SCEV
Establishing Pre and Post Conditions

BPF MicroConference, Lisbon 2019
John Fastabend, Cilium
SCEV:

Scaler Evolution: Technique to understand how variables change with execution.
SCEV:

Scaler Evolution: Technique to understand how variables change with execution.

Example: Generate set of conditions required to establish loop is safe to run.

Safety Properties:
(1) Loop terminates
(2) Memory access are in bounds
  C[100] = {...} ; x = 99; C[x] //safe; C[x + 1] //unsafe
SCEV: Example

```c
for (j = 0; j < 300; j++) {
    if (j & 1) {
        m = PT_REGS_RC(ctx);
        i = m + j;
    } else {
        m = j;
    }
    sum += i * m;
}
```
SCEV: Example

r4 = 0
r2 = 0
r3 = 0
goto +10 <LBB0_1>

LBB0_3:
  r0 *= r3
  r0 += r4
  r2 += 1
  r5 = r2
  r5 <<= 32
  r5 >>= 32
  r4 = r0
  if r5 != 300 goto +2 <LBB0_1>
end_loop

exit

LBB0_1:
  r5 = r2
  r5 &= 1
  r0 = r2
  if r5 == 0 goto -14 <LBB0_3>
  r0 = *(u64 *)(r1 + 80)
  r3 = r2
  r3 += r0
  goto -18 <LBB0_3>
SCEV: Example

\[
\begin{align*}
    r4 &= 0 \\
    r2 &= 0 \\
    r3 &= 0 \\
    \text{goto } +10 \text{ <LBB0_1>} \\
\end{align*}
\]

**LBB0_3:**

\[
\begin{align*}
    r0 &= r3 \\
    r0 &= r4 \\
    r2 &= 1 \\
    r5 &= r2 \\
    r5 &= 32 \\
    r5 &= 32 \\
    r4 &= r0 \\
    \text{if } r5 \neq 300 \text{ goto } +2 \text{ <LBB0_1>} \\
\end{align*}
\]

\[
\begin{align*}
    \text{end_loop} \\
    \text{exit}
\end{align*}
\]

**LBB0_1:**

\[
\begin{align*}
    r5 &= r2 \\
    r5 &= 1 \\
    r0 &= r2 \\
    \text{if } r5 == 0 \text{ goto } -14 \text{ <LBB0_3>} \\
    r0 &= *(u64 *)(r1 + 80) \\
    r3 &= r2 \\
    r3 &= r0 \\
    \text{goto } -18 \text{ <LBB0_3>} \\
\end{align*}
\]

\[
\begin{align*}
    r5: (\_\_\_, \text{mov, } r2)
\end{align*}
\]
SCEV: Example

r4 = 0
r2 = 0
r3 = 0
goto +10 <LBB0_1>

LBB0_3:
r0 *= r3
r0 += r4
r2 += 1
r5 = r2
r5 <<= 32
r5 >>= 32
r4 = r0
if r5 != 300 goto +2 <LBB0_1>
end_loop
exit

LBB0_1:
r5 = r2
r5 &= 1
r0 = r2
if r5 == 0 goto -14 <LBB0_3>

r0 = *(u64 *)(r1 + 80)
r3 += r0
goto -18 <LBB0_3>
SCEV: Example

r4 = 0
r2 = 0
r3 = 0
goto +10 <LBB0_1>

LBB0_3:
  r0 *= r3
  r0 += r4
  r2 += 1
  r5 = r2
  r5 <<= 32
  r5 >>= 32
  r4 = r0
  if r5 != 300 goto +2 <LBB0_1>
end_loop
exit

LBB0_1:
  r5 = r2
  r5 &= 1
  r0 = r2
  if r5 == 0 goto -14 <LBB0_3>
  r0 = *(u64 *)(r1 + 80)
  r3 = r2
  r3 += r0
  goto -18 <LBB0_3>
SCEV: Example

r4 = 0
r2 = 0
r3 = 0
goto +10 <LBB0_1>

LBB0_3:

r0 *= r3  
r0: ( _ , =, r2, *, r3)
r0 += r4  
  r0: ( _ , =, r2, *, r3, +, r4)
r2 += 1  
  r2: ( _ , +, 1)
r5 = r2   
  r5: ( _ , =, r2)
r5 <<= 32
r5 >>= 32
r4 = r0
if r5 != 300 goto +2 <LBB0_1>

end_loop

exit

LBB0_1:

r5 = r2  
  r5: ( _ , =, r2)
r5 &= 1  
  r5: ( _ , =, r2, &, 1)
r0 = r2  
  r0: ( _ , =, r2)
if r5 == 0 goto -14 <LBB0_3>
r0 = *(u64 *)(r1 + 80)
r3 = r2  
  r3: = r2
r3 += r0
if r5 == 0 goto -14 <LBB0_3>
goto -18 <LBB0_3>
SCEV: Example

r4 = 0
r2 = 0
r3 = 0
goto +10 <LBB0_1>

LBB0_3:
r0 *= r3                        r0: ( _ , =, r2, *, r3)
r0 += r4                        r0: ( _ , =, r2, *, r3, +, r4)
r2 += 1                         r2: ( _ , +, 1)
r5 = r2                  r5: ( _ , =, r2)     r5': (_, +, 1)
r5 <<= 32                      r5': (_, +, 1, << 32)
r5 >>= 32                      r5': (_, +, 1, >> 32)
r4 = r0                                        r4: (_, =, r0)
if r5 != 300 goto +2 <LBB0_1>
end_loop
exit

