Allocation Removal by Partial Evaluation in a Tracing JIT

Carl Friedrich Bolz[1] Antonio Cuni[1] Maciej Fijałkowski[2]
Michael Leuschel[1] Samuele Pedroni[3] Armin Rigo[1]

[1]Heinrich-Heine-Universität Düsseldorf, STUPS Group, Germany

[2]merlinux GmbH, Hildesheim, Germany

[3]Open End, Göteborg, Sweden

2011 Workshop on Partial Evaluation and Program Manipulation, January 24, 2011

Dynamic Languages are Slow

- Interpretation overhead
- Type dispatching
- Boxing of primitive types

Dynamic Languages are Slow

- Interpretation overhead
- Type dispatching
- Boxing of primitive types
  - A lot of allocation of short-lived objects

Dynamic Languages are Slow: Example

```
x = a + b; y = x + c
```

evaluated in an interpreter:

Dynamic Languages are Slow: Example

`x = a + b; y = x + c`

evaluated in an interpreter:

1. What's the type of a? `Integer`
2. What's the type of b? `Integer`

Dynamic Languages are Slow: Example

`x = a + b; y = x + c`

evaluated in an interpreter:

1. What's the type of a? `Integer`
2. What's the type of b? `Integer`
3. unbox a
4. unbox b
5. compute the sum

Dynamic Languages are Slow: Example

`x = a + b; y = x + c`

evaluated in an interpreter:

1. What's the type of a? `Integer`
2. What's the type of b? `Integer`
3. unbox a
4. unbox b
5. compute the sum
6. box the result as an `Integer`
7. store into x

Dynamic Languages are Slow: Example

`x = a + b; y = x + c`

evaluated in an interpreter:

1. What's the type of a? `Integer`
2. What's the type of b? `Integer`
3. unbox a
4. unbox b
5. compute the sum
6. box the result as an `Integer`
7. store into x
8. What's the type of x? `Integer`
9. What's the type of c? `Integer`

Dynamic Languages are Slow: Example

`x = a + b; y = x + c`

evaluated in an interpreter:

1. What's the type of a? `Integer`
2. What's the type of b? `Integer`
3. unbox a
4. unbox b
5. compute the sum
6. box the result as an `Integer`
7. store into x
8. What's the type of x? `Integer`
9. What's the type of c? `Integer`
10. unbox x
11. unbox c
12. compute the sum

Dynamic Languages are Slow: Example

`x = a + b; y = x + c`

evaluated in an interpreter:

1. What's the type of a? `Integer`
2. What's the type of b? `Integer`
3. unbox a
4. unbox b
5. compute the sum
6. box the result as an `Integer`
7. store into x
8. What's the type of x? `Integer`
9. What's the type of c? `Integer`
10. unbox x
11. unbox c
12. compute the sum
13. box the result as an `Integer`
14. store into y

What to do?

- Use a JIT compiler
- **Add an optimization that can deal with heap operations**

What to do?

- Use a JIT compiler
- **Add an optimization that can deal with heap operations**
- optimize short-lived objects
- remove some of the redundant type checks

Overview

1. Experimental Context

2. Proposed Optimization

3. Benchmarks

Our Experimental Context is the PyPy Project

A general environment for implementing dynamic languages

Our Experimental Context is the PyPy Project

A general environment for implementing dynamic languages

Approach

- write an interpreter for the language in RPython
- compilable to an efficient C-based VM

Our Experimental Context is the PyPy Project

A general environment for implementing dynamic languages

Approach

- write an interpreter for the language in RPython
- compilable to an efficient C-based VM
- (RPython is a restricted subset of Python)

PyPy Uses a Novel Tracing JIT

the feature that makes PyPy interesting:

- a meta-JIT, applicable to many languages
- needs a few source-code hints (or user annotations) in the interpreter
- JIT is a tracing JIT compiler

Tracing JITs Compile by Observing an Interpreter

- VM contains both an interpreter and the tracing JIT compiler
- JIT works by observing and logging what the interpreter does
  - for interesting, commonly executed code paths
  - produces a linear list of operations (trace)
- trace is optimized and then turned into machine code

The Advantages of Tracing JITs

- Traces are interesting linear pieces of code
- most of the time correspond to loops
- everything called in the trace is inlined
- can perform good optimizations on the trace
- rarer paths run by the interpreter

Example Trace

Trace of `x = a + b; y = x + c`:

```
guard_class(a, Integer)
guard_class(b, Integer)
i1 = get(a, intval)
i2 = get(b, intval)
i3 = int_add(i1, i2)
x = new(Integer)
set(x, intval, i3)
```

Example Trace

Trace of `x = a + b; y = x + c`:

```
guard_class(a, Integer)
guard_class(b, Integer)
i1 = get(a, intval)
i2 = get(b, intval)
i3 = int_add(i1, i2)
x = new(Integer)
set(x, intval, i3)

guard_class(x, Integer)
guard_class(c, Integer)
i4 = get(x, intval)
i5 = get(c, intval)
i6 = int_add(i4, i5)
y = new(Integer)
set(y, intval, i6)
return(y)
```

Optimized Example Trace

Trace of `x = a + b; y = x + c`:

```
guard_class(a, Integer)
guard_class(b, Integer)
i1 = get(a, intval)
i2 = get(b, intval)
i3 = int_add(i1, i2)
~~x = new(Integer)~~
~~set(x, intval, i3)~~

~~guard_class(x, Integer)~~
guard_class(c, Integer)
~~i4 = get(x, intval)~~
i5 = get(c, intval)
i6 = int_add(i3, i5)
y = new(Integer)
set(y, intval, i6)
return(y)
```

Contribution of our Paper

- a simple, efficient and effective optimization of heap operations in a trace
- using online partial evaluation
- fully implemented and in use in large-scale interpreters

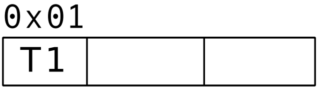
Contribution of our Paper

- a simple, efficient and effective optimization of heap operations in a trace
- using online partial evaluation
- fully implemented and in use in large-scale interpreters

Ingredients

- a slightly simplified model for objects on the heap
- operational semantics of trace operations that manipulate the heap
- optimization rules for those operations, following the operational semantics

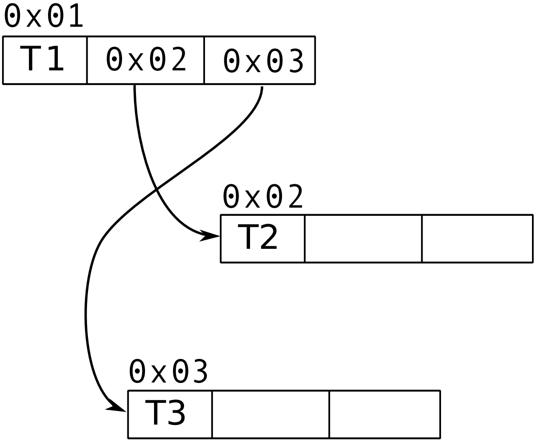
Heap Model



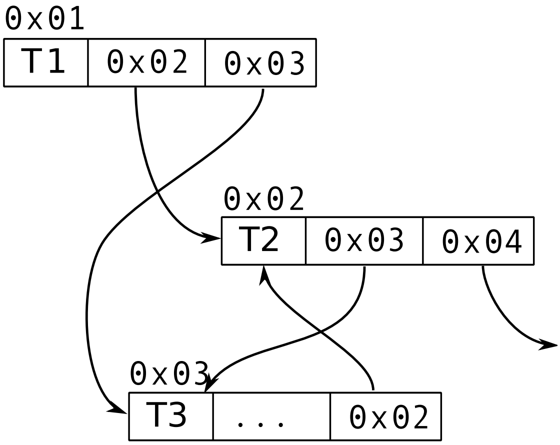
0x01

T1

Heap Model



Heap Model

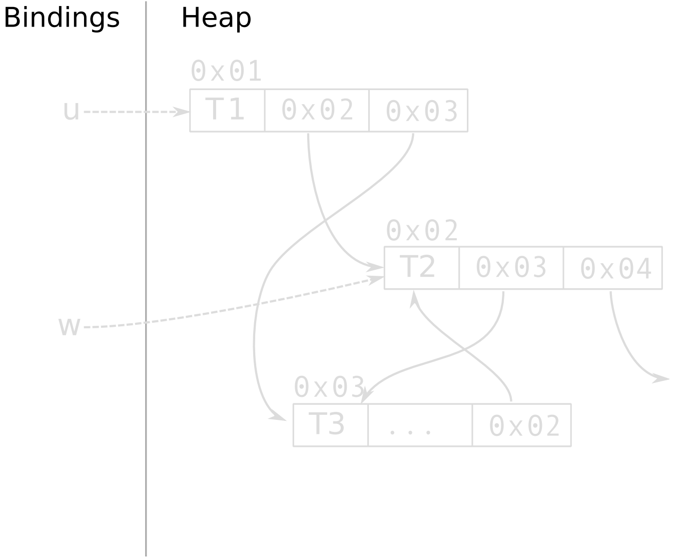


Operations Manipulating Heap Objects

- `v = new(T)` makes a new object
- `u = get(w, F)` reads a field out of an object
- `set(v, F, w)` writes a field of an object
- `guard(v, T)` checks the type of an object

