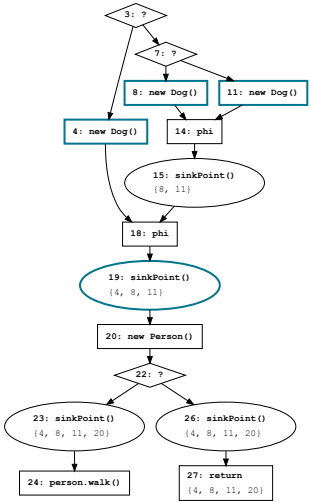
Allocation Sink Analysis: Cost Model

- Allocation cost weighted by execution frequency
  it is cheaper to sink $\{4, 8, 11, 20\}$ to 23
  it is **free** to sink $\{4, 8, 11, 20\}$ below 27!
- "GC pressure"
  weight allocation cost also by object size
- Code size



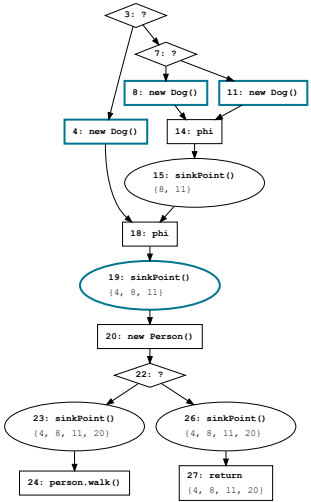
Allocation Sink Analysis: Cost Model

- Allocation cost weighted by execution frequency
  it is cheaper to sink $\{4, 8, 11, 20\}$ to 23
  it is **free** to sink $\{4, 8, 11, 20\}$ below 27!
- "GC pressure"
  weight allocation cost also by object size
- Code size
  it is cheaper to sink $\{4, 8, 11\}$ to 19



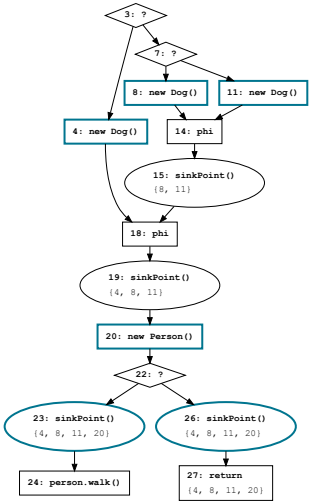
Allocation Sink Analysis: Cost Model

- Allocation cost weighted by execution frequency
  it is cheaper to sink $\{4, 8, 11, 20\}$ to 23
  it is **free** to sink $\{4, 8, 11, 20\}$ below 27!
- "GC pressure"
  weight allocation cost also by object size
- Code size
  it is cheaper to sink $\{4, 8, 11\}$ to 19
  sinking $\{4, 8, 11\}$ to 19



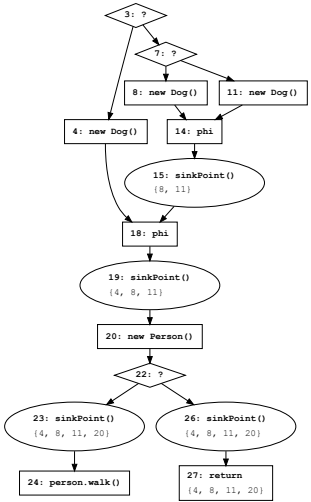
Allocation Sink Analysis: Cost Model

- Allocation cost weighted by execution frequency
  it is cheaper to sink $\{4, 8, 11, 20\}$ to 23
  it is **free** to sink $\{4, 8, 11, 20\}$ below 27!
- "GC pressure"
  weight allocation cost also by object size
- Code size
  it is cheaper to sink $\{4, 8, 11\}$ to 19
  sinking $\{4, 8, 11\}$ to 19 is cheaper than sinking them (and 20) to both 23 and 26



