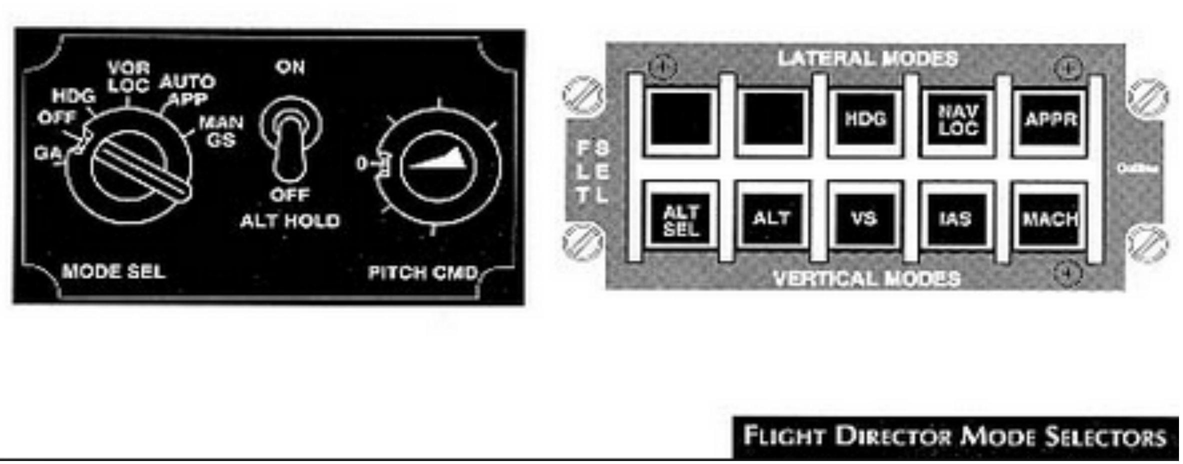
Static resource analysis with application to avionics systems

By: **Bishoksan Kafle** (Roskilde Univ.)
Sup: Jorge A. Navas (NASA Ames)
Joint work with
John P. Gallagher (Roskilde Univ.)

Outline

- Motivation
- Summary of our approach
- Horn logic: language for analysis
- Lexicographic ordering and Bound computation
- Conclusion and future work

Motivation (I)



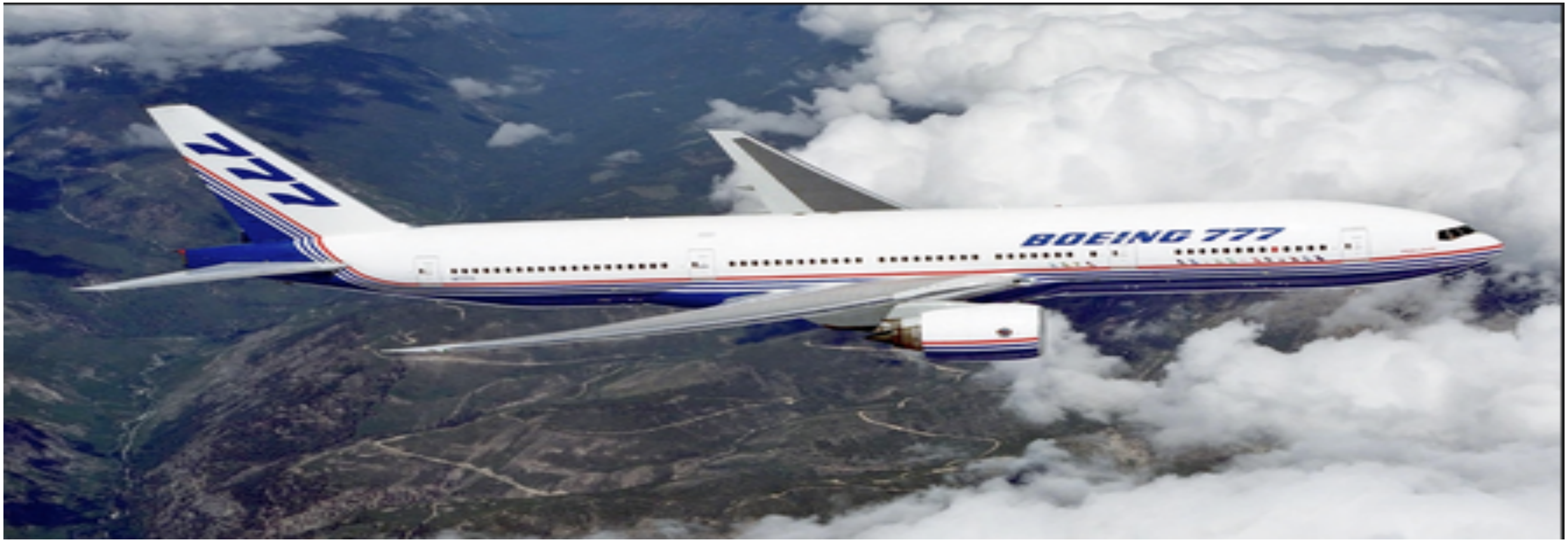
- Reactive systems

```
While(true){
    Switch(event){
    Case 1: mode1()
    Case 2: mode2()
    Default: modeDefault()
}
}
```

- Operates on different modes

Motivation (II)

Inferring resource consumption (memory, energy, time etc.) of each mode is useful e.g. to prevent side-channel attacks.