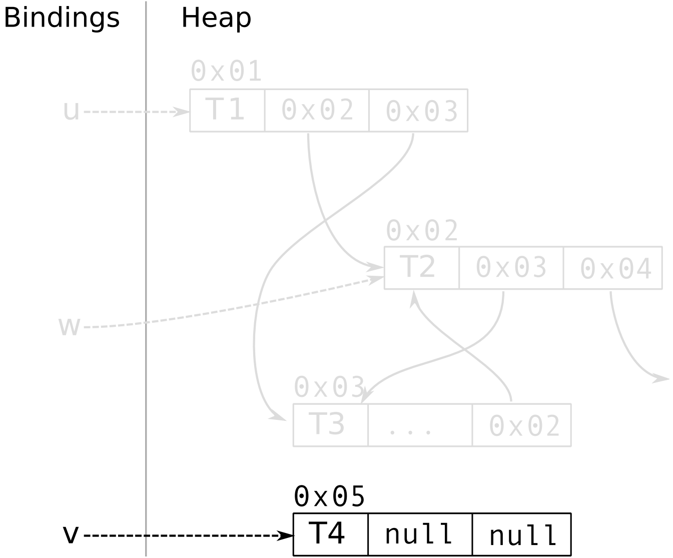
Operations: New

v=new(T4)



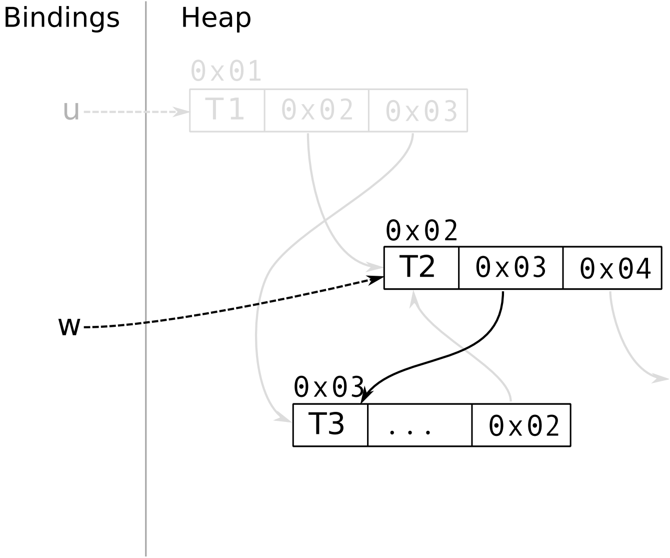
Operations: New

v=new(T4)



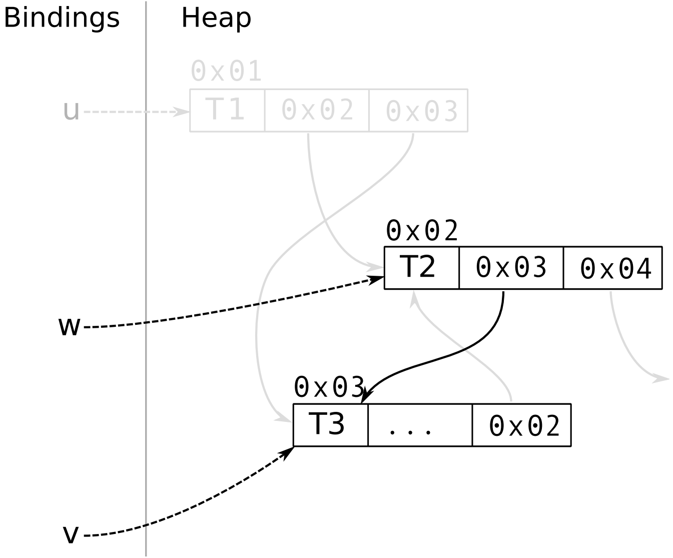
Operations: Get

v=get(w, L)



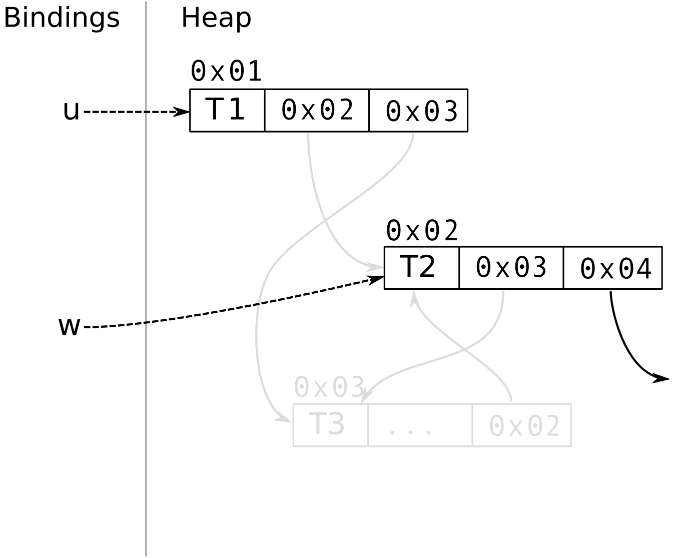
Operations: Get

v=get(w, L)



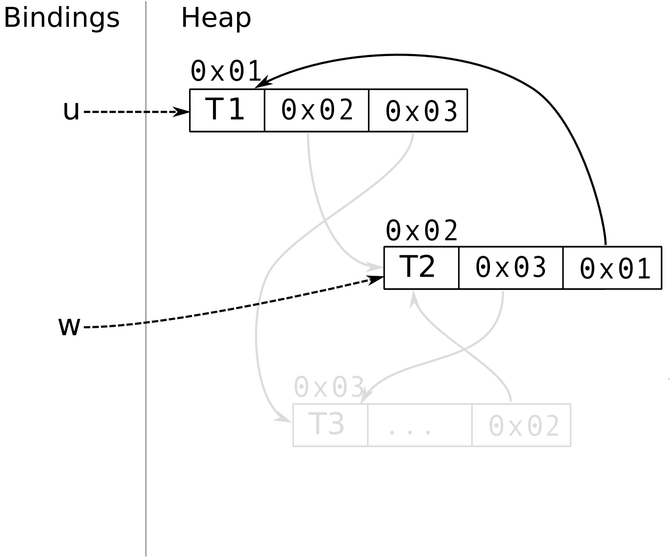
Operations: Set

set(w, R, u)



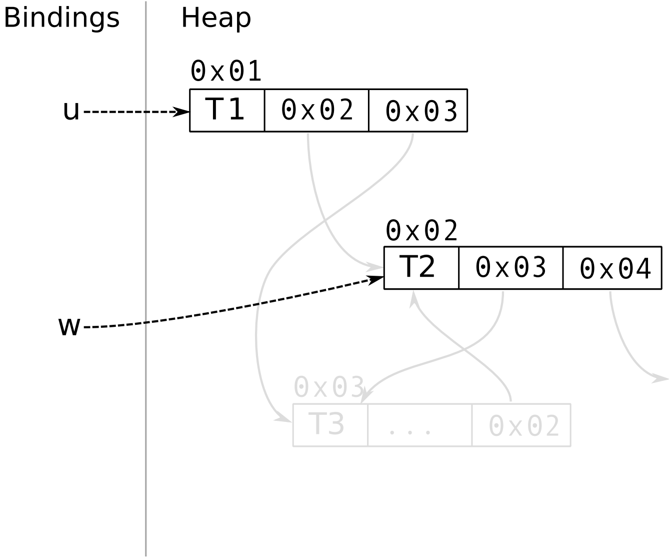
Operations: Set

set(w, R, u)



