Resource analysis of C program by analyzing its Horn clause representation

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

DANSAS, Odense, 20-08-2015

Resource 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;
 }
```

- Amount of resource consumed by this program
- Part of code responsible for the highest amount of resource consumption
- Resource consumed by the inner loop etc.

- Resource = time, energy, memory etc.

Complexity Bound/Bound

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

    Verifying if some energy budgets are met

Reusability

Can we reuse the work done in program verification and termination for bound analysis?

Do we have a common language on which we can base our analyses?

The answers are affirmative ☺

1. Our approach for bound analysis:

   Program invariants (verification)

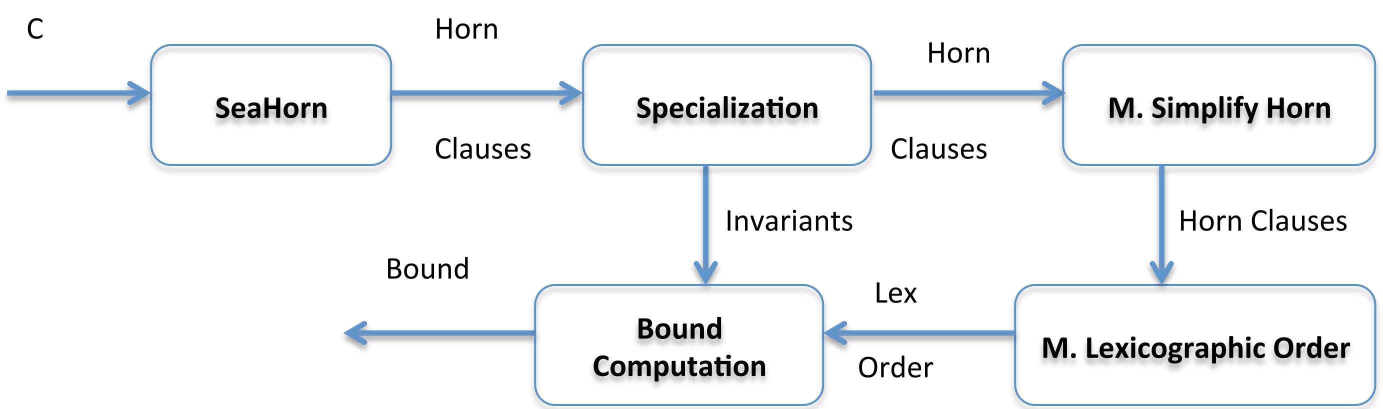
   Termination arguments (termination)

   Bound computation (bound)

2. Horn clauses as a common language

◆ Prolog systems are great tools to exploit Horn clauses

Architecture 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

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

Linear clause (Transition system)

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

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

Restricting the shape of constraints

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

Horn clauses

Restricting the shape of constraints

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

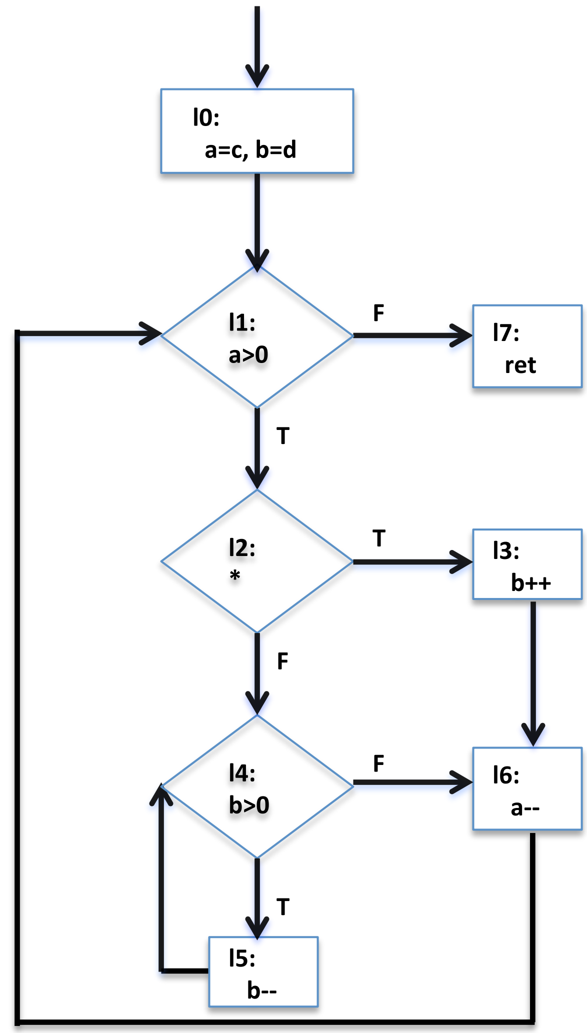
Allowing parameters

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

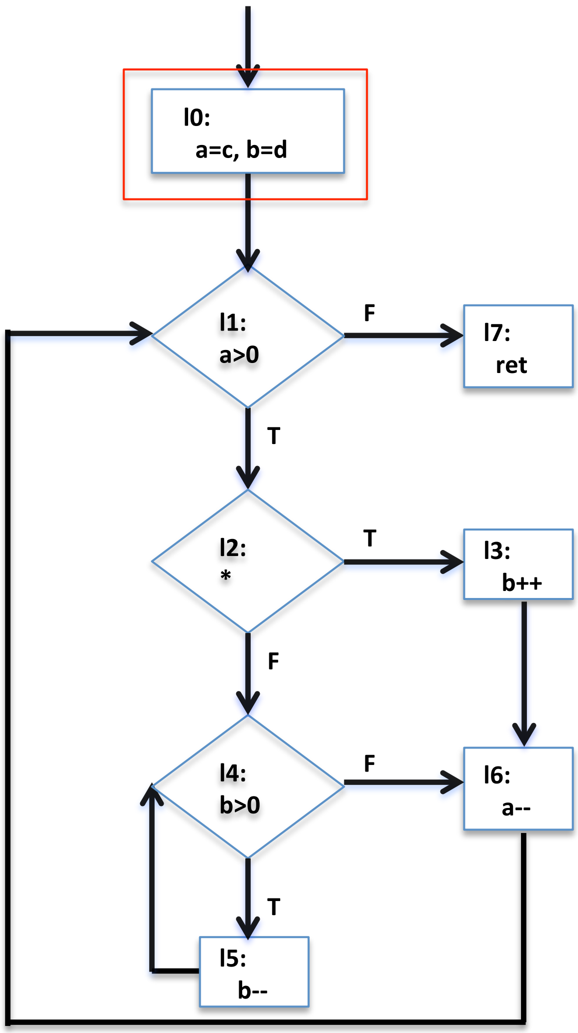
Params are program's input params eg. c and d

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;
 }
```

