# Decompilers

# Resources

- Cifuentes Thesis: Reverse Compilation Techniques
- I will be posting other resources on the website

# What is a Decompiler?

```
; __unwind {
push    r13
xor     ecx, ecx        ; n
xor     esi, esi        ; buf
mov     edx, 2          ; modes
push    r12
lea     r13, fmt        ; "I'm on %u!"
push    rbp
push    rbx
sub     rsp, 28h
mov     rdi, cs:stdin   ; stream
mov     rax, fs:28h
mov     [rsp+48h+var_30], rax
xor     eax, eax
lea     r12, [rsp+48h+n]
lea     rbp, [rsp+48h+ptr]
call    _setvbuf
mov     rdi, cs:stdout  ; stream
xor     ecx, ecx        ; n
xor     esi, esi        ; buf
mov     edx, 2          ; modes
call    _setvbuf
mov     rdi, cs:stderr  ; stream
xor     ecx, ecx        ; n
xor     esi, esi        ; buf
mov     edx, 2          ; modes
call    _setvbuf
xor     esi, esi
lea     rdi, aWhatSHappening ; "What's happening out there?"
call    sub_2560
lea     rdi, aShallIDescribe ; "Shall I describe it to you? Or would yo"...
call    sub_25E0
xor     esi, esi
lea     rdi, aHahaha    ; "Hahaha!"
call    sub_2560
lea     rdi, aTheUrukLeaderC ; "The Uruk Leader cries out once, encoura"...
call    sub_27C0
lea     rdi, aDarthoHold ; "Dartho! (Hold!)"
```

```
__int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  __ssize_t v3; // rax
  unsigned int v4; // ebx
  char *ptr; // [rsp+8h] [rbp-40h]
  size_t n; // [rsp+10h] [rbp-38h]
  unsigned __int64 v8; // [rsp+18h] [rbp-30h]

  v8 = __readfsqword(0x28u);
  setvbuf(stdin, 0LL, 2, 0LL);
  setvbuf(stdout, 0LL, 2, 0LL);
  setvbuf(stderr, 0LL, 2, 0LL);
  sub_2560("What's happening out there?", 0LL);
  sub_25E0("Shall I describe it to you? Or would you like me to find you a box?");
  sub_2560("Hahaha!", 0LL);
  sub_27C0(
    "The Uruk Leader cries out once, encouraging the Uruk-hai to start roaring and thumping their spears furiously. The w"
    "oman and children in the caves huddle together in feat. Suddently, Aldor, the old man next to Haleth, loses his grip"
    " and releases his arrow prematurely, shooting an Uruk-hai in the neck.");
  sub_2640("Dartho! (Hold!)");
  sub_27C0(
    "The Uruk-hai army stop their roaring and thumping. With a hollow groan, the Uruk that was shot collapsed to the grou"
    "nd. The other Uruk-hai bare their teeth and roar with angr. With a cry, the Uruk-hai leader thrusts his weapon in th"
    "e air and the Uruk-hai army starts charging.");
  sub_2640("So it begins.");
```

# Phases of Decompilation

1. Loading
2. Disassembly
3. Lifting
4. Dataflow Analysis
5. Type Inference
6. CodeGen
7. Name Recovery???

```
def _decompile(self):

    if self.func.is_simprocedure:
        return

    # convert function blocks to AIL blocks
    clinic = self.project.analyses.Clinic(self.func,
                                          kb=self.kb,
                                          optimization_passes=self._optimization_passes,
                                          sp_tracker_track_memory=self._sp_tracker_track_memory)

    # recover regions
    ri = self.project.analyses.RegionIdentifier(self.func, graph=clinic.graph, kb=self.kb)

    # structure it
    rs = self.project.analyses.RecursiveStructurer(ri.region, kb=self.kb)

    # simplify it
    s = self.project.analyses.RegionSimplifier(rs.result, kb=self.kb)

    codegen = self.project.analyses.StructuredCodeGenerator(self.func, s.result, cfg=self._cfg, kb=self.kb)

    self.clinic = clinic
    self.codegen = codegen
```

# Loading

- You've done this!

# Disassembly

- You've done this too!

