

BEYOND OVERDUBBING

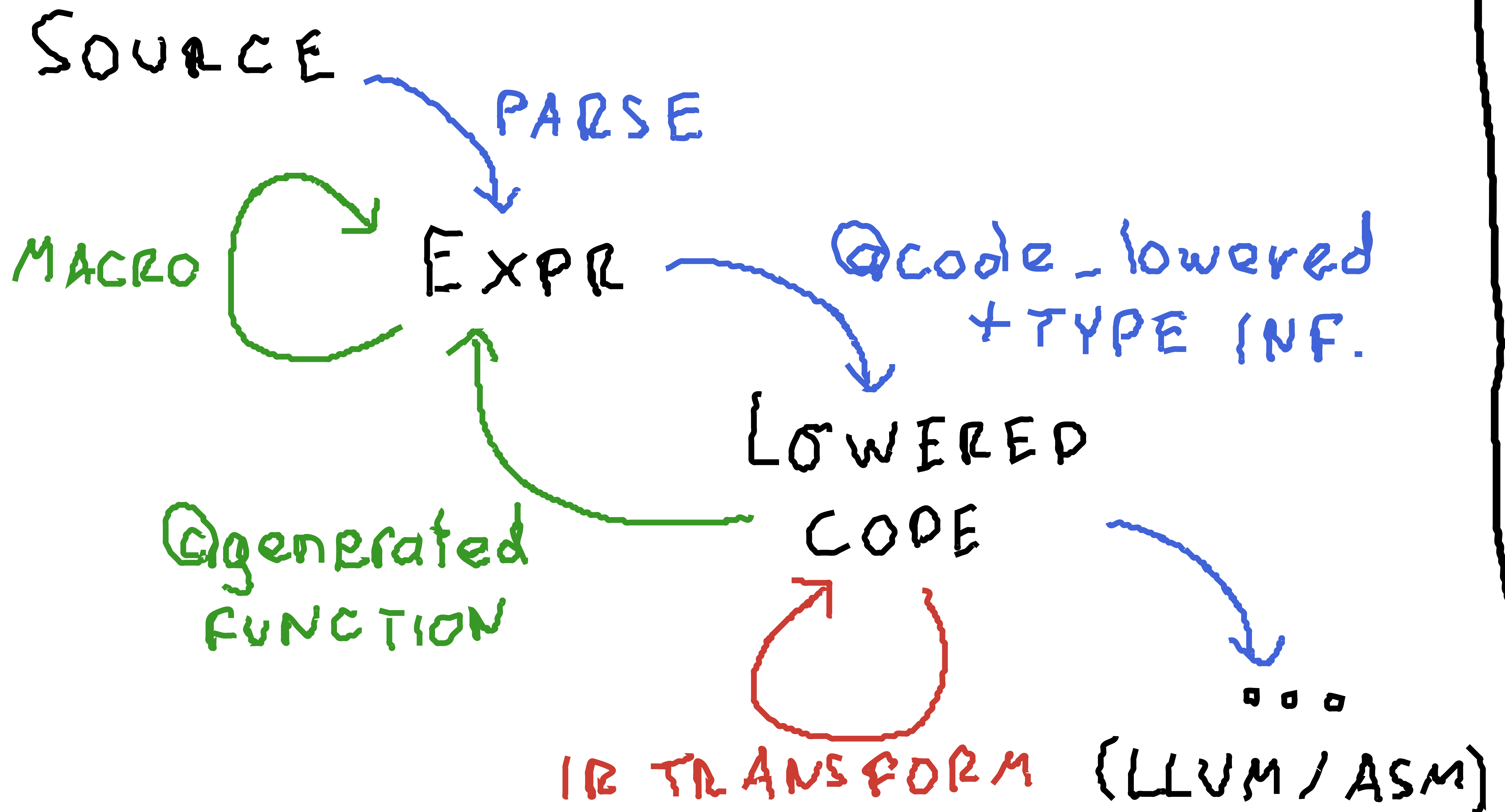
BUILDING A GENERIC IR TRACKER

PHILIPP GABLER
(@phipsgabler)

