Termination analysis of programs with multiphase control-flow

Samir Genaim

Universidad Complutense de Madrid

Automatic Termination Analysis

Proofs by Ranking Functions

Linear

Lexico. Linear

Multiphase Linear

Abstract and then Prove Termination

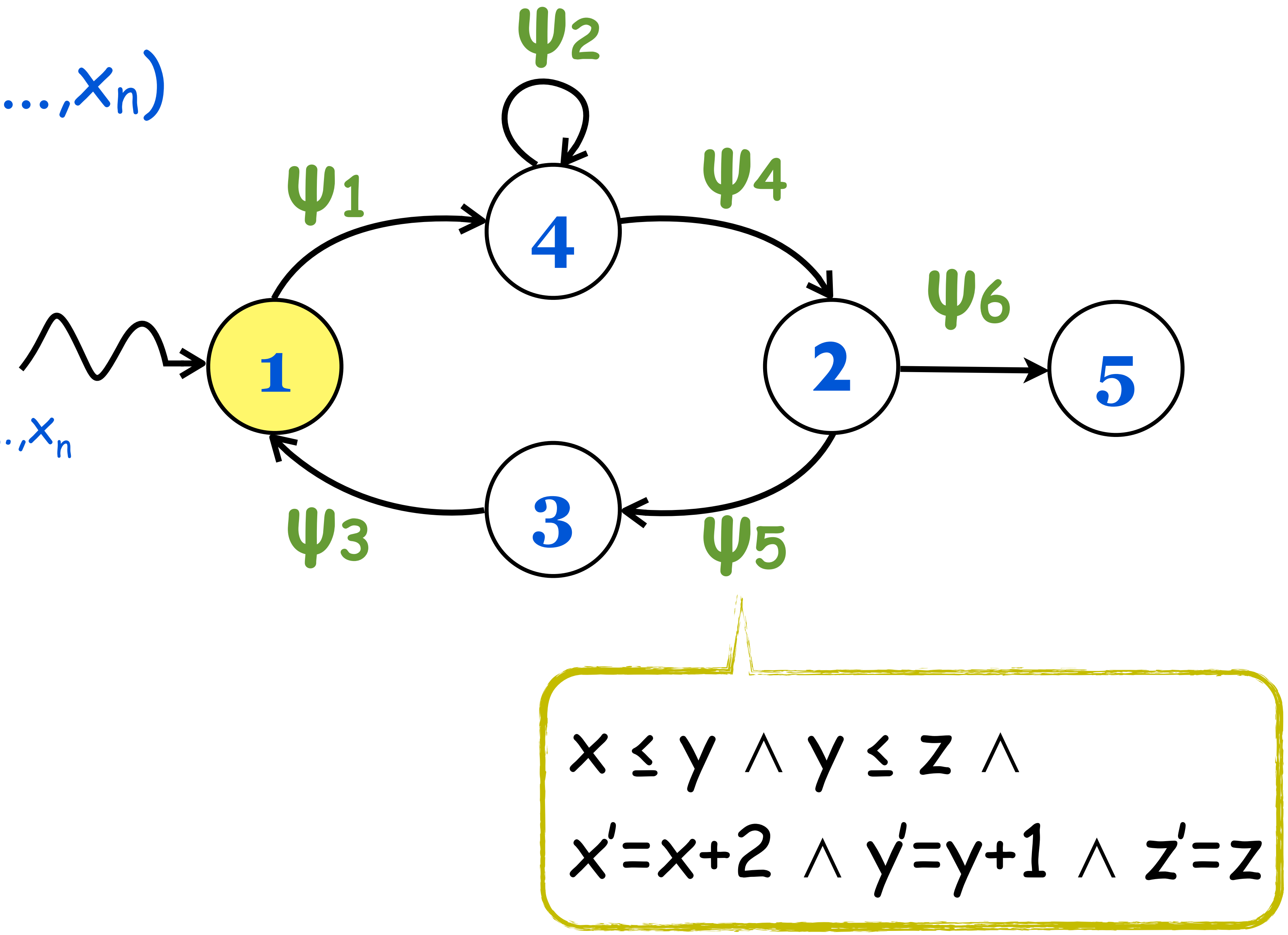
Linear-Constraint Abstraction

Complexity Bounds from Ranking Functions

Linear-Constraint Programs

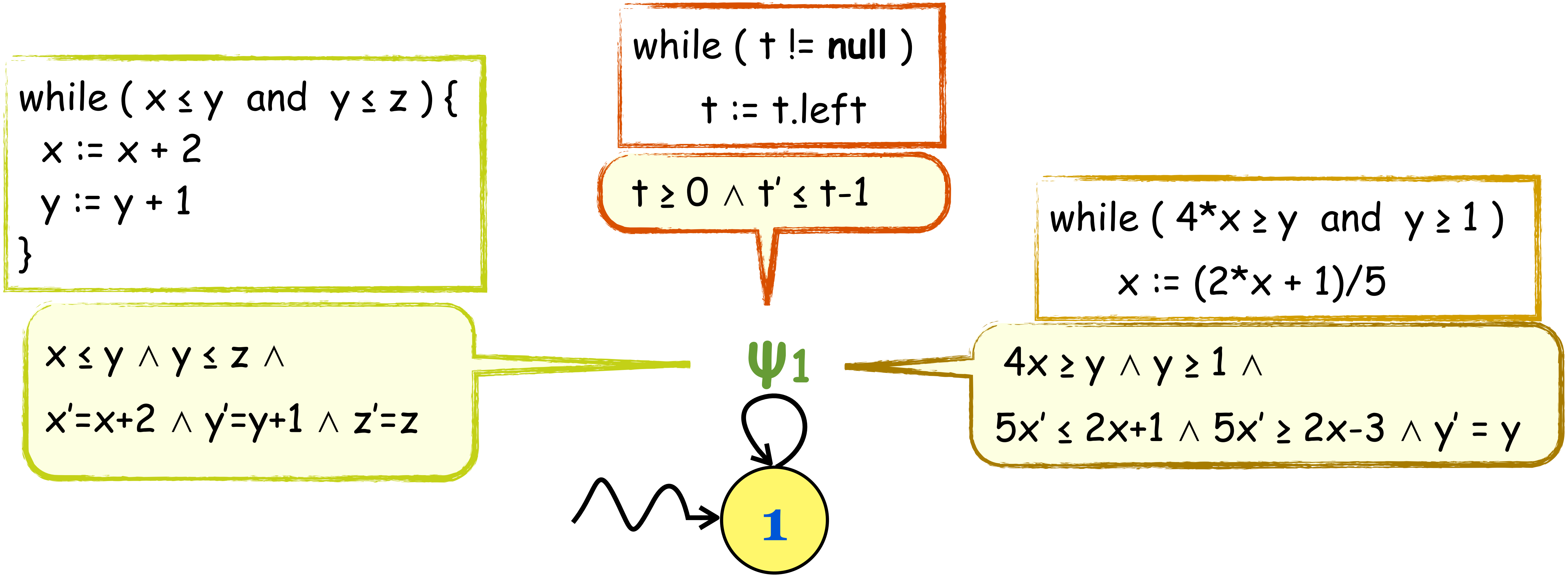
state = $(x_1,\ldots,x_n)$