# Lifting

- Intermediate Representation (IR)
  - Concise description of instruction semantics
- Three-argument IR
- Examples: BAP, VEX
- Abstract across architectures
  - Common abstractions?
  - Representing memory? Registers?
  - Organized in basic blocks

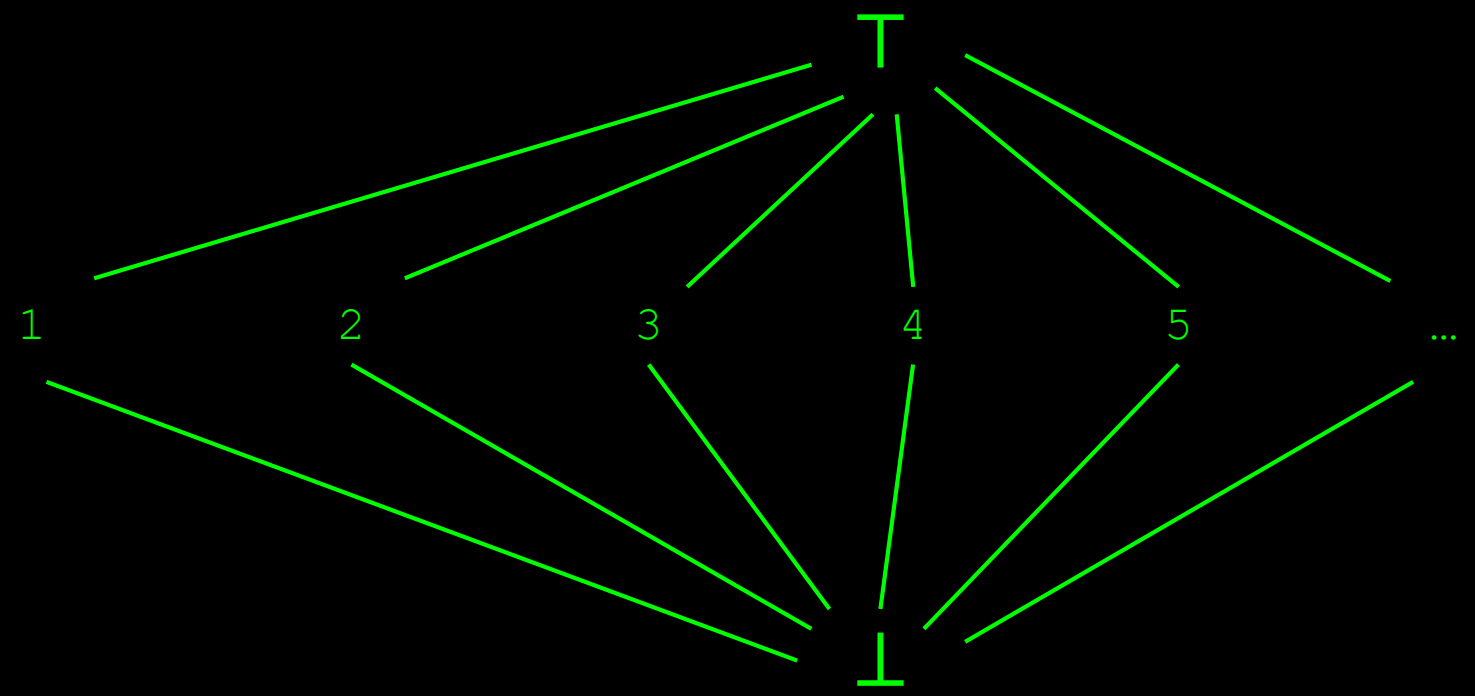
```
In [13]: list(p.kb.functions['main'].blocks)[1].vex.pp()
IRSB {
   t0:Ity_I64 t1:Ity_I32 t2:Ity_I64 t3:Ity_I64 t4:Ity_I64 t5:I

   00 | ------ IMark(0x4040cd, 7, 0) ------
   01 | PUT(rsi) = 0x000000000041887e
   02 | ------ IMark(0x4040d4, 5, 0) ------
   03 | PUT(rdi) = 0x0000000000000006
   04 | PUT(rip) = 0x00000000004040d9
   05 | ------ IMark(0x4040d9, 6, 0) ------
   06 | t3 = LDle:I64(0x0000000000421f08)
   07 | t7 = GET:I64(rsp)
   08 | t6 = Sub64(t7,0x0000000000000008)
   09 | PUT(rsp) = t6
   10 | STle(t6) = 0x00000000004040df
   11 | t8 = Sub64(t6,0x0000000000000080)
   12 | ====== AbiHint(0xt8, 128, t3) ======
   NEXT: PUT(rip) = t3; Ijk_Call
}
```