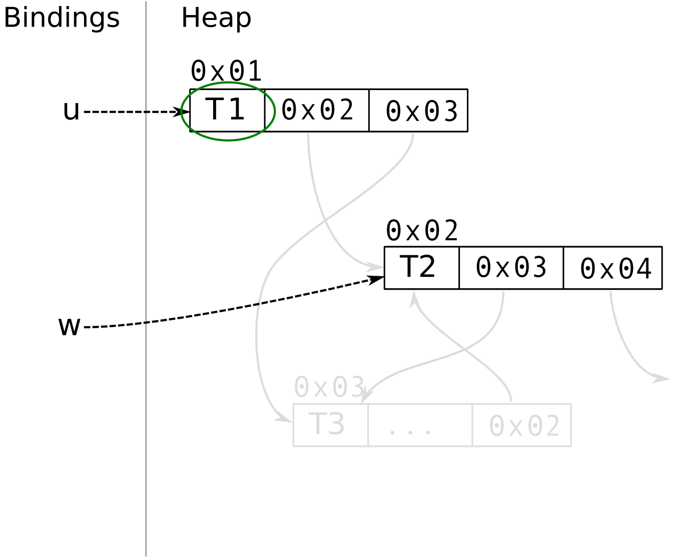
Operations: Guard

guard(u, T1)



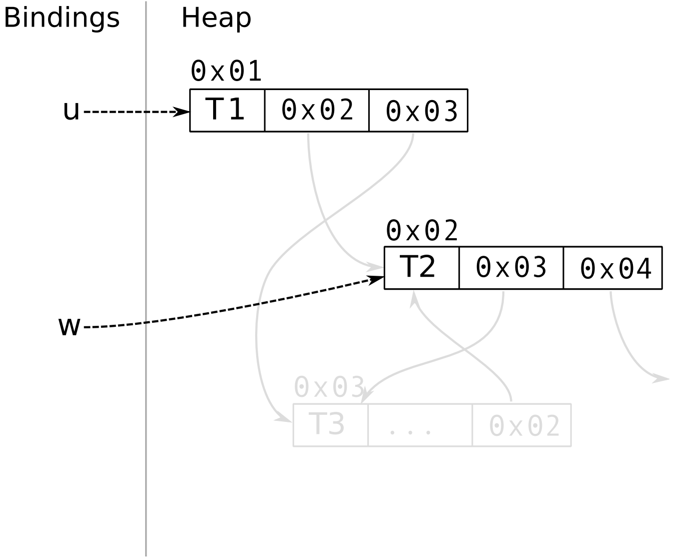
Operations: Guard

guard(u, T1)



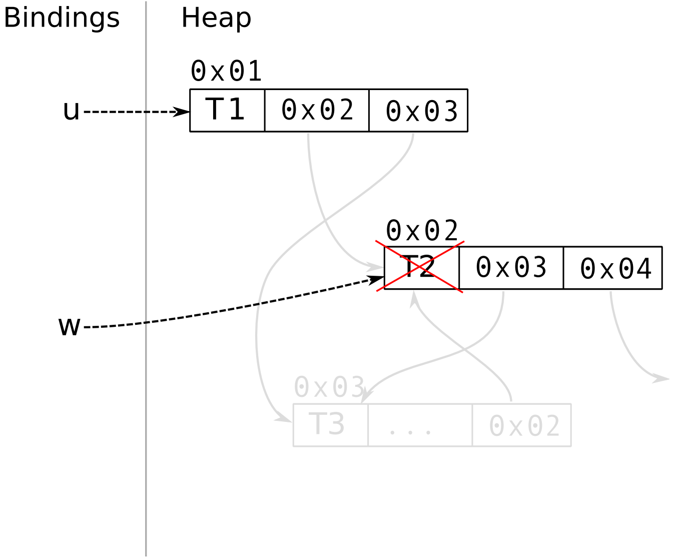
Operations: Guard

guard(w, T1)



Operations: Guard

guard(w, T1)



Optimization by Online Partial Evaluation

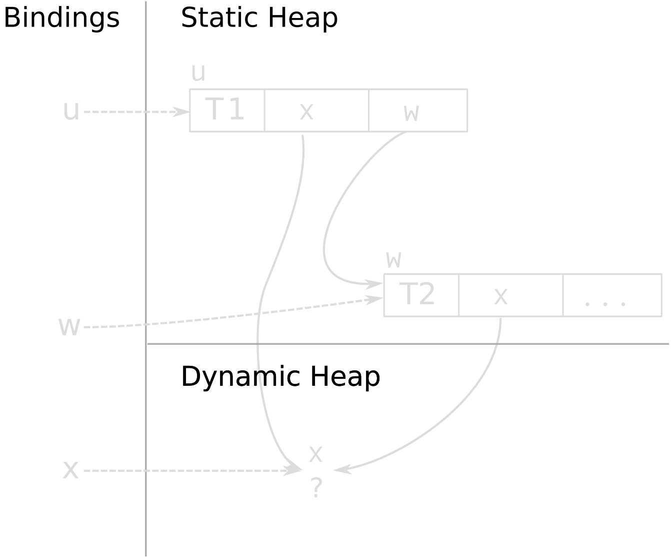
- Trace is optimized using online partial evaluation
- part of the runtime heap is modelled in the static heap
- static heap contains objects that are allocated within the trace
- (as opposed to before the trace is executed)

Optimization by Online Partial Evaluation

- Trace is optimized using online partial evaluation
- part of the runtime heap is modelled in the static heap
- static heap contains objects that are allocated within the trace
- (as opposed to before the trace is executed)
- operations acting on the static heap can be executed
- follows operational semantics
- all others need to be residualized
- all fairly straightforward

Optimizing New

v=new(T4)



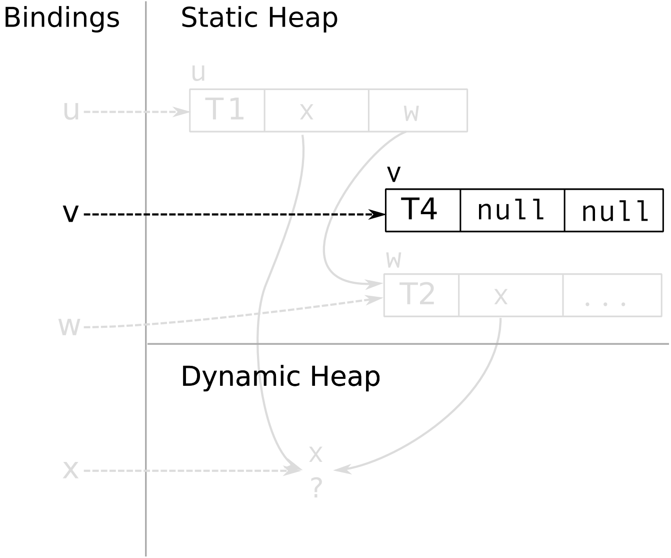
Bindings

Static Heap

Dynamic Heap

Optimizing New

~~v=new(T4)~~



Bindings

Static Heap

u

T1	x	w

v

T4	null	null

w

T2	x	. . .

u

v

w

Dynamic Heap

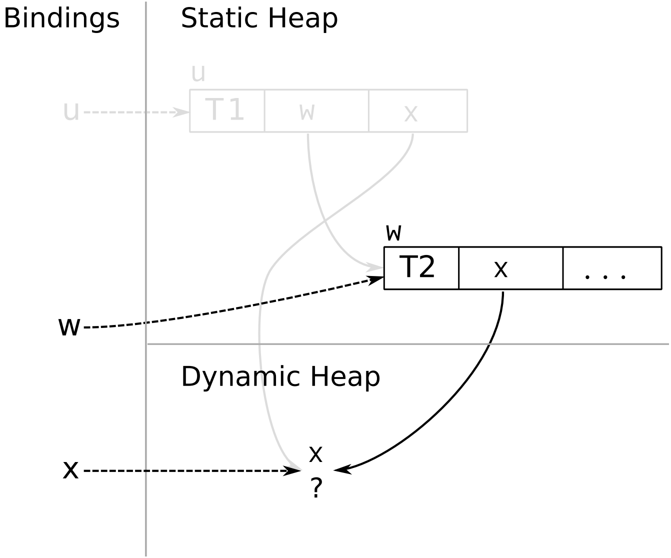
x

x

?