initial states ψ_0:
a formula over $x_1,\ldots,x_n$



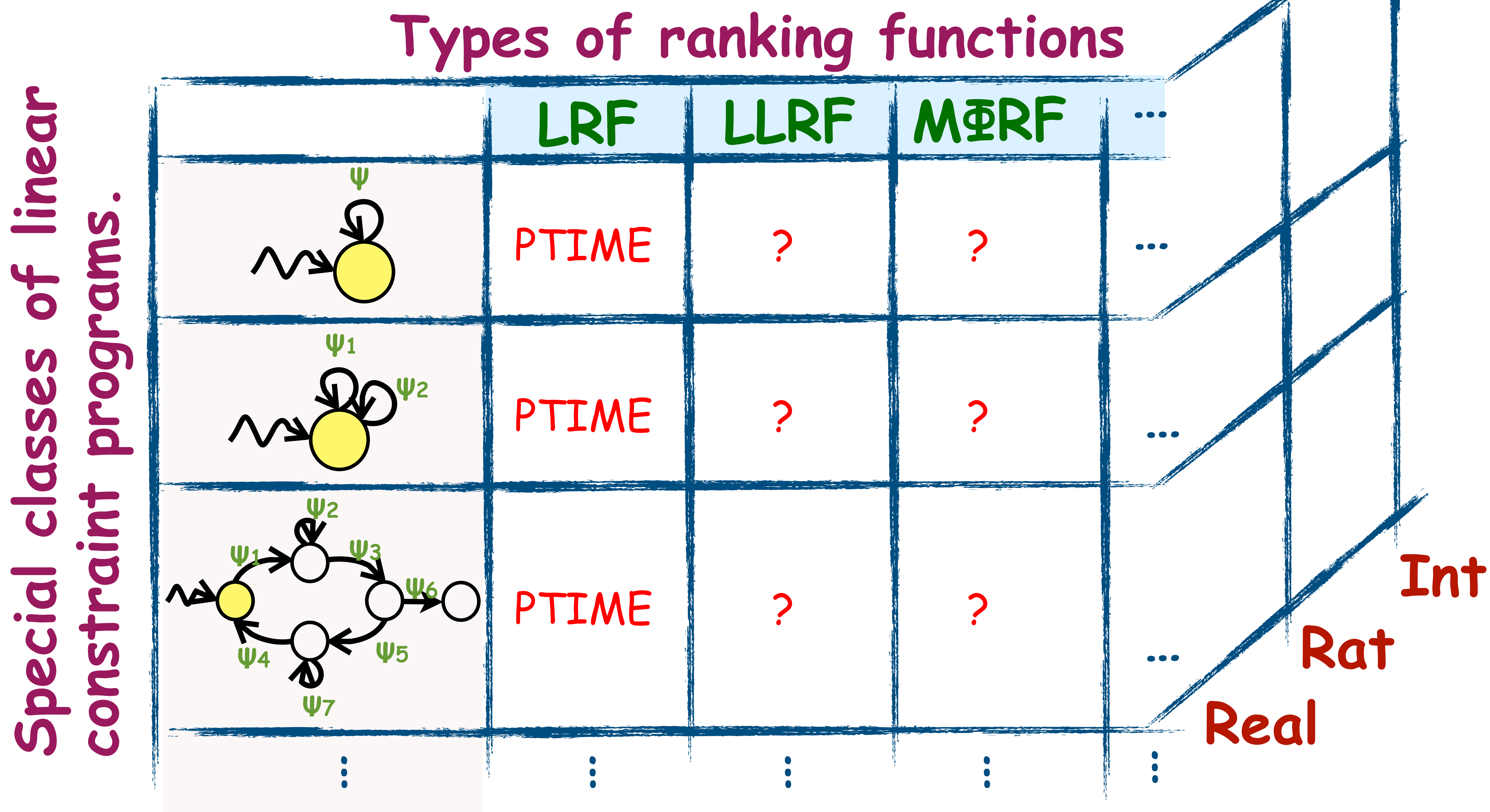
ψ_i are conjunctions of linear constraints over the variables $x_1,\ldots,x_n,x_1',\ldots,x_n'$

Single Path Linear-Constraint Loops



- In many cases the termination proof boils down to termination of SLC loops.
- Interesting questions of decidability of termination in general for this setting.

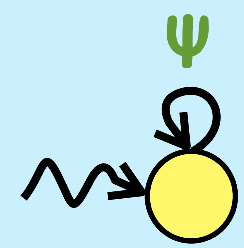
Decidability Questions



- How hard is it to decide if there exists a RF of a specific type, for a given class of programs?
- Develop synthesis algorithms (including loop bounds).

Linear Ranking Functions (by Ex.)

```
while ( x ≤ y ) {
    x := x + 2
    y := y + 1
}
```



$\psi = \{\, x \le y,\ x'=x+2,\ y'=y+1 \,\}$

- $f(x,y) = y-x$ is a linear ranking function (**LRF**)
  - **non-negative** in all (enabled) states: $f(x,y) \ge 0$
  - **strictly decreasing**: $f(x,y)-f(x',y') \ge 1$

