

Resources Control Graphs

Jean-Yves Moyen

LIPN – CNRS



Motivations

- Programs analysis deal about uniform properties:
 - Do **all** the executions terminate?
 - Are **all** the executions performed within a given time/space bound ?
- Importance of the use of resources.
 - Time and space are resources usually considered.
 - Termination is usually reduced to finding a decreasing well-founded ordering, *i.e.* total usage of a finite resource.
 - Only specific resources may be considered (non-overflow of a specific buffer or stack).
- Design a single tool for this kind of analysis.

The core idea

Control Flow Graphs

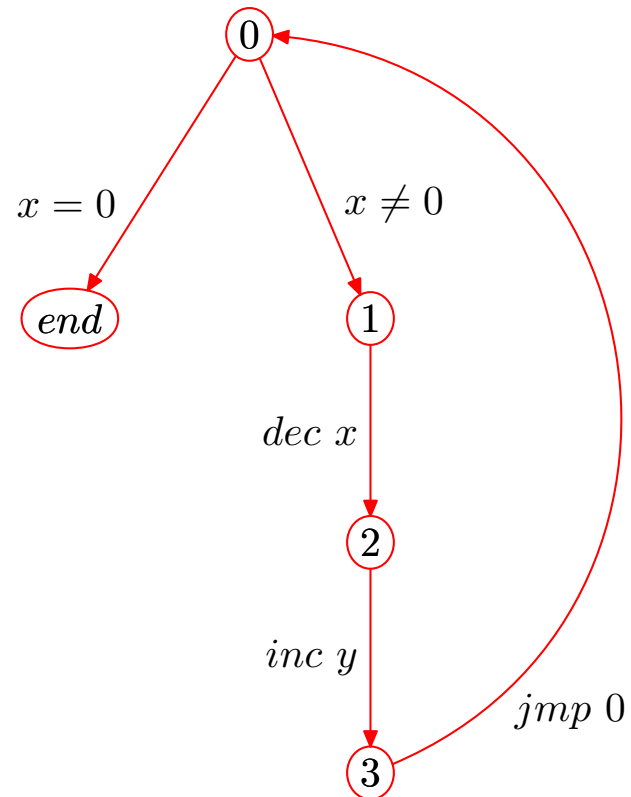
We're working here over counters machines.

```
0 : if  $x = 0$  jmp end  
1 :  $x --$   
2 :  $y ++$   
3 : jmp 0  
end : end
```

Control Flow Graphs

We're working here over counters machines.

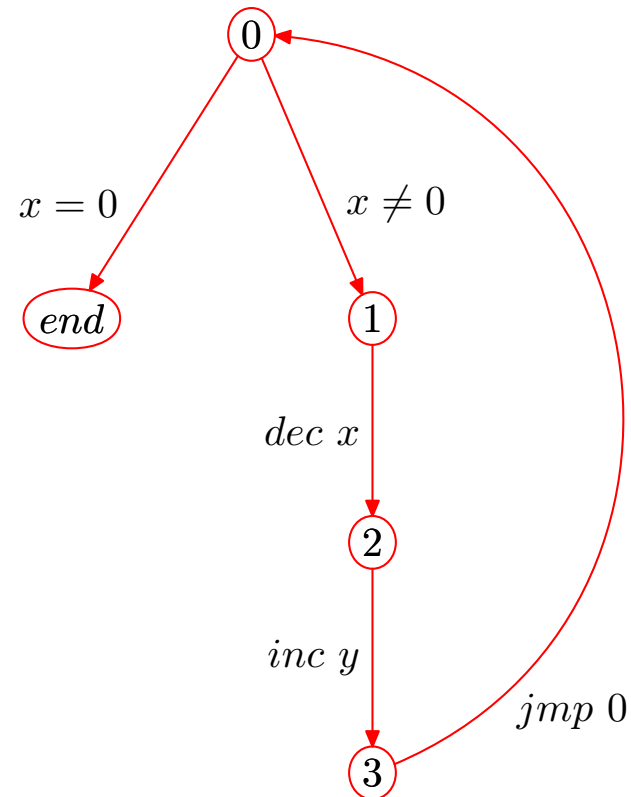
0 : if $x = 0$ jmp *end*
1 : $x --$
2 : $y ++$
3 : jmp 0
end : end



Control Flow Graphs

We're working here over counters machines.

```
0 : if  $x = 0$  jmp end  
1 :  $x --$   
2 :  $y ++$   
3 : jmp 0  
end : end
```

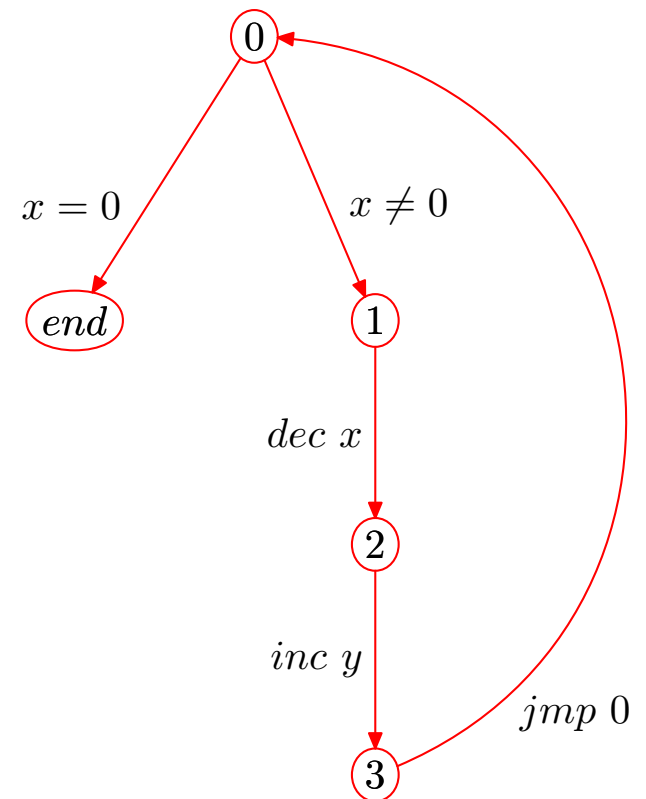


An execution of the program \rightsquigarrow A path in the CFG.