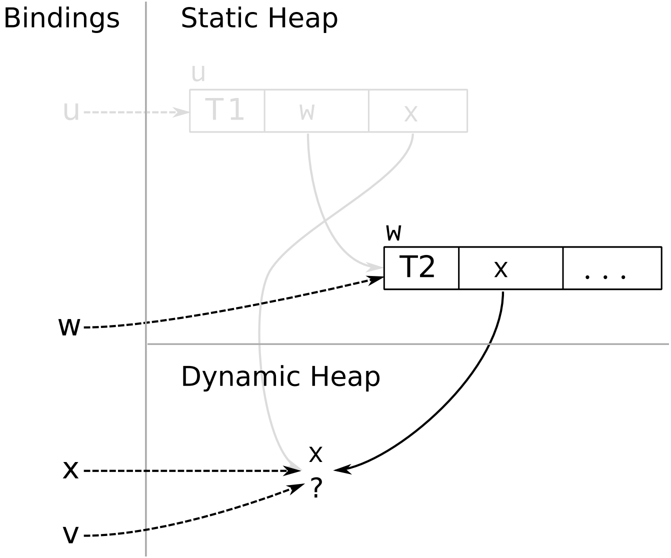
Optimizing Get

v=get(w, L)



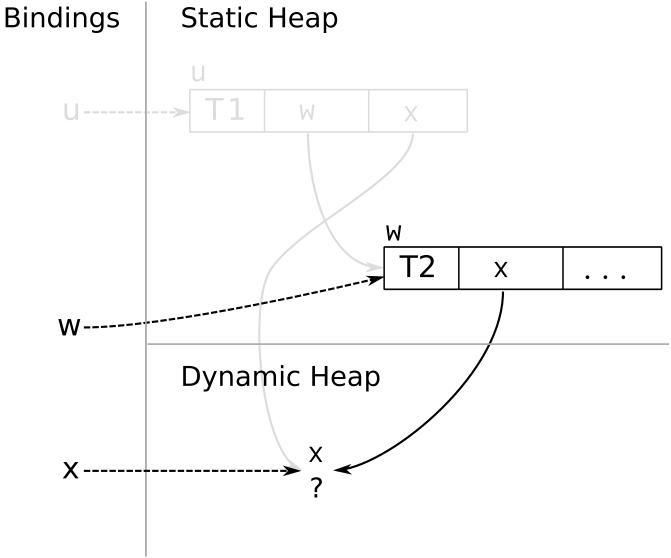
Optimizing Get

~~v=get(w, L)~~

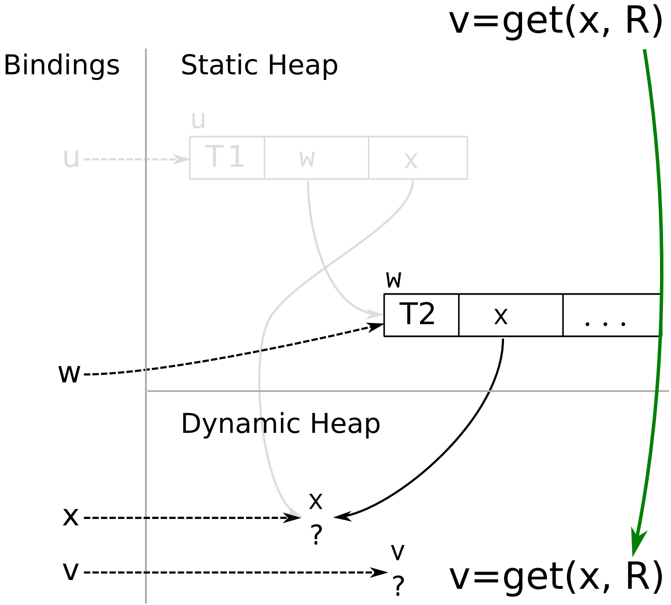


Optimizing Get

v=get(x, R)

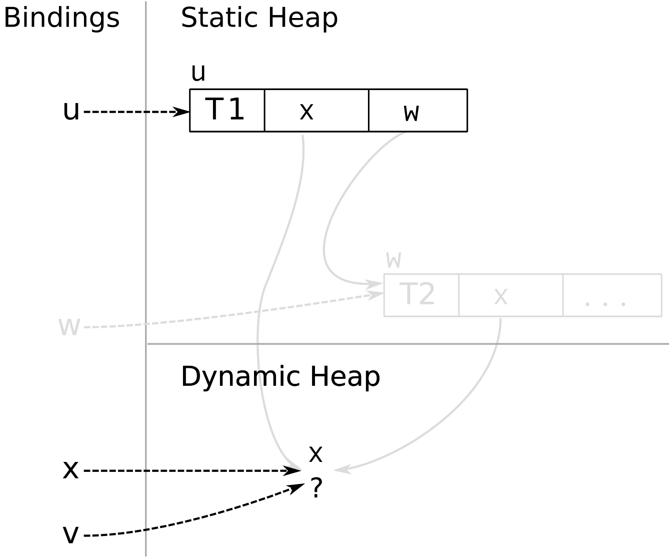


Optimizing Get



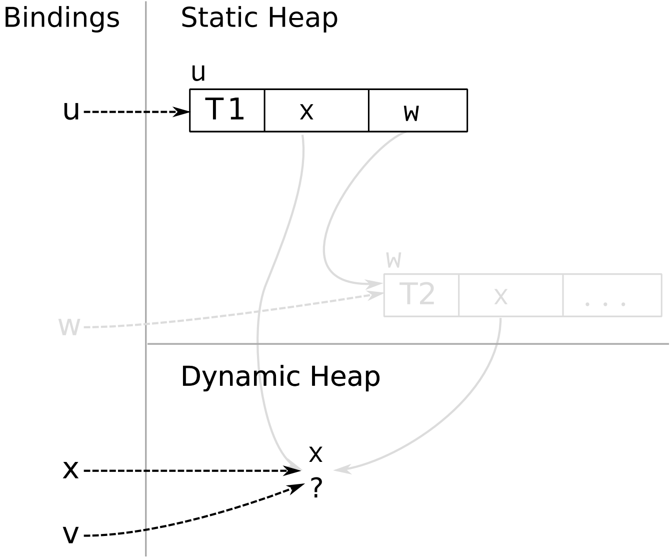
Optimizing Guard

guard(u, T1)



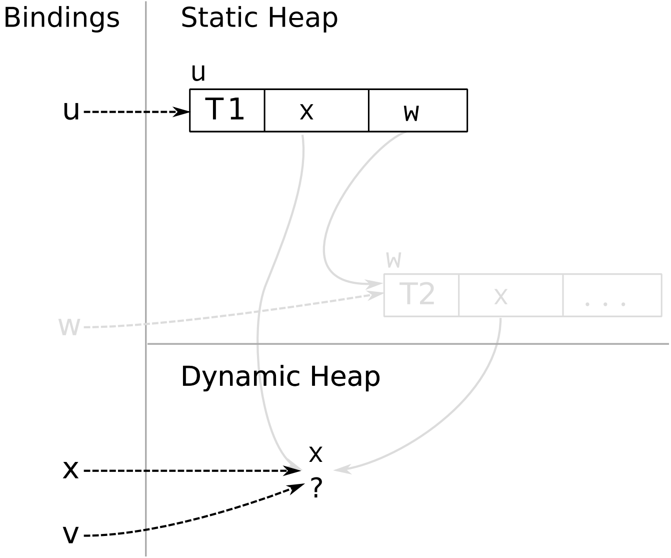
Optimizing Guard

~~guard(u, T1)~~



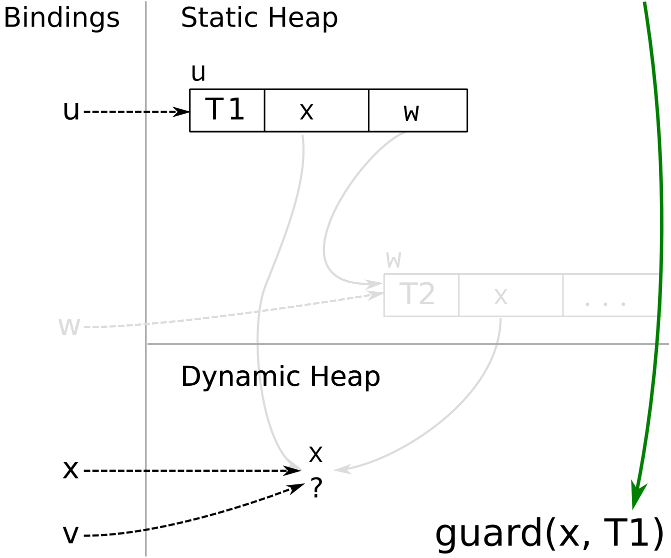
Optimizing Guard

guard(v, T1)



Optimizing Guard

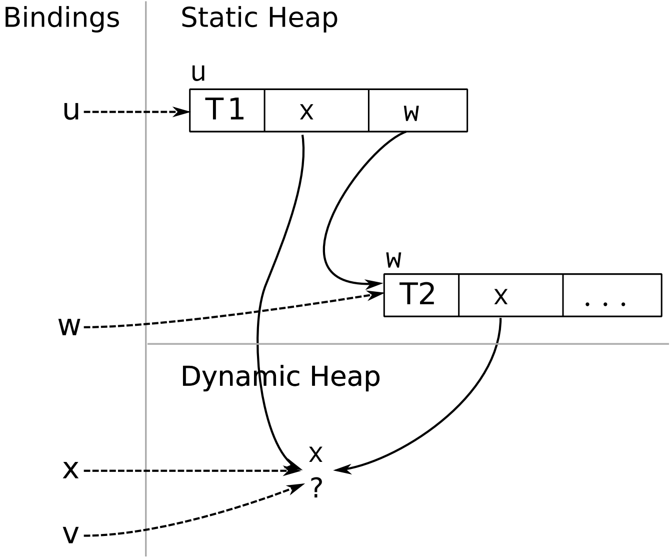
guard(v, T1)



guard(x, T1)

Optimizing Set

set(u,R,v)



Bindings

Static Heap

u

T1	x	w

w

T2	x	. . .