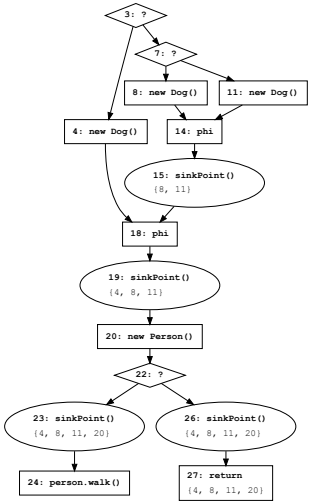
Allocation Sink Analysis: Cost Model

- Allocation cost weighted by execution frequency
  it is cheaper to sink $\{4, 8, 11, 20\}$ to 23
  it is **free** to sink $\{4, 8, 11, 20\}$ below 27!
- "GC pressure"
  weight allocation cost also by object size
- Code size
  it is cheaper to sink $\{4, 8, 11\}$ to 19
  sinking $\{4, 8, 11\}$ to 19 is cheaper than sinking them (and 20) to both 23 and 26
- Heap accesses



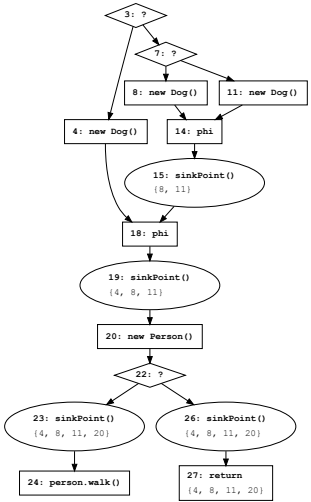
Allocation Sink Analysis: Cost Model

- Allocation cost weighted by execution frequency
  it is cheaper to sink $\{4, 8, 11, 20\}$ to 23
  it is **free** to sink $\{4, 8, 11, 20\}$ below 27!
- "GC pressure"
  weight allocation cost also by object size
- Code size
  it is cheaper to sink $\{4, 8, 11\}$ to 19
  sinking $\{4, 8, 11\}$ to 19 is cheaper than sinking them (and 20) to both 23 and 26
- Heap accesses
- Identity operations

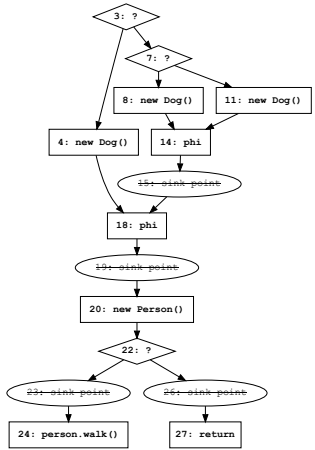


Allocation Sink Analysis: Cost Model

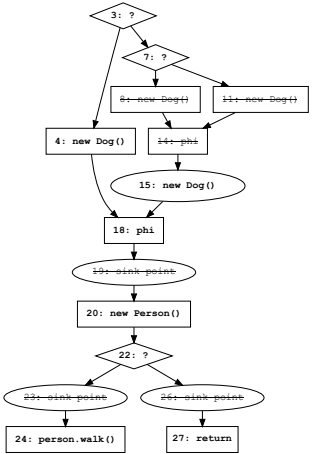
- Allocation cost weighted by execution frequency
  it is cheaper to sink $\{4, 8, 11, 20\}$ to 23
  it is **free** to sink $\{4, 8, 11, 20\}$ below 27!
- "GC pressure"
  weight allocation cost also by object size
- Code size
  it is cheaper to sink $\{4, 8, 11\}$ to 19
  sinking $\{4, 8, 11\}$ to 19 is cheaper than sinking them (and 20) to both 23 and 26
- Heap accesses
- Identity operations
- Synchronization
- ...



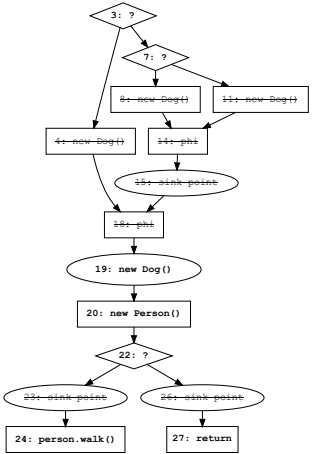
Allocation Sink Analysis: Sinking A (Original)