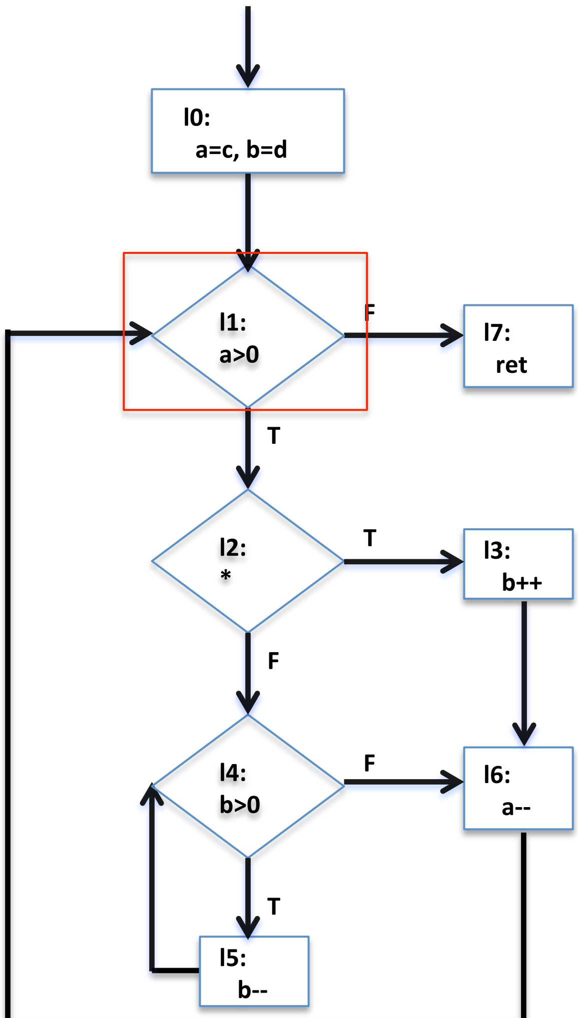


Horn clauses Generation (2)