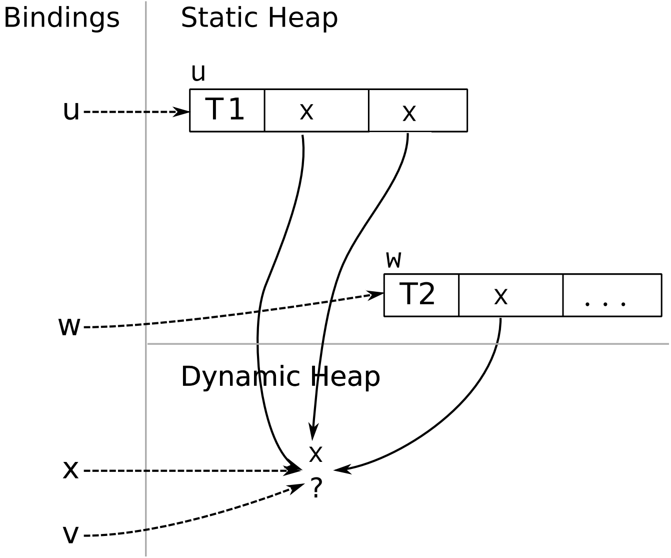
Dynamic Heap

x

?

Optimizing Set

~~set(u,R,v)~~



Lifting

Problem: What happens if we write a static object into a dynamic one?

Lifting

Problem: What happens if we write a static object into a dynamic one?

- lose track of the static object because of possibility of aliasing
- need to lift the static object
- lifting produces operations that recreate the static object

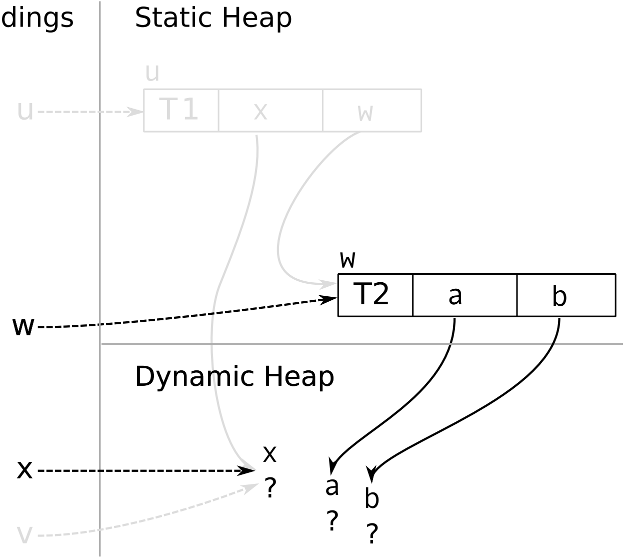
Lifting

Problem: What happens if we write a static object into a dynamic one?

- lose track of the static object because of possibility of aliasing
- need to lift the static object
- lifting produces operations that recreate the static object
- needs to be careful due to recursive structures

Optimizing Set

set(x,R,w)

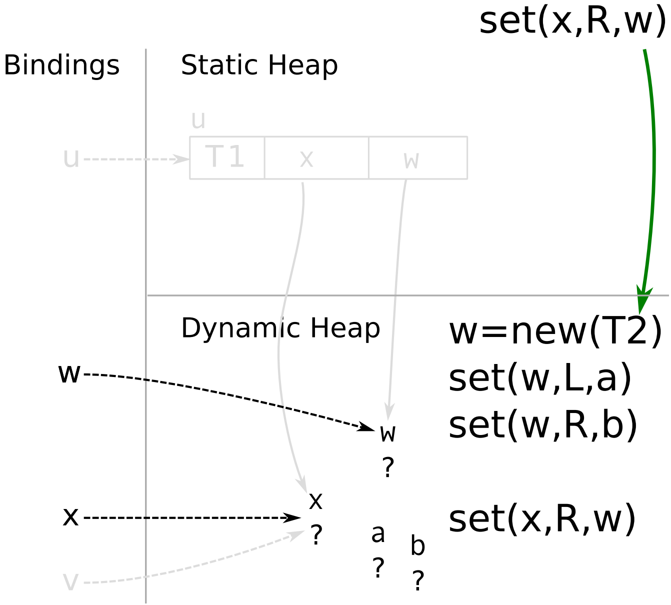


Bindings

Static Heap

Dynamic Heap

Optimizing Set



w=new(T2)
set(w,L,a)
set(w,R,b)

set(x,R,w)

Properties of the Optimization

- output trace is never longer than input trace
- runtime linear in the length of the trace

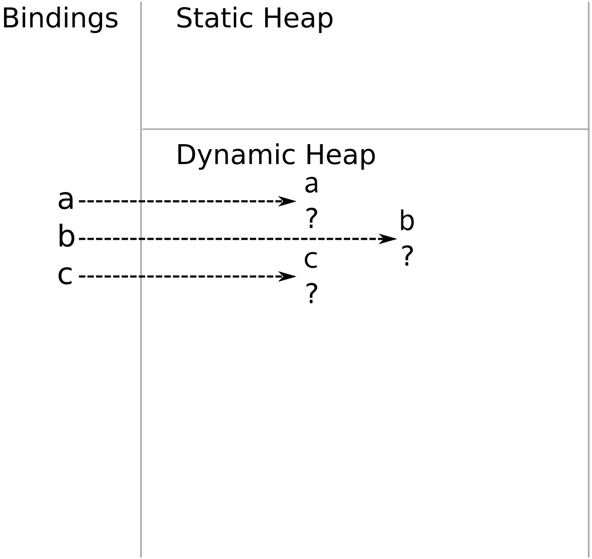
Properties of the Optimization

- output trace is never longer than input trace
- runtime linear in the length of the trace

Implementation

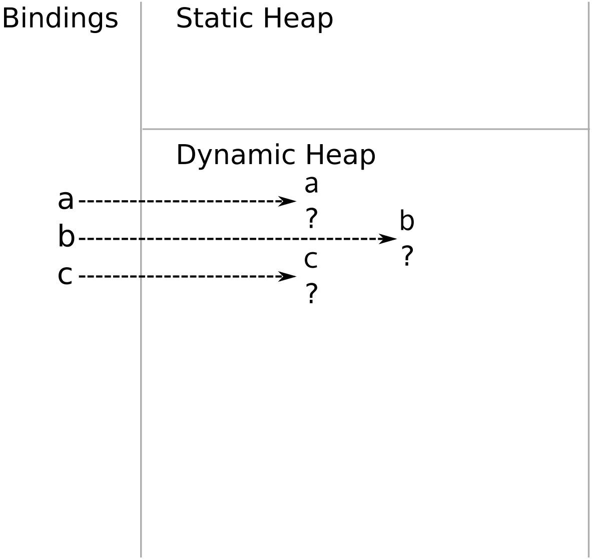
- about 400 lines of code
- some added complexity over presentation
- objects with arbitrary numbers of fields
- array support

Optimizing the Example Trace



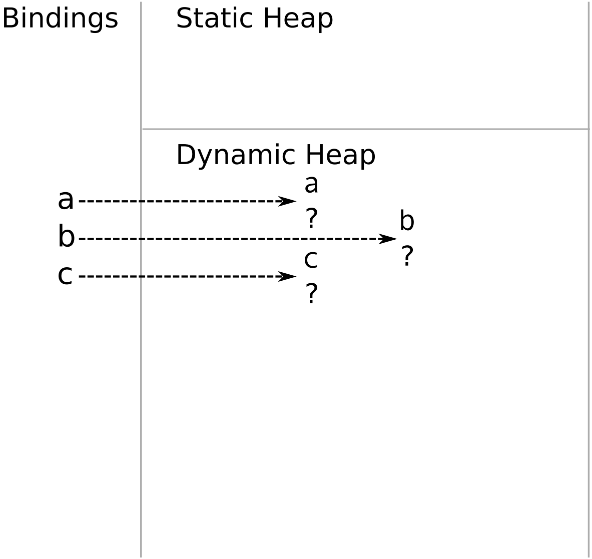
```
guard(a, Int)
guard(b, Int)
i1 = get(a, intval)
i2 = get(b, intval)
i3 = int_add(i1, i2)
x = new(Int)
set(x, intval, i3)
guard(x, Int)
guard(c, Int)
i4 = get(x, intval)
i5 = get(c, intval)
i6 = int_add(i4, i5)
y = new(Int)
set(y, intval, i6)
return(y)
```