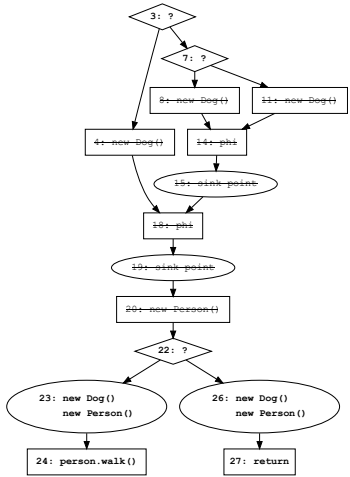
Allocation Sink Analysis: Sinking B (Better Code Size)



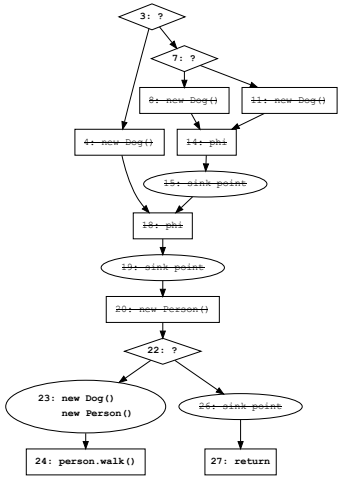
Allocation Sink Analysis: Sinking C (Even Better Code Size)



Allocation Sink Analysis: Sinking D (As Bad as Original)



Allocation Sink Analysis: Sinking E (Best)



Partial Scalar Replacement

- Given
  - an IR

Partial Scalar Replacement

- Given
  - an IR
  - a projected CFG

Partial Scalar Replacement

- Given
  - an IR
  - a projected CFG
  - an assignment of allocations to sink points

Partial Scalar Replacement

- Given
  - an IR
  - a projected CFG
  - an assignment of allocations to sink points
- Transform the IR such that

Partial Scalar Replacement

- Given
  - an IR
  - a projected CFG
  - an assignment of allocations to sink points
- Transform the IR such that
  - Allocations are materialized at their assigned sink points

Partial Scalar Replacement

- Given
  - an IR
  - a projected CFG
  - an assignment of allocations to sink points
- Transform the IR such that
  - Allocations are materialized at their assigned sink points
  - Allocations are scalar-replaced until their materialization point

Partial Scalar Replacement

- Given
  - an IR
  - a projected CFG
  - an assignment of allocations to sink points
- Transform the IR such that
  - Allocations are materialized at their assigned sink points
  - Allocations are scalar-replaced until their materialization point
  - Deoptimization points are instrumented to materialize allocations if needed

Partial Scalar Replacement

- Given
  - an IR
  - a projected CFG
  - an assignment of allocations to sink points
- Transform the IR such that
  - Allocations are materialized at their assigned sink points
  - Allocations are scalar-replaced until their materialization point
  - Deoptimization points are instrumented to materialize allocations if needed
- Algorithm

Partial Scalar Replacement

- Given
  - an IR
  - a projected CFG
  - an assignment of allocations to sink points
- Transform the IR such that
  - Allocations are materialized at their assigned sink points
  - Allocations are scalar-replaced until their materialization point
  - Deoptimization points are instrumented to materialize allocations if needed
- Algorithm
  - Similar to Stadler, traverse projected CFG forwards, keeping track of contents of each virtual allocated object

Partial Scalar Replacement

- Given
  - an IR
  - a projected CFG
  - an assignment of allocations to sink points
- Transform the IR such that
  - Allocations are materialized at their assigned sink points
  - Allocations are scalar-replaced until their materialization point
  - Deoptimization points are instrumented to materialize allocations if needed
- Algorithm
  - Similar to Stadler, traverse projected CFG forwards, keeping track of contents of each virtual allocated object
  - Optimize virtual object operations “on the fly”

Partial Scalar Replacement

- Given
  - an IR
  - a projected CFG
  - an assignment of allocations to sink points
- Transform the IR such that
  - Allocations are materialized at their assigned sink points
  - Allocations are scalar-replaced until their materialization point
  - Deoptimization points are instrumented to materialize allocations if needed
- Algorithm
  - Similar to Stadler, traverse projected CFG forwards, keeping track of contents of each virtual allocated object
  - Optimize virtual object operations “on the fly”
    - resolve field loads

