Underapproximating Procedure Summaries for Recursive Integer Programs

Filip Konecny (EPFL, Lausanne)

joint work with
Radu Iosif (Verimag/CNRS, Grenoble)
Pierre Ganty (IMDEA, Madrid)

Procedure Summaries

- Relations between the values of the input and output variables of a given procedure
- Abstractions/specifications of the behavior of a functional unit in the program
- Computing summaries is useful for:
  - compositional verification
  - equivalence checking
  - termination proofs
  - abstraction refinement

Computing Summaries

- Expensive fixed point computations
- Difficult problem in the presence of:
  - recursive calls, integer parameters, local variables, return values
- Abstract interpretation methods produce conservative over-approximations
- Precise acceleration methods compute under-approximations, but usually do not cope with recursive calls

Underapproximating Summaries

- For a recursive program P, we compute an increasing sequence of under-approximations:

$$[[P]]^{(1)} \subseteq [[P]]^{(2)} \subseteq \ldots \subseteq [[P]]$$

- Each $[[P]]^{(i)}$ captures executions of unbounded length and stack usage
- Each $[[P]]^{(i)}$ can be computed by looking at a non-recursive program

Example

```
int P(int x) {

  assume(x >= 0);

  if (x > 0) {
     x = P(x-1);
     x = x+2;
  } else {
     x = x+10;
  }

  return x;
}
```

Example

```
int P(int x) {

  assume(x >= 0);

  if (x > 0) {
    x = P(x-1);
    x = x+2;
  } else {
    x = x+10;
  }

  return x;
}
```

x = n

Example

```
int P(int x) {

  assume(x >= 0);

  if (x > 0) {
    x = P(x-1);
    x = x+2;
  } else {
    x = x+10;
  }

  return x;
}
```

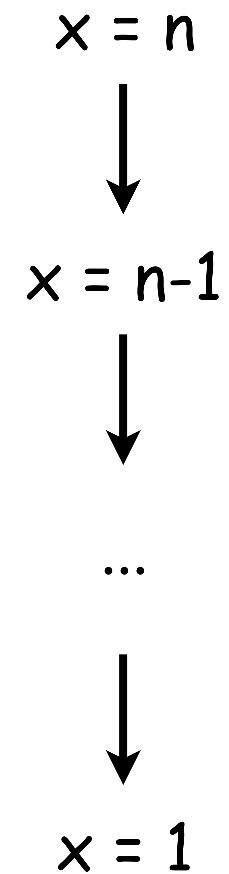
x = n

↓

x = n-1

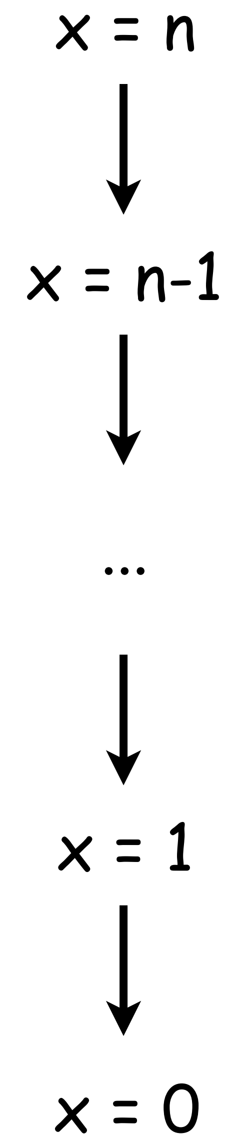
Example

```
int P(int x) {

  assume(x >= 0);

  if (x > 0) {
    x = P(x-1);
    x = x+2;
  } else {
    x = x+10;
  }

  return x;
}
```



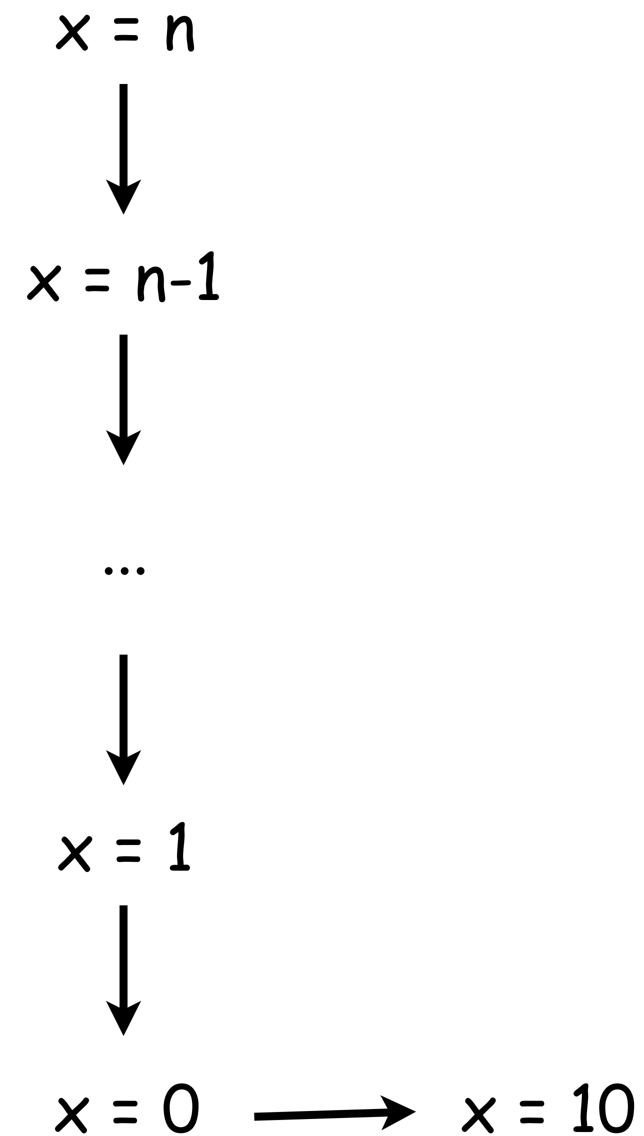
Example

```
int P(int x) {

  assume(x >= 0);

  if (x > 0) {
    x = P(x-1);
    x = x+2;
  } else {
    x = x+10;
  }

  return x;
}
```



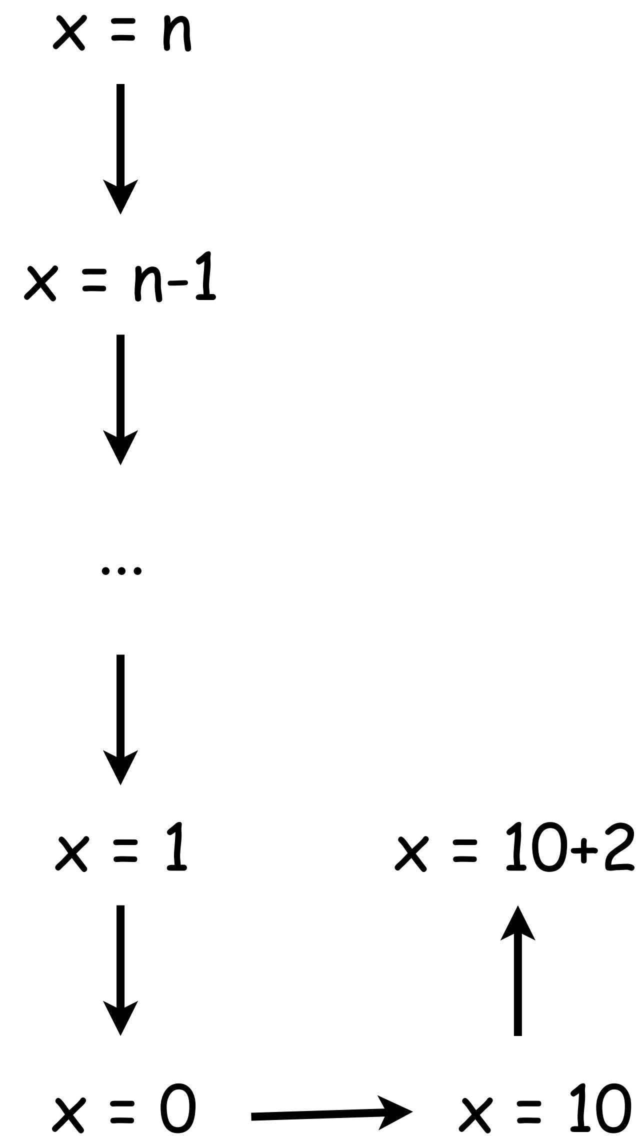
Example

```
int P(int x) {

  assume(x >= 0);

  if (x > 0) {
    x = P(x-1);
    x = x+2;
  } else {
    x = x+10;
  }

  return x;
}
```



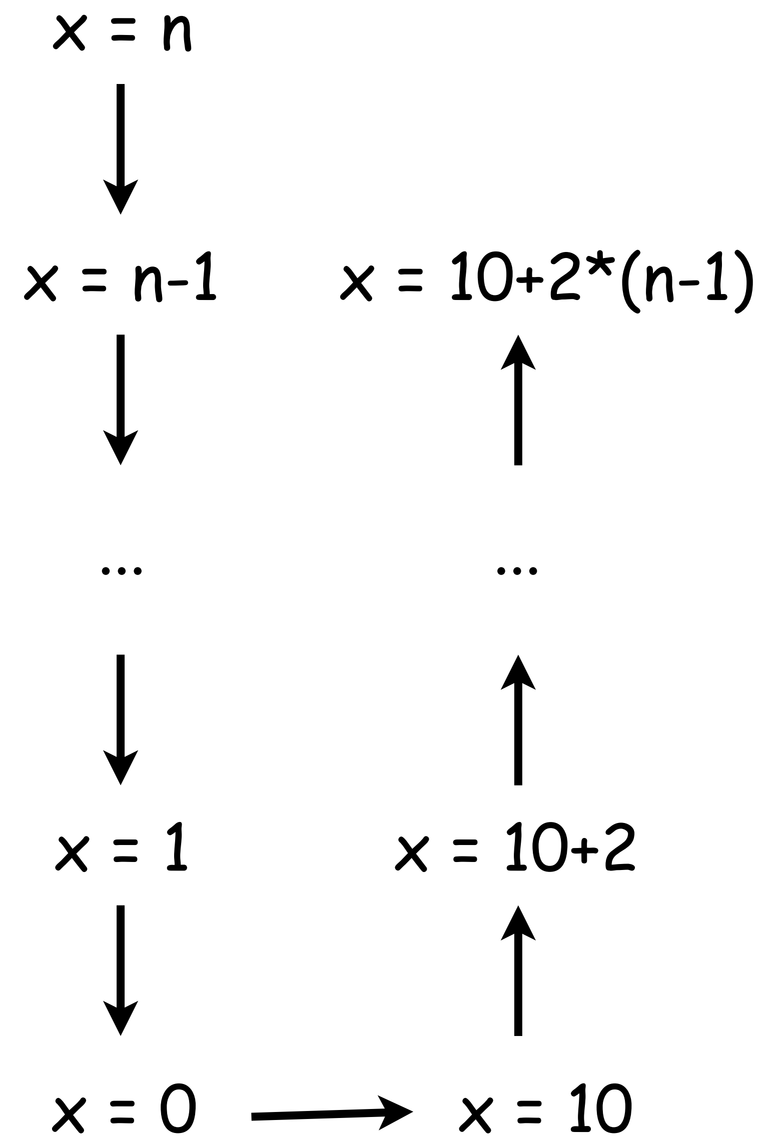
Example

```
int P(int x) {

  assume(x >= 0);

  if (x > 0) {
    x = P(x-1);
    x = x+2;
  } else {
    x = x+10;
  }

  return x;
}
```



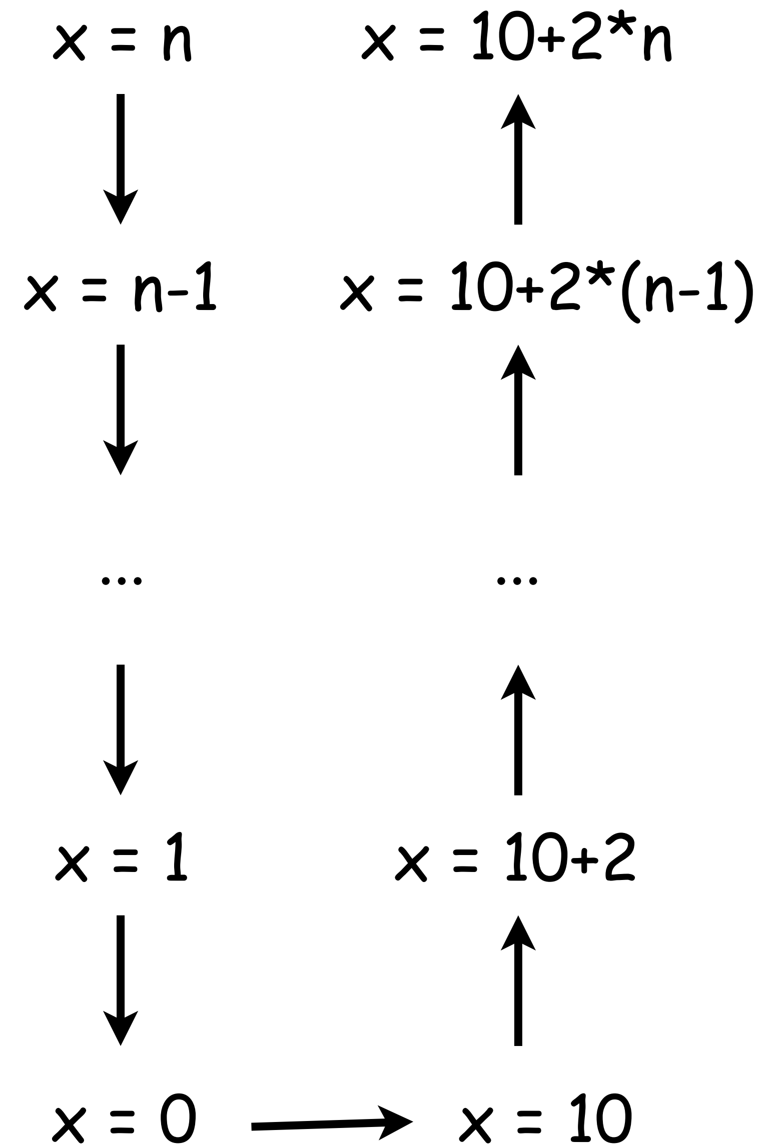
Example

```
int P(int x) {

  assume(x >= 0);

  if (x > 0) {
    x = P(x-1);
    x = x+2;
  } else {
    x = x+10;
  }

  return x;
}
```