Petri nets and VASS

Resource usage, *i.e.* values of variables, can be seen as the number of token in states of a Petri net.

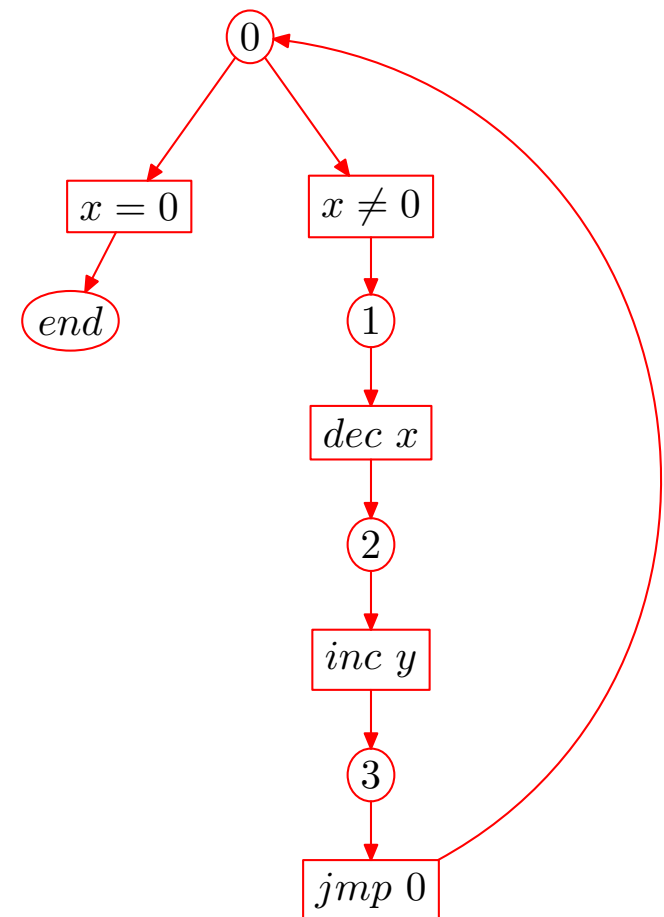
0 : if $x = 0$ jmp *end*
1 : $x --$
2 : $y ++$
3 : jmp 0
end : end



Petri nets and VASS

Resource usage, *i.e.* values of variables, can be seen as the number of token in states of a Petri net.

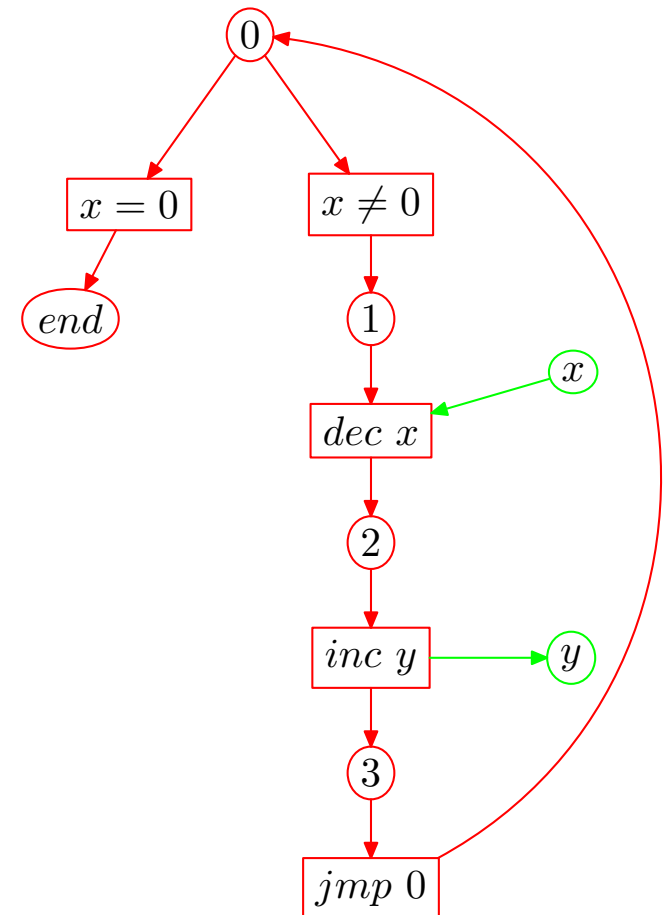
0 : if $x = 0$ jmp *end*
1 : $x --$
2 : $y ++$
3 : jmp 0
end : end



Petri nets and VASS

Resource usage, *i.e.* values of variables, can be seen as the number of token in states of a Petri net.