Optimizing the Example Trace



```
guard(a, Int)
guard(b, Int)
i1 = get(a, intval)
i2 = get(b, intval)
i3 = int_add(i1, i2)
x = new(Int)
set(x, intval, i3)
guard(x, Int)
guard(c, Int)
i4 = get(x, intval)
i5 = get(c, intval)
i6 = int_add(i4, i5)
y = new(Int)
set(y, intval, i6)
return(y)
```

Optimizing the Example Trace



Bindings

Static Heap

Dynamic Heap

a

b

c

a

?

b

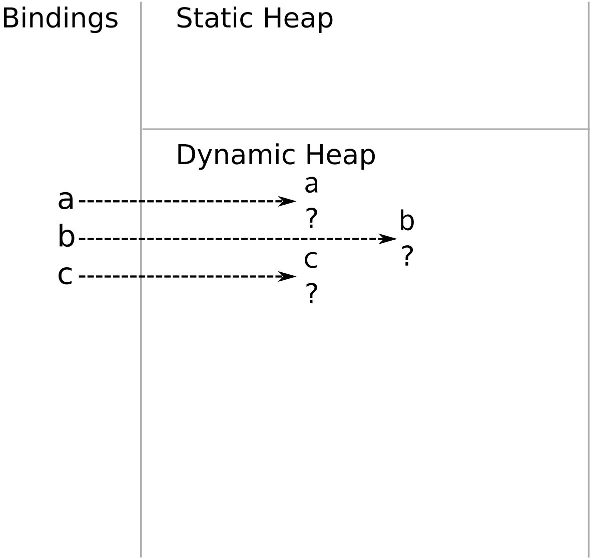
?

c

?

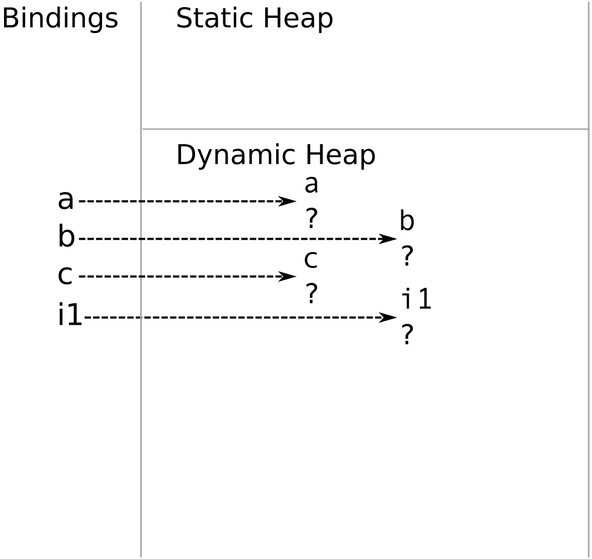
```
guard(a, Int)
guard(b, Int)
i1 = get(a, intval)
i2 = get(b, intval)
i3 = int_add(i1, i2)
x = new(Int)
set(x, intval, i3)
guard(x, Int)
guard(c, Int)
i4 = get(x, intval)
i5 = get(c, intval)
i6 = int_add(i4, i5)
y = new(Int)
set(y, intval, i6)
return(y)
```

Optimizing the Example Trace



```
guard(a, Int)
guard(b, Int)
i1 = get(a, intval)
i2 = get(b, intval)
i3 = int_add(i1, i2)
x = new(Int)
set(x, intval, i3)
guard(x, Int)
guard(c, Int)
i4 = get(x, intval)
i5 = get(c, intval)
i6 = int_add(i4, i5)
y = new(Int)
set(y, intval, i6)
return(y)
```

Optimizing the Example Trace



Bindings

Static Heap

Dynamic Heap

a

b

c

i1

a
?

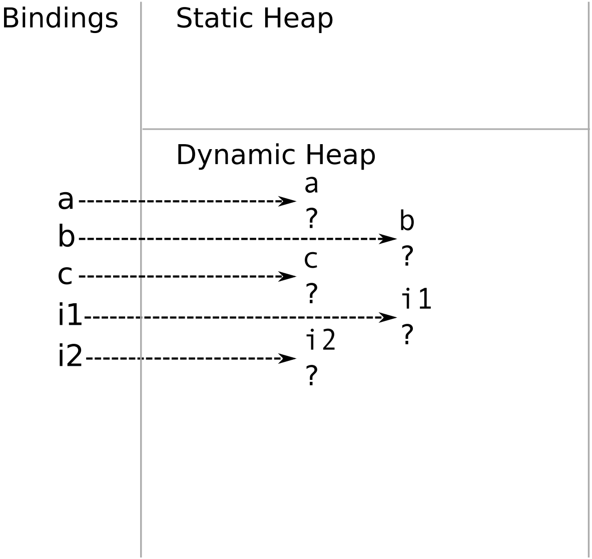
b
?

c
?

i 1
?

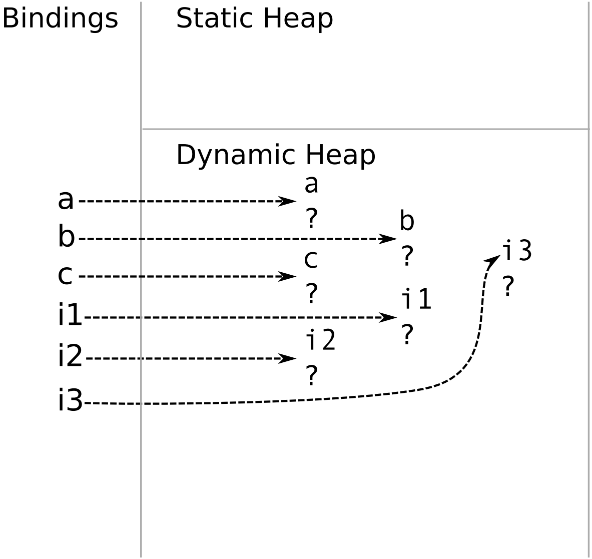
```
guard(a, Int)
guard(b, Int)
i1 = get(a, intval)
i2 = get(b, intval)
i3 = int_add(i1, i2)
x = new(Int)
set(x, intval, i3)
guard(x, Int)
guard(c, Int)
i4 = get(x, intval)
i5 = get(c, intval)
i6 = int_add(i4, i5)
y = new(Int)
set(y, intval, i6)
return(y)
```

Optimizing the Example Trace



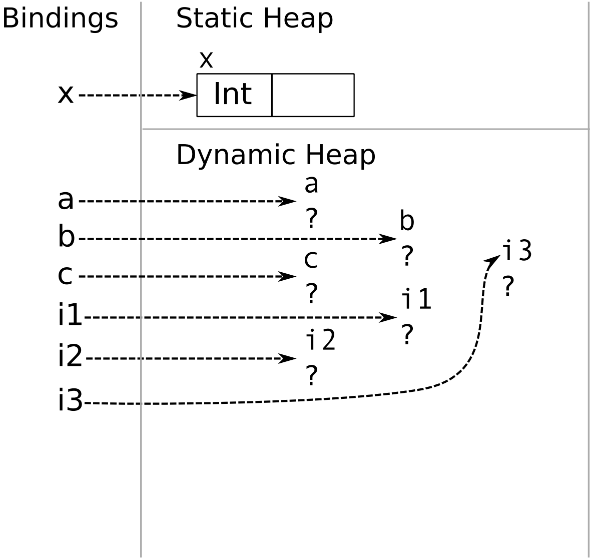
```
guard(a, Int)
guard(b, Int)
i1 = get(a, intval)
i2 = get(b, intval)
i3 = int_add(i1, i2)
x = new(Int)
set(x, intval, i3)
guard(x, Int)
guard(c, Int)
i4 = get(x, intval)
i5 = get(c, intval)
i6 = int_add(i4, i5)
y = new(Int)
set(y, intval, i6)
return(y)
```

Optimizing the Example Trace



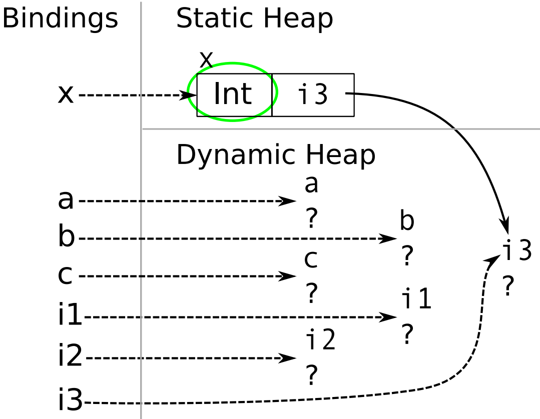
```
guard(a, Int)
guard(b, Int)
i1 = get(a, intval)
i2 = get(b, intval)
i3 = int_add(i1, i2)
x = new(Int)
set(x, intval, i3)
guard(x, Int)
guard(c, Int)
i4 = get(x, intval)
i5 = get(c, intval)
i6 = int_add(i4, i5)
y = new(Int)
set(y, intval, i6)
return(y)
```