# Data Flow Analysis

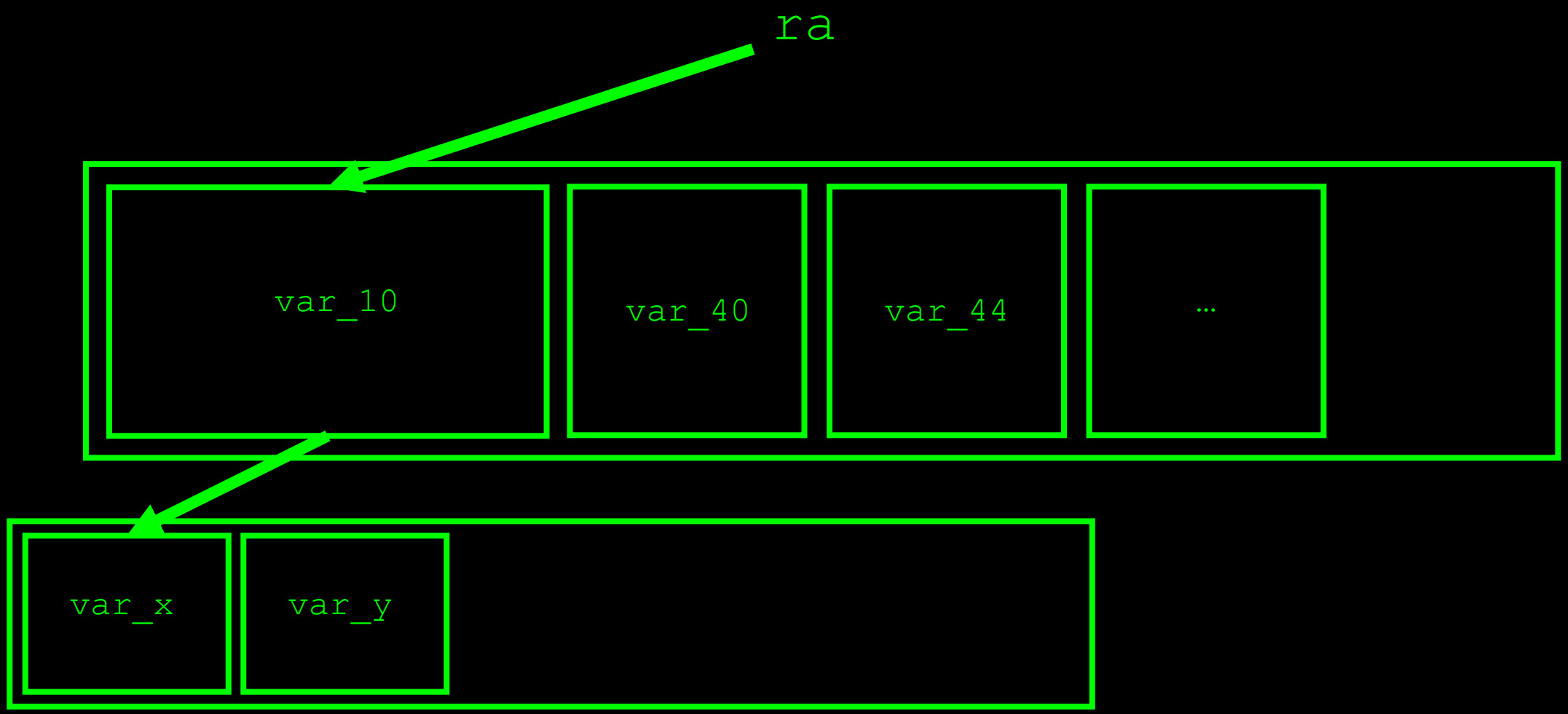
- Constant Propagation
- Calling Convention Analysis
- Condition Code Propagation
- Register Copy Elimination
- Stack Frame Analysis
- Dead Register & Condition Code Elimination
- Variable Recovery