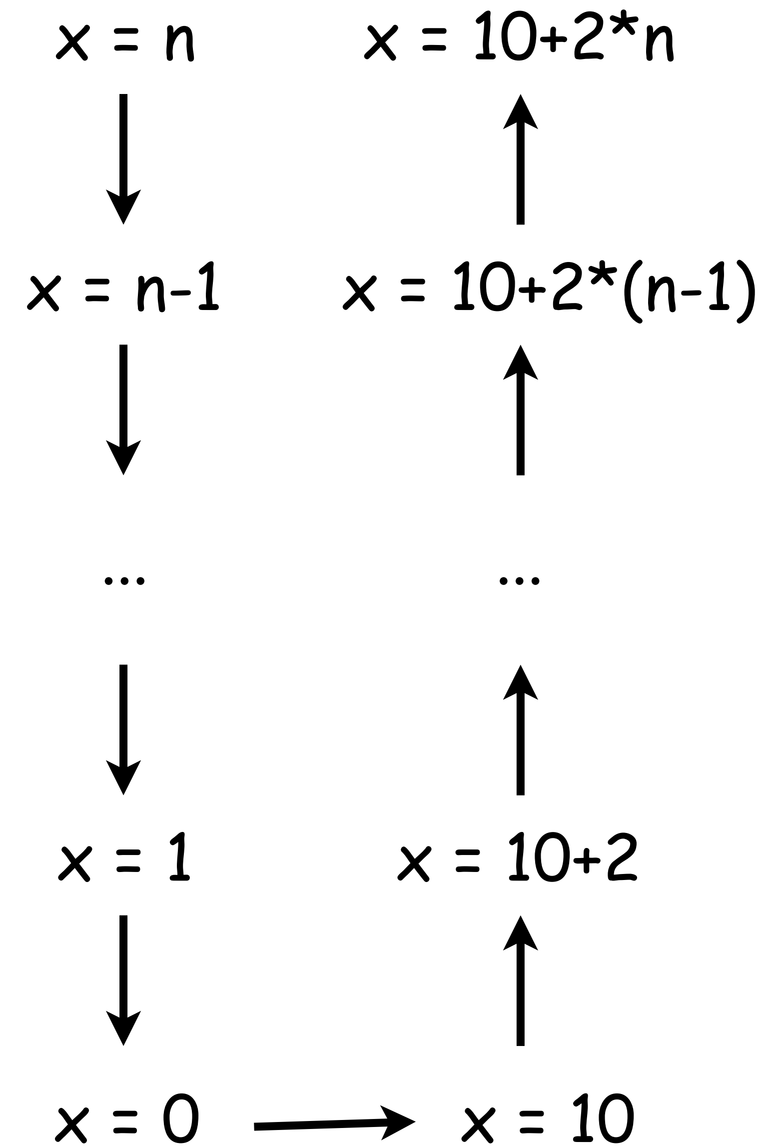
Example

```
int P(int x) {

  assume(x >= 0);

  if (x > 0) {
    x = P(x-1);
    x = x+2;
  } else {
    x = x+10;
  }

  return x;
}
```



Example

```
int P(int x) {

  assume(x >= 0);

  if (x > 0) {
    x = P(x-1);
    x = x+2;
  } else {
    x = x+10;
  }

  return x;
}
```



Example

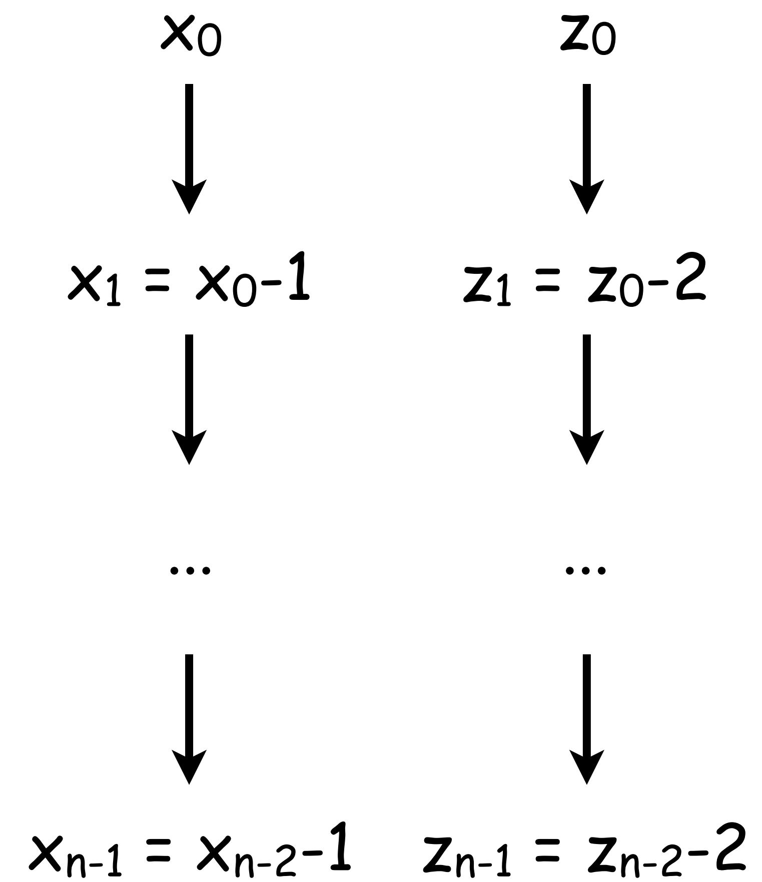
x_0 z_0

Example

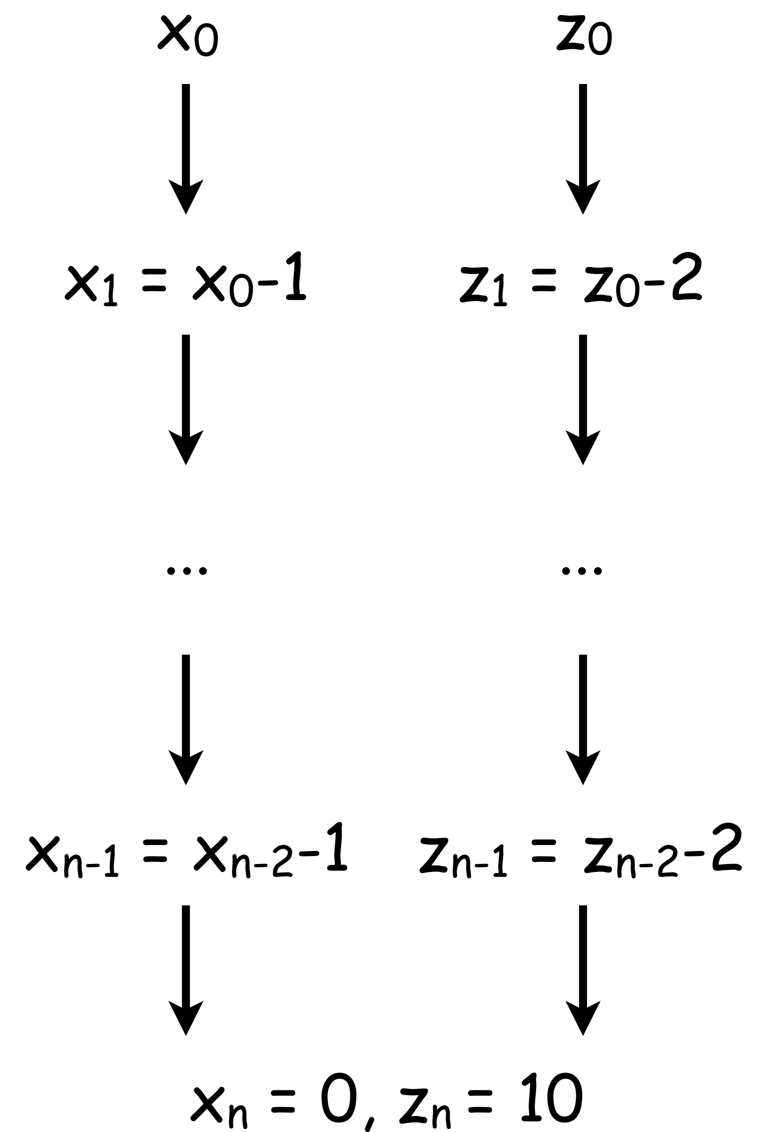
x_0 → $x_1 = x_0 - 1$

z_0 → $z_1 = z_0 - 2$

Example

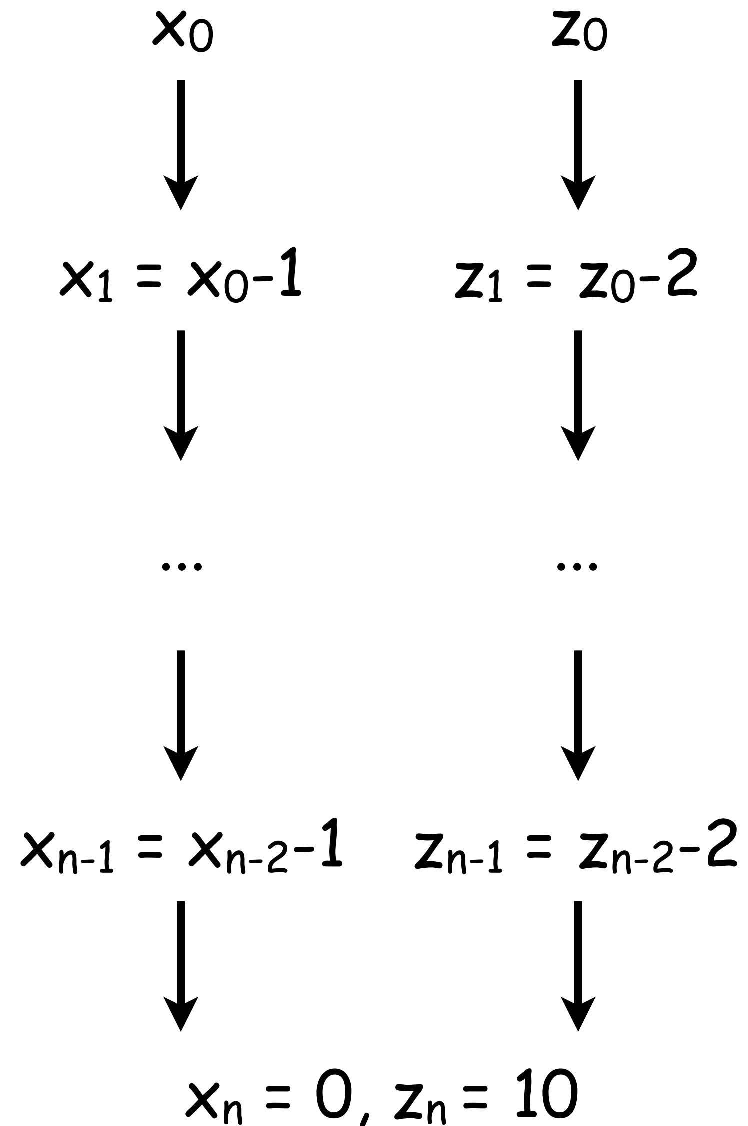


Example



Example

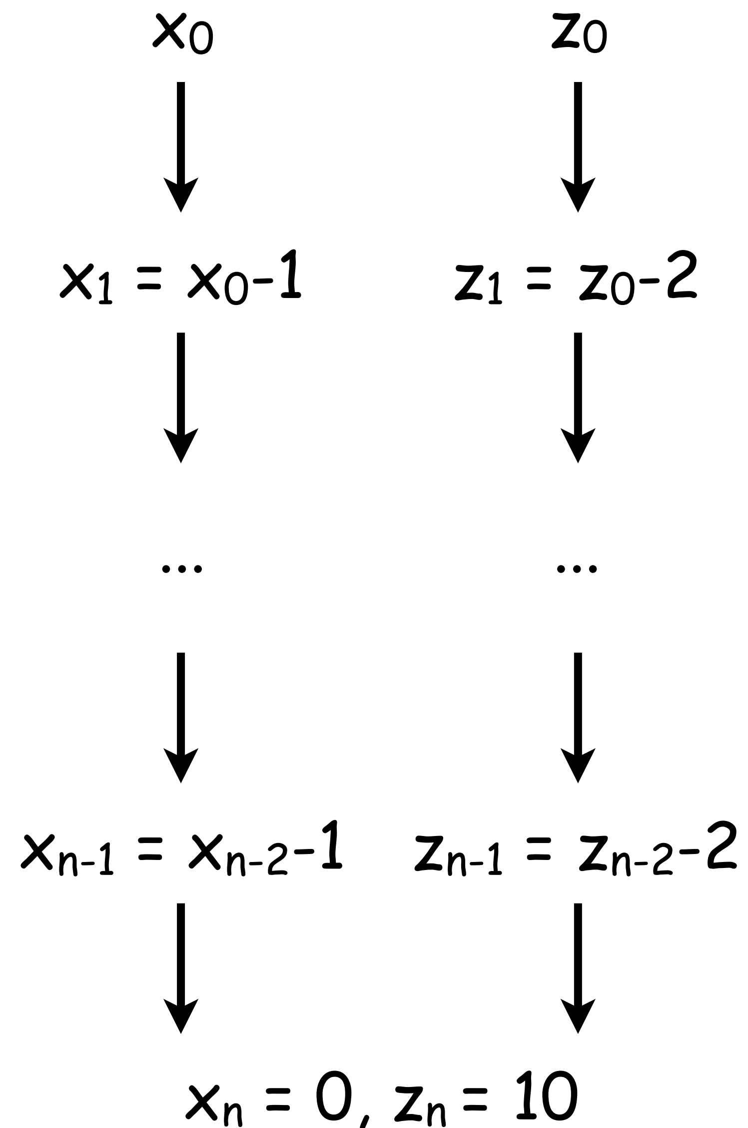
```
void Q(int x,z) {

  assume(x >= 0);

  while (x > 0) {
    x = x-1;
    z = z-2;
  }

  assume(z=x+10);
}
```



Example

@requires: z=10+2*x

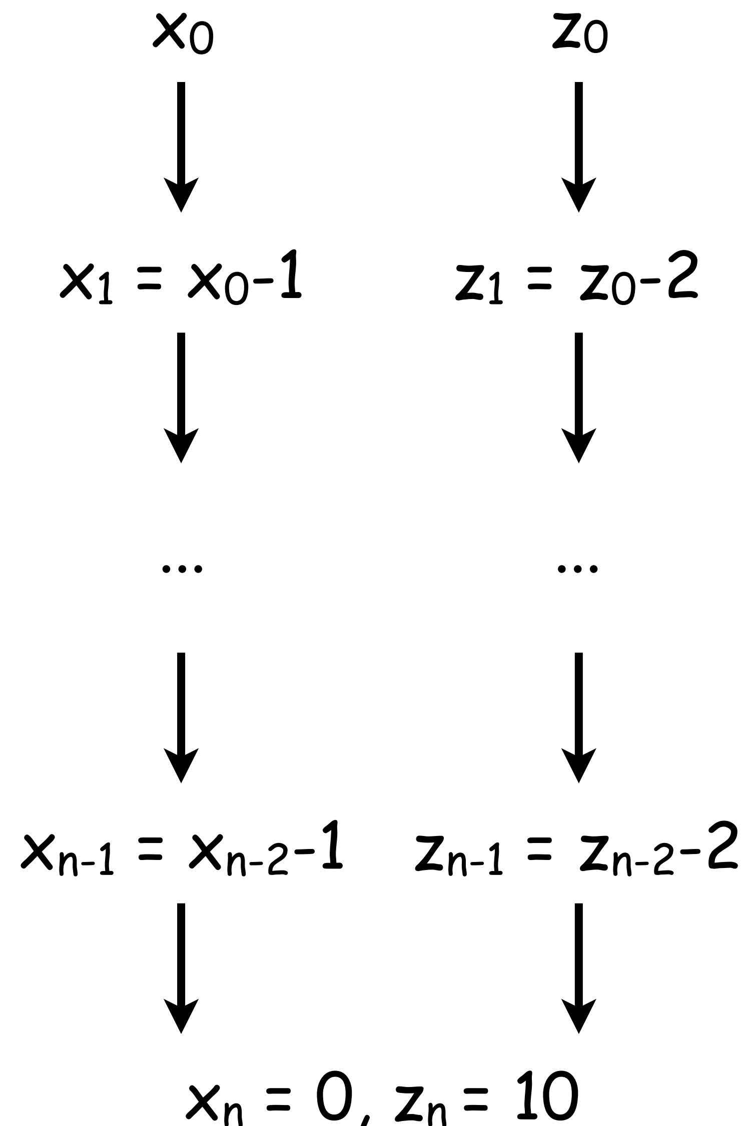
```
void Q(int x,z) {

  assume(x >= 0);

  while (x > 0) {
    x = x-1;
    z = z-2;
  }

  assume(z=x+10);
}
```



Example

@requires: z=10+2*x

```
void Q(int x,z) {

  assume(x >= 0);

  while (x > 0) {
    x = x-1;
    z = z-2;
  }

  assume(z=x+10);
}
```

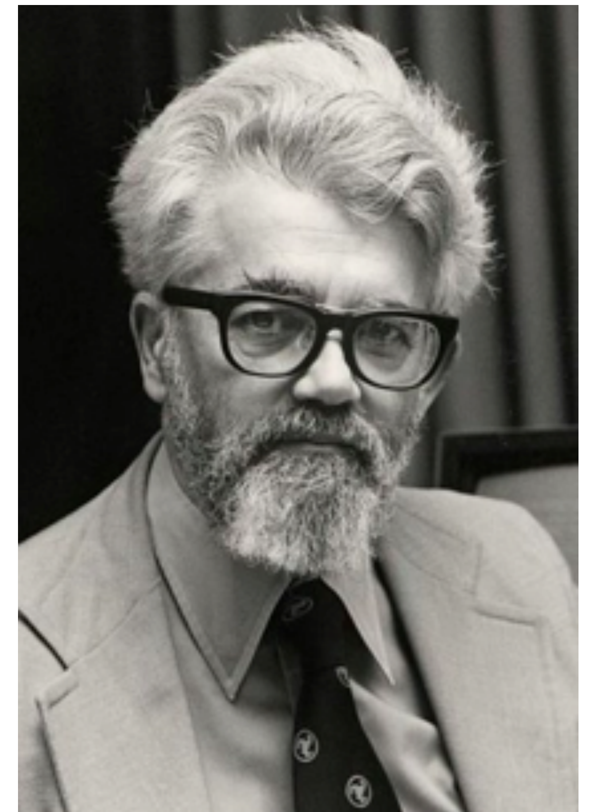


The summary of P is the precondition of Q

Not so fast ...