Bound

- Symbolic expression in terms of program's input parameters (upper bound)
- Understanding program performance (complexity)
- **Resource bound = bound*suitable resource measure**
- Useful for resource analysis and verification

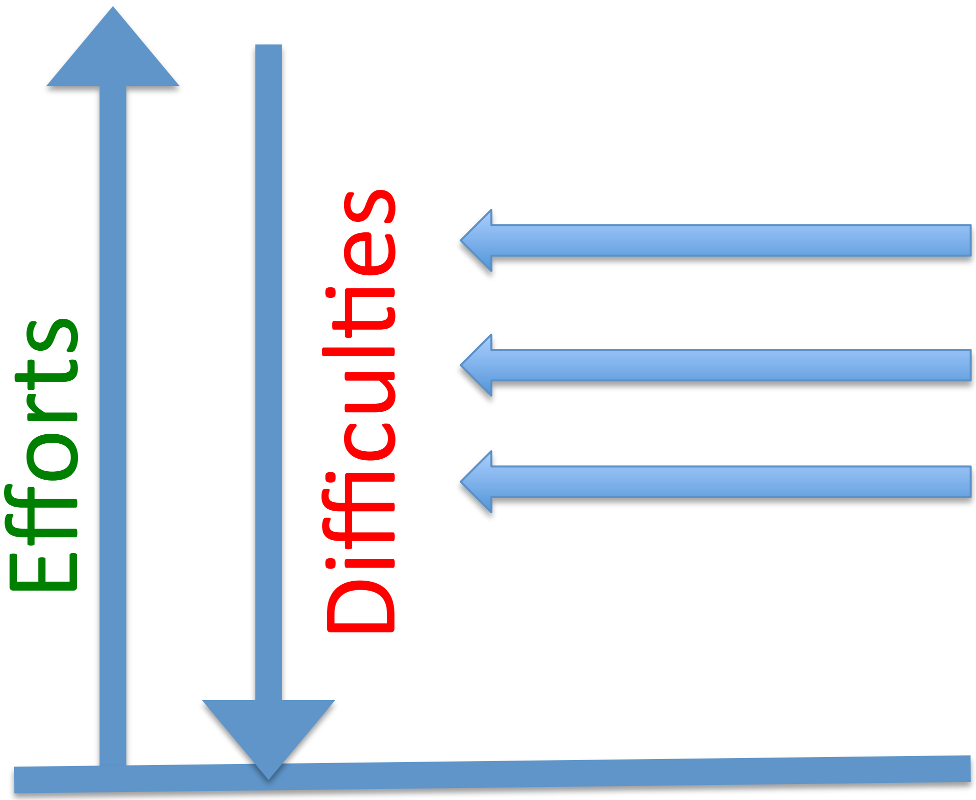
    Verifying if some energy budgets are met

Bound analysis can answer…

```
main( uint c,  uint d){
0.    int a=c,b=d;
1.    while (a>0){
2.     if (*)
3.       b++;
        else
4.       while (b>0)
5.           b--;
6.      a--; }
7.     return 0;
 }
```

- Nr. of visits to a statement (e.g. line 3)
- Nr of loop iterations
- Nr of calls to a function

Automatic analysis



- Verification (Q)
- Termination (Q)
- Bound(N)

Qualitative=Q

Quantitative=N

1. Can we reuse the work from other analyses?
2. Do we have a common language on which we can base our analyses?

The answers are affirmative ☺

1. The roadmap for bound analysis:

   Program invariants (verification)

   Termination arguments (termination)