Optimizing the Example Trace



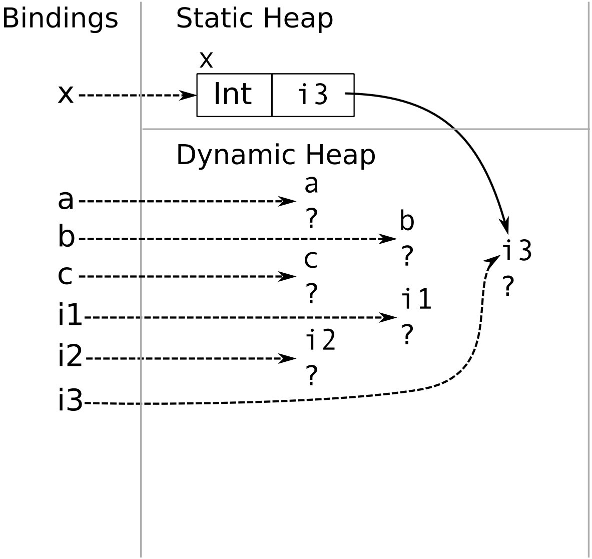
```
guard(a, Int)
guard(b, Int)
i1 = get(a, intval)
i2 = get(b, intval)
i3 = int_add(i1, i2)
~~x = new(Int)~~
set(x, intval, i3)
guard(x, Int)
guard(c, Int)
i4 = get(x, intval)
i5 = get(c, intval)
i6 = int_add(i4, i5)
y = new(Int)
set(y, intval, i6)
return(y)
```

Optimizing the Example Trace



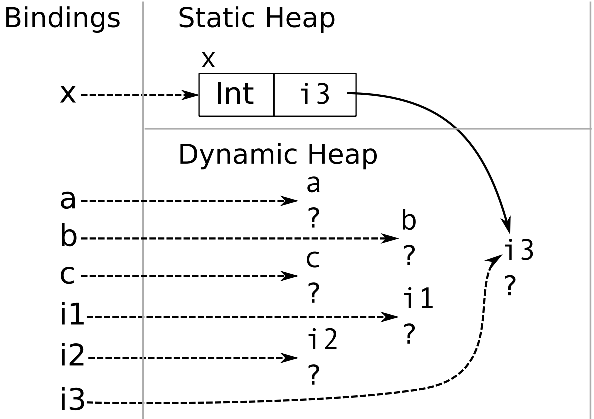
```
guard(a, Int)
guard(b, Int)
i1 = get(a, intval)
i2 = get(b, intval)
i3 = int_add(i1, i2)
~~x = new(Int)~~
~~set(x, intval, i3)~~
guard(x, Int)
guard(c, Int)
i4 = get(x, intval)
i5 = get(c, intval)
i6 = int_add(i4, i5)
y = new(Int)
set(y, intval, i6)
return(y)
```

Optimizing the Example Trace



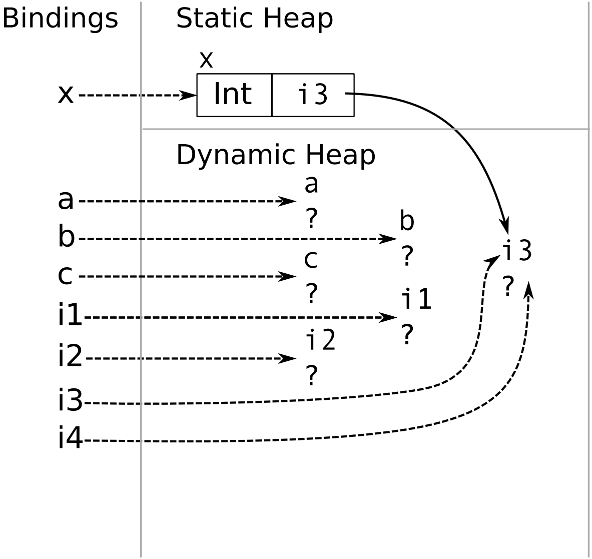
```
guard(a, Int)
guard(b, Int)
i1 = get(a, intval)
i2 = get(b, intval)
i3 = int_add(i1, i2)
~~x = new(Int)~~
~~set(x, intval, i3)~~
~~guard(x, Int)~~
guard(c, Int)
i4 = get(x, intval)
i5 = get(c, intval)
i6 = int_add(i4, i5)
y = new(Int)
set(y, intval, i6)
return(y)
```

Optimizing the Example Trace



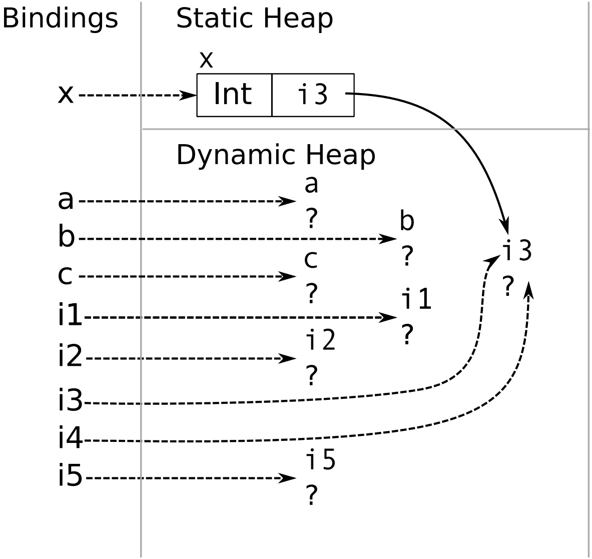
guard(a, Int)
guard(b, Int)
i1 = get(a, intval)
i2 = get(b, intval)
i3 = int_add(i1, i2)
~~x = new(Int)~~
~~set(x, intval, i3)~~
~~guard(x, Int)~~
guard(c, Int)
i4 = get(x, intval)
i5 = get(c, intval)
i6 = int_add(i4, i5)
y = new(Int)
set(y, intval, i6)
return(y)

Optimizing the Example Trace



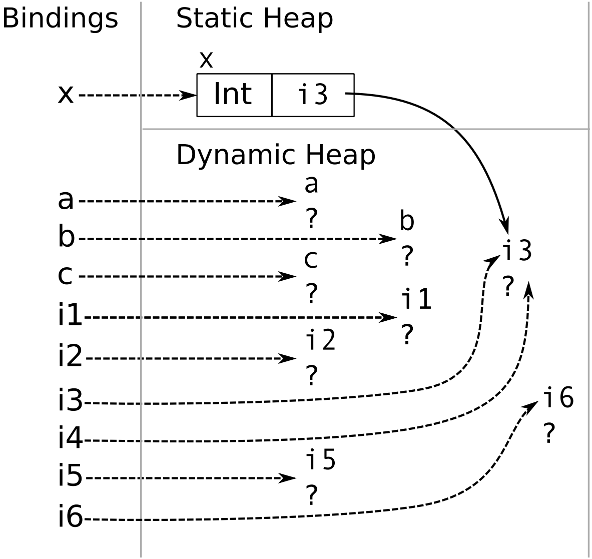
guard(a, Int)
guard(b, Int)
i1 = get(a, intval)
i2 = get(b, intval)
i3 = int_add(i1, i2)
~~x = new(Int)~~
~~set(x, intval, i3)~~
~~guard(x, Int)~~
guard(c, Int)
~~i4 = get(x, intval)~~
i5 = get(c, intval)
i6 = int_add(i4, i5)
y = new(Int)
set(y, intval, i6)
return(y)

Optimizing the Example Trace



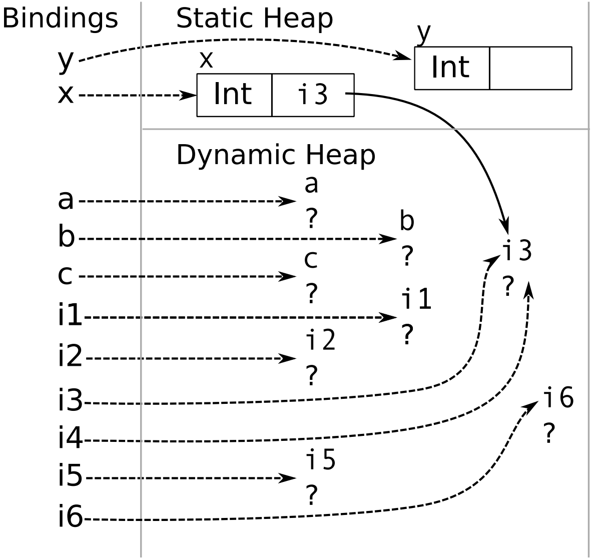
guard(a, Int)
guard(b, Int)
i1 = get(a, intval)
i2 = get(b, intval)
i3 = int_add(i1, i2)
~~x = new(Int)~~
~~set(x, intval, i3)~~
~~guard(x, Int)~~
guard(c, Int)
~~i4 = get(x, intval)~~
i5 = get(c, intval)
i6 = int_add(i4, i5)
y = new(Int)
set(y, intval, i6)
return(y)

Optimizing the Example Trace



guard(a, Int)
guard(b, Int)
i1 = get(a, intval)
i2 = get(b, intval)
i3 = int_add(i1, i2)
~~x = new(Int)~~
~~set(x, intval, i3)~~
~~guard(x, Int)~~
guard(c, Int)
~~i4 = get(x, intval)~~
i5 = get(c, intval)
i6 = int_add(i3, i5)
y = new(Int)
set(y, intval, i6)
return(y)

Optimizing the Example Trace



Bindings

Static Heap

y

x

x

Int

i3

y

Int

Dynamic Heap

a

b

c

i1

i2

i3

i4

i5

i6

a
?

b
?

c
?

i1
?

i2
?