OVERVIEW

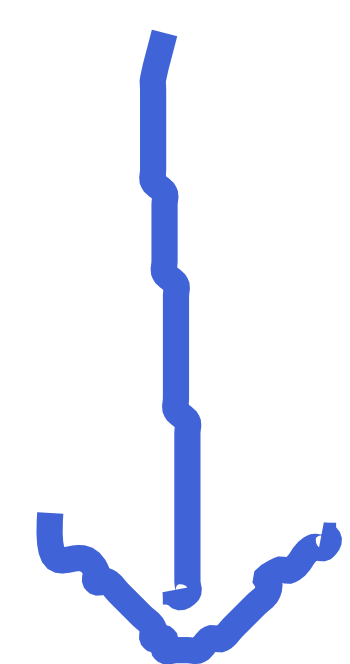
- FROM ZERO TO IRTRACKER
- IRTOOLS & CASSETTE FROM
A USER PERSPECTIVE

COMPILER PASSES 101



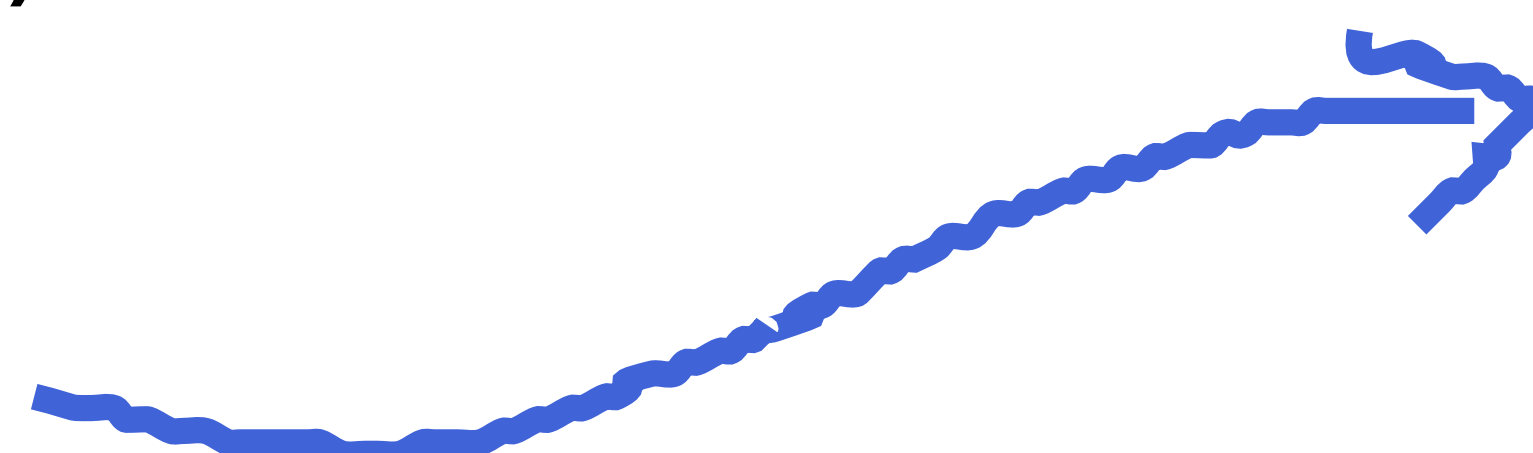
SLOPPY 'IF's

```
julia> f(x) = x ? 1 : 0  
f (generic function with 1 method)
```



ORIGINAL
IR

```
julia> @code_ir f(0.0)  
1: (%1, %2)  
   br 2 unless %2  
   return 1  
2:  
   return 0
```



TRANSFORMED
IR

```
julia> @code_ir sloppyifs(f, 0.0)  
1: (%1, %2)  
   %3 = Base.getfield(%2, 1)  
   %4 = Base.getfield(%2, 2)  
   %5 = Base.convert(Bool, %4)  
   br 2 unless %5  
   return 1  
2:  
   return 0
```

NEW
STUFF

CASSETTE OVERDUBBING

```
julia> @code_lowered 1/2
```

```
CodeInfo(  
59 1 -
```

```
    %1 = (Base.float)(x)  
    %2 = (Base.float)(y)  
    %3 = %1 / %2  
    return %3  
)
```

```
julia> @code_lowered overdub(Ctx(), /, 1, 2)
```

```
CodeInfo(  
59 1 -
```

```
    #self# = getfield(##args, 1)
```

```
    x = getfield(##args, 2)
```

```
    y = getfield(##args, 3)
```

```
    prehook(##ctx, float, x)
```

```
    %5 = overdub(##ctx, float, x)
```

```
    posthook(##ctx, %5, float, x)
```

```
    %7 = %5
```

```
    prehook(##ctx, float, y)
```

```
    %9 = overdub(##ctx, float, y)
```

```
    posthook(##ctx, %9, float, y)
```

```
    %11 = %9
```

```
    prehook(##ctx, /, %7, %11)
```

```
    %13 = overdub(##ctx, /, %7, %11)
```

```
    posthook(##ctx, %13, /, %7, %11)
```

```
    %15 = %13
```

```
    return %15  
)
```