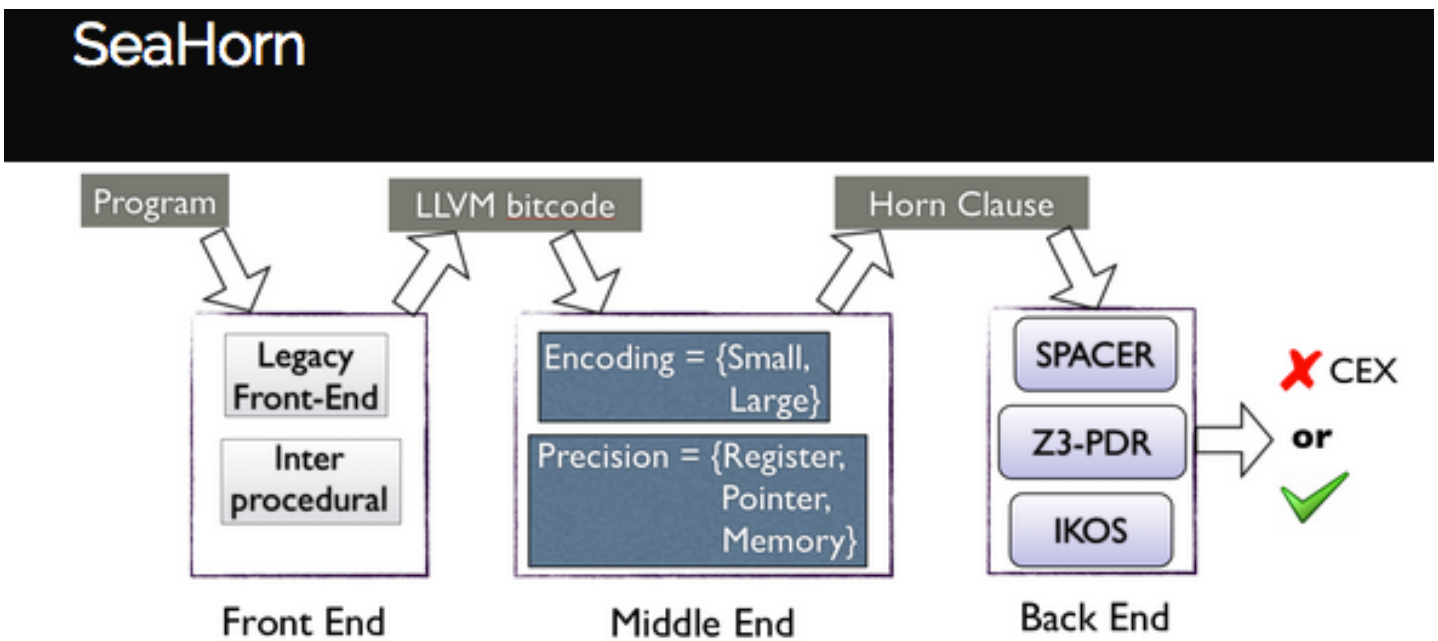
   Bound computation (bound)

2. Horn logic (fragment of FOL) as a common language

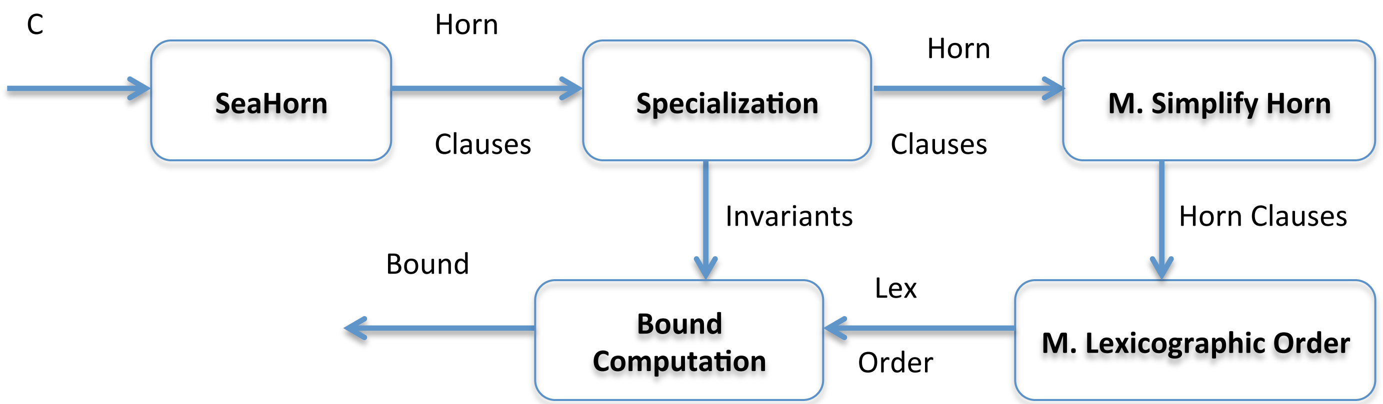
The answers are affirmative ☺

2. Horn logic (fragment of FOL) as a common language

3. SeaHorn a great tool to exploit Horn logic



Overview of our tool-chain



[1] Moritz Sinn, Florian Zuleger, Helmut Veith:
A Simple and Scalable Static Analysis for Bound Analysis and Amortized Complexity Analysis. CAV 2014

Horn clause

$$p(X) \leftarrow \phi \land p_1(X_1) \land \ldots \land p_k(X_k)$$

Linear clause (Transition system)