- Programs may have more complicated recursion schemes for which this simple idea does not apply

```
int mc91(int x) {
    if (x >= 101)
        return x-10
    else
        return mc91(mc91(x+11))
}
```

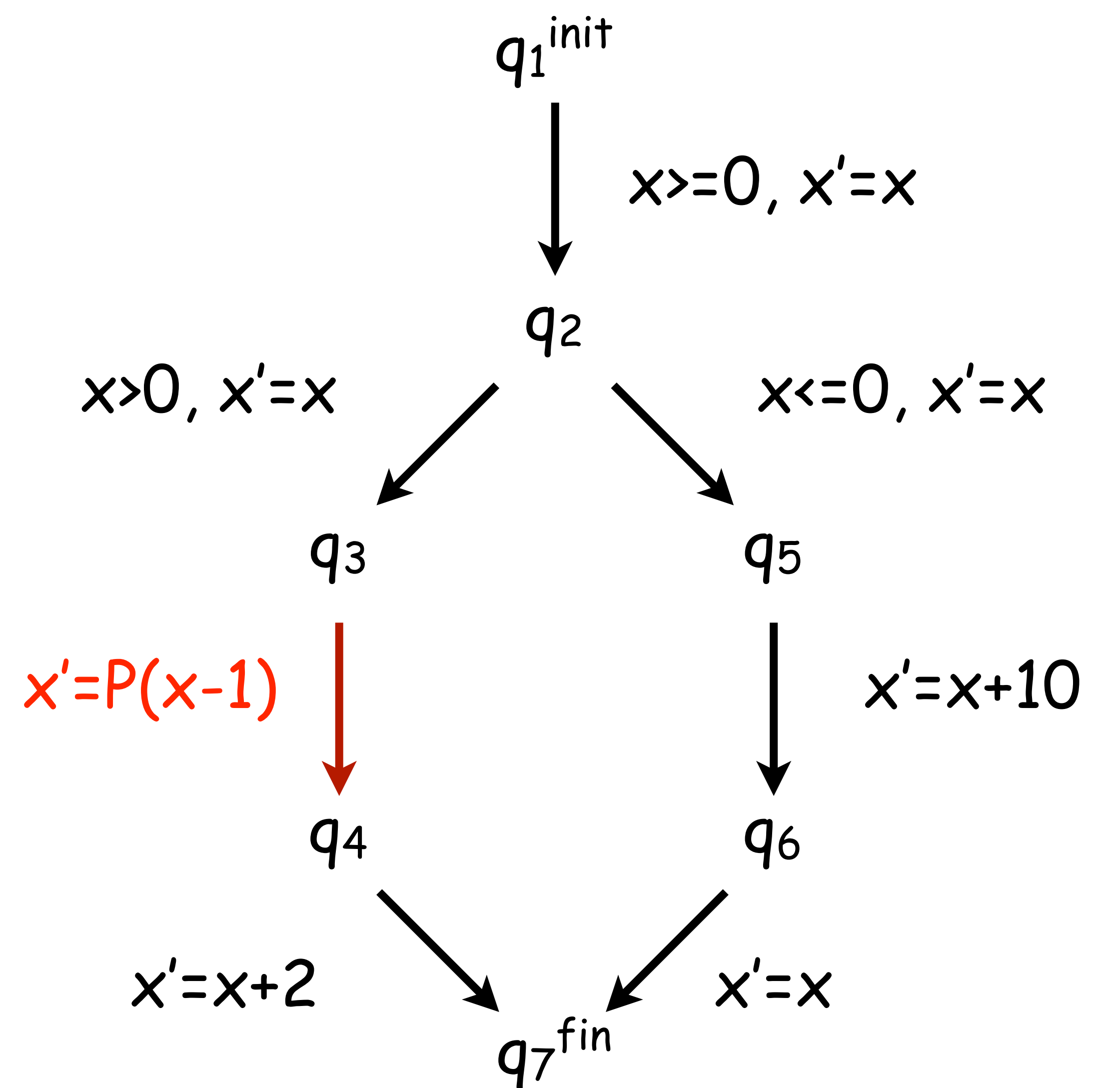


John McCarthy
1927-2011

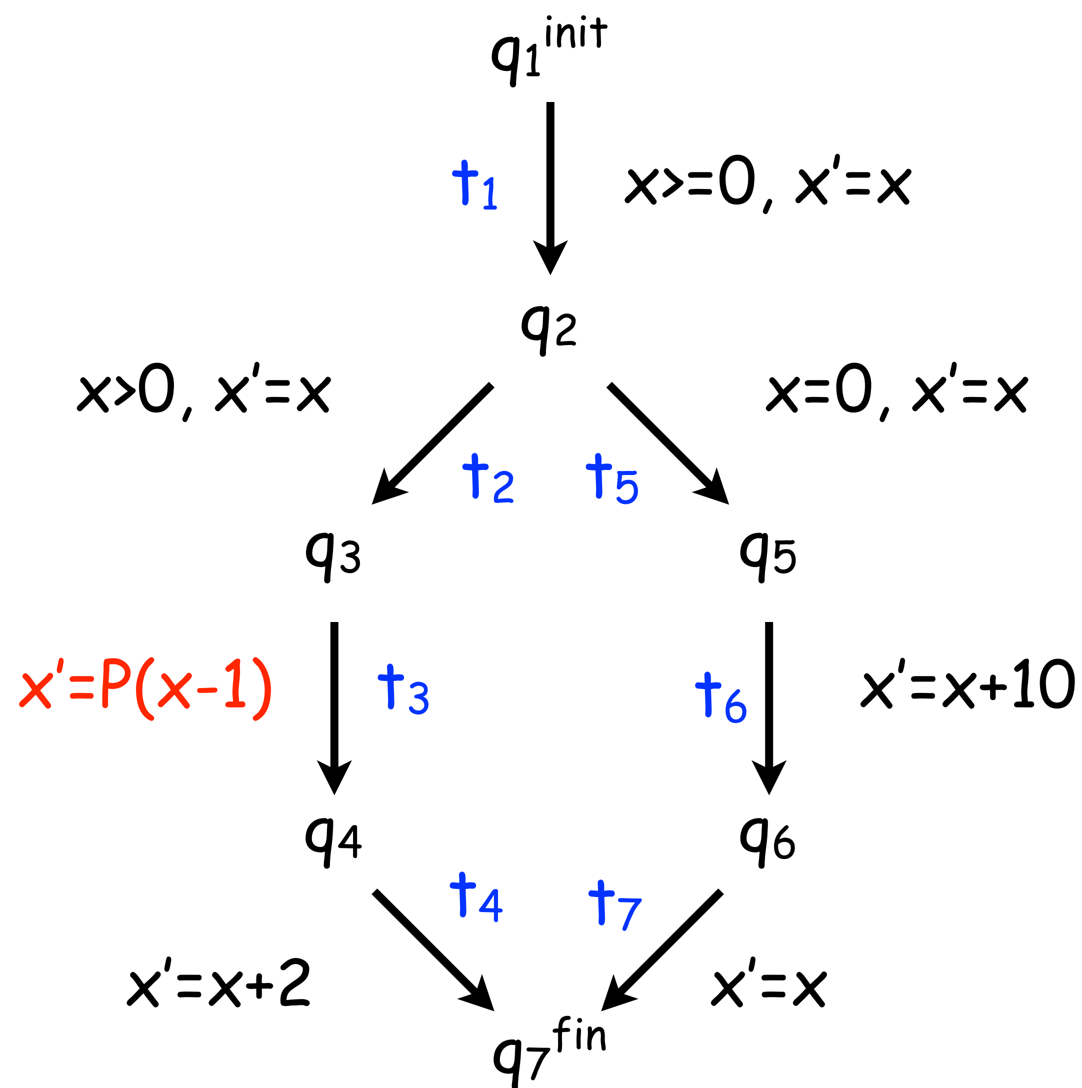
Preliminaries

Recursive Programs as CFGs

```
int P(int x) {

  assume(x >= 0);

  if (x > 0) {
    x = P(x-1);
    x = x+2;
  } else {
    x = x+10;
  }

  return x;
}
```