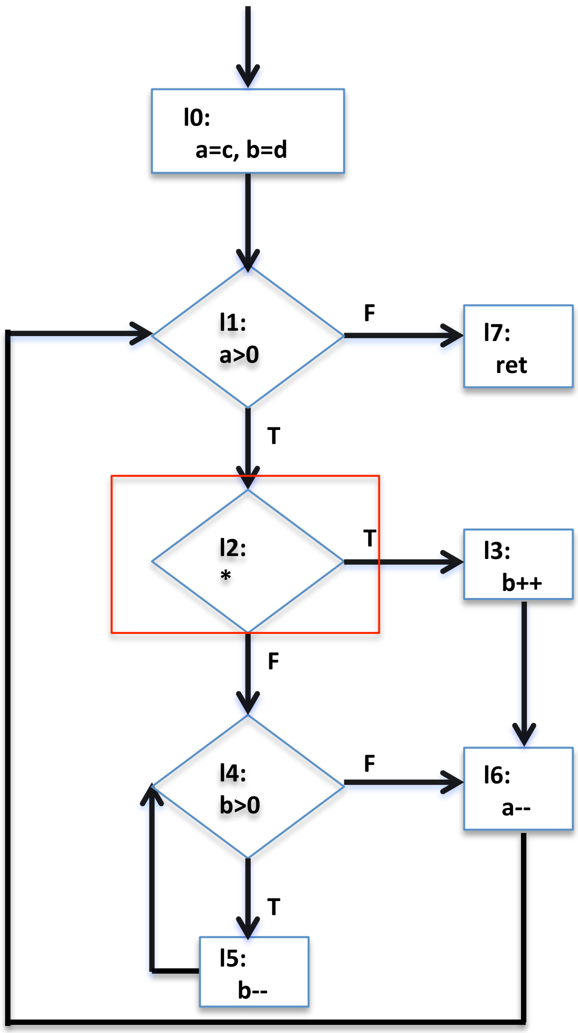

```
p0(A,B,C,D) ← true.
p1(A,B,C,D) ← A=C,B=D, C>=0,D>=0,
              p0(A,B,C,D).
p1(A,B,C,D) ← A=A1-1,p6(A1,B,C,D).
p2(A,B,C,D) ← A>0,p1(A,B,C,D).
p3(A,B,C,D) ← p2(A,B,C,D).
p4(A,B,C,D) ← p2(A,B,C,D).
p4(A,B,C,D) ← B=B1-1,p5(A,B1,C,D).
p5(A,B,C,D) ← B>0,p4(A,B,C,D).
p6(A,B,C,D) ← B=<0,p4(A,B,C,D).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



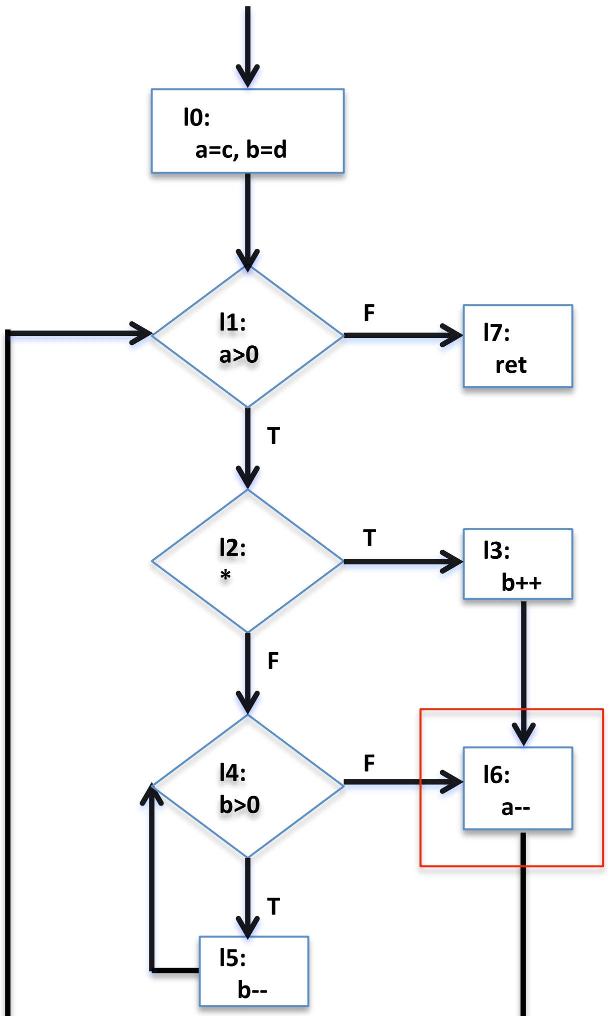