$$p(X') \leftarrow \phi \land q(X)$$

Restricting the shape of constraints

$$p(X') \leftarrow X' \leq X + K \land q(X), K \in \mathbb{Z}^n$$

Horn clauses

Restricting the shape of constraints

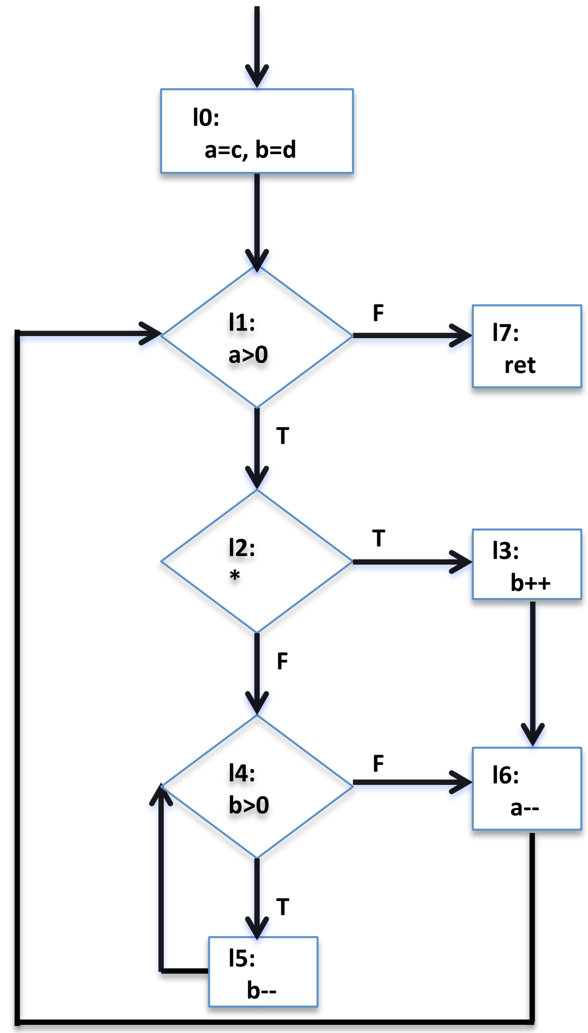
$$p(X') \leftarrow X' \leq X + K \wedge q(X), K \in \mathbb{Z}^n$$

Allowing parameters

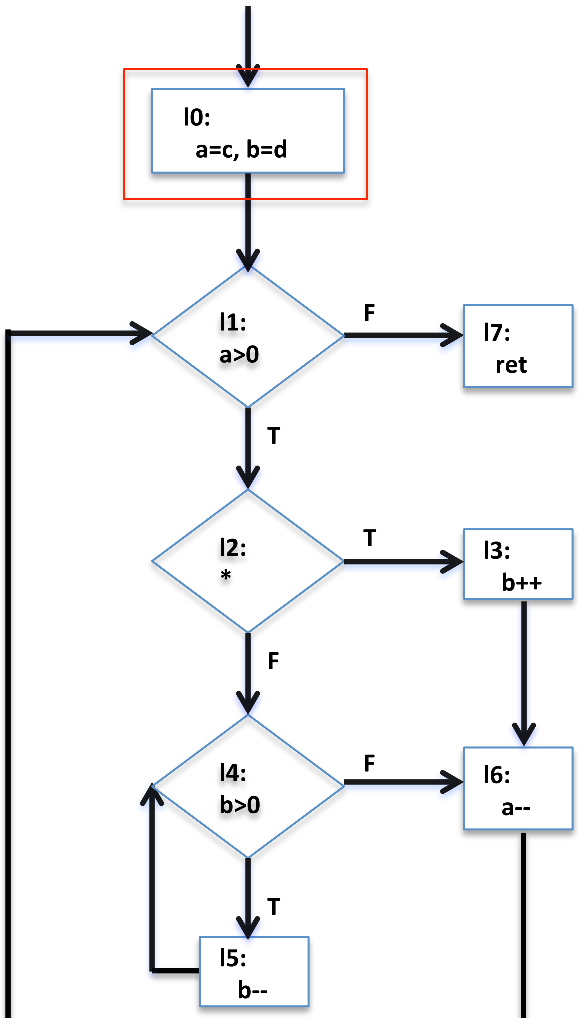
$$p(X') \leftarrow X' \leq X + K \wedge q(X), k_i \in \mathbb{Z} \cup Params$$

Horn clause generation (1)

```
 main( uint c,  uint d){
0.    int a=c,b=d;
1.    while (a>0){
2.     if (*)
3.        b++;
       else
4.        while (b>0)
5.            b--;
6.      a--; }
7.     return 0;
 }
```

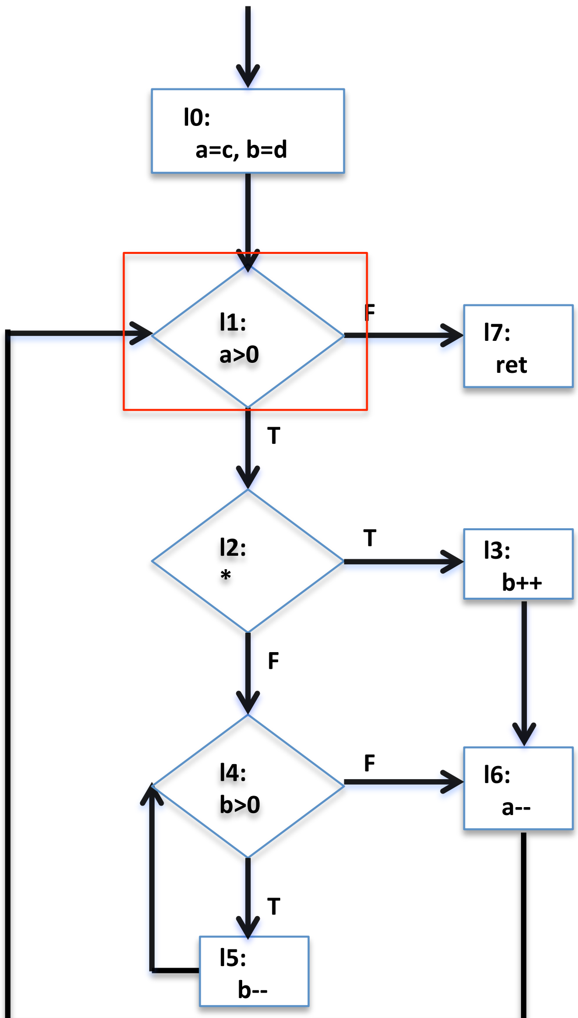


Horn clauses Generation (2)