Recursive Programs as VPGs



$Q_1^{init} \to t_1 Q_2$

$Q_2 \to t_2 Q_3$

$Q_2 \to t_5 Q_5$

$Q_3 \to \langle t_3 \, Q_1^{init} \, t_3 \rangle \, Q_4$

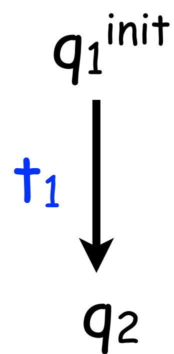
$Q_4 \to t_4$

$Q_5 \to t_6 Q_6$

$Q_6 \to t_7$

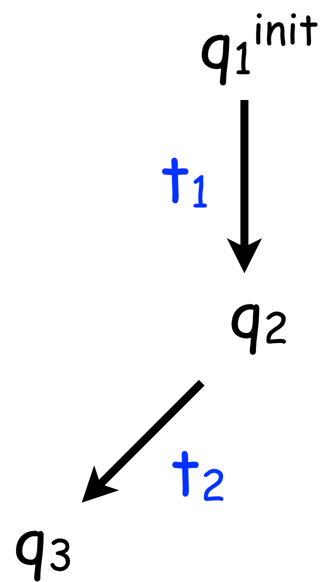
Executions as Derivations

Executions as Derivations



$Q_1^{init} \Rightarrow t_1 Q_2$

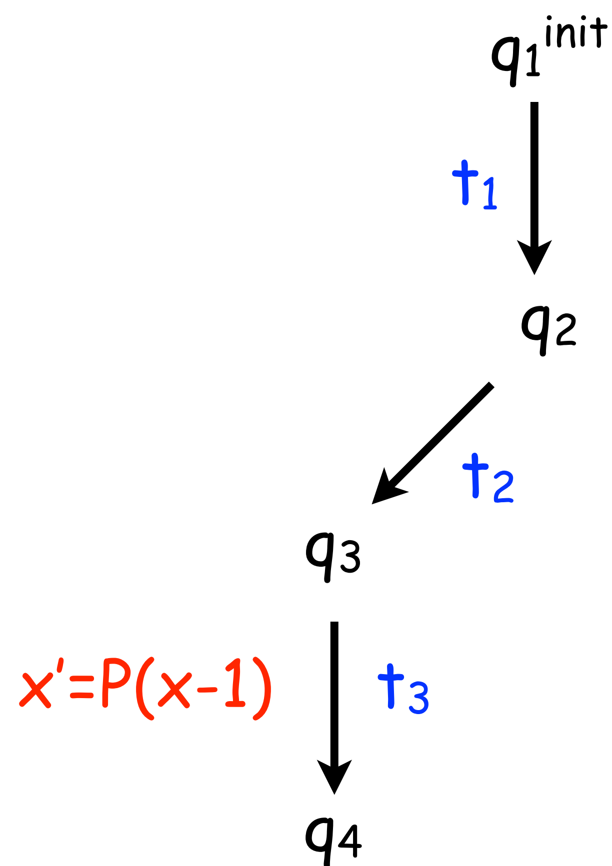
Executions as Derivations



Q_1^{init} => $t_1\ Q_2$

=> $t_1\ t_2\ Q_3$

Executions as Derivations

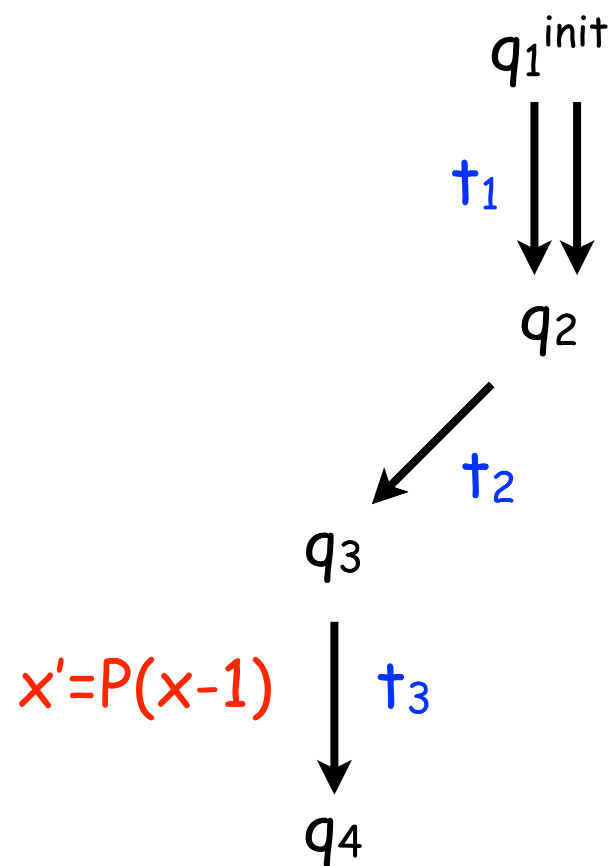


$Q_1^{init} \Rightarrow t_1 \, Q_2$

$\Rightarrow t_1 \, t_2 \, Q_3$

$\Rightarrow t_1 \, t_2 \, \langle t_3 \, Q_1^{init} \, t_3 \rangle \, Q_4$

Executions as Derivations



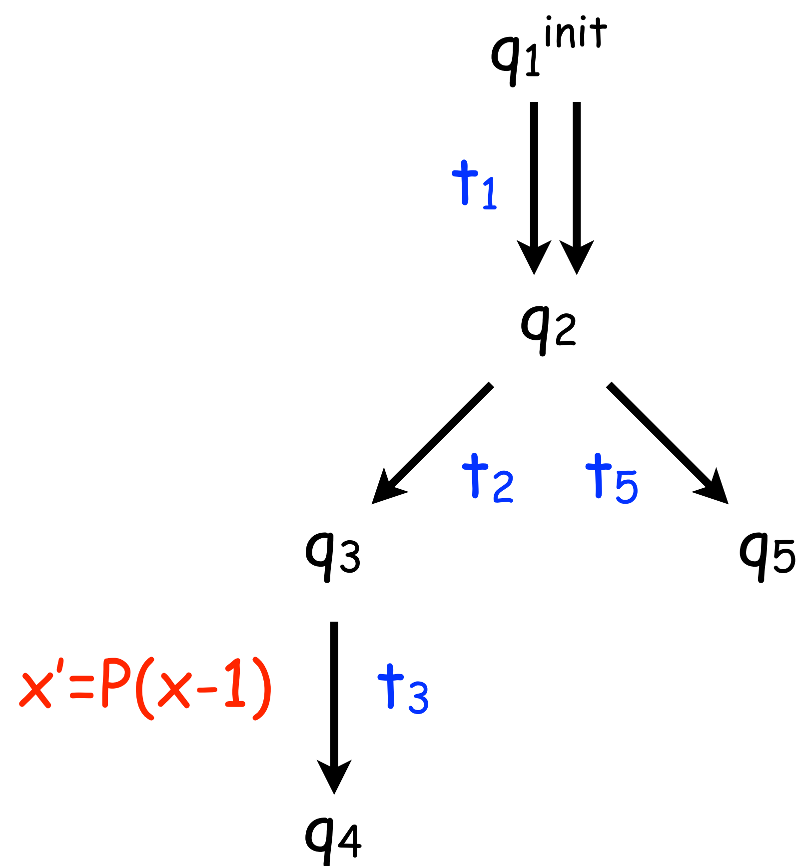
Q_1^{init} => t_1 Q_2

=> t_1 t_2 Q_3

=> t_1 t_2 <t_3 Q_1^{init} t_3> Q_4

=> t_1 t_2 <t_3 t_1 Q_2 t_3> Q_4

Executions as Derivations



Q_1^{init} => $t_1\ Q_2$

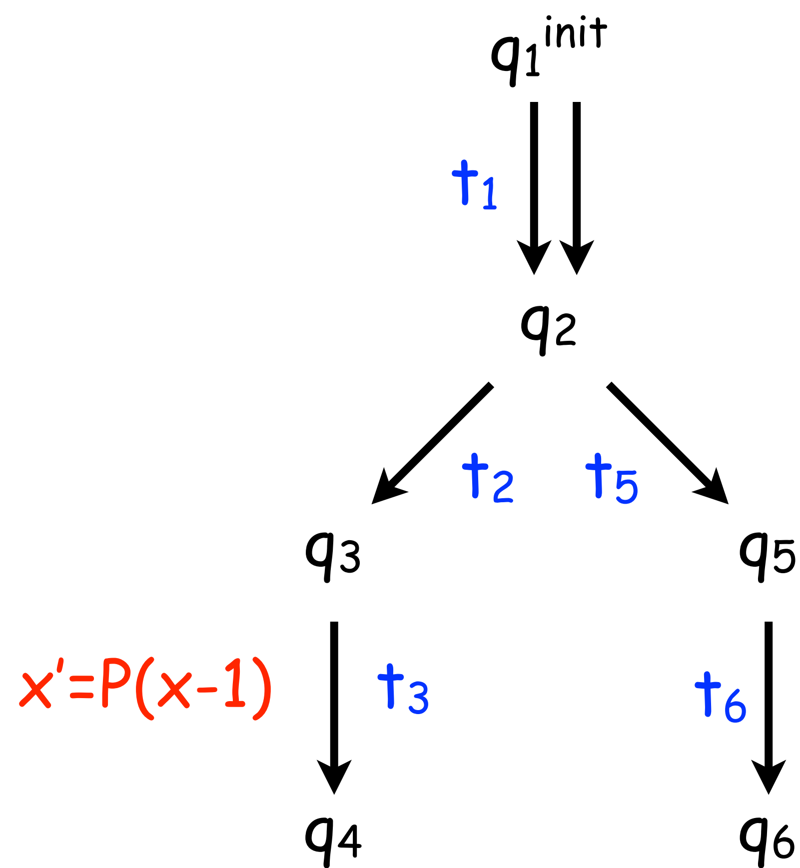
=> $t_1\ t_2\ Q_3$

=> $t_1\ t_2$ <$t_3\ Q_1^{init}\ t_3$> Q_4

=> $t_1\ t_2$ <$t_3\ t_1\ Q_2\ t_3$> Q_4

=> $t_1\ t_2$ <$t_3\ t_1\ t_5\ Q_5\ t_3$> Q_4

Executions as Derivations



Q_1^{init} => $t_1\ Q_2$

=> $t_1\ t_2\ Q_3$

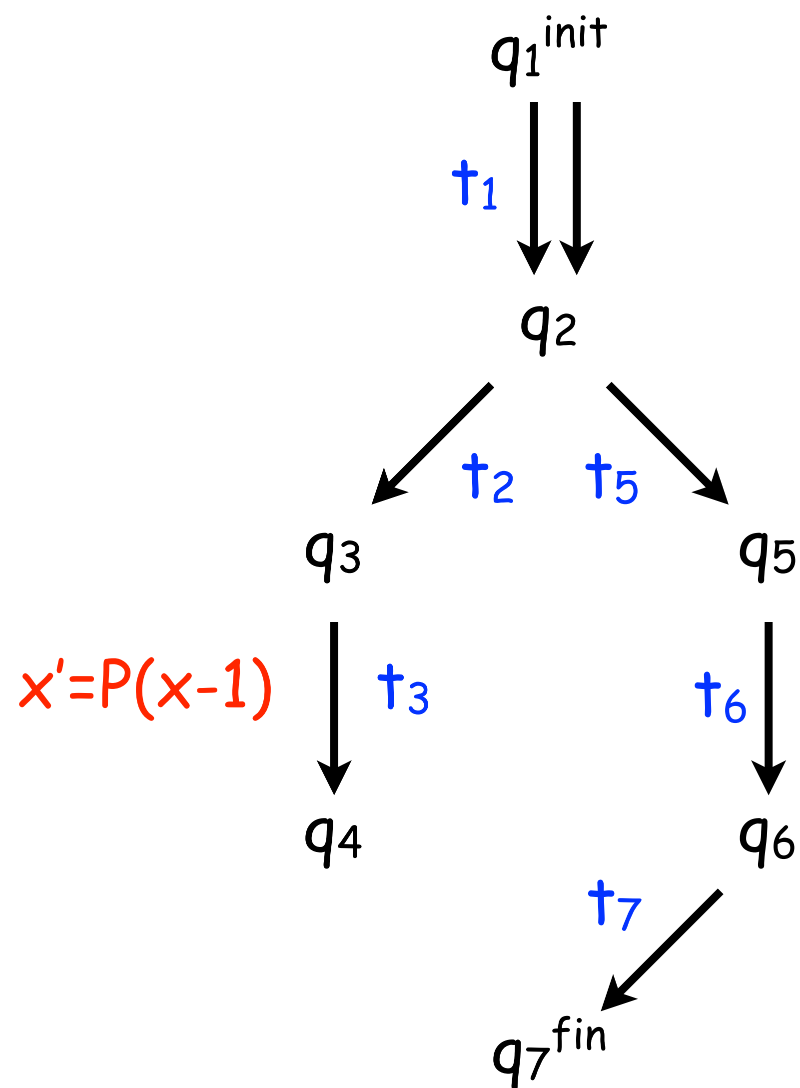
=> $t_1\ t_2 < t_3\ Q_1^{init}\ t_3 >\ Q_4$

=> $t_1\ t_2 < t_3\ t_1\ Q_2\ t_3 >\ Q_4$

=> $t_1\ t_2 < t_3\ t_1\ t_5\ Q_5\ t_3 >\ Q_4$

=> $t_1\ t_2 < t_3\ t_1\ t_5\ t_6\ Q_6\ t_3 >\ Q_4$

Executions as Derivations



$Q_1^{init} \Rightarrow t_1\ Q_2$

$\Rightarrow t_1\ t_2\ Q_3$

$\Rightarrow t_1\ t_2\ \langle t_3\ Q_1^{init}\ t_3 \rangle\ Q_4$

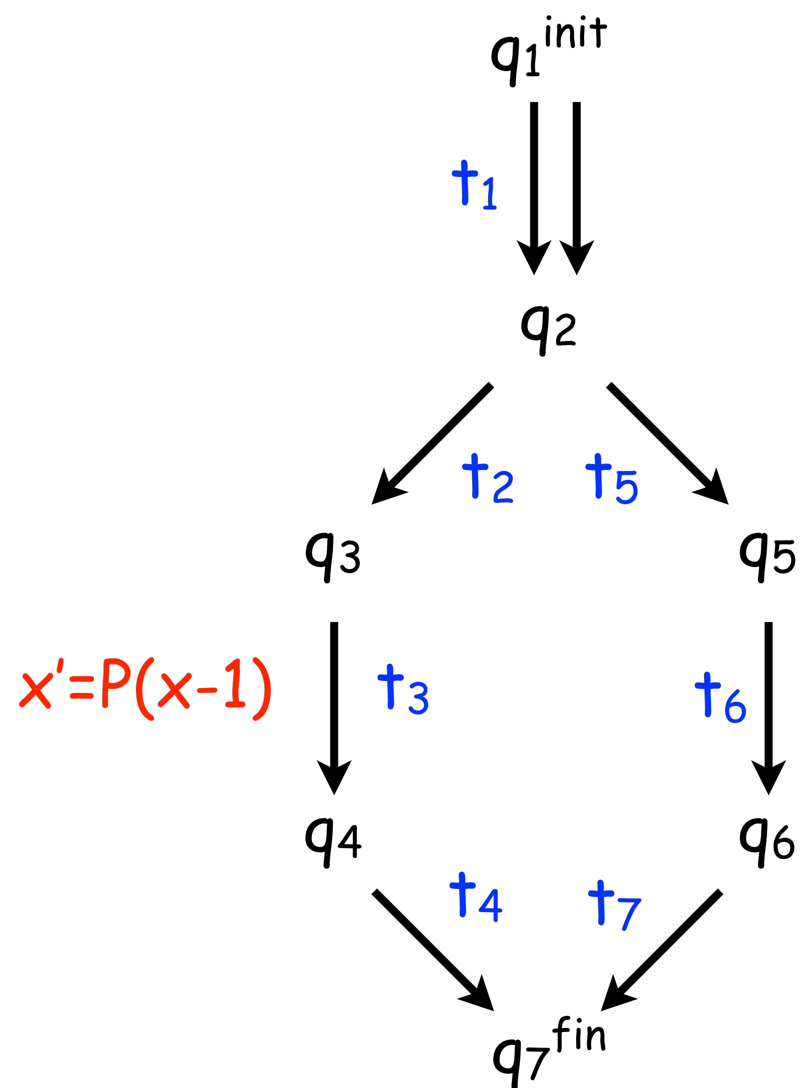
$\Rightarrow t_1\ t_2\ \langle t_3\ t_1\ Q_2\ t_3 \rangle\ Q_4$

$\Rightarrow t_1\ t_2\ \langle t_3\ t_1\ t_5\ Q_5\ t_3 \rangle\ Q_4$

$\Rightarrow t_1\ t_2\ \langle t_3\ t_1\ t_5\ t_6\ Q_6\ t_3 \rangle\ Q_4$

$\Rightarrow t_1\ t_2\ \langle t_3\ t_1\ t_5\ t_6\ t_7\ t_3 \rangle\ Q_4$

Executions as Derivations



Q_1^{init} => $t_1\ Q_2$

=> $t_1\ t_2\ Q_3$

=> $t_1\ t_2 < t_3\ Q_1^{init}\ t_3 >\ Q_4$

=> $t_1\ t_2 < t_3\ t_1\ Q_2\ t_3 >\ Q_4$

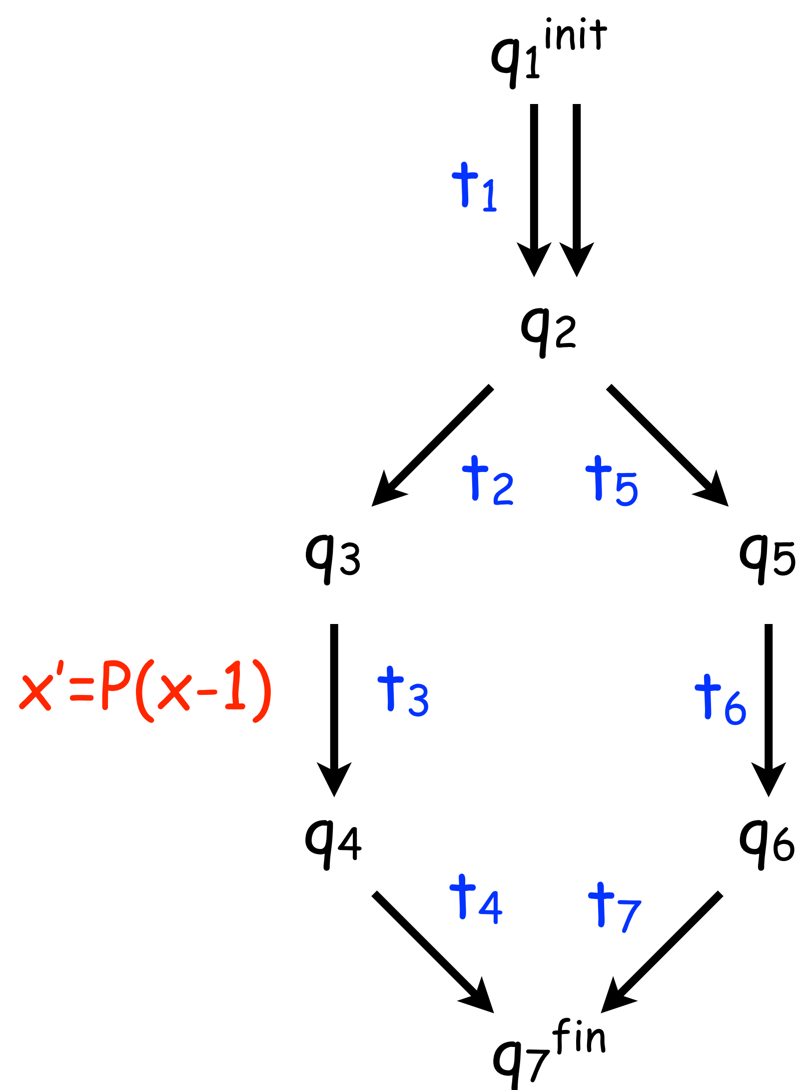
=> $t_1\ t_2 < t_3\ t_1\ t_5\ Q_5\ t_3 >\ Q_4$