0 : if $x = 0$ jmp *end*
1 : $x --$
2 : $y ++$
3 : jmp 0
end : *end*



Petri nets and VASS

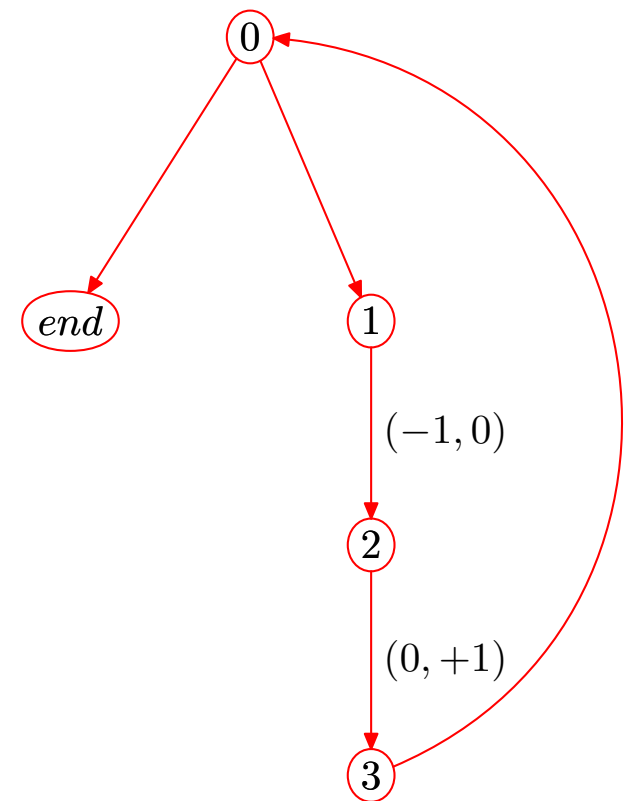
Values of variables can be stored in a vector thus leading to a Vectors Addition System with States (VASS).

0 : if $x = 0$ jmp *end*
1 : $x - -$
2 : $y + +$
3 : jmp 0
end : end

Petri nets and VASS

Values of variables can be stored in a vector thus leading to a Vectors Addition System with States (VASS).

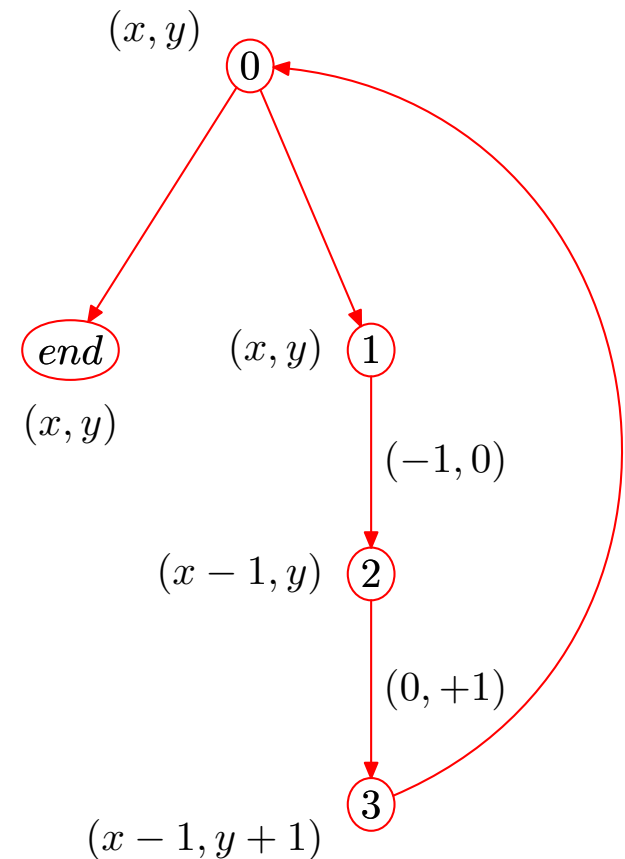
0 : if $x = 0$ jmp *end*
1 : $x --$
2 : $y ++$
3 : jmp 0
end : end



Petri nets and VASS

Values of variables can be stored in a vector thus leading to a Vectors Addition System with States (VASS).

0 : if $x = 0$ jmp *end*
1 : $x - -$
2 : $y + +$
3 : jmp 0
end : end



Resource Systems with States

Resource Systems with States

A RSS is a quintuple (G, V, V^+, W, ω) where:

Resource Systems with States

A RSS is a quintuple (G, V, V^+, W, ω) where:

- G is a directed graph.
 - $S = \{s_1, \dots, s_n\}$ are the vertices.
 - $A = \{a_1, \dots, a_m\}$ are the edges.

Resource Systems with States

A RSS is a quintuple (G, V, V^+, W, ω) where:

- G is a directed graph.
 - $S = \{s_1, \dots, s_n\}$ are the vertices.
 - $A = \{a_1, \dots, a_m\}$ are the edges.
- For each vertex s_i , there is:
 - A set of *valuations* V_i
 - A set of *admissible valuations* V_i^+ .
- $V = \bigcup V_i, V^+ = \bigcup V_i^+$.

Resource Systems with States

A RSS is a quintuple (G, V, V^+, W, ω) where:

- G is a directed graph.
 - $S = \{s_1, \dots, s_n\}$ are the vertices.
 - $A = \{a_1, \dots, a_m\}$ are the edges.
- For each vertex s_i , there is:
 - A set of *valuations* V_i
 - A set of *admissible valuations* V_i^+ .
- $V = \bigcup V_i, V^+ = \bigcup V_i^+$.
- $W_{i,j} = \mathcal{F}(V_i, V_j), W = \bigcup W_{i,j}$ is the set of weights.
- $\omega : A \rightarrow W$ such that if $a = (s_i, s_j)$, then $\omega(a) \in W_{i,j}$.

Configurations and walks

Let $G = (S, A)$ be a graph.

- A vertex is an element of S .
- A *path* is a sequence of vertices s^1, \dots, s^n such that there is an edge between s^i and s^{i+1} .