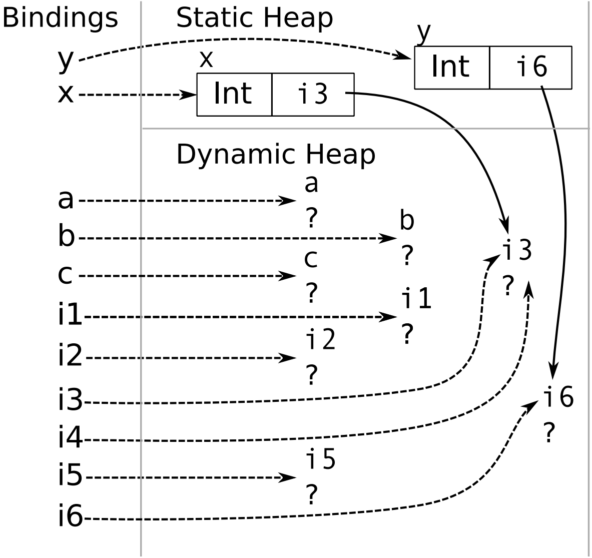
i3
?

i5
?

i6
?

guard(a, Int)
guard(b, Int)
i1 = get(a, intval)
i2 = get(b, intval)
i3 = int_add(i1, i2)
~~x = new(Int)~~
~~set(x, intval, i3)~~
~~guard(x, Int)~~
guard(c, Int)
~~i4 = get(x, intval)~~
i5 = get(c, intval)
i6 = int_add(i3, i5)
~~y = new(Int)~~
set(y, intval, i6)
return(y)

Optimizing the Example Trace



guard(a, Int)

guard(b, Int)

i1 = get(a, intval)

i2 = get(b, intval)

i3 = int_add(i1, i2)

~~x = new(Int)~~

~~set(x, intval, i3)~~

~~guard(x, Int)~~

guard(c, Int)

~~i4 = get(x, intval)~~

i5 = get(c, intval)

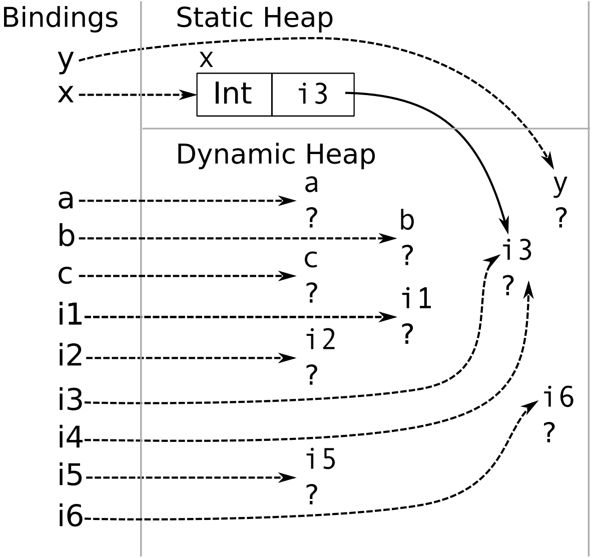
i6 = int_add(i3, i5)

~~y = new(Int)~~

~~set(y, intval, i6)~~

return(y)

Optimizing the Example Trace



```
guard(a, Int)
guard(b, Int)
i1 = get(a, intval)
i2 = get(b, intval)
i3 = int_add(i1, i2)
~~x = new(Int)~~
~~set(x, intval, i3)~~
~~guard(x, Int)~~
guard(c, Int)
~~i4 = get(x, intval)~~
i5 = get(c, intval)
i6 = int_add(i3, i5)
y = new(Int)
set(y, intval, i6)
return(y)
```

Benchmark Results

- to evaluate the optimization we used PyPy's Python interpreter with real-world programs
    - interpreter is about 30'000 lines of code

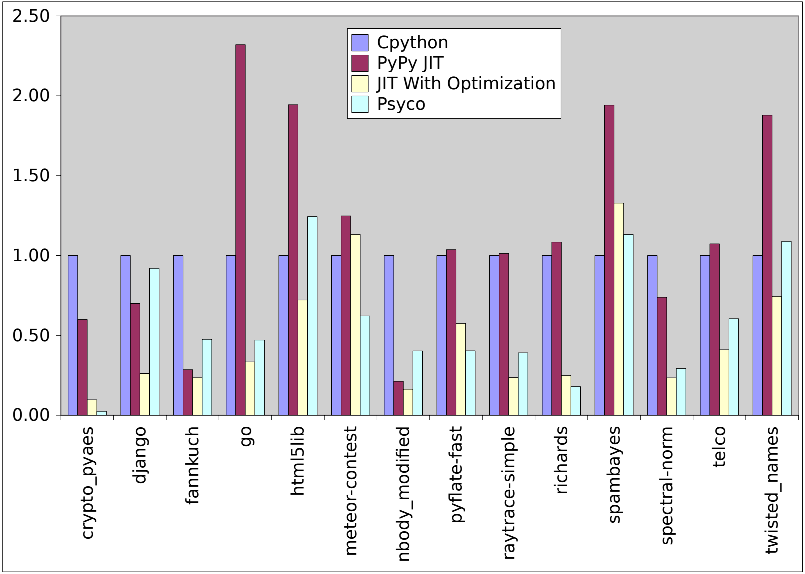
Benchmark Results

- to evaluate the optimization we used PyPy's Python interpreter with real-world programs
  - interpreter is about 30'000 lines of code
- optimization can remove
  - 70% of all `new` operations
  - 14% of all `get`/`set` operations
  - 93% of all `guard` operations

Benchmark Results

- to evaluate the optimization we used PyPy's Python interpreter with real-world programs
  - interpreter is about 30'000 lines of code
- optimization can remove
  - 70% of all `new` operations
  - 14% of all `get/set` operations
  - 93% of all `guard` operations
- Timings improve by a factor between 1.1 and 6.95
- outperforming standard Python on all benchmarks but two

Benchmark


2.50
2.00
1.50
1.00
0.50
0.00
Cpython
PyPy JIT
JIT With Optimization
Psyco
crypto_pyaes
django
fannkuch
go
html5lib
meteor-contest
nbody_modified
pyflate-fast
raytrace-simple
richards
spambayes
spectral-norm
telco
twisted_names

Conclusion

- We propose a very simple partial-evaluation-based optimization for tracing JITs of dynamic languages that:
  - can remove a lot of allocations and type checks in practical programs.
  - is efficient and effective.
  - has no control issues because all control decisions are made by the tracing JIT.

Conclusion

- We propose a very simple partial-evaluation-based optimization for tracing JITs of dynamic languages that:
  - can remove a lot of allocations and type checks in practical programs.
  - is efficient and effective.
  - has no control issues because all control decisions are made by the tracing JIT.
- We claim that this is a general strategy to get rid of control problems by simply observing the runtime behaviour of the program.

Backup Slides

What About Correctness?

- We haven't proven correctness yet
- should not be too hard
- lifting needs to be carefully handled

Comparison to Escape Analysis

- Effect very similar to escape analysis
- Escape analysis needs a complex upfront analysis
- our optimization automatically has a lot of context, due to the inlining tracing does
- our optimization can optimize operations on objects even if they escape later

Comparison to ”Dynamic Typing”/Boxing Analysis

- those optimizations work ahead of time
- don’t work for many dynamic languages, where the source simply does not contain enough information

Python Example:

```
def sum(container, initial):
    result = initial
    for element in container:
        result = result + element
    return result
```