```
p0(A,B,C,D) ←  true.
p1(A,B,C,D) ←  A=C,B=D, C>=0,D>=0,
               p0(A,B,C,D).
p1(A,B,C,D) ←  A=A1-1,p6(A1,B,C,D).
p2(A,B,C,D) ←  A>0,p1(A,B,C,D).
p3(A,B,C,D) ←  p2(A,B,C,D).
p4(A,B,C,D) ←  p2(A,B,C,D).
p4(A,B,C,D) ←  B=B1-1,p5(A,B1,C,D).
p5(A,B,C,D) ←  B>0,p4(A,B,C,D).
p6(A,B,C,D) ←  B=<0,p4(A,B,C,D).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

Example

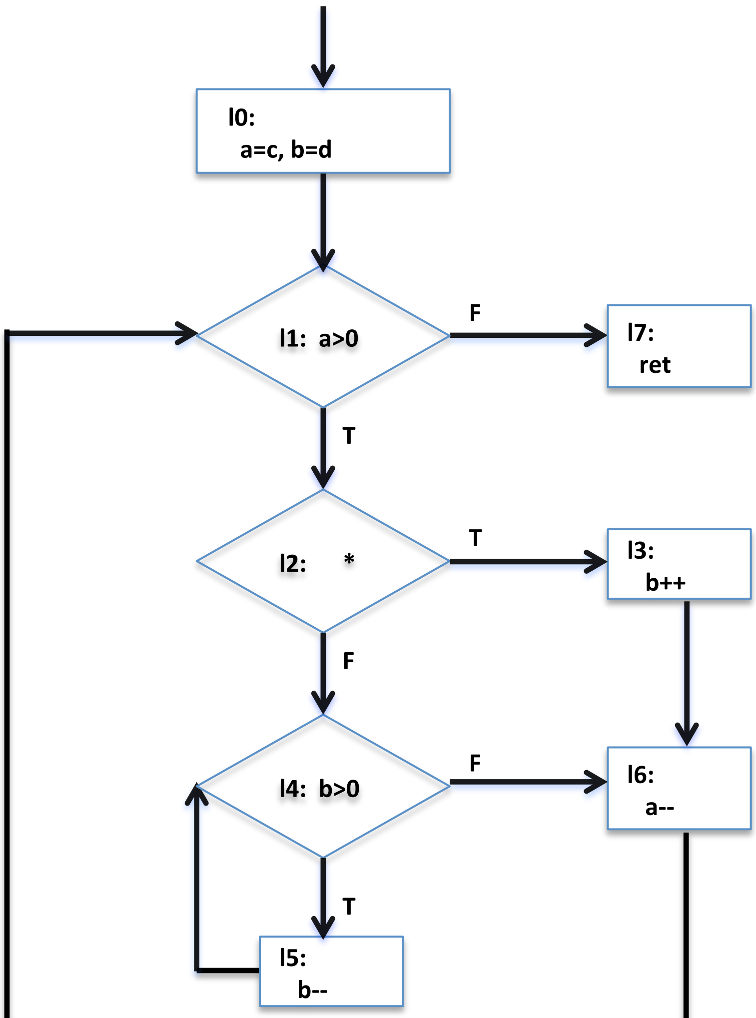
```
main( uint c,  uint d){
    int a=c,b=d;
    while (a>0){
     if (*)
        b++;
       else
        while (b>0)
            b--;
       a--; }
     return 0;
 }
```