LRFs and Alternatives …

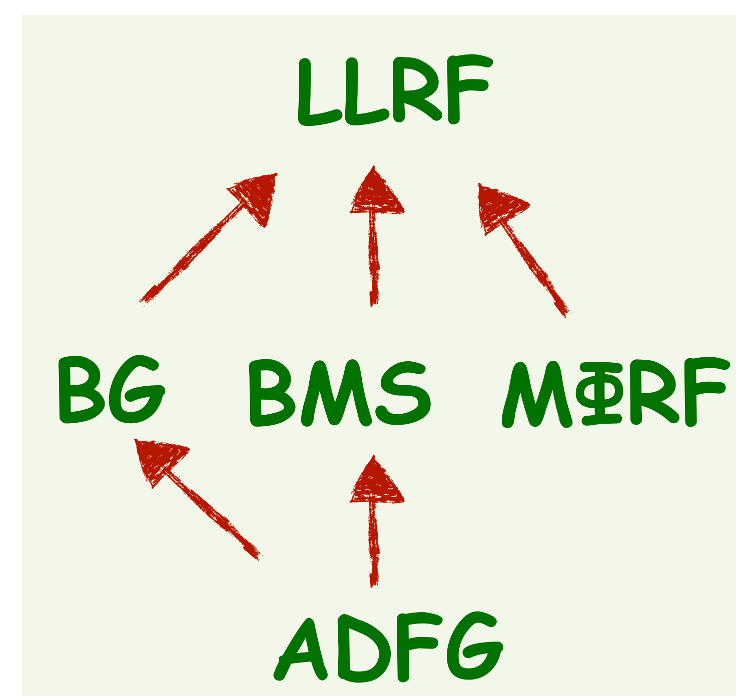
- There are complete algorithms for synthesising LRFs over rationals and integers, even for complex control flow (PTIME / coNP-complete)
  - Sohn and van Gelder (1991)
  - Feautrier (1992)
  - Colón and Sipma (2001)
  - Podelski and Rybalchenko (2004)
  - Mesnard and Serebrenik (2008)
  - Alias, Darte, Feautrier, Gonnord (2010)
  - Ben Amram and Genaim (2013)
  - …
- LRFs do not suffice for all loops… **Lexicographic Linear Ranking Functions (LLRFs)** are a very common alternative.

Types of LLRF

$\langle f_1,\dots,f_i,\dots,f_d \rangle$ is a **LLRF** for a set of transitions **T** iff for any $\vec{x}''=(\vec{x},\vec{x}') \in$ **T** there is $1 \le i \le d$ such that

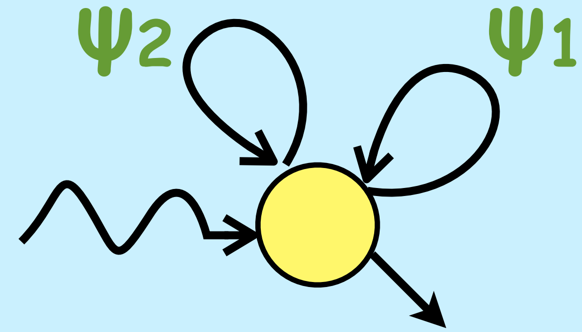
1. $f_i(\vec{x}) \ge 0$ **non-negative**
2. $f_i(\vec{x})-f_i(\vec{x}') \ge 1$ **decreasing**
3. $\forall j<i.\ f_j(\vec{x})-f_j(\vec{x}') \ge 0$ **non-increasing**

- **BG-LLRF** [Ben-Amram and Genaim, JACM'14]: $\forall j \le i.\ f_j(\vec{x}) \ge 0$
- **ADFG-LLRF** [Alias et al., SAS'10]: ...
- **BMS-LLRF** [Bradley et al., CAV'05]: ...
- **MΦRF**: $\forall j<i.\ f_j(\vec{x})-f_j(\vec{x}') \ge 1$



Examples of programs with **LLRFs**

```
while (x ≥ 0 ∧  y ≥ 0 ) {
   if (*) {
     x := x-1
     y := *
   } else {
     y := y-1
   }
}
```

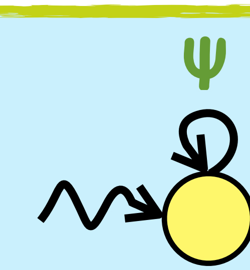


$\psi_1 = \{ x \geq 0, y \geq 0, x' = x-1 \}$

$\psi_2 = \{ x \geq 0, y \geq 0, x' = x, y' = y-1 \}$

(BG,ADFG,BMS)-LLRF <x,y>

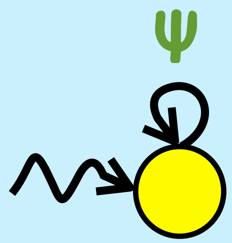
```
while (x ≥ -y ∧ y ≤ 9 ∧ 1 ≥ z ≥ 0) {
   x := x+y+10z-15
   y := y-z
}
```