IR TOOLS DYNAMO

```
@dynamo function sloppyifs(f, args...)
  ir = IR(f, args...)
  ir === nothing && return
```

```
  for block in blocks(ir)
    bblock = BasicBlock(block)
    for b in eachindex(branches(bblock))
      branch = bblock.branches[b]
      if isconditional(branch)
        converted = push!(block, xcall(:convert, Bool, branch.condition))
        bblock.branches[b] = Branch(branch; condition=converted)
      end
    end
  end
end
```

```
  return ir
```

```
end
```

julia> @code_ir f(0.0)

1: (%1, %2)

✓ br 2 unless %2

* return 1

2:

return 0

Convert(Bool, %2)

IR TRACKER'S IMPLEMENTATION

```
julia> @code_tracked f(true)
1: (%3, %4, %1, %2)
    %5 = saveir!(%4, ...)
    %6 = TapeConstant(%1)
    %7 = trackedargument(%4, %6, nothing, 1, $($1:%1))
    %8 = record!(%4, %7)
    %9 = TapeConstant(%2)
    %10 = trackedargument(%4, %9, nothing, 1, $($1:%2))
    %11 = record!(%4, %10)
    %12 = tuple()
    %13 = trackedvariable(%4, $($2), %2)
    %14 = trackedjump(%4, 2, %12, %13, $($1:&1))
    %15 = trackedreturn(%4, $(<1>), $($1:&2))
    br 2 (%14) unless %2
    br 3 (1, %15)
2: (%16)
    %17 = record!(%4, %16)
    %18 = trackedreturn(%4, $(<0>), $($2:&1))
    br 3 (0, %18)
3: (%19, %20)
    %21 = record!(%4, %20)
    return %19
```

The diagram illustrates the control flow of the IR code. Green arrows show the initial flow from block 1 to block 2. A blue arrow shows a jump from block 1 to block 2. Red arrows show jumps from block 1 to block 3 and from block 2 to block 3.

IR TRACKER'S RESULT

```
julia> @code ir f(true)
```

```
1: (%1, %2)
   br 2 unless %2
   return 1
```

```
2:     return 0
```

```
julia> @code ir (x -> f(true) + x)(1)
```

```
1: (%1, %2)
   %3 = Main.f(true)
   %4 = %3 + %2
   return %4
```

```
julia> track(x -> f(true) + x, 1)
```

$$\langle \text{var}\text{"\#7\#8"}() \rangle (\langle 1 \rangle, () \dots) \rightarrow 2 :: \text{Int64}$$

```
@1: [Arg:§1:%1] var"#7#8"():var"#7#8"
```

```
@2: [Arg:$1:%2] 1::Int64
```

```
@3: [§1:%3] <f>(<true>, ()...) → 1::Int64
```

```
@1: [Arg:$1:%1] @3#1 → f::typeof(f)
```

```
@2: [Arg:$1:%2] @3#2 → true::Bool
```

```
@3: [§1:&2] return <1>
```

```
@4:  [§1:%4]  ⟨+⟩ (@3, @2, ()...) → 2::Int64
```

```
@1: [Arg:$1:%1] @4#1 → +::typeof(+)
```

```
@2: [Arg:$1:%2] @4#2 → 1::Int64
```

```
@3: [Arg:§1:%3] @4#3 → 1::Int64
```

```
@4:  [%1:%4]  (add int) (@2, @3) → 2::Int64
```

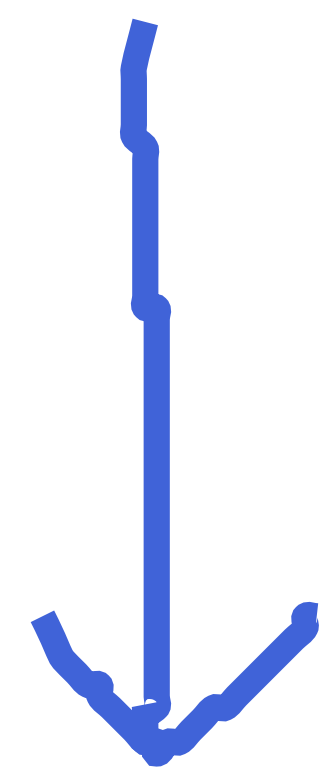
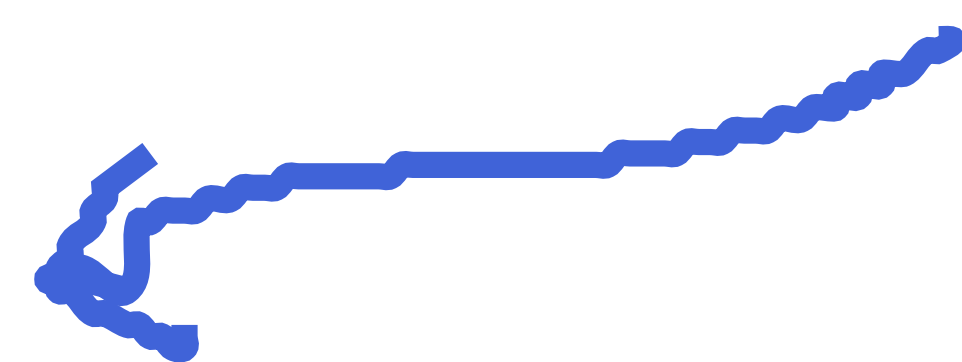
```
@5:  [$1:&1] return @4 → 2::Int64
```

```
@5:  [$1:&1] return @4 → 2::Int64
```