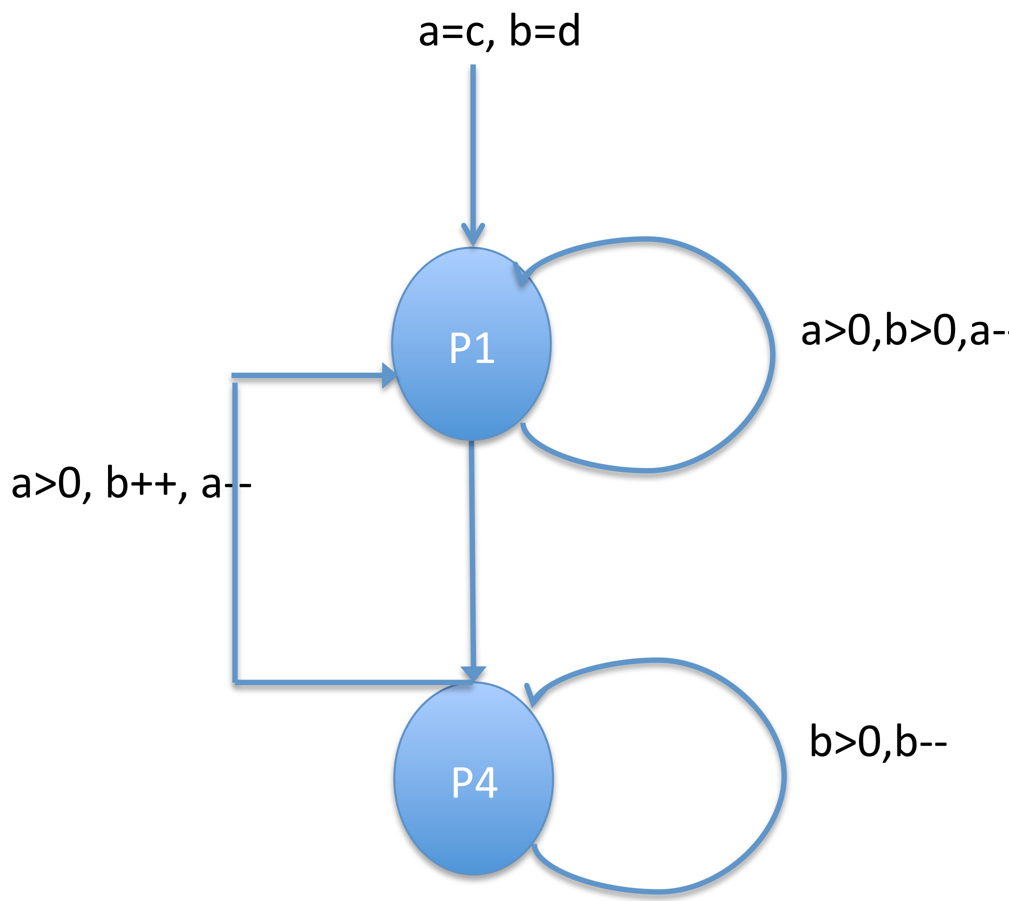
- Outer loop: can be executed at most c times
- The counter for the inner loop can be incremented by outer at most c times
- So inner loop can be executed at most c+d times

Bound=c+d

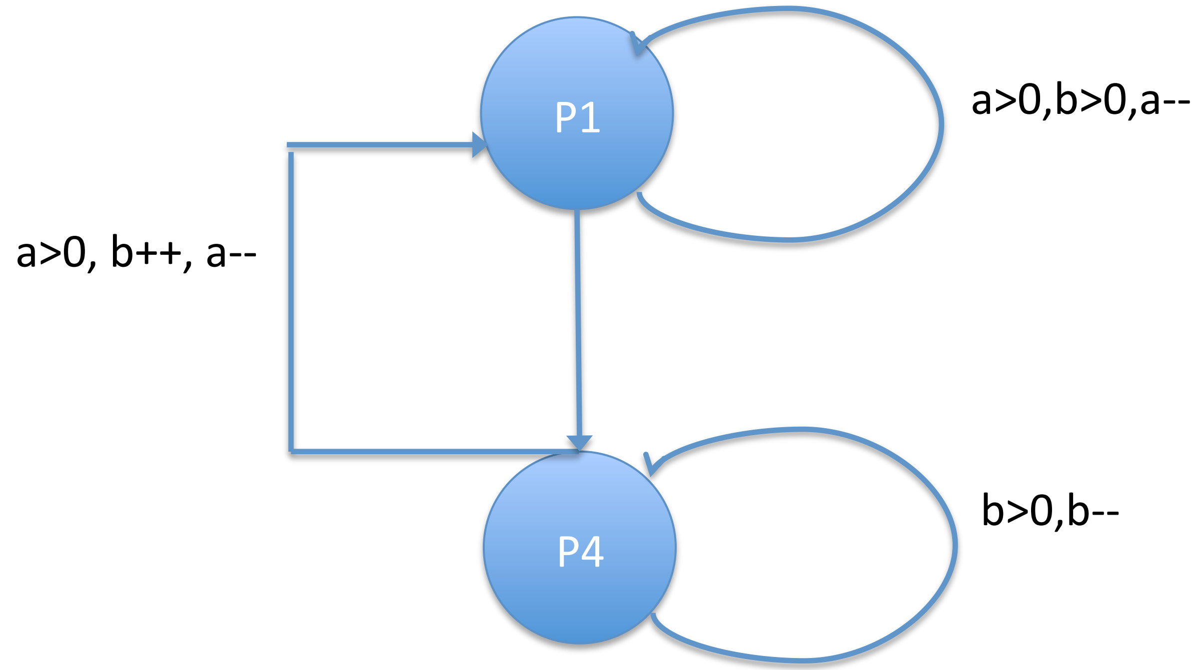
How to derive this automatically?