Partial Scalar Replacement

- Given
  - an IR
  - a projected CFG
  - an assignment of allocations to sink points
- Transform the IR such that
  - Allocations are materialized at their assigned sink points
  - Allocations are scalar-replaced until their materialization point
  - Deoptimization points are instrumented to materialize allocations if needed
- Algorithm
  - Similar to Stadler, traverse projected CFG forwards, keeping track of contents of each virtual allocated object
  - Optimize virtual object operations “on the fly”
    - resolve field loads
    - remove synchronization

Partial Scalar Replacement

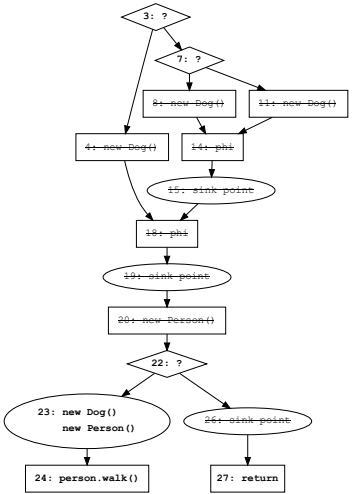
- Given
  - an IR
  - a projected CFG
  - an assignment of allocations to sink points
- Transform the IR such that
  - Allocations are materialized at their assigned sink points
  - Allocations are scalar-replaced until their materialization point
  - Deoptimization points are instrumented to materialize allocations if needed
- Algorithm
  - Similar to Stadler, traverse projected CFG forwards, keeping track of contents of each virtual allocated object
  - Optimize virtual object operations “on the fly”
    - resolve field loads
    - remove synchronization
    - resolve identity comparisons and subtype checks

Partial Scalar Replacement

- Given
  - an IR
  - a projected CFG
  - an assignment of allocations to sink points
- Transform the IR such that
  - Allocations are materialized at their assigned sink points
  - Allocations are scalar-replaced until their materialization point
  - Deoptimization points are instrumented to materialize allocations if needed
- Algorithm
  - Similar to Stadler, traverse projected CFG forwards, keeping track of contents of each virtual allocated object
  - Optimize virtual object operations “on the fly”
    - resolve field loads
    - remove synchronization
    - resolve identity comparisons and subtype checks
    - …

Partial Scalar Replacement: Output for Sinking E

```
static void run(int dogLifeStage) {
    Dog dog;
    if (dogLifeStage == 2) {
        dog = new Dog();
        dog.age = 25;
    } else {
        if (dogLifeStage == 1) {
            dog = new Dog();
            dog.age = 12;
        } else {
            dog = new Dog();
            dog.age = 3;
        }
        dog.name = ”Bobby”;
    }
    Person person = new Person();
    person.pet = dog;
    if (dogLifeStage < 2) {
        person.walk();
    }
}
```




```
static void run(int dogLifeStage) {
    int age;
    String name;
    if (dogLifeStage == 2) {
      age = 25;
      name = null;
    } else {
        if (dogLifeStage == 1) {
            age = 12;
        } else {
            age = 3;
        }
        name = ”Bobby”;
    }
    if (dogLifeStage < 2) {
        Dog dog = new Dog();
        dog.age = age;
        dog.name = name;
        Person person = new Person();
        person.pet = dog;
        person.walk();
    }
}
```


Control-Flow Sensitivity

Frequency-Aware Escape Analysis

Example

Challenges

Challenges: CFG Projection

- Many relevant operations are fixed: allocations, phis, calls, sink points, synchronization, …

Challenges: CFG Projection

- Many relevant operations are fixed: allocations, phis, calls, sink points, synchronization, …
- Others are at least pinned: stores

Challenges: CFG Projection

- Many relevant operations are fixed: allocations, phis, calls, sink points, synchronization, …
- Others are at least pinned: stores
- Others are floating: identity comparisons, subtype checks, …

Challenges: CFG Projection

- Many relevant operations are fixed: allocations, phis, calls, sink points, synchronization, …
- Others are at least pinned: stores
- Others are floating: identity comparisons, subtype checks, …
- While scalarizing: which operations belong to the “virtual lifetime” of the allocation?