Configurations and walks

Let $G = (S, A)$ be a graph.

- A vertex is an element of S .
- A *path* is a sequence of vertices s^1, \dots, s^n such that there is an edge between s^i and s^{i+1} .

Let $R = (G, V, V^+, W, \omega)$ be a RSS.

- A *configuration* is a couple (s_i, x) with $x \in V_i$.
- A configuration (s_i, x) is *admissible* if $x \in V_i^+$.

Configurations and walks

Let $G = (S, A)$ be a graph.

- A vertex is an element of S .
- A *path* is a sequence of vertices s^1, \dots, s^n such that there is an edge between s^i and s^{i+1} .

Let $R = (G, V, V^+, W, \omega)$ be a RSS.

- A *configuration* is a couple (s_i, x) with $x \in V_i$.
- A configuration (s_i, x) is *admissible* if $x \in V_i^+$.
- A *walk* is a sequence of configurations $(s^1, x^1), \dots, (s^n, x^n)$ such that:
 - s^1, \dots, s^n is a path.
 - If a^i is the edge between s^i and s^{i+1} , then $x^{i+1} = \omega(a^i)(x^i)$.

Turing Machines

A TM can be represented by a RSS in the following way:

- Underlying graph is the automaton of the TM.
- Valuations (and admissible valuations) are bi-infinite strings over $\{0, 1\}$.
- Weights perform the corresponding operations.

Each execution of the TM corresponds to an (admissible) walk in the RSS and each (admissible) walk in the RSS corresponds to an execution of the TM.

By Rice's theorem, extensional properties of RSS are not decidable.

Facts and notations

Let $R = (G, V, V^+, W, \omega)$ be a RSS. Let $(s^0, x^0), \dots, (s^n, x^n)$ be a walk following edges a^1, \dots, a^n .

- $x^n = \omega(a^n) \circ \dots \circ \omega(a^1)(x^0)$.
- Functions composition is done in reverse order.
- Weight functions are usually somewhat uniform.
- We write $f \circledast g$ instead of $g \circ f$ and $x \circledast f$ instead of $f(x)$.
- $x^n = x^0 \circledast \omega(a^1) \circledast \dots \circledast \omega(a^n)$.
- \circledast is associative.
- \overline{W} is the closure of $\bigcup \omega(a_i)$ by \circledast .

Ordering, termination, resource awareness

An *ordered RSS* is a RSS $R = (G, V, V^+, W, \omega)$ together with a **well partial order** \prec over V .

Ordering, termination, resource awareness

An *ordered RSS* is a RSS $R = (G, V, V^+, W, \omega)$ together with a **well partial order** \prec over V .

- R is *monotonic* if all functions in \overline{W} are monotonic (increasing).
- R is *positive* if $v \in V^+$ and $v \prec v'$ implies $v' \in V^+$.

Ordering, termination, resource awareness

An *ordered RSS* is a RSS $R = (G, V, V^+, W, \omega)$ together with a **well partial order** \prec over V .

- R is *monotonic* if all functions in \overline{W} are monotonic (increasing).
- R is *positive* if $v \in V^+$ and $v \prec v'$ implies $v' \in V^+$.
- R *uniformly terminates* if there is no infinite admissible walk.
- R is *f -resource aware* if for each admissible walk $(s^0, x^0), \dots, (s^i, x^i)$, $x^i \prec f(x^0)$ (f increasing).

A first result

R does not uniformly terminate $\Rightarrow \exists$ an admissible cycle $(s, x) \xrightarrow{*} (s, x')$ such that $x \preceq x'$.
If R is monotonic and positive, the converse is true.

A first result

R does not uniformly terminate $\Rightarrow \exists$ an admissible cycle $(s, x) \xrightarrow{*} (s, x')$ such that $x \preceq x'$.
If R is monotonic and positive, the converse is true.

If an infinite admissible walk exists, extract from it an infinite sequence of configurations with the same vertex. Since \prec is a well ordering, $x \preceq x'$ can be extracted from this sequence.

A first result

R does not uniformly terminate $\Rightarrow \exists$ an admissible cycle $(s, x) \xrightarrow{*} (s, x')$ such that $x \preceq x'$.
If R is monotonic and positive, the converse is true.

If an infinite admissible walk exists, extract from it an infinite sequence of configurations with the same vertex. Since \prec is a well ordering, $x \preceq x'$ can be extracted from this sequence.

If the cycle exists, follow it infinitely many times. Monotonicity ensures that valuations reached increase and positivity that this keeps everything admissible.

Resource Control Graph

Let p be a program. It's Resource Control Graph (RCG) is a RSS where:

- The underlying graph is the Control Flow Graph.
- Admissible valuations approximate the state of memory.

Approximations allow to restrict weighting functions to some uniform family of functions (e.g. $\lambda x.x + \alpha$).

Resource Control Graph

Let p be a program. Its Resource Control Graph (RCG) is a RSS where:

- The underlying graph is the Control Flow Graph.
- Admissible valuations approximate the state of memory.

Approximations allow to restrict weighting functions to some uniform family of functions (e.g. $\lambda x.x + \alpha$).

To each execution of the program corresponds an admissible walk in the RCG.

Resource Control Graph

Let p be a program. Its Resource Control Graph (RCG) is a RSS where:

- The underlying graph is the Control Flow Graph.
- Admissible valuations approximate the state of memory.

Approximations allow to restrict weighting functions to some uniform family of functions (e.g. $\lambda x.x + \alpha$).

To each execution of the program corresponds an admissible walk in the RCG.

Unif. term. of the RCG \Rightarrow u.t. of the program.