# Constant Propagation



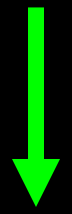

- resolving indirection
- deobfuscation

# Variable Recovery



# Register Copy Elimination

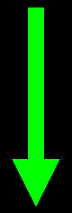
```
3   ax = si          ; def = {ax}      use = {}
4   dx:ax = ax       ; def = {dx, ax}  use = {ax}
5   tmp = dx:ax      ; def = {tmp}     use = {dx, ax}
6   ax = tmp / di    ; def = {ax}      use = {tmp}
8   dx = 3           ; def = {dx}      use = {}
9   ax = ax * dx     ; def = {ax}      use = {ax, dx}
10  si = ax          ; def = {}        use = {ax}
```



```
10 si = (si / di) * 3
```

# Dead Condition Code Elimination

```
14   cmp [bp-6]:[bp-8], dx:ax   ; cc-def = ZF, CF, SF
15   jg B2                      ; cc-use = SF
```

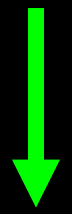


```
14   cmp [bp-6]:[bp-8], dx:ax   ; cc-def = SF
```

Important when lifted to IR

# Condition Code Propagation

```
14   cmp [bp-6]:[bp-8], dx:ax   ; cc-def = SF
15   jg B2                      ; cc-use = SF
```

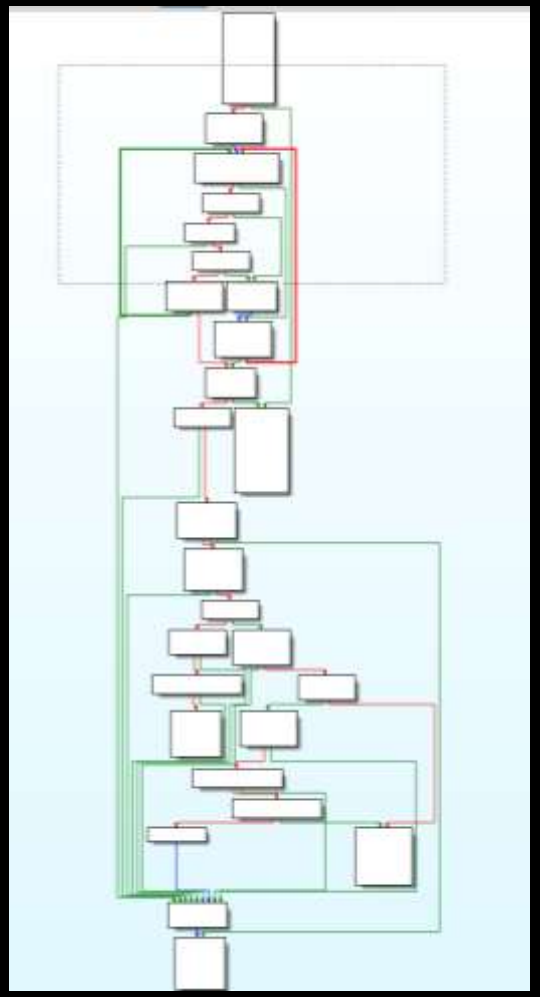


```
15   jcond ([bp-6]:[bp-8] > dx:ax) B2
```