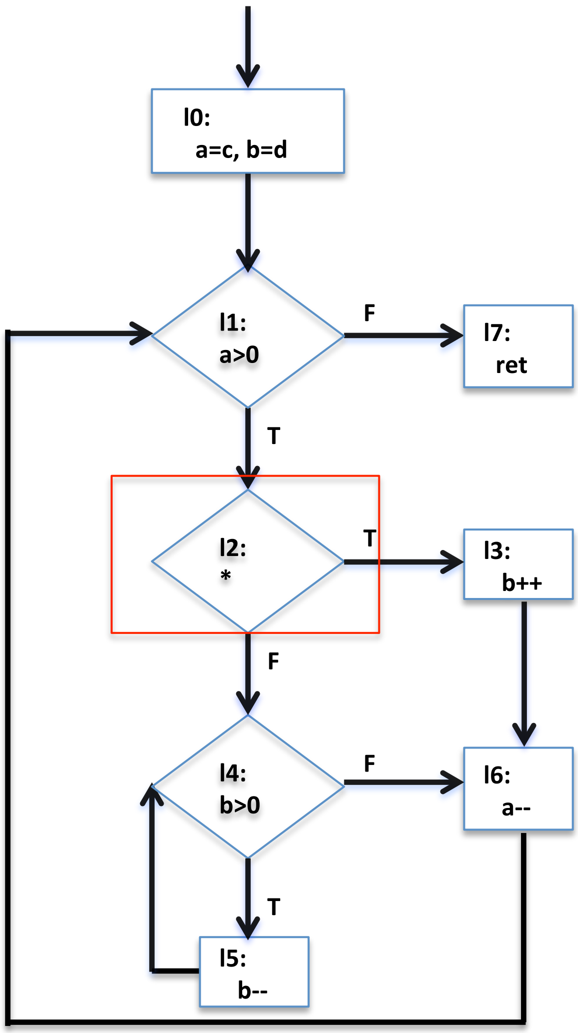

```
p0(A,B,C,D) ←  true.
p1(A,B,C,D) ←  A=C,B=D, C>=0,D>=0,
              p0(A,B,C,D).
p1(A,B,C,D) ←  A=A1-1,p6(A1,B,C,D).
p2(A,B,C,D) ←  A>0,p1(A,B,C,D).
p3(A,B,C,D) ←  p2(A,B,C,D).
p4(A,B,C,D) ←  p2(A,B,C,D).
p4(A,B,C,D) ←  B=B1-1,p5(A,B1,C,D).
p5(A,B,C,D) ←  B>0,p4(A,B,C,D).
p6(A,B,C,D) ←  B=<0,p4(A,B,C,D).
p6(A,B,C,D) ←  B=B1+1,p3(A,B1,C,D).
p7(A,B,C,D) ←  A=<0,p1(A,B,C,D).
ret         ←  p7(A,B,C,D).
```

Horn clauses Generation



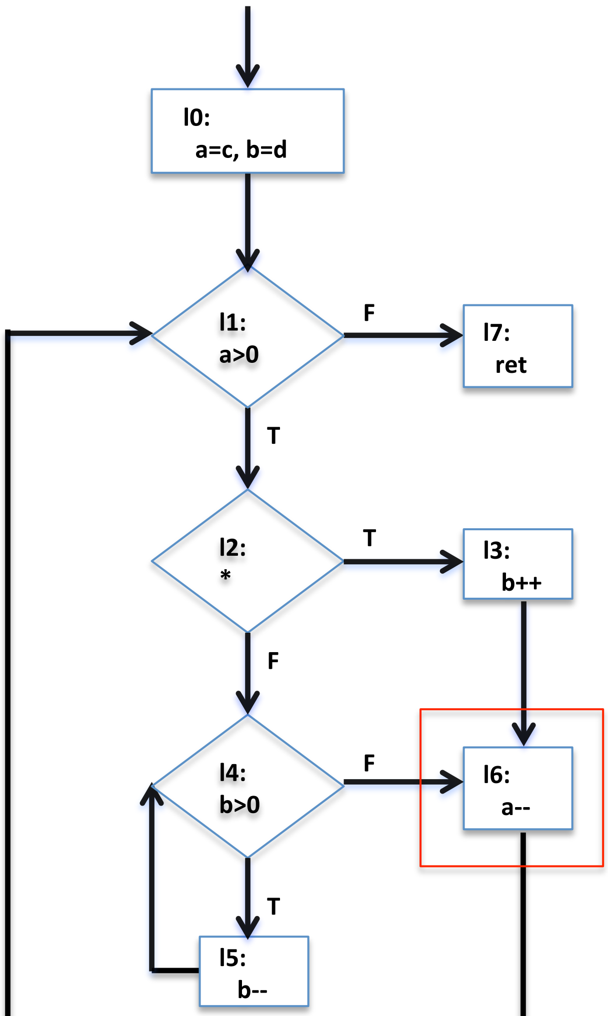

```
p0(A,B,C,D) ← true.
p1(A,B,C,D) ← A=C,B=D, C>=0,D>=0,
              p0(A,B,C,D).
p1(A,B,C,D) ← A=A1-1,p6(A1,B,C,D).
p2(A,B,C,D) ← A>0,p1(A,B,C,D).
p3(A,B,C,D) ← p2(A,B,C,D).
p4(A,B,C,D) ← p2(A,B,C,D).
p4(A,B,C,D) ← B=B1-1,p5(A,B1,C,D).
p5(A,B,C,D) ← B>0,p4(A,B,C,D).
p6(A,B,C,D) ← B=<0,p4(A,B,C,D).
p6(A,B,C,D) ← B=B1+1,p3(A,B1,C,D).
p7(A,B,C,D) ← A=<0,p1(A,B,C,D).
ret         ← p7(A,B,C,D).
```