Weighted graphs and Non Size Increasingness

Weighted graphs as RSS

- A weighted graph is a graph $G = (S, A)$ together with a weighting function $\omega : A \rightarrow \mathbb{Z}$.
- The weight of a path is the sum of the weights of each edges.

Weighted graphs as RSS

- A weighted graph is a graph $G = (S, A)$ together with a weighting function $\omega : A \rightarrow \mathbb{Z}$.
- The weight of a path is the sum of the weights of each edges.

This is similar to RSS with $W = \mathbb{Z}$ and $\circ = +$.
 $V = \mathbb{Z}$ and $V^+ = \mathbb{N}$.

Weighted graphs as RSS

- A weighted graph is a graph $G = (S, A)$ together with a weighting function $\omega : A \rightarrow \mathbb{Z}$.
- The weight of a path is the sum of the weights of each edges.

This is similar to RSS with $W = \mathbb{Z}$ and $\circ = +$.

$V = \mathbb{Z}$ and $V^+ = \mathbb{N}$.

A weighted graph uniformly terminates if and only if it contains no cycle of weight ≥ 0 .

It is $\lambda x.x + \alpha$ -resource aware if it contains no cycle of weight > 0 .

Both criteria can be decided in polynomial time.

Non Size Increasingness

Consider assembly-like programs working over lists of integers.

Non Size Increasingness

Consider assembly-like programs working over lists of integers. Build the RCG with the following approximation:

- The memory is only represented by its total space usage.
- Space usage of integers is 1.
- Space usage of lists is their length.

Non Size Increasingness

Consider assembly-like programs working over lists of integers. Build the RCG with the following approximation:

- The memory is only represented by its total space usage.
- Space usage of integers is 1.
- Space usage of lists is their length.

This leads to the following RCG:

- $V^+ = \mathbb{N}$ (total space usage is ≥ 0).
- $\omega(\text{cons}) = \lambda x.x + 1$, $\omega(\text{tail}) = \lambda x.x - 1$.
- $\overline{W} = \bigcup \lambda x.x + \alpha$, $V = \mathbb{Z}$.

NSI (2)

If the RCG is $\lambda x.x + \alpha$ -resource aware, then the program is NSI.

If the RCG is $\lambda x.\beta x + \alpha$ -resource aware, then the program is LINSPEACE.

NSI (2)

If the RCG is $\lambda x.x + \alpha$ -resource aware, then the program is NSI.

If the RCG is $\lambda x.\beta x + \alpha$ -resource aware, then the program is LINSPEACE.

Admissible valuations play exactly the same role as M. Hofmann's diamonds: the valuation is equal to the number of diamonds needed in the current state.

Going further

- We can use more types if we have an appropriate size function.
- We can choose not to consider all the lists, *i.e.* only control some buffers.
- We can similarly control depth of data stacks (`push`, `pop`).
- We can similarly control depth of recursion stack (`call`, `return`).

Size Change Principle

Original SCP

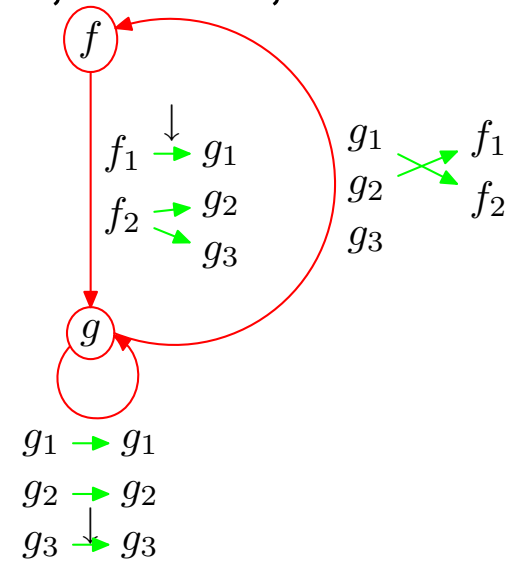
(Lee, Jones, Ben Amram)

$$\begin{aligned}f(a + 1, b) &\rightarrow g(a, b, b) \\g(a, b, c + 1) &\rightarrow g(a, b, c) \\g(a, b, 0) &\rightarrow f(b, a)\end{aligned}$$

Original SCP

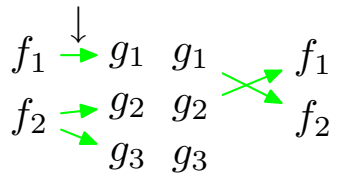
$$\begin{aligned} f(a + 1, b) &\rightarrow g(a, b, b) \\ g(a, b, c + 1) &\rightarrow g(a, b, c) \\ g(a, b, 0) &\rightarrow f(b, a) \end{aligned}$$

(Lee, Jones, Ben Amram)



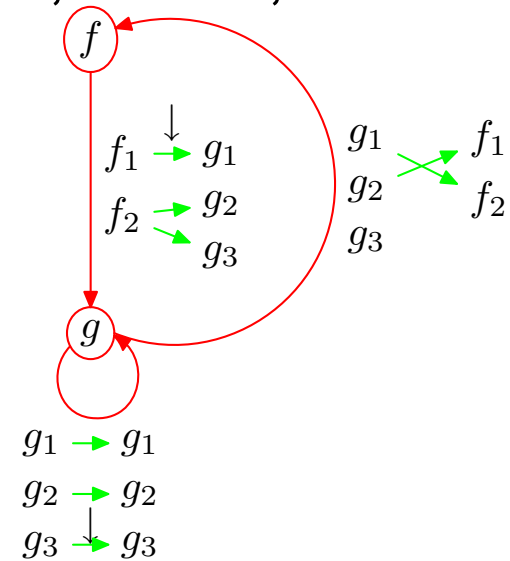
Original SCP

$$\begin{aligned} f(a + 1, b) &\rightarrow g(a, b, b) \\ g(a, b, c + 1) &\rightarrow g(a, b, c) \\ g(a, b, 0) &\rightarrow f(b, a) \end{aligned}$$



Multipaths, threads.