Single path linear constraint loops





Special form of Horn clauses (Abstraction of program)




```
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)
```

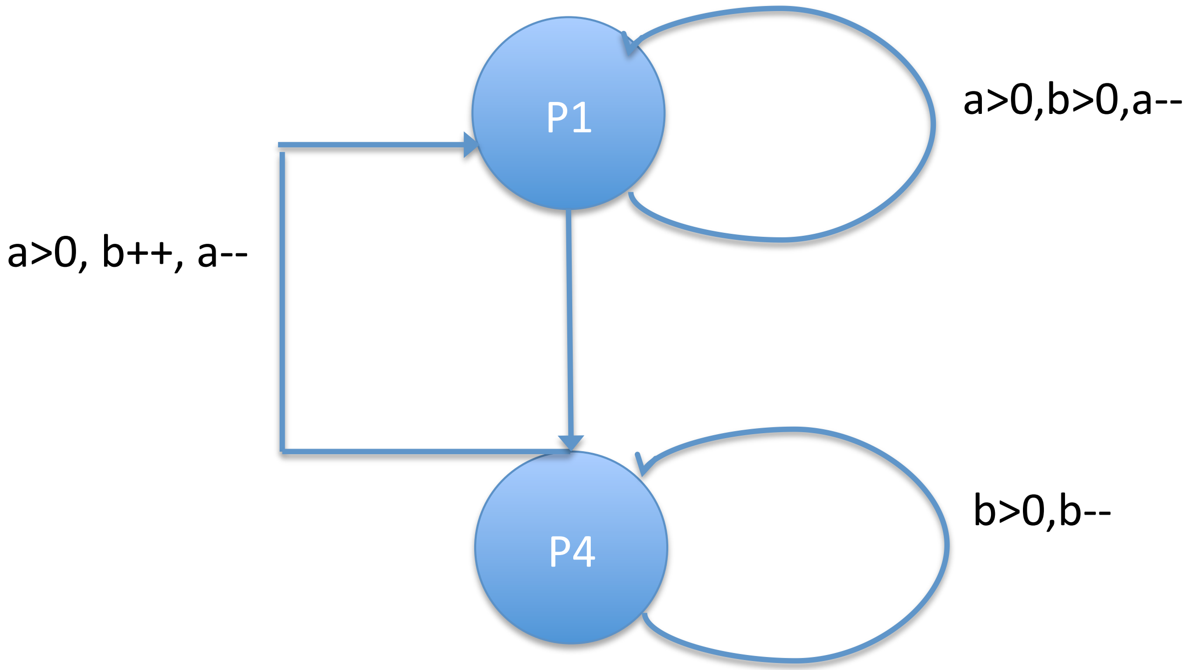
InitVal: A=c, B=d

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 = max. bound of each transition= c+d

Resource consumption



$$\sum_{i \in Loops} \text{cost\_i*bound\_i}$$

What is the role of Invariants?

- Deriving the special form of Horn clauses (with difference bound constraints)
- Derives new constraints which are useful to find updates of ranking functions
- Computing initial values of ranking functions

Conclusions and Future work

- Resource analysis of C programs using Horn clauses

In the future:

- Inter-procedural bound analysis
- Generating Recurrence equation and solving them for bound computation

Thank you!

Questions & suggestions?