Horn clauses Generation




```
p0(A,B,C,D) ←  true.
p1(A,B,C,D) ←  A=C,B=D, C>=0,D>=0,
               p0(A,B,C,D).
p1(A,B,C,D) ←  A=A1-1,p6(A1,B,C,D).
p2(A,B,C,D) ←  A>0,p1(A,B,C,D).
p3(A,B,C,D) ←  p2(A,B,C,D).
p4(A,B,C,D) ←  p2(A,B,C,D).
p4(A,B,C,D) ←  B=B1-1,p5(A,B1,C,D).
p5(A,B,C,D) ←  B>0,p4(A,B,C,D).
p6(A,B,C,D) ←  B=<0,p4(A,B,C,D).
p6(A,B,C,D) ←  B=B1+1,p3(A,B1,C,D).
p7(A,B,C,D) ←  A=<0,p1(A,B,C,D).
ret         ←  p7(A,B,C,D).
```

Horn clauses Generation




```
p0(A,B,C,D) ← true.
p1(A,B,C,D) ← A=C,B=D, C>=0,D>=0,
              p0(A,B,C,D).
p1(A,B,C,D) ← A=A1-1,p6(A1,B,C,D).
p2(A,B,C,D) ← A>0,p1(A,B,C,D).
p3(A,B,C,D) ← p2(A,B,C,D).
p4(A,B,C,D) ← p2(A,B,C,D).
p4(A,B,C,D) ← B=B1-1,p5(A,B1,C,D).
p5(A,B,C,D) ← B>0,p4(A,B,C,D).
p6(A,B,C,D) ← B=<0,p4(A,B,C,D).
p6(A,B,C,D) ← B=B1+1,p3(A,B1,C,D).
p7(A,B,C,D) ← A=<0,p1(A,B,C,D).
ret         ← p7(A,B,C,D).
```

Clauses can be viewed as set of equations!

What is the bound of this program?

```
 main( uint c,  uint d){
0.   int a=c,b=d;
1.   while (a>0){
2.    if (*)
3.      b++;
     else
4.      while (b>0)
5.          b--;
6.     a--; }
7.    return 0;
 }
```

Example

```
 main( uint c,  uint d){
0.   int a=c,b=d;
1.   while (a>0){
2.    if (*)
3.      b++;
      else
4.      while (b>0)
5.          b--;
6.     a--; }
7.    return 0;
 }
```

Bound=2*c+d

Example

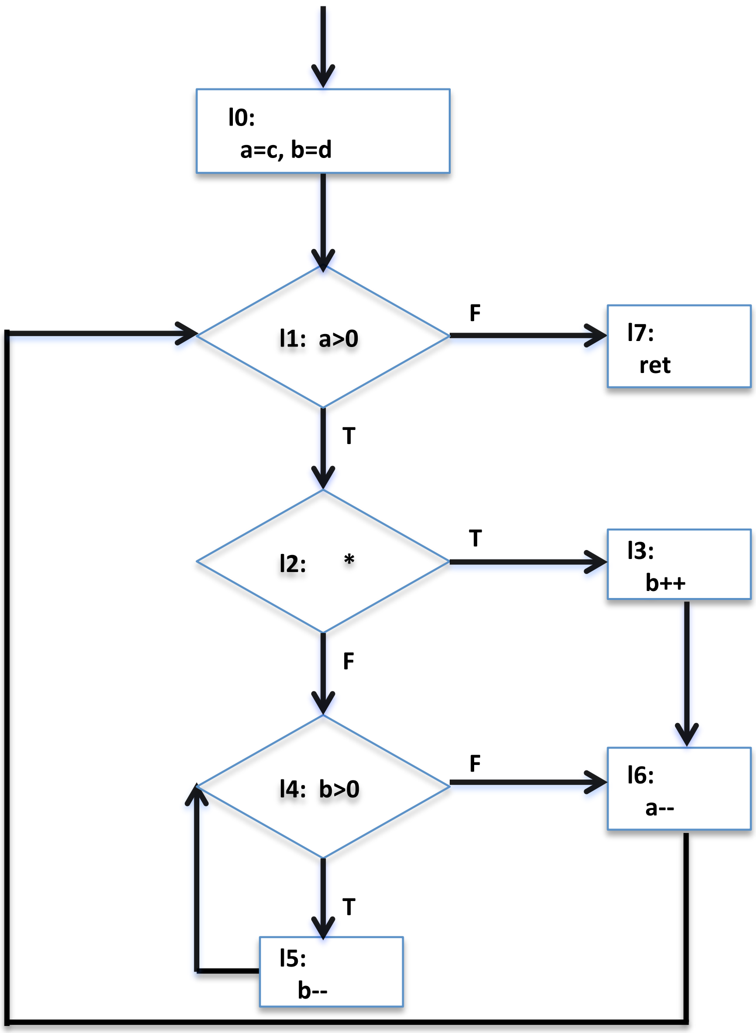
```
main( uint c,  uint d){
    int a=c,b=d;
    while (a>0){
     if (*)
       b++;
      else
       while (b>0)
           b--;
      a--; }
     return 0;
 }
```