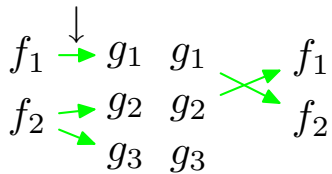
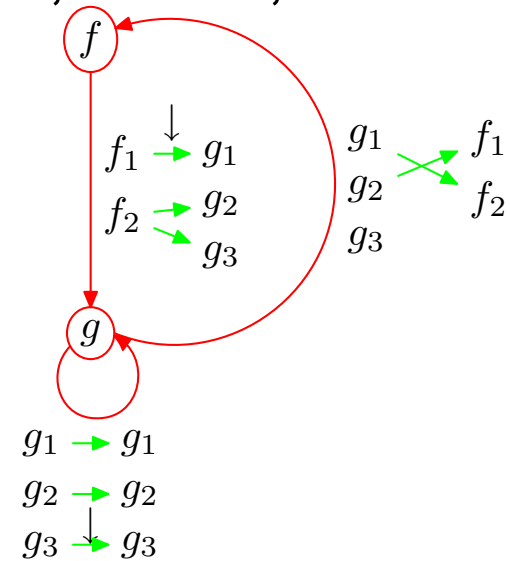
(Lee, Jones, Ben Amram)



Original SCP

$$\begin{aligned}
 f(a + 1, b) &\rightarrow g(a, b, b) \\
 g(a, b, c + 1) &\rightarrow g(a, b, c) \\
 g(a, b, 0) &\rightarrow f(b, a)
 \end{aligned}$$

(Lee, Jones, Ben Amram)



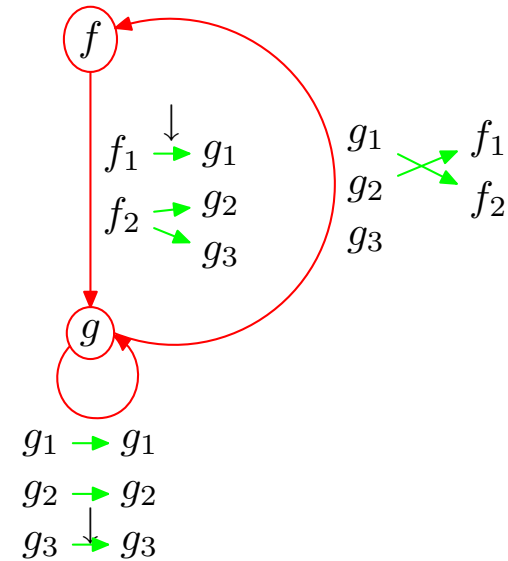
Multipaths, threads.

$FLOW^\omega$, $DESC^\omega$

$FLOW^\omega = DESC^\omega \Rightarrow$ termination
(PSPACE-complet).

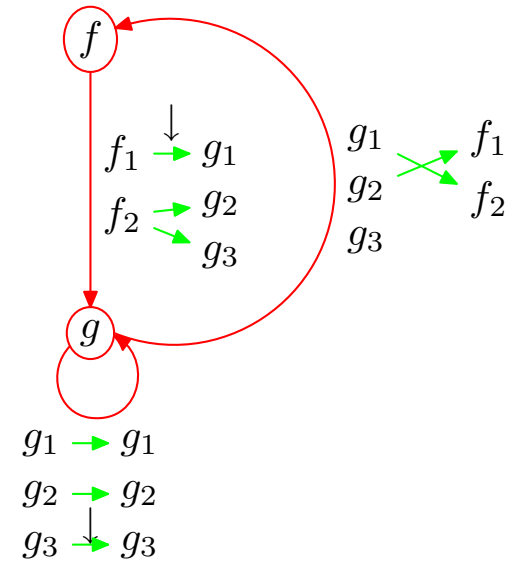
SCP as a RSS

$$\begin{aligned} f(a + 1, b) &\rightarrow g(a, b, b) \\ g(a, b, c + 1) &\rightarrow g(a, b, c) \\ g(a, b, 0) &\rightarrow f(b, a) \end{aligned}$$



SCP as a RSS

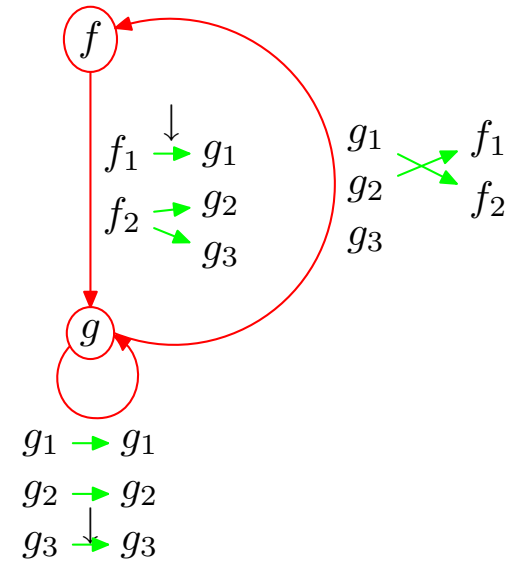
$$\begin{aligned} f(a + 1, b) &\rightarrow g(a, b, b) \\ g(a, b, c + 1) &\rightarrow g(a, b, c) \\ g(a, b, 0) &\rightarrow f(b, a) \end{aligned}$$