=> $t_1\ t_2 < t_3\ t_1\ t_5\ t_6\ Q_6\ t_3 >\ Q_4$

=> $t_1\ t_2 < t_3\ t_1\ t_5\ t_6\ t_7\ t_3 >\ Q_4$

=> $t_1\ t_2 < t_3\ t_1\ t_5\ t_6\ t_7\ t_3 >\ t_4$

Executions as Derivations

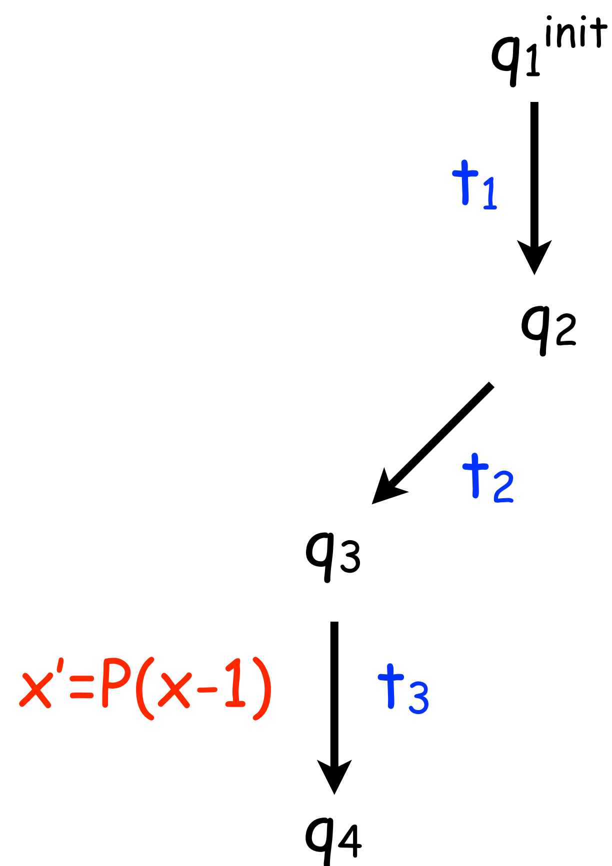


$$
\begin{aligned}
Q_1^{init} &\Rightarrow t_1\, Q_2 \\
&\Rightarrow t_1\, t_2\, Q_3 \\
&\Rightarrow t_1\, t_2 \langle t_3\, Q_1^{init}\, t_3 \rangle\, Q_4 \\
&\Rightarrow t_1\, t_2 \langle t_3\, t_1\, Q_2\, t_3 \rangle\, Q_4 \\
&\Rightarrow t_1\, t_2 \langle t_3\, t_1\, t_5\, Q_5\, t_3 \rangle\, Q_4 \\
&\Rightarrow t_1\, t_2 \langle t_3\, t_1\, t_5\, t_6\, Q_6\, t_3 \rangle\, Q_4 \\
&\Rightarrow t_1\, t_2 \langle t_3\, t_1\, t_5\, t_6\, t_7\, t_3 \rangle\, Q_4 \\
&\Rightarrow t_1\, t_2 \langle t_3\, t_1\, t_5\, t_6\, t_7\, t_3 \rangle\, t_4
\end{aligned}
$$

In order execution

Executions as Derivations

Executions as Derivations

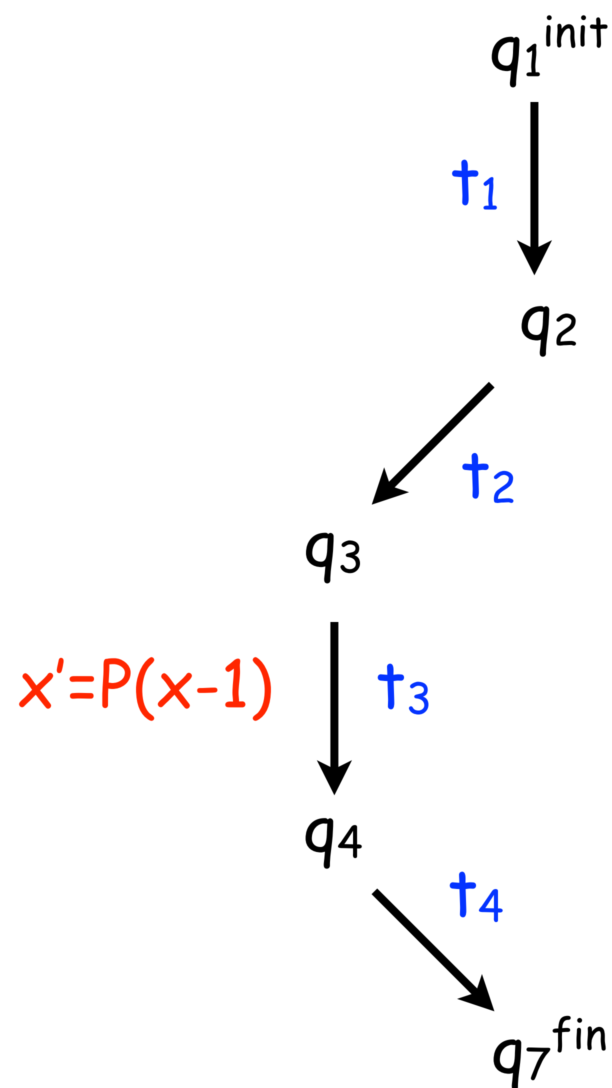


Q_1^{init} => $t_1\ Q_2$

=> $t_1\ t_2\ Q_3$

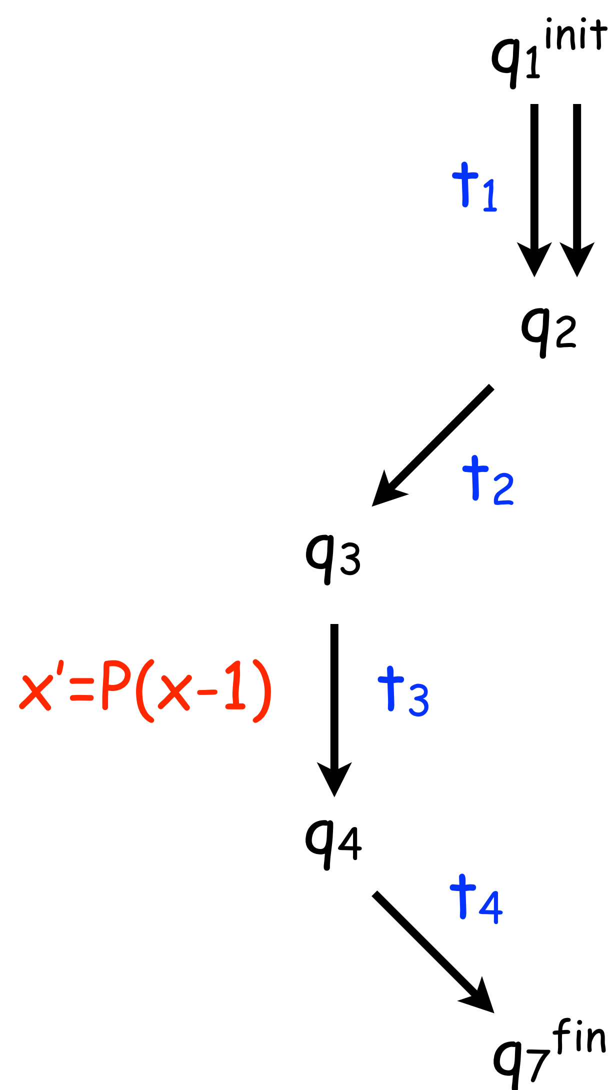
=> $t_1\ t_2 < t_3\ Q_1^{init}\ t_3 >\ Q_4$

Executions as Derivations



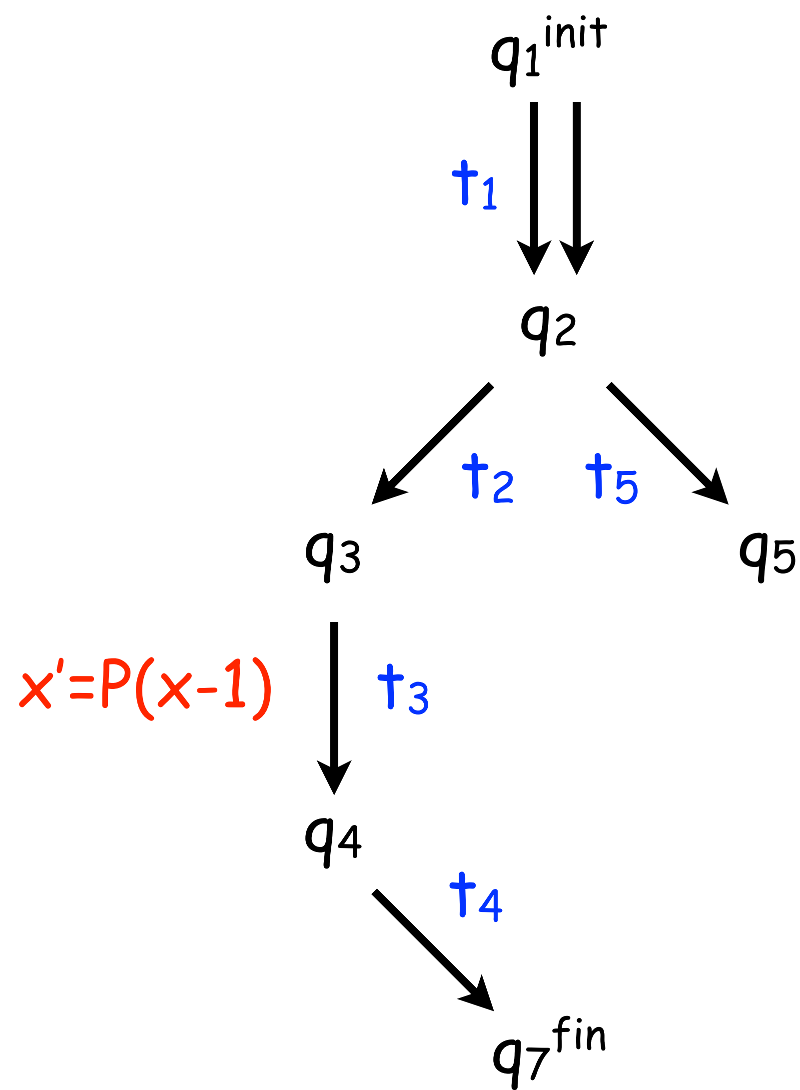
$$Q_1^{init} \Rightarrow t_1\, Q_2$$
$$\Rightarrow t_1\, t_2\, Q_3$$
$$\Rightarrow t_1\, t_2 \langle t_3\, Q_1^{init}\, t_3 \rangle\, Q_4$$
$$\Rightarrow t_1\, t_2 \langle t_3\, Q_1^{init}\, t_3 \rangle\, t_4$$

Executions as Derivations



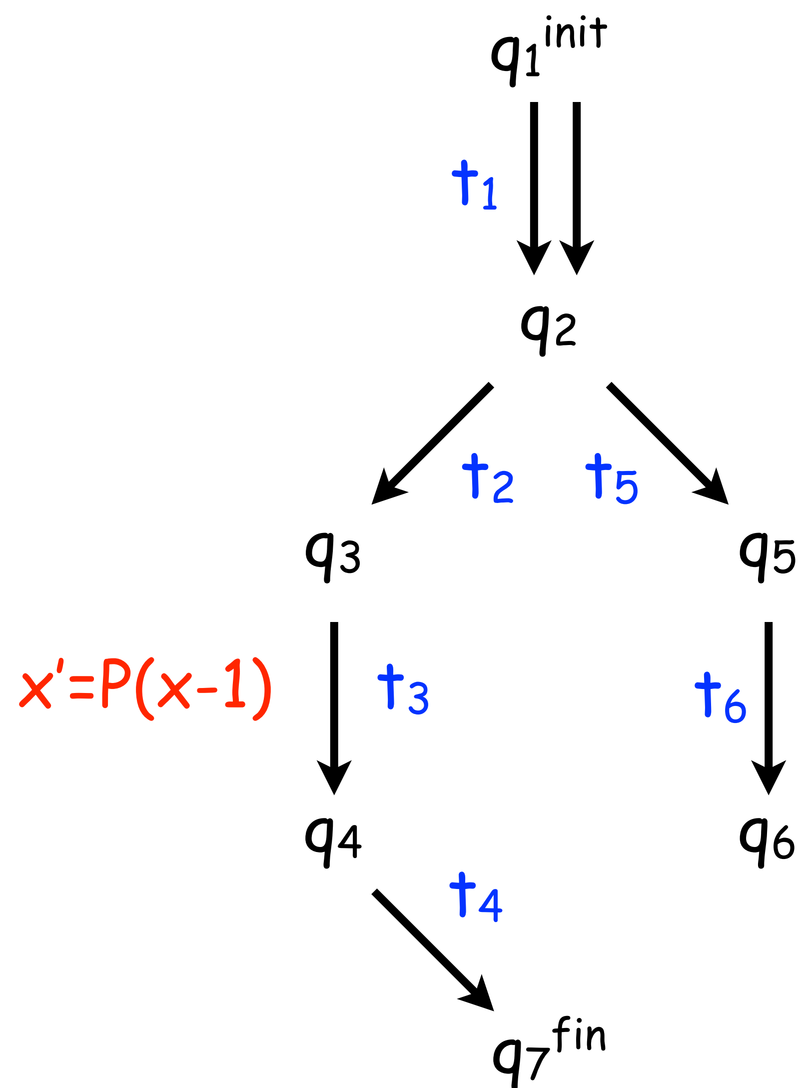
$$
\begin{aligned}
Q_1^{init} &\Rightarrow t_1\, Q_2 \\
&\Rightarrow t_1\, t_2\, Q_3 \\
&\Rightarrow t_1\, t_2 < t_3\, Q_1^{init}\, t_3 > Q_4 \\
&\Rightarrow t_1\, t_2 < t_3\, Q_1^{init}\, t_3 > t_4 \\
&\Rightarrow t_1\, t_2 < t_3\, t_1\, Q_2\, t_3 > t_4
\end{aligned}
$$

Executions as Derivations



$$Q_1^{init} \Rightarrow t_1\, Q_2$$
$$\Rightarrow t_1\, t_2\, Q_3$$
$$\Rightarrow t_1\, t_2 <t_3\, Q_1^{init}\, t_3> Q_4$$
$$\Rightarrow t_1\, t_2 <t_3\, Q_1^{init}\, t_3> t_4$$
$$\Rightarrow t_1\, t_2 <t_3\, t_1\, Q_2\, t_3> t_4$$
$$\Rightarrow t_1\, t_2 <t_3\, t_1\, t_5\, Q_5\, t_3> t_4$$

Executions as Derivations



$Q_1^{init} \Rightarrow t_1\, Q_2$

$\Rightarrow t_1\, t_2\, Q_3$

$\Rightarrow t_1\, t_2 < t_3\, Q_1^{init}\, t_3 > Q_4$

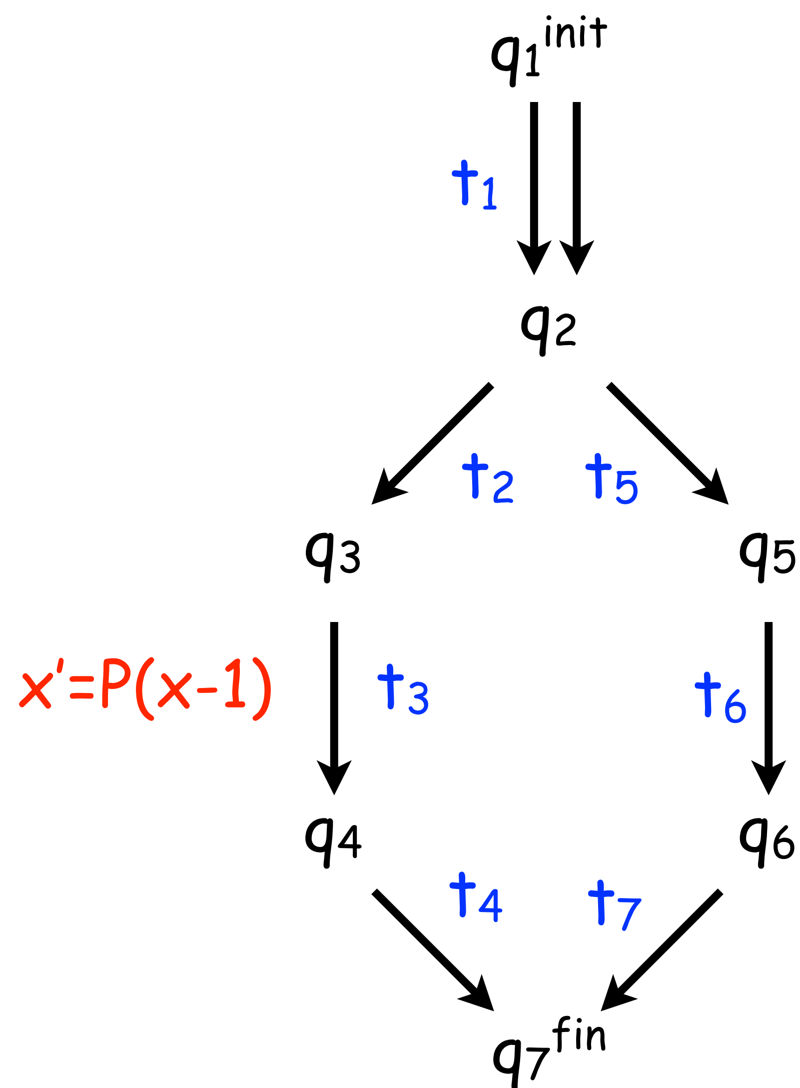
$\Rightarrow t_1\, t_2 < t_3\, Q_1^{init}\, t_3 > t_4$