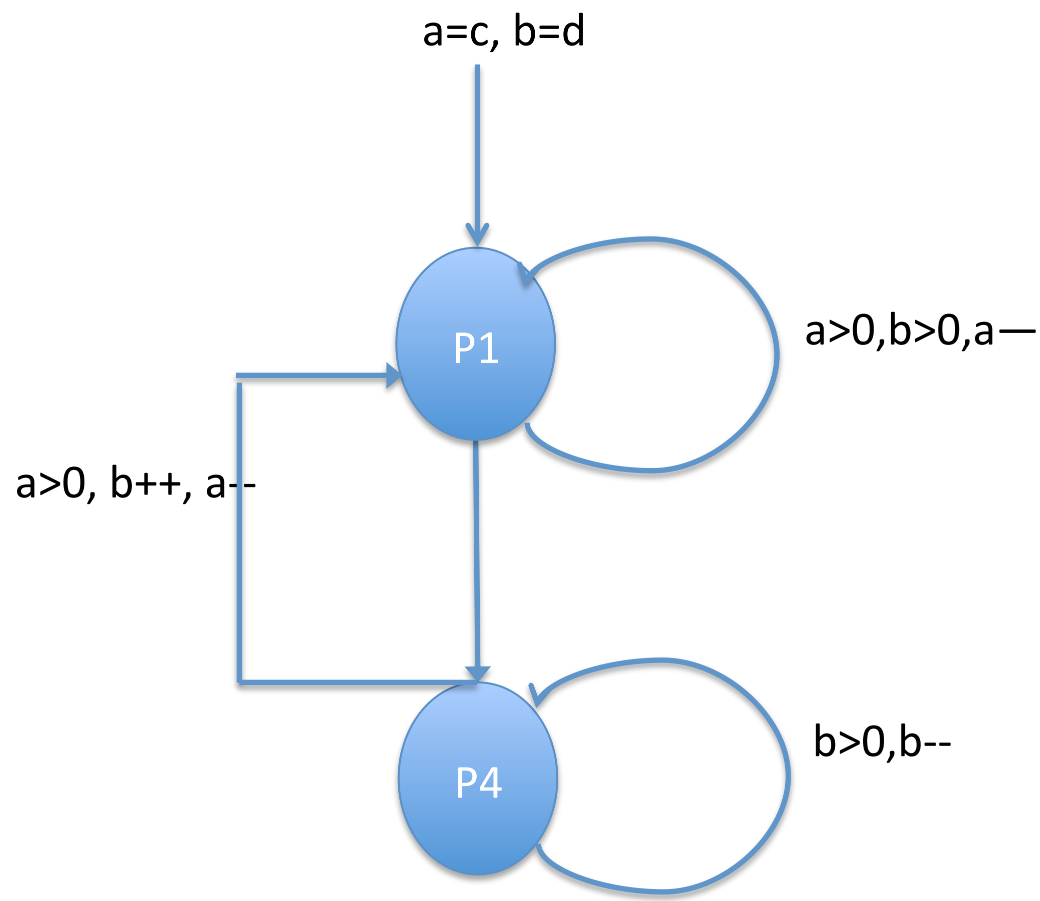
- Outer loop: can be executed at most c times
- The counter for the inner loop can be incremented by outer at most c times
- So inner loop can be executed at most c+d times