$W = \text{SCG}, \circledast = \text{SCG-composition}.$

SCP as a RSS

$$\begin{aligned}
 f(a+1, b) &\rightarrow g(a, b, b) \\
 g(a, b, c+1) &\rightarrow g(a, b, c) \\
 g(a, b, 0) &\rightarrow f(b, a)
 \end{aligned}$$

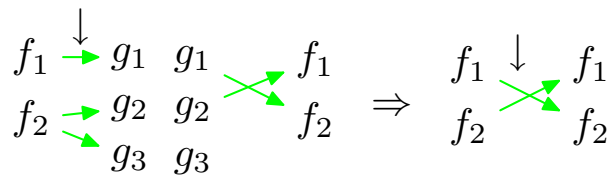
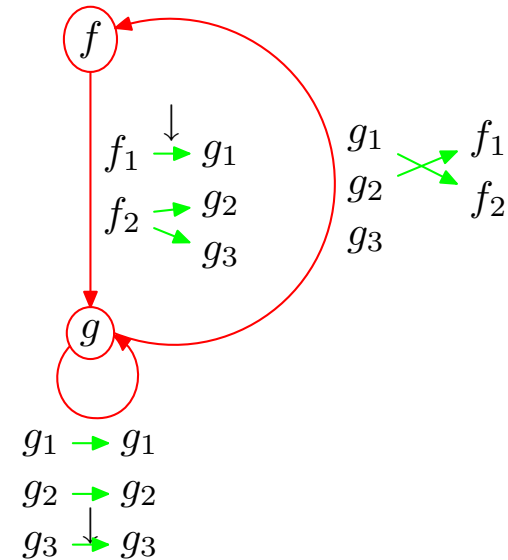


$$\begin{array}{ccc}
 \begin{array}{ccc}
 f_1 \xrightarrow{\downarrow} g_1 & g_1 \xrightarrow{\times} f_1 \\
 f_2 \xrightarrow{\quad} g_2 & g_2 \xrightarrow{\quad} f_2 \\
 & g_3 \xrightarrow{\quad} f_2
 \end{array}
 & \Rightarrow &
 \begin{array}{ccc}
 f_1 \xrightarrow{\downarrow} f_1 \\
 f_2 \xrightarrow{\quad} f_2 \\
 & \xrightarrow{\quad} f_2
 \end{array}
 \end{array}$$

$W = \text{SCG}, \circlearrowleft = \text{SCG-composition.}$

SCP as a RSS

$$\begin{aligned} f(a+1, b) &\rightarrow g(a, b, b) \\ g(a, b, c+1) &\rightarrow g(a, b, c) \\ g(a, b, 0) &\rightarrow f(b, a) \end{aligned}$$

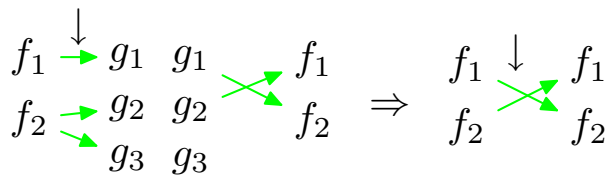
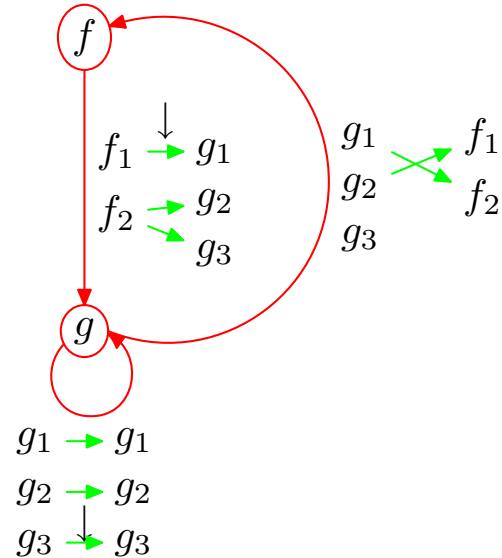


$W = \text{SCG}$, $\circledast = \text{SCG-composition}$.

$V = \mathbb{N}^k$ (number of \downarrow), V^+ if all component $\leq n$.

SCP as a RSS

$$\begin{aligned}
 f(a+1, b) &\rightarrow g(a, b, b) \\
 g(a, b, c+1) &\rightarrow g(a, b, c) \\
 g(a, b, 0) &\rightarrow f(b, a)
 \end{aligned}$$



$W = \text{SCG}$, $\circledast = \text{SCG-composition}$.

$V = \mathbb{N}^k$ (number of \downarrow), V^+ if all component $\leq n$.

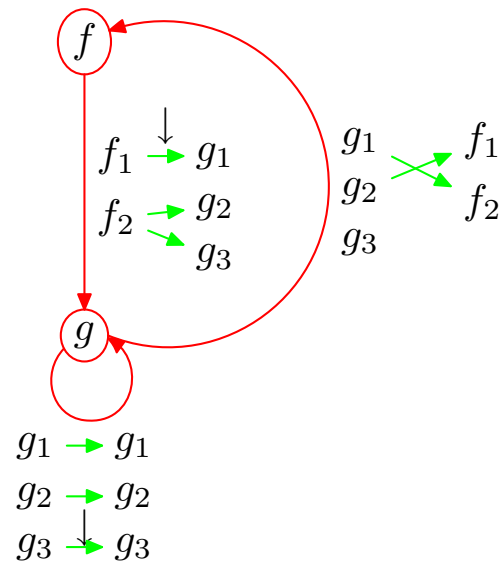
SCP \equiv uniform termination for all n .
 Not u.t. $\Leftrightarrow \exists$ cycle of dec. idempotent weight.