# Stack Frame Analysis

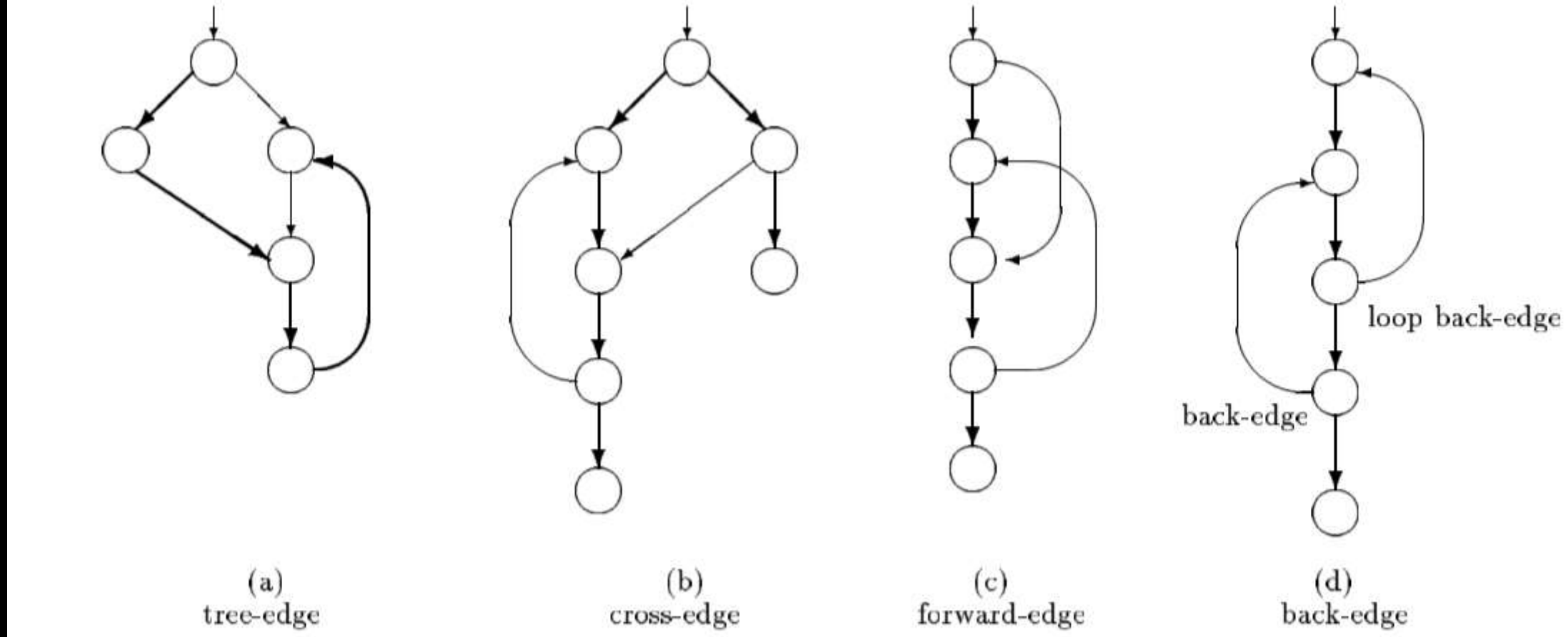
- Abstract Domain: {rsp, rbp} × {1,2,…} and ⊤ and ⊥
- Track constant offsets off of rbp and rsp
- Determine overapproximation of stack from size at all program points in function

# Control Flow Structuring



```
if (…) {
    …
    while (…) {
        …
        goto lbl
    }
} else  if (…) {
    …
} else {
    if (…) {
lbl:    for (…) {
                …
            }
    }
    …
}
```

# Control Flow Structuring



# Control Flow Structuring

- Basic Pattern Matching
- Cifuentes Thesis
- Phoenix
- No More Goto
- Focus of next week

# Type Inference

- Early approaches: Unification-Based
  - Relies on the equality constraint Type(a) == Type(b)
  - Problems:
    - Overunification
    - Not expressive enough (missing fields, pointer passing)
- TIE: subtying!
  - Relies on the subtyping constraint $Vals(Type(a)) \subseteq Vals(Type(b))$
- Retypd: polymorphic types! existential types!
  - (i.e. generics and interfaces)

# Codegen

- Now we have functions, IR, etc. But how do we get C?
- Translate semantics of IR and control flow into pseudocode

# Future Directions: Name Recovery

- Now we have variables, but they don't mean much!
- Can we automate recovering variables names?
- DEEP LEARNING!!!
- DIRE, Debin

```
v2 = a2;
v3 = *a1;
v4 = (_BYTE *)a1[1];
if ( (unsigned __int64)v4 <= *a1 )
  return sub_2840(*a1, v4, &off_73C0);
v5 = (_BYTE *)*a1;
v6 = 0LL;
v7 = 0LL;
do
{
  while ( *v5 != 45 )
  {
ABEL_3:
    if ( v4 == ++v5 )
      goto LABEL_9;
  }
  if ( !v7 )
    goto LABEL_29;
  if ( v6 )
    return 0xFFFFFFFFLL;
  if ( !v7 )
  {
ABEL_29:
    v7 = v5;
```