DEPENDENCY EXTRACTION

```
julia> @model function test0(x)
    λ ~ Gamma(2.0, inv(3.0))
    m ~ Normal(0, sqrt(1 / λ))
    x ~ Normal(m, sqrt(1 / λ))
end
```

TURING MODEL

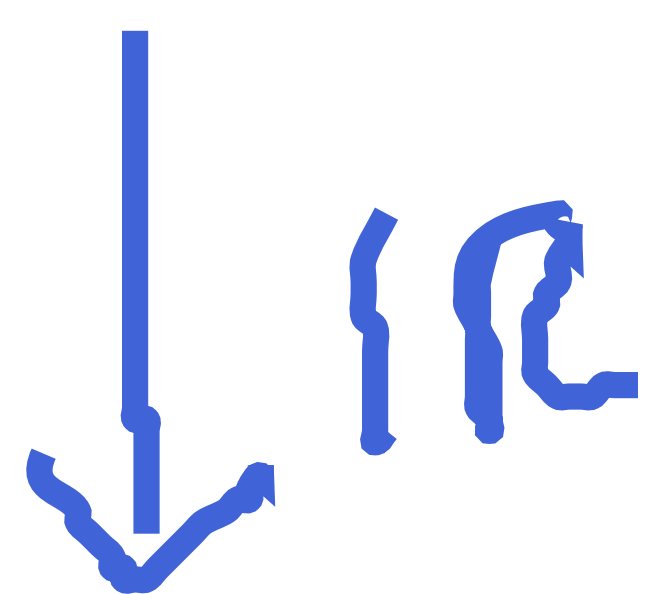


TRACK + EXTRACT

```
julia> trackdependencies(test0(1.4))
⟨2⟩ = 1.4
⟨4:λ⟩ ~ Gamma(2.0, 0.3333333333333333) → 1.0351608689025245
⟨5⟩ = /(1, ⟨4:λ⟩) → 0.9660334253749353
⟨6⟩ = sqrt(⟨5⟩) → 0.982869994137035
⟨8:m⟩ ~ Normal(0, ⟨6⟩) → -2.0155543806491205
⟨9⟩ = /(1, ⟨4:λ⟩) → 0.9660334253749353
⟨10⟩ = sqrt(⟨9⟩) → 0.982869994137035
⟨12:x⟩ ~ Normal(⟨8:m⟩, ⟨10⟩) ← ⟨2⟩
```

WISHFUL THINKING

```
geom(n, beta) =  
  rand() < beta ? n : geom(n + 1, beta)
```



```
1: (%1, %2, %3)  
  %4 = Main.rand()  
  %5 = %4 < %3  
  br 2 (%5) unless %5  
  return %2  
2:  
  %6 = %2 + 1  
  %7 = Main.geom(%6, %3)  
  return %7
```

REFINED
IR
OVERDUBS

```
1: (%1, %4, %2, %3)  
  %5 = quoted(%4, Arg, :(%1), %1)  
  %6 = quoted(%4, Arg, :(%2), %2)  
  %7 = quoted(%4, Arg, :(%3), %3)  
  %8 = quoted(%4, Const, Main.rand)  
  %9 = overdub(%4, Call, :(%4), %8)  
  %10 = quoted(%4, Const, <)  
  %11 = overdub(%4, Call, :(%5), %10, %9, %7)  
  %12 = quoted(%4, CondBranch, 2, %11)  
  %13 = quoted(%4, Return, %6)  
  br 2 unless %11  
  br 3 (%13)  
2:  
  %14 = quoted(%4, Const, +)  
  %15 = quoted(%4, Const, 1)  
  %16 = overdub(%4, Call, :(%6), %14, %6, %5)  
  %17 = quoted(%4, Const, Main.geom)  
  %18 = overdub(%4, Call, :(%7), %17, %16, %7)  
  %19 = quoted(%4, Return, %18)  
  br 3 (%19)  
3 (%20):  
  %21 = overdub(%4, Return, %20)  
  return %20
```

LAST SLIDE

- SPECIAL THANKS TO:
MIKE INNES, THE TURING TEAM
- FIND SLIDES & TEXT AT:
github.com/phipsyabler/beyond_overdubbing