Challenges: CFG Projection

- Many relevant operations are fixed: allocations, phis, calls, sink points, synchronization, …
- Others are at least pinned: stores
- Others are floating: identity comparisons, subtype checks, …
- While scalarizing: which operations belong to the “virtual lifetime” of the allocation?

```
1  static boolean createStoreAndCheck() {
2      Object o = new Object();
3      ...
4      sinkPoint();
5      ...
6      store(o); // o escapes, pointed by static field
7      return o == stored;  // floating, no direct IR relation
8                           // with the escaping store
9  }
```


Challenges: CFG Projection

- Many relevant operations are fixed: allocations, phis, calls, sink points, synchronization, …
- Others are at least pinned: stores
- Others are floating: identity comparisons, subtype checks, …
- While scalarizing: which operations belong to the “virtual lifetime” of the allocation?

```
1  static boolean createStoreAndCheck() {
2      Object o = new Object();
3      ...
4      sinkPoint();
5      ...
6      store(o); // o escapes, pointed by static field
7      return o == stored;  // floating, no direct IR relation
8                           // with the escaping store
9  }
```

- More generally, can we always project the CFG we need from C2’s IR?

Challenges: Partial Scalar Replacement

- Ideally, we would like to reuse existing infrastructure

Challenges: Partial Scalar Replacement

- Ideally, we would like to reuse existing infrastructure
- But reusing the current C2 strategy of “split unique types” + IGVN does not seem feasible
  - due to allocation merge limitation

Challenges: Partial Scalar Replacement

- Ideally, we would like to reuse existing infrastructure
- But reusing the current C2 strategy of “split unique types” + IGVN does not seem feasible
  - due to allocation merge limitation
- Might have to implement independently

Challenges: Other

- Preservation of identity semantics

Challenges: Other

- Preservation of identity semantics
- Sink point analysis
  - how to do it efficiently
  - what is the right granularity
  - effect on other optimizations

Challenges: Other

- Preservation of identity semantics
- Sink point analysis
  - how to do it efficiently
  - what is the right granularity
  - effect on other optimizations
  - **also an opportunity**: reuse analysis to bias inlining

Challenges: Other

- Preservation of identity semantics
- Sink point analysis
  - how to do it efficiently
  - what is the right granularity
  - effect on other optimizations
  - **also an opportunity**: reuse analysis to bias inlining
- Heuristic (e.g. frequency estimation)

Challenges: Other

- Preservation of identity semantics
- Sink point analysis
  - how to do it efficiently
  - what is the right granularity
  - effect on other optimizations
  - **also an opportunity**: reuse analysis to bias inlining
- Heuristic (e.g. frequency estimation)
- Compilation time
- …

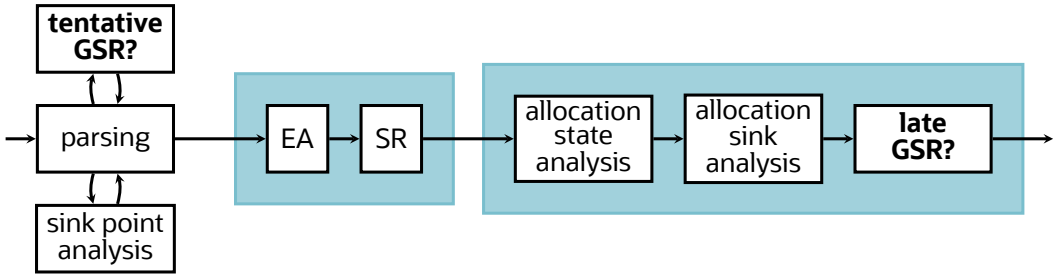
Plan and Status

- ✓ Study literature
- ✓ Study C2 implementation
- ✓ Study opportunities in benchmarks
- ► Gather tests and micro-benchmarks
- ► Prototype general (partial, merge-agnostic) SR (highest uncertainty)

  Apply general SR to current EA (intermediate contribution)

  Implement rest of frequency-aware escape analysis

Plan and Status: Tentative vs. Late General SR



- Tentative GSR: construct dual (heap/scalar) representation during parsing
- Late GSR: infer scalar representation on-demand, after escape analysis