$\Rightarrow t_1\, t_2 < t_3\, t_1\, Q_2\, t_3 > t_4$

$\Rightarrow t_1\, t_2 < t_3\, t_1\, t_5\, Q_5\, t_3 > t_4$

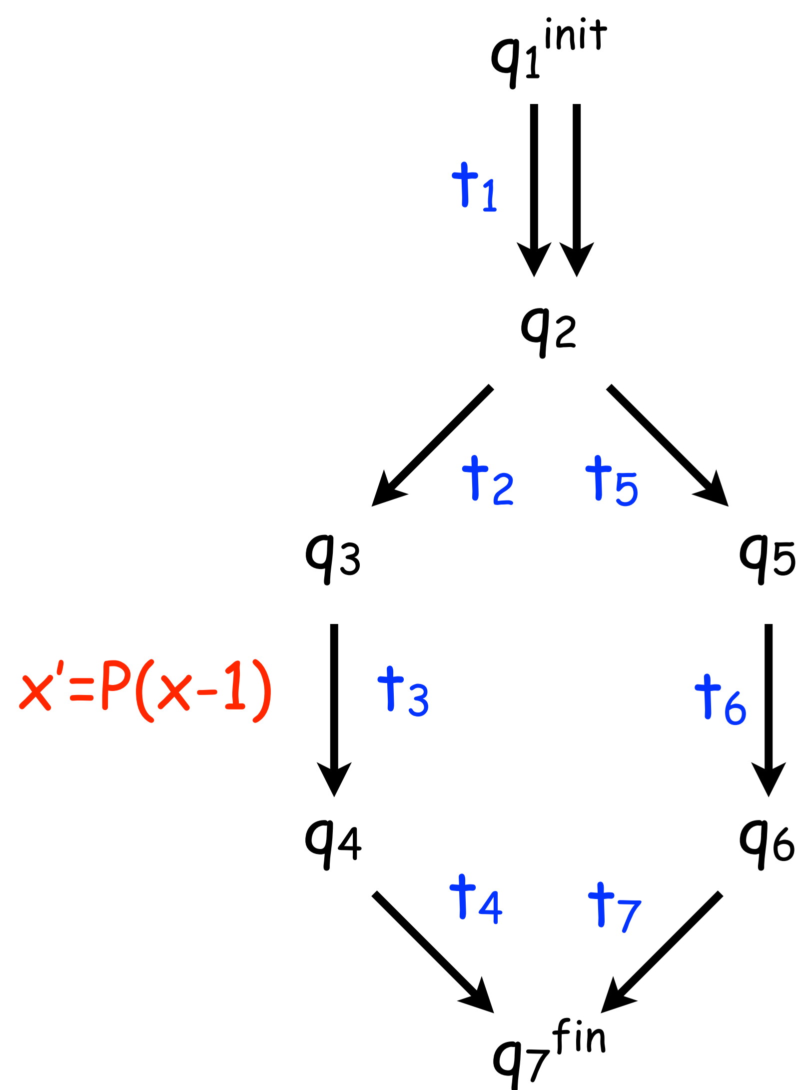
$\Rightarrow t_1\, t_2 < t_3\, t_1\, t_5\, t_6\, Q_6\, t_3 > t_4$

Executions as Derivations



$$Q_1^{init} \Rightarrow t_1\, Q_2$$
$$\Rightarrow t_1\, t_2\, Q_3$$
$$\Rightarrow t_1\, t_2 < t_3\, Q_1^{init}\, t_3 > Q_4$$
$$\Rightarrow t_1\, t_2 < t_3\, Q_1^{init}\, t_3 > t_4$$
$$\Rightarrow t_1\, t_2 < t_3\, t_1\, Q_2\, t_3 > t_4$$
$$\Rightarrow t_1\, t_2 < t_3\, t_1\, t_5\, Q_5\, t_3 > t_4$$
$$\Rightarrow t_1\, t_2 < t_3\, t_1\, t_5\, t_6\, Q_6\, t_3 > t_4$$
$$\Rightarrow t_1\, t_2 < t_3\, t_1\, t_5\, t_6\, t_7\, t_3 > t_4$$

Executions as Derivations

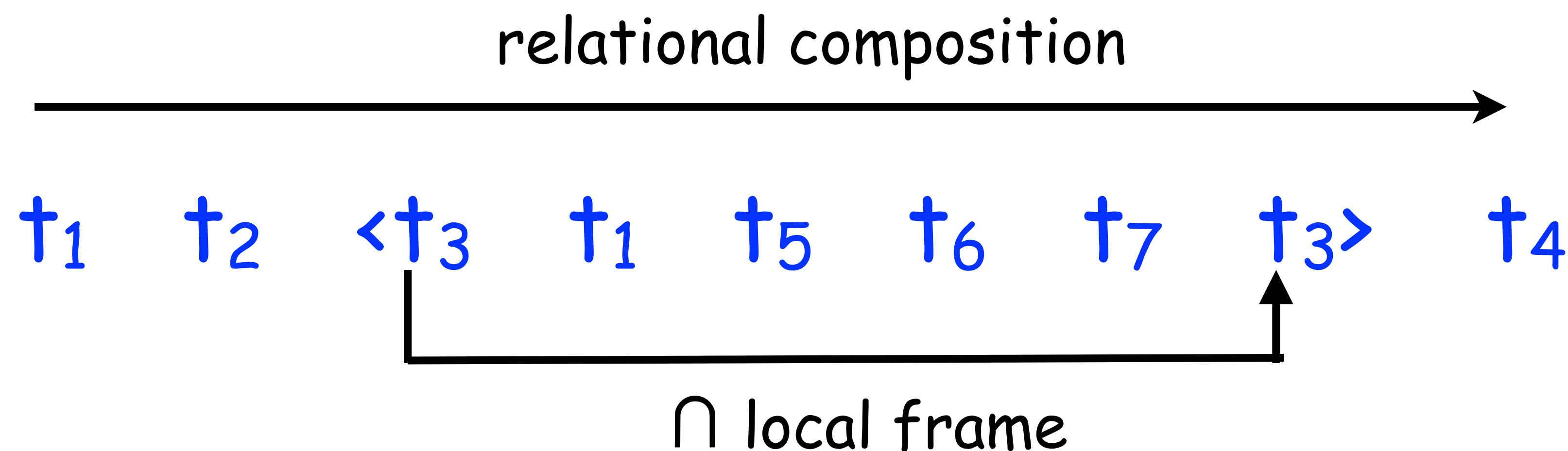


$$
\begin{aligned}
Q_1^{init} &\Rightarrow t_1\, Q_2 \\
&\Rightarrow t_1\, t_2\, Q_3 \\
&\Rightarrow t_1\, t_2\, <t_3\, Q_1^{init}\, t_3>\, Q_4 \\
&\Rightarrow t_1\, t_2\, <t_3\, Q_1^{init}\, t_3>\, t_4 \\
&\Rightarrow t_1\, t_2\, <t_3\, t_1\, Q_2\, t_3>\, t_4 \\
&\Rightarrow t_1\, t_2\, <t_3\, t_1\, t_5\, Q_5\, t_3>\, t_4 \\
&\Rightarrow t_1\, t_2\, <t_3\, t_1\, t_5\, t_6\, Q_6\, t_3>\, t_4 \\
&\Rightarrow t_1\, t_2\, <t_3\, t_1\, t_5\, t_6\, t_7\, t_3>\, t_4
\end{aligned}
$$

Out of order execution

Executions as Nested Words

- Each inter-procedurally valid path in the program is a (nested) word in the language of the VPG
- The semantics of a nested word w is a relation [[w]]



- For a program P to which correspond a VPG G define:

$$[[P]] = \bigcup_{w \in L(G)} [[w]]$$

Index Bounded Under-approximations

Derivation Index

A derivation of a context free grammar G

$$Q \Rightarrow \alpha_1 \Rightarrow \alpha_2 \ldots \Rightarrow \alpha_n$$

has index k if each α_i **has at most k occurrences of variables** (non-terminals)

The language $L^{(k)}(G)$ of index k is the set of words that can be obtained by a derivation of index k

Bounded Index Semantics

For a program P to which corresponds a VPG G define:

$$[[P]]^{(k)} = \{ [[w]] \mid w \in L^{(k)}(G) \}$$

- $[[P]]^{(1)} \subseteq [[P]]^{(2)} \subseteq \ldots$ forms an increasing sequence
- $[[P]] = \cup_{k>0} [[P]]^{(k)}$ i.e. the limit is the summary of P
- $[[P]]^{(k)}$ is the precondition of a non-recursive program

Bounded Index Semantics

For a program P to which corresponds a VPG G define:

$$[[P]]^{(k)} = \{ [[w]] \mid w \in L^{(k)}(G) \}$$

- $[[P]]^{(1)} \subseteq [[P]]^{(2)} \subseteq \ldots$ forms an increasing sequence
- $[[P]] = \cup_{k>0} [[P]]^{(k)}$ i.e. the limit is the summary of P
- $[[P]]^{(k)}$ is the precondition of a non-recursive program

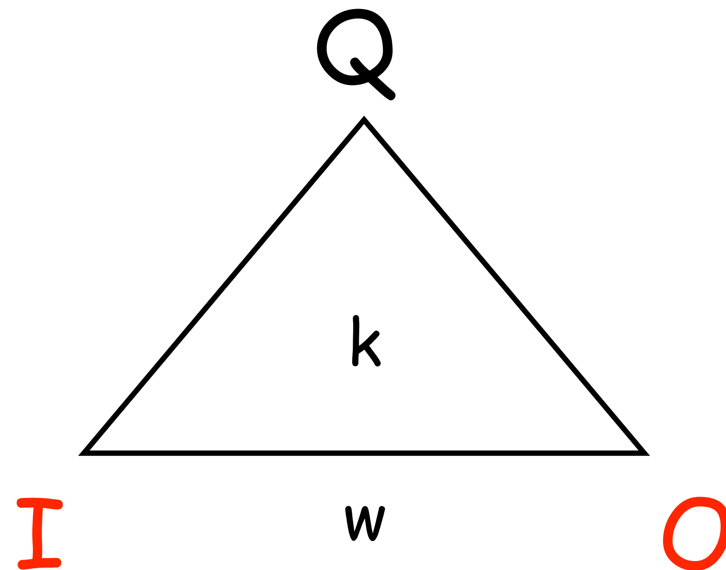
We actually compute for each nonterminal Q:

$$[[Q]]^{(k)} = \{ [[w]] \mid Q \Rightarrow^{(k)} w \}$$

The generation of Q_k

X - local variables of all procedures

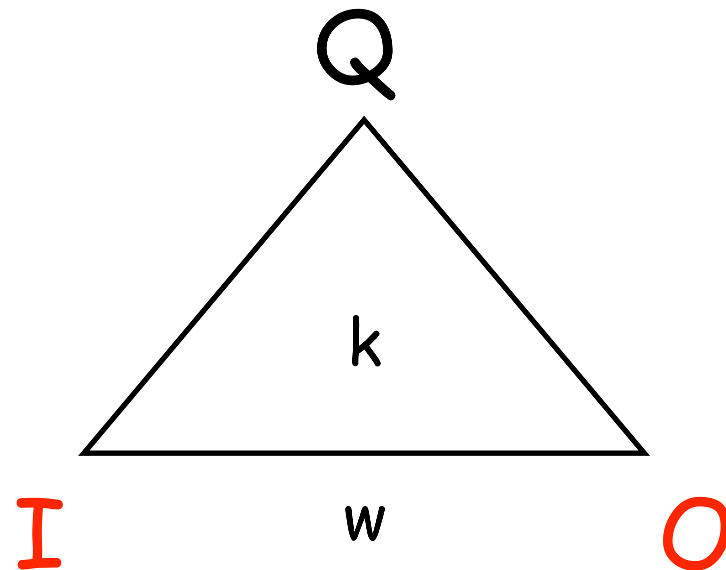
(I,O) ∈ [[Q]] $^{(k)}$ **IFF** (I,O) ∈ [[w]], for some Q =>$^{(k)}$ w



The generation of Q_k

X - local variables of all procedures

$(I,O) \in [[Q]]^{(k)}$ **IFF** $(I,O) \in [[w]]$, for some $Q \Rightarrow^{(k)} w$

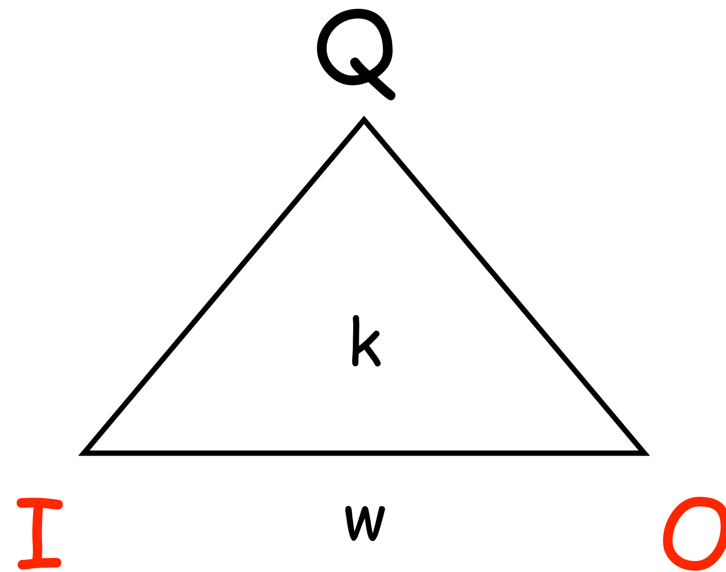


IFF $query^k_Q(I,O)$ returns

The generation of Q_k

X - local variables of all procedures

(I,O) ∈ [[Q]] $^{(k)}$ **IFF** (I,O) ∈ [[w]], for some Q =>$^{(k)}$ w



IFF query^k_Q(I,O) returns

procedure $\text{query}^k_Q(X_I, X_o)$
var X_J, X_K, X_L ;

The generation of Q_k

procedure $query^{k}_{Q}(X_I, X_o)$
var X_J, X_K, X_L ;

choose production rule $Q \rightarrow \alpha$;

The generation of Q_k

the current index

procedure $\text{query}^k{}_Q(X_I, X_o)$
var X_J, X_K, X_L ;

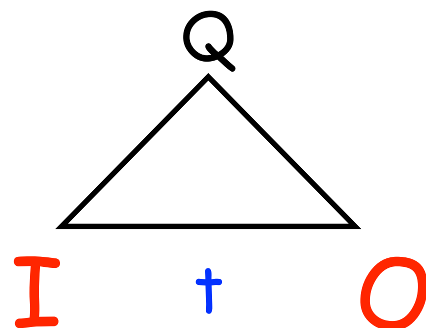
the current nonterminal

choose production rule Q -> α;