BG-LLRF <x,y>

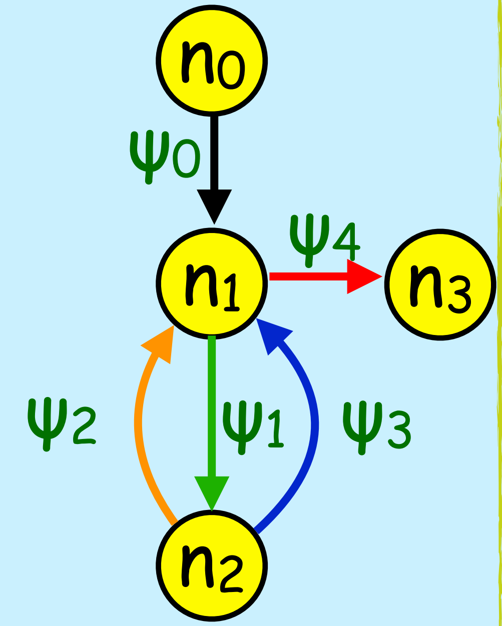
MΦRFs and Multiphase Behaviour

```
while (x ≥ -z) {
   x := x+y
   y := y+z
   z := z-1
}
```



MΦRF ⟨z,y,x⟩

```
while (x ≥ 1) {
   if ( y < z ) {
      y := y+1
   } else {
      x := x-1
   }
}
```



LLRF ⟨z-y-1,x-1⟩

- ⟨z-y-1,x-1⟩ is not a **MΦRF**, but it induces a multiphase behaviour since once a component is negative, it cannot be used anymore.
- if we add y:=y+1 to the else branch, ⟨z-y-1,x-1⟩ would be a **MΦRF** as well.

Outline

- Algorithmic and complexity aspects of **MΦRFs**
  - Mainly for SLC Loops.
  - Inference algorithms.
  - Complexity of decision problems.
  - ...

Based on works with Amir Ben-Amram and Jesús Domenech

- Using control-flow refinement (CFR) for termination analysis of programs with multiphase behaviour
  - Partial evaluation as a CFR technique.
  - Applications of CFR to other analyses.
  - ...

Based on works with John Gallagher and Jesús Domenech

- Concluding remarks.

The (Bounded) MΦRF Problems

Decision problems (d-)MΦRF

Instance: A set of transitions T

Question: Does there exist a MΦRF for T (of length d)?

- The MΦRF problem seeks tuples of any length.
- The d-MΦRF assume that the length d of the tuple is part of the input or the problem.
- We are seeking complexity classification, and corresponding synthesis algorithms for this problem.

d-MΦRFs for SLC Loops

Theorem [Ben-Amram and Genaim, CAV'17]

MΦRFs have the same power as Nested-Linear Ranking Functions **(NLRFs)** for SLC loops

$\langle f_1,\dots,f_d\rangle$ is a **NLRF** for a set of transitions **T** iff

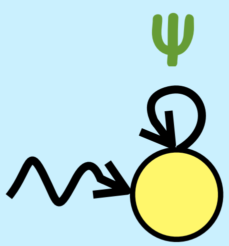
$$\forall\, \vec{x}'' \in T \Rightarrow \Delta f_1(\vec{x}'') \geq 1 \;\wedge$$
$$\Delta f_2(\vec{x}'') + f_1(\vec{x}) \geq 1 \wedge \dots \wedge \Delta f_d(\vec{x}'') + f_{d-1}(\vec{x}) \geq 1 \wedge$$
$$f_d(\vec{x}) \geq 0$$

Notation: $\Delta f_i(\vec{x}'') \equiv f_i(\vec{x}) - f_i(\vec{x}')$

- For SLC loops **T** is a **polyhedron**, so we can use **Farkas' Lemma** to get a complete PTIME synthesis procedure for **NLRFs** [Leike and Heizmann, LMCS 2015].
- It is also complete for general linear-constraint programs, but in such case there is no equivalence to **MΦRFs**.

Example of a **NLRF**

```
while (x ≥ -z) {
   x' = x+y
   y' = y+z
   z' = z-1
}
```



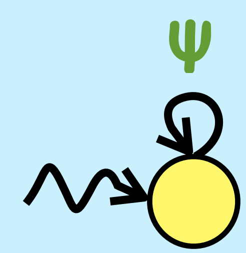
- The loop has the **MΦRF** $\langle z,y,x \rangle$, which is not a **NLRF** since, for example, x if not non-negative on all states.
- but it has a **NLRF** $\langle z,y+z,x+z \rangle$

MΦRFs vs. LLRFs for SLC loops

Theorem [Ben-Amram and Genaim, CAV'17]

If a SLC loop has a **LLRF** of length **d**, then it has a **MΦRF** of length **d**

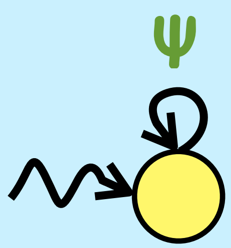
```
while (x ≥ -y ∧ y ≤ 9 ∧ 1 ≥ z ≥ 0) {
   x' = x+y+10z-15
   y' = y-z
}
```



- It has the **LLRF** ‹y,x› which is not a **MΦRF** (y does not decrease on all transitions) …
- … but it has the **MΦRF** ‹x+10y,25x+25y+6›

Loop Bounds from MΦRFs

```
while (x ≥ 0) {
    x' = x+y
    y' = y-1
}
```



- This loop has a **MΦRFs** ⟨y,x⟩, can we use it to obtain a loop bound?
- Can we infer loop bounds for SLC loops that have **MΦRFs** in general?