Bound=2*c+d

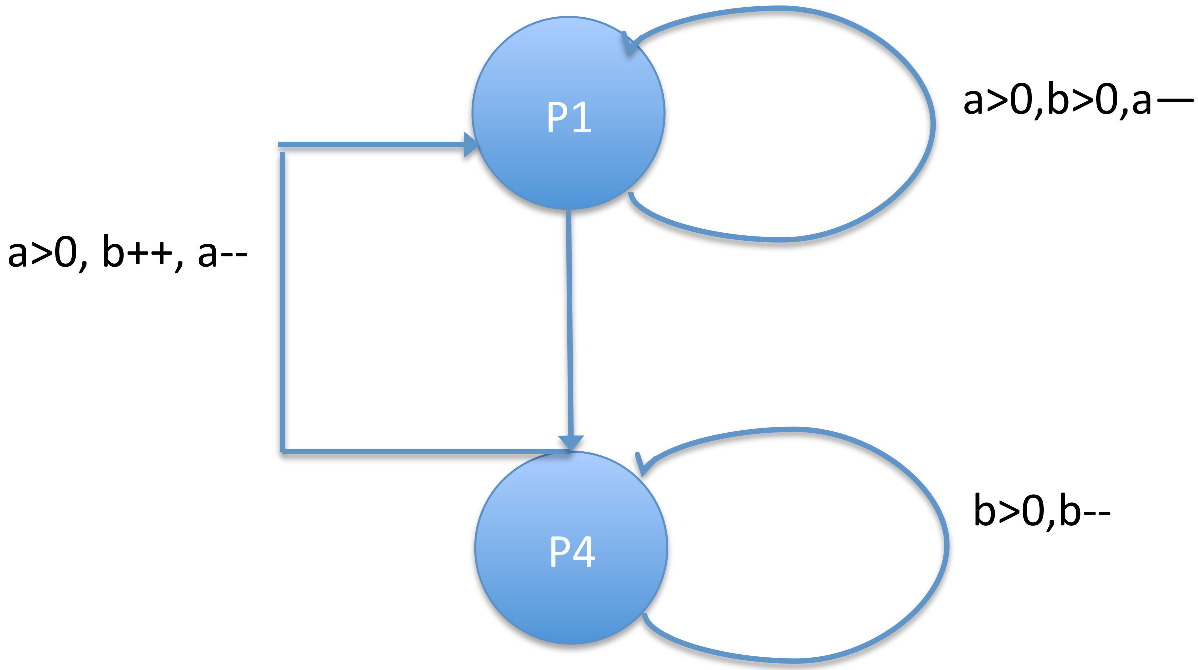
How to derive this automatically?

Single path linear constraint loops





Special form of Horn clauses (VASS)




```
p1(A,B,C,D) ←  A=<E-1,B=<F+1,p1(E,F,C,D)
p1(A,B,C,D) ←  A=<E-1,B=<F,p1(E,F,C,D)
p4(A,B,C,D) ←  A=<E,B=<F-1,p4(E,F,C,D)
```

Lexicographic ranking function

```
T1: p1(A,B,C,D) ←  A=<E-1,B=<F+1,p1(E,F,C,D)
T2: p1(A,B,C,D) ←  A=<E-1,B=<F,p1(E,F,C,D)
T3: p4(A,B,C,D) ←  A=<E,B=<F-1,p4(E,F,C,D)
```

- <a,a,b>
- Either a is decreasing; or
- b is decreasing and a is not increasing

Bound computation

```
T1: p1(A,B,C,D) ← A=<E-1,B=<F+1,p1(E,F,C,D)
T2: p1(A,B,C,D) ← A=<E-1,B=<F,p1(E,F,C,D)
T3: p4(A,B,C,D) ← A=<E,B=<F-1,p4(E,F,C,D)
```

InitVal: a=c, b=d

Lex: <a,a,b>

- Bound(T1)= InitVal (a)=c
- Bound(T2)= InitVal (a) + Bound(T1)*increment(a,T1) =c+0=c

Bound computation

```
T1: p1(A,B,C,D) ← A=<E-1,B=<F+1,p1(E,F,C,D)
T2: p1(A,B,C,D) ← A=<E-1,B=<F,p1(E,F,C,D)
T3: p4(A,B,C,D) ← A=<E,B=<F-1,p4(E,F,C,D)
```

InitVal: a=c, b=d

Lex: <a,a,b>

- Bound(T3)= InitVal (b)+
  Bound(T1)*increment(b,T1)+
  Bound(T2)*increment(b,T2) = d+c*1+0=c+d

  Overall bound = 3*c+d

All looks simple, what is the role of Invariants?

- Deriving the special form of Horn clauses (with difference bound constraints)
- During specialization (removing infeasible paths)
- Computing initial values of ranking functions

But our tool derives: 4(c+d)-1.

Conclusions and Future work

- Bound analysis using Horn logic,
- Application to avionics systems

In the future:

- Compositional bound analysis
- Combination of over-approximation with under-approximation for bound analysis

Thank you!

Amortized analysis

- Derives upper bound
- More realistic in practice
- Concerned with the overall cost of a sequence of operations
- Standard example stack
- N operations
- Push-constant time
- Pop many linear
- Amortized complexity of n operations is linear

Bounds on some programs

```
1. singleloop
for(i=0;i<n;i++){
    }

bound=n

//2. doubleloop
for (i=0;i< n;i++){
    for (j=0;j<m;j++){
    }
}

bound=n+m+n*m
```

```
//3. Sinn et. al Example 1
    a=n;
    b=0;
    while(a>0){
        a--;b++;
        while(b>0){
            b--;
            for (int i=n-1;i>0;i--)
            if (a>0) {
                a--;b++;
            }
        }
    }

bound=3*n^2+n-1
```