Using matrices

$$\begin{aligned} f(a+1, b) &\rightarrow g(a, b, b) \\ g(a, b, c+1) &\rightarrow g(a, b, c) \\ g(a, b, 0) &\rightarrow f(b, a) \end{aligned}$$

$$\begin{array}{ccc} f_1 \xrightarrow{\downarrow} g_1 & g_1 \xrightarrow{\times} f_1 \\ f_2 \xrightarrow{\quad} g_2 & g_2 \xrightarrow{\quad} f_2 \\ & g_3 \xrightarrow{\quad} f_2 \end{array} \Rightarrow \begin{array}{ccc} f_1 \xrightarrow{\downarrow} f_1 \\ f_2 \xrightarrow{\times} f_2 \end{array}$$

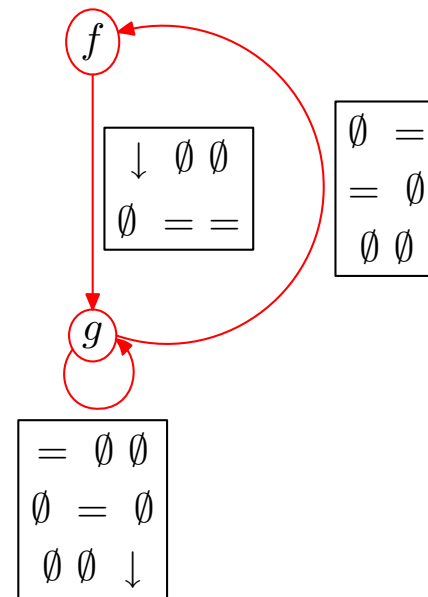
(Abel and Altenkirch)



Using matrices

$$\begin{aligned} f(a + 1, b) &\rightarrow g(a, b, b) \\ g(a, b, c + 1) &\rightarrow g(a, b, c) \\ g(a, b, 0) &\rightarrow f(b, a) \end{aligned}$$

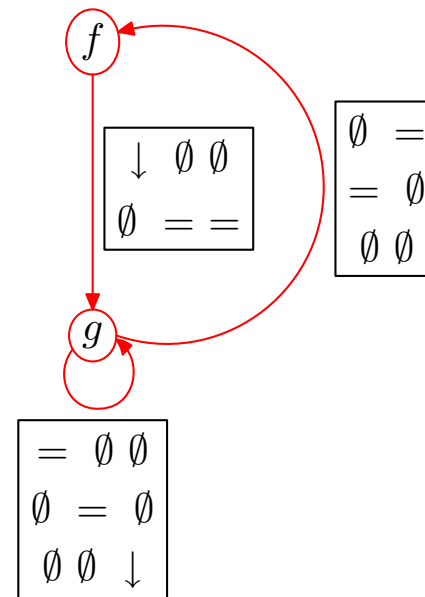
(Abel and Altenkirch)



Using matrices

$$\begin{aligned}
 f(a + 1, b) &\rightarrow g(a, b, b) \\
 g(a, b, c + 1) &\rightarrow g(a, b, c) \\
 g(a, b, 0) &\rightarrow f(b, a)
 \end{aligned}$$

(Abel and Altenkirch)



$$\begin{bmatrix} \downarrow & \emptyset & \emptyset \\ \emptyset & = & = \end{bmatrix} \otimes \begin{bmatrix} \emptyset & = \\ = & \emptyset \\ \emptyset & \emptyset \end{bmatrix} \Rightarrow \begin{bmatrix} \emptyset & \downarrow \\ = & \emptyset \end{bmatrix}$$

W : matrices over the three valued set.

\otimes : matrices multiplication.

Matrices Multiplication Systems with States

MMSS

A MMSS is a RSS where:

- $V_i = bZ^{k_i}, V_i^+ = \mathbb{N}^{k_i}$

- $W_{i,j} = \mathcal{M}_{k_i,k_j}$

- $\circ = \circledast = \times$

If we use column-vectors instead of row-vectors, we need to transpose matrices and perform multiplication in reverse order.

Uniform termination

Uniform termination of MMSS is not decidable

Uniform termination

Uniform termination of MMSS is not decidable

Reduce to uniform termination of counter machines.

- $x++$, $x--$: keep 1 as first component of vector.
- $x \neq 0$: $x--$; $x++$.
- $x = 0$: multiplication by -1 .

Polynomial bound

(Niggl and Wunderlich)

- Use a matrice over $\{0, 1, \infty\}$ as a certificate for polynomiality of instructions.
- Certificate for $P_1; P_2$: $M_1 \times M_2$.
- Certificate for tests: \max .
- Certificate for loops: closure.

Shape of the certificate of a program can lead to polytime bound.

Difficulty of instructions

- $x++$, $x--$, $x \neq 0$ can be modelised with VASS (*i.e.* with vectors).
- $x = 0$ needs MMSS (*i.e.* a matrix) to be modelised. (it is a well-known fact that Petri nets cannot test if a place is empty)
- First order program can be represented by several matrices, *i.e.* a tensor (given an enumeration of the edges).
- Tensor Multiplication Systems should be able to modelise high order programs.
- The higher the order, the more dimensions are required.

Conclusion

- New tool, generalisation of existing one (Petri nets/VASS).
- Several analysis can be rewritten using this tool.
- Is it possible to combine these analysis ?
- Can we also rewrite other analysis ?
- Can we discover new analysis ?