The generation of Q_k

$(I,O) \in [[Q]]^{(k)}$ iff $(I,O) \in [[w]]$, for some $Q \Rightarrow^{(k)} w$

1. $Q \Rightarrow t$ and $(I,O) \in [[t]]$

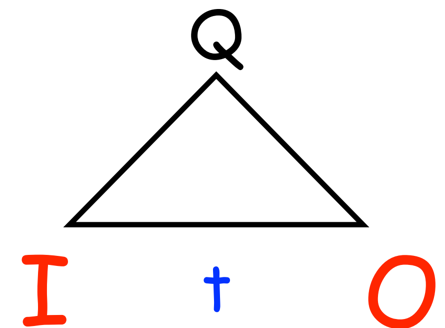


The generation of Q_k

procedure $query^{k}_{Q}(X_I, X_o)$
var X_J, X_K, X_L ;

choose production rule Q -> α;

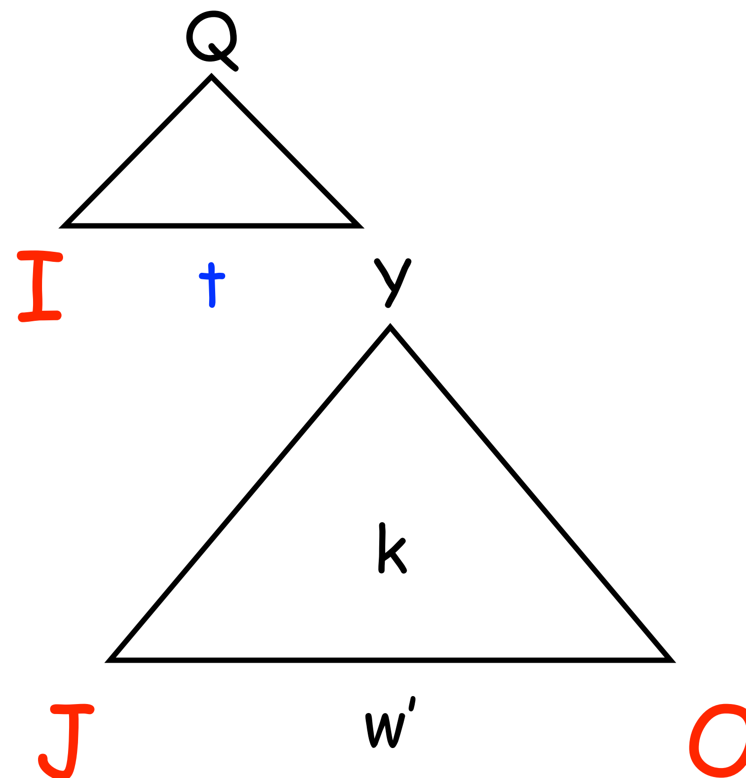
case Q -> t : **assume [[t]]**(X_I, X_o);



The generation of Q_k

$(I,O) \in [[Q]]^{(k)}$ iff $(I,O) \in [[w]]$, for some $Q \Rightarrow^{(k)} w$

2. $Q \Rightarrow^{(k)} ty \Rightarrow^{(k)} tw'$



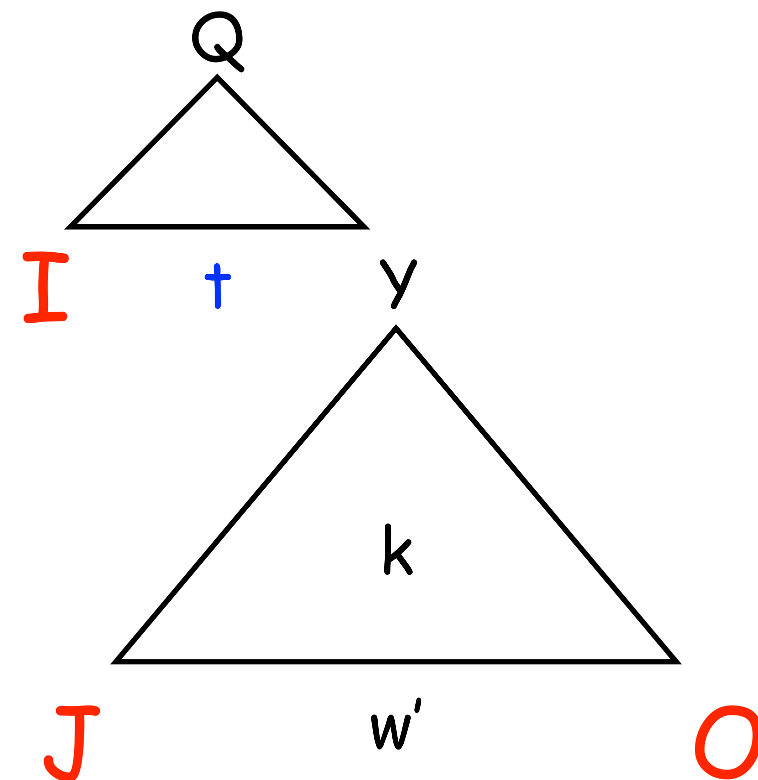
The generation of Q_k

procedure $\text{query}^{k}_{Q}(X_I, X_o)$
var X_J, X_K, X_L ;

choose production rule Q -> α;

case Q -> t Y :

havoc X_J;
assume [[t]](X_I, X_J);
$\text{query}^{k}_{Y}(X_J, X_o)$;



The generation of Q_k

procedure $\text{query}^{k}_{Q}(X_I, X_o)$

var X_J, X_K, X_L ;

var PC = Q;

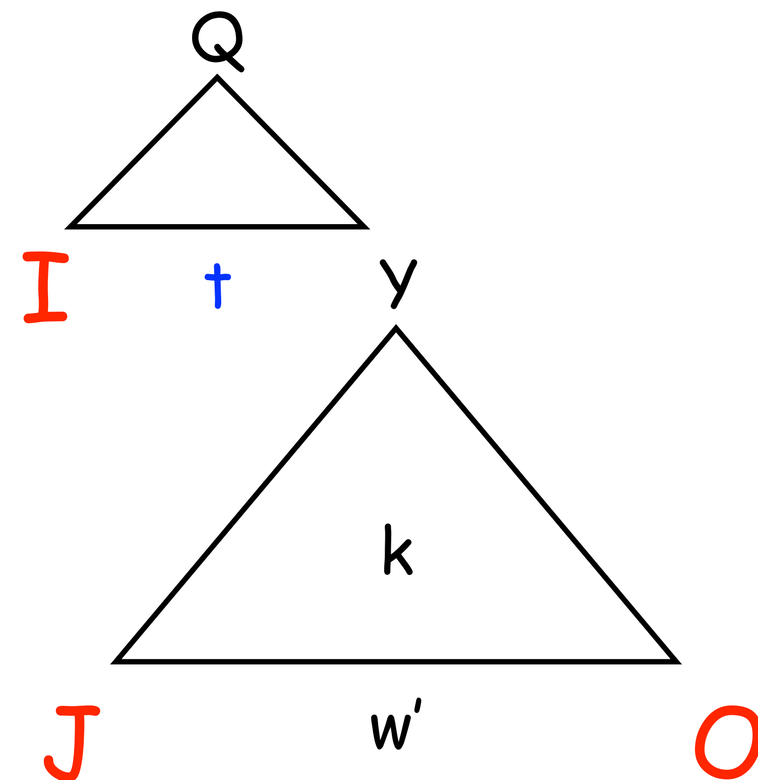
choose production rule PC -> α;

case Q -> t Y :

havoc X_J;

assume [[t]](X_I, X_J);

$X_i = X_J$; PC = Y;



The generation of Q_k

procedure $query^{k}_{Q}(X_I, X_o)$

var X_J, X_K, X_L ;

var PC = Q;

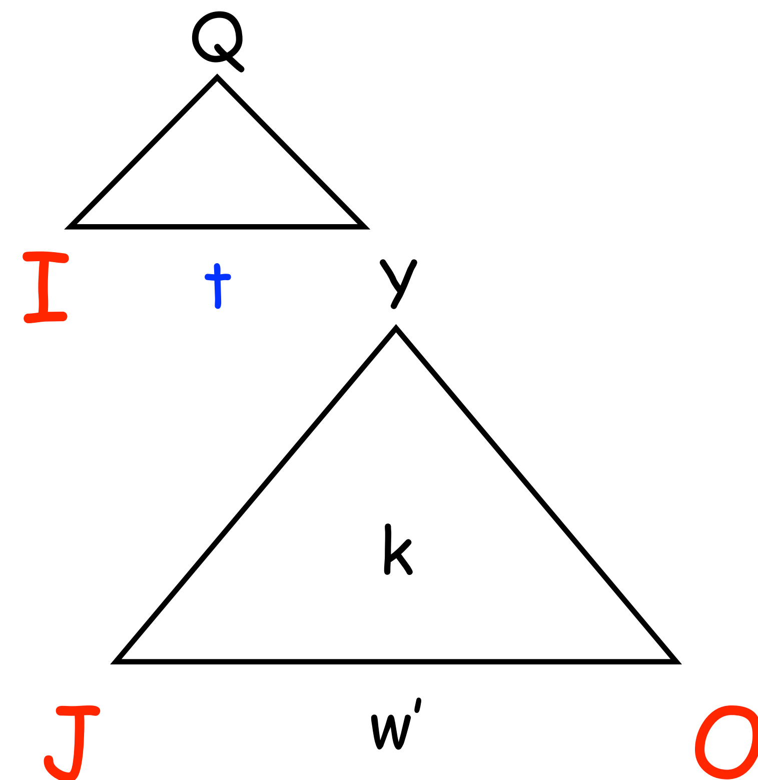
choose production rule PC -> α;

case Q -> t Y :

  havoc X_J;

  assume [[t]](X_I, X_J);

  $X_i = X_J$; PC = Y;



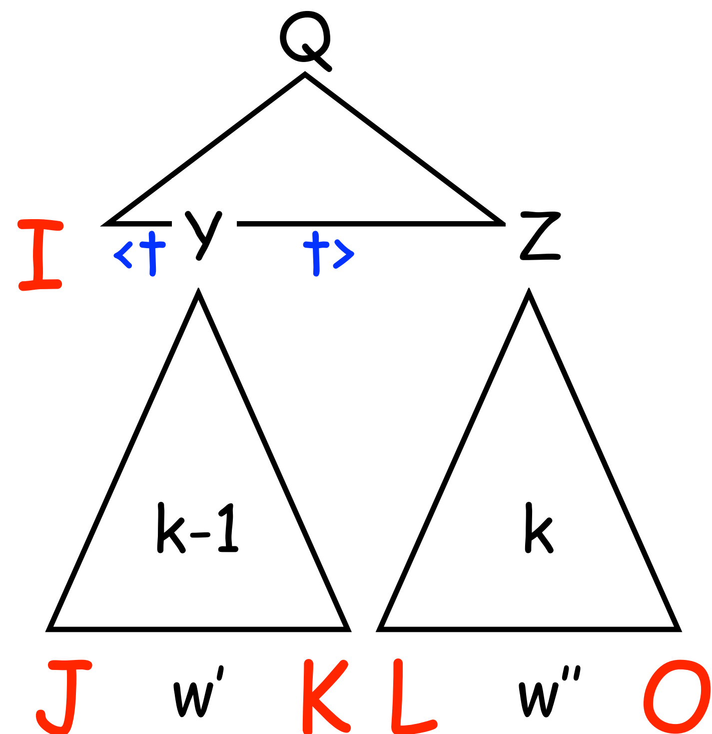
The generation of Q_k

$(I,O) \in [[Q]]^{(k)}$ iff $(I,O) \in [[w]]$, for some $Q \Rightarrow^{(k)} w$

3. $Q \Rightarrow^{(k)}$ <t Y t> Z $\Rightarrow^{(k)} w$

$Y \Rightarrow^{(k-1)} w'$ or

$Z \Rightarrow^{(k)} w''$



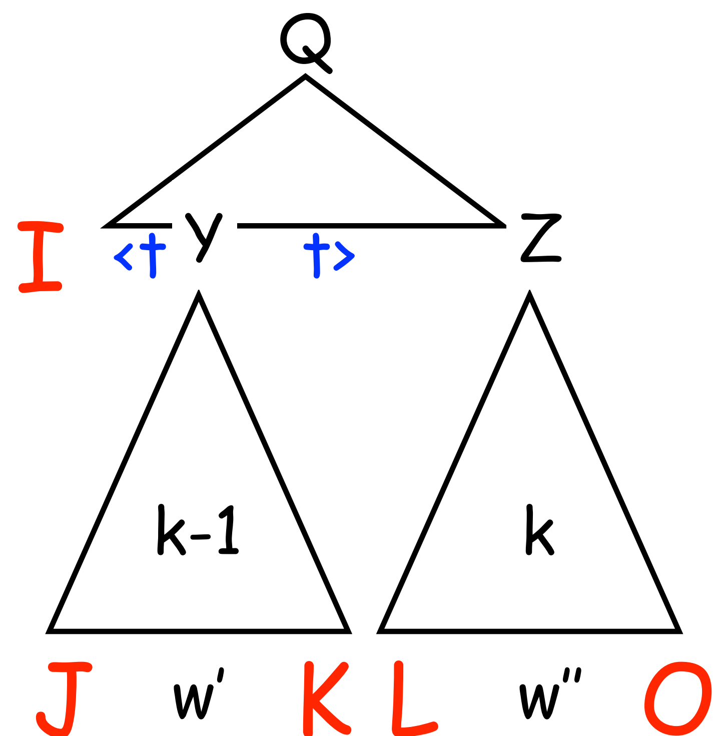
The generation of Q_k

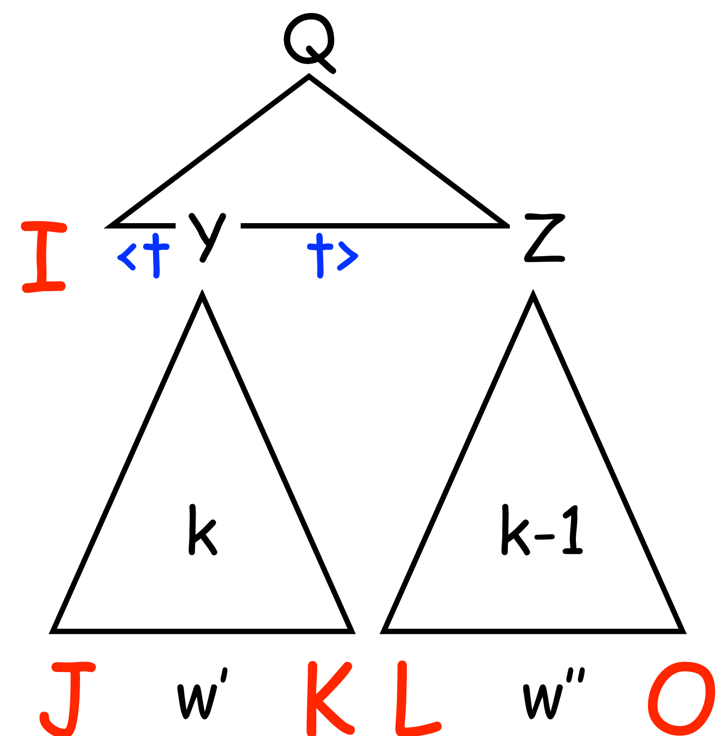
$(I,O) \in [[Q]]^{(k)}$ iff $(I,O) \in [[w]]$, for some $Q \Rightarrow^{(k)} w$