LBB0_1:
r5 = r2                        r5: ( _ , =, r2)
r5 &= 1                         r5: ( _ , =, r2, &, 1)
r0 = r2                        r0: ( _ , =, r2)
if r5 == 0 goto -14 <LBB0_3>
r0 = *(u64 *)(r1 + 80)
r3 = r2
r3 += r0
r3 += r0
goto -18 <LBB0_3>
SCEV: Example

\[
\begin{align*}
\text{r4} &= 0 \\
\text{r2} &= 0 \\
\text{r3} &= 0 \\
\text{goto} &+10 <\text{LBB0}_1> \\
\text{LBB0}_3: &
\begin{align*}
\text{r0} &= \text{r3} &\text{r0}: (\_, =, \text{r2}, *, \text{r3}) \\
\text{r0} &= \text{r4} &\text{r0}: (\_, =, \text{r2}, *, \text{r3}, +, \text{r4}) \\
\text{r2} &= +1 &\text{r2}: (\_, +, 1) \\
\text{r5} &= \text{r2} &\text{r5}: (\_, =, \text{r2}) \\
\text{r5} &= 32 &\text{r5}: (\_, =, \text{r2})
\end{align*}
\begin{align*}
\text{r5} &= \text{r5} &\text{r5}' : (\_, +, 1) \\
\text{r5} &= \text{r5} &\text{r5}' : (\_, +, 1, << 32) \\
\text{r4} &= \text{r4} &\text{r4} : (\_, =, \text{r0})
\end{align*}
\begin{align*}
\text{if} \text{r5} \neq 300 &\text{goto} +2 <\text{LBB0}_1> \\
\text{end\_loop} \\
\text{exit}
\end{align*}
\end{align*}
\]
SCEV: Example

REG#0: (unknown, unknown (MOV), <>)
REG#1: (none)
REG#2: (unknown, add, 1)
REG#3: (unknown, add (MOV), REG0)
REG#4: (unknown, unknown (MOV), <>)
REG#5: (unknown, add, 1)
REG#6: (none)
REG#7: (none)
REG#8: (none)
REG#9: (none)
REG#10: (none)

REG#0: (unknown, unknown (MOV), <>)
REG#1: (none)
REG#2: (unknown, add, 1)
REG#3: (unknown, add (MOV), REG0)
REG#4: (unknown, unknown (MOV), <>)
REG#5: (unknown, add, 1)
REG#6: (none)
REG#7: (none)
REG#8: (none)
REG#9: (none)
REG#10: (none)
SCEV: Example

REG#0: ( <unknown>, unknown (MOV), <> )
REG#1: (none)
REG#2: ( <unknown>, add, 1)
REG#3: ( <unknown>, add (MOV), REG0)
REG#4: ( <unknown>, unknown (MOV), <> )
REG#5: ( <unknown>, add, 1)
REG#6: (none)
REG#7: (none)
REG#8: (none)
REG#9: (none)
REG#10: (none)

REG#0: ( <unknown>, unknown (MOV), <> )
REG#1: (none)
REG#2: ( <unknown>, add, 1)
REG#3: ( <unknown>, add (MOV), REG0)
REG#4: ( <unknown>, unknown (MOV), <> )
REG#5: ( <unknown>, add, 1)
REG#6: (none)
REG#7: (none)
REG#8: (none)
REG#9: (none)
REG#10: (none)

TBD: implement "*" operator logic
SCEV: Example

r3 = 0

... if r5 != 300 goto +2 <LBB0_1>

REG#5: ( <unknown>, add, 1)

Terminate?

(a) Is r5 monotonic? Yes
(b) Tripcount, 300
(c) Memory safe? Yes
SCEV: Details

SCEV Foldings:

\[(x, +, a, +, b) \rightarrow (x, + (a+b))\]
SCEV: Details

Memory Safety in general:

(1) When we see memory read/write push entire state on to stack with reference

(2) Complete SCEV pass, calculate trip count

(3) Check memory access from stack
SCEV: Details

BPF stack:

Track register + stack_slot so that all expressions are unique.

(reg#, slot) \rightarrow (REG#5, curr): ( <unknown>, add, 1)
SCEV: What Else

- Operators &, |, >>, <<, need a solution for folding
- Works for loops, let's try functions
- Testing and more testing
- SCEV implement folding operations
Thanks! Questions?