3. $Q \Rightarrow^{(k)}$ <t Y t> Z $\Rightarrow^{(k)} w$

$Y \Rightarrow^{(k-1)} w'$ or $Y \Rightarrow^{(k)} w'$

$Z \Rightarrow^{(k)} w''$ $\quad$ $Z \Rightarrow^{(k-1)} w''$





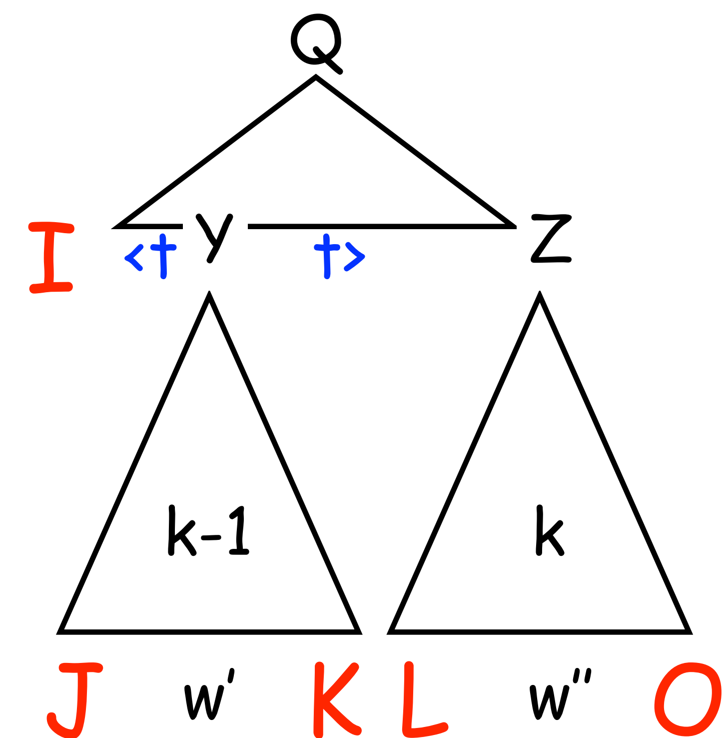
The generation of Q_k

procedure $query^k_Q(X_I, X_o)$
var X_J, X_K, X_L ;

choose production rule Q -> α;

case Q -> <t Y t> Z :

    havoc X_J, X_K, X_L;
    assume [[<t]](X_I, X_J);
    assume [[t>]](X_K, X_L);
    assume $\Phi_t(X_I, X_L)$;
    $query^{k-1}_Y(X_J, X_K)$;
    $query^k_Z(X_L, X_O)$;

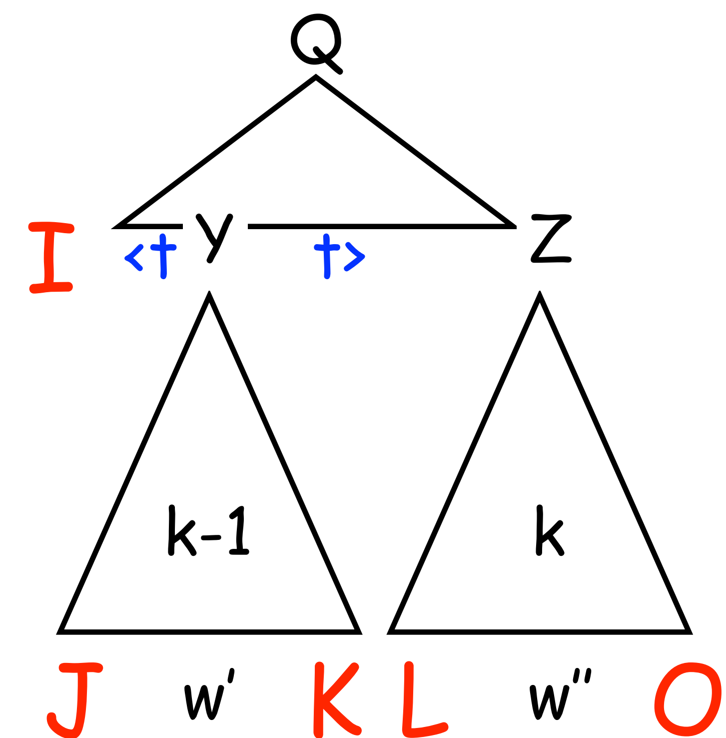


The generation of Q_k

procedure $\text{query}^{k}_{Q}(X_I, X_o)$
var X_J, X_K, X_L ;
var PC = Q;
choose production rule PC -> α;

case Q -> <t Y t> Z :

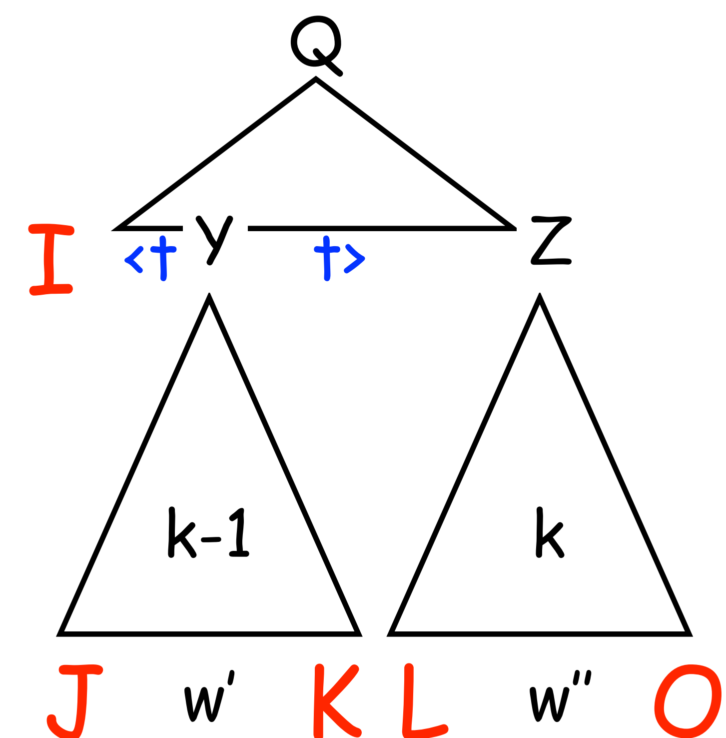
havoc X_J, X_K, X_L;
assume [[<t]](X_I, X_J);
assume [[t>]](X_K, X_L);
assume $\Phi_t(X_I, X_L)$;
$\text{query}^{k-1}_{Y}(X_J, X_K)$;
$X_i = X_L$; PC = Z;



The generation of Q_k

procedure $query^{k}_{Q}(X_I, X_o)$
var X_J, X_K, X_L ;
var PC = Q;
choose production rule PC -> α;

case Q -> <t Y t> Z :
  havoc X_J, X_K, X_L;
  assume [[<t]](X_I, X_J);
  assume [[t>]](X_K, X_L);
  assume $\Phi_t(X_I, X_L)$;
  $query^{k-1}_{Y}(X_J, X_K)$;
  $X_i = X_L$; PC = Z;



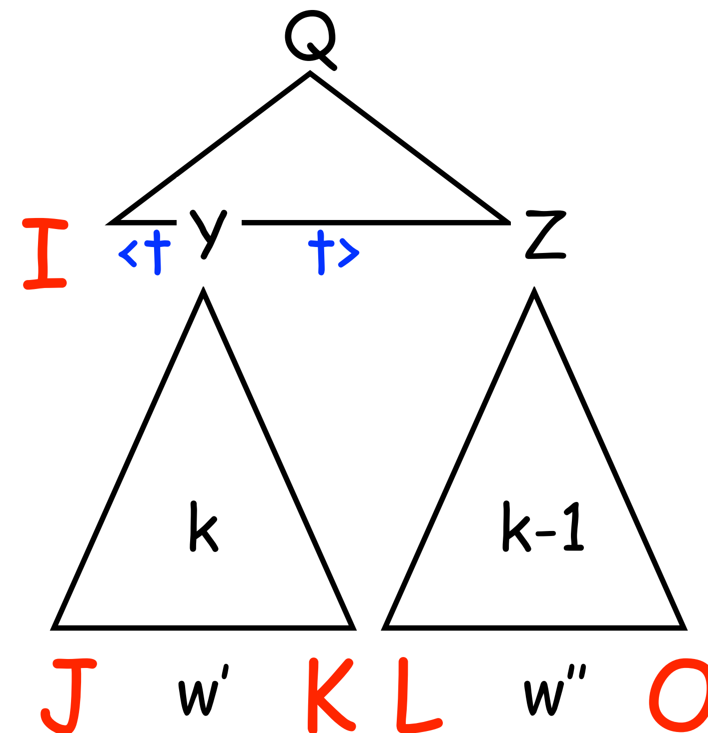
The generation of Q_k

procedure $query^{k}_{Q}(X_I, X_o)$
var X_J, X_K, X_L ;

choose production rule Q -> α;

case Q -> <t Y t> Z :

havoc X_J, X_K, X_L;
assume [[<t]](X_I, X_J);
assume [[t>]](X_K, X_L);
assume $\Phi_t(X_I, X_L)$;
$query^{k-1}_{Z}(X_L, X_O)$;
$query^{k}_{Y}(X_J, X_K)$;



The generation of Q_k

procedure $query^{k}_{Q}(X_I, X_o)$

var X_J, X_K, X_L ;

var PC = Q;

choose production rule PC -> α;

case Q -> <t Y t> Z :

havoc X_J, X_K, X_L;

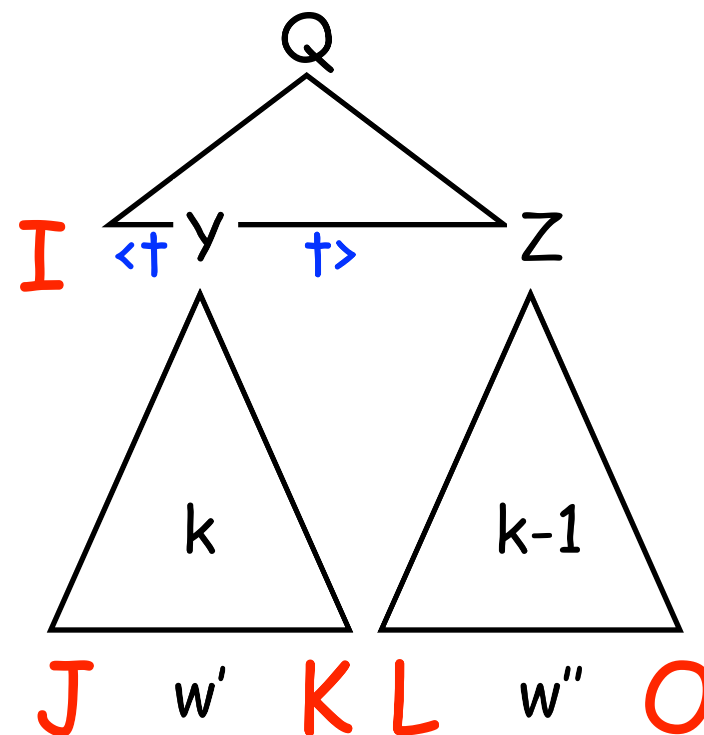
assume [[<t]](X_I, X_J);

assume [[t>]](X_K, X_L);

assume $\Phi_t(X_I, X_L)$;

$query^{k-1}_{Z}(X_L, X_O)$;

$X_i = X_J$; $X_O = X_K$; PC = Y;



The generation of Q_k

procedure $\text{query}^{k}{}_{Q}(X_I, X_o)$

var X_J, X_K, X_L ;

var PC = Q;

choose production rule PC -> α;

case Q -> <t Y t> Z :

havoc X_J, X_K, X_L;

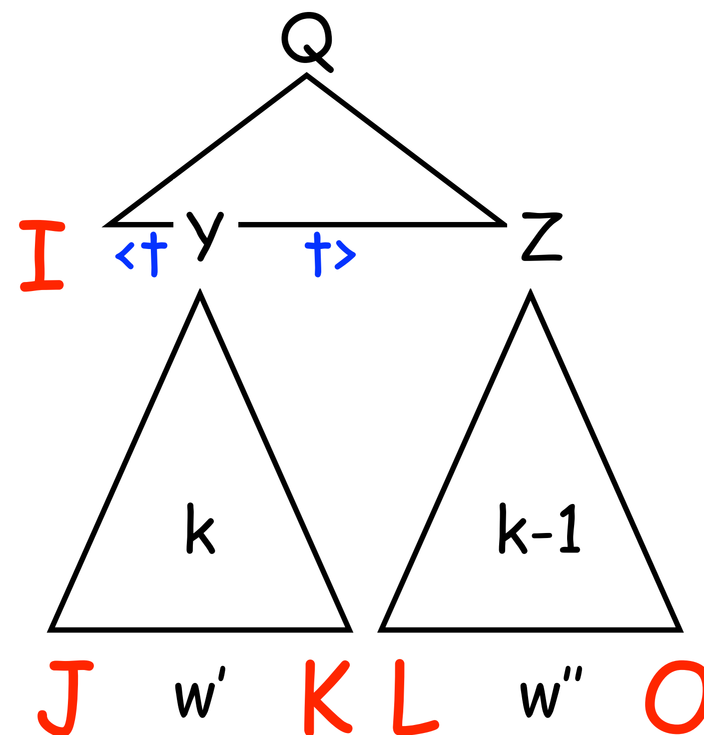
assume $[[<t]](X_I, X_J)$;

assume $[[t>]](X_K, X_L)$;

assume $\Phi_t(X_I, X_L)$;

$\text{query}^{k-1}{}_{Z}(X_L, X_O)$;

$X_i = X_J$; $X_O = X_K$; PC = Y;



A Completness Result

Bounded Languages

- A language $L \subseteq \Sigma^*$ is said to be bounded iff exists words $w_1, w_2, \ldots w_n \in \Sigma^*$ such that $\mathbf{L} \subseteq \mathbf{w_1^* w_2^* \ldots w_n^*}$
- A recursive program P is said to be bounded iff L(G) is bounded, where G is its corresponding VPG
- A bounded non-recursive program is said to be flat

Periodic Relations

- The transitive closure of a relation is Presburger definable for the following periodic relations:
  - octagonal relations: $\bigwedge \pm x \pm y \leq c$, $c \in Z$
  - affine relations $x' = Ax + b$, where $A \in Z^{n \times n}$, $b \in Z^n$, and the set of powers of A is finite
- A program is said to be periodic iff all its statements are definable by periodic relations

Bounded Periodic Programs

- P is bounded => [[P]] = $[[P]]^{(n)}$ for n <= #locations in P
- P is bounded => Q_k is flat for all k > 0
- P is periodic => the precondition of Q_k is Presburger definable and can be computed using acceleration
- Bounded index under-approximation terminates with precise result for bounded periodic programs
- Available tools based on acceleration: FAST, FLATA

Experimental Results

Experimental Results

Program	Time(s)	k
timesTwo	0.7	2
leq	0.7	2
parity	0.8	2
plus	3.4	2
$McCarthy_{a=2}$	3.7	3
$McCarthy_{a=8}$	45.1	4
$McCarthy_{b=12}$	5.7	3
$McCarthy_{b=13}$	19.1	3
$McCarthy_{b=14}$	24.2	3

```
int MC_a(int x) {
  if (x >= 101)
    return x-10
  else
    return (mc_a)^a(x+10a-9)
}
```

```
int MC_b(int x) {
  if (x >= 101)
    return x-10
  else
    return (mc_b)^2(x+b)
}
```

Conclusions

- Under-approximation based on limiting the index of the derivations that produce execution traces
- The summary of a recursive program can be successively under-approximated by a sequence of non-recursive programs, capturing possibly infinite behaviors
- Complete method for bounded periodic programs
- Implementation and experimental results