$(x_0,y_0) \rightarrow (x_0+y_0,y_0-1) \rightarrow (x_0+y_0+(y_0-1),y_0-2) \rightarrow \ldots \rightarrow (x_0+O(y_0^2),-1)$

Theorem [Ben-Amram and Genaim, CAV'17]

- **MΦRFs** imply linear loop bounds for SLC loops (a linear combination of its components).
- **NLRFs** imply linear loop bounds for general linear-constraint programs.

The UnBounded Version of MΦRF

- One can apply the **d-MΦRF** iteratively, which guarantees finding a **MΦRF** if one exists, but what if it does not exist? When to stop?
- Is there a theoretical bound on the length of the **MΦRFs**, given the loop?
- We are not aware of any such length-bound, and, moreover, unlike the case of **BG-LLRFs**, it does not depend only on the number of variables (or constraints) [Ben-Amram and Genaim, CAV'17].
- We are seeking a more direct algorithm, which is not based on **d-MΦRF** [Ben-Amram, Domenech, and Genaim, SAS'19].

Synthesising BG-LLRFs

$\langle f_1,\ldots,f_i,\ldots,f_d \rangle$ is a **BG-LLRF** for **Q** iff for each $(\vec{x},\vec{x}') \in Q$ there is $1 \leq i \leq d$ such that

1.	$\forall j \leq i \quad f_j(\vec{x}) \geq 0$	**non-negative**
2.	$f_i(\vec{x})-f_i(\vec{x}') \geq 1$	**decreasing**
3.	$\forall j<i. \; f_j(\vec{x})-f_j(\vec{x}') \geq 0$	**non-increasing**

- $f_1(\vec{x}) \geq 0$ and $f_1(\vec{x})-f_1(\vec{x}') \geq 0$ holds **for any** $(\vec{x},\vec{x}') \in Q$
- $f_1(\vec{x})-f_1(\vec{x}') \geq 1$ holds **for some** $(\vec{x},\vec{x}') \in Q$
- We continue with the SLC loop $Q_1 \equiv Q \wedge f_1(\vec{x})-f_1(\vec{x}') \leq 0$
- There is an optimal choice for f_1

Synthesising MΦRFs

$\langle f_1,\ldots,f_i,\ldots,f_d \rangle$ is a **MΦRF** for **Q** iff for each $(\vec{x},\vec{x}') \in$ **Q** there is $1 \leq i \leq d$ such that

1.		$f_i(\vec{x}) \geq 0$	**non-negative**
2.	$\forall j \leq i$	$f_j(\vec{x})-f_j(\vec{x}') \geq 1$	**decreasing**

- $f_1(\vec{x})-f_1(\vec{x}') \geq 1$ holds **for any** $(\vec{x},\vec{x}') \in$ **Q**
- $f_1(\vec{x}) \geq 0$ holds **for some** $(\vec{x},\vec{x}') \in$ **Q**
- We continue with the SLC loop $Q_1 \equiv Q \wedge f_1(\vec{x}) \leq 0$
- Is there an optimal choice for f_1? Unfortunately no ...

Synthesising MΦRFs

From the equivalence of **MΦRF** and **NLRFs**, we know that if a **SLC** loop **Q** has a **MΦRF**, then it has one of optimal length $\langle f_1,\ldots,f_d \rangle$ where the f_d is non-negative on all enabled states, i.e., $f_d(\vec{x}) \geq 0$ for any $(\vec{x},\vec{x}') \in Q$

- $g(\vec{x}) \geq 0$ holds **for any** $(\vec{x},\vec{x}') \in Q$
- $g(\vec{x})-g(\vec{x}') > 0$ holds **for some** $(\vec{x},\vec{x}') \in Q$
- Continue with the SLC loop $Q_1 \equiv Q \wedge g(\vec{x})-g(\vec{x}') \leq 0$
- If we succeed to build a **MΦRF** τ of length k for Q_1, then we can use g and τ to get one of length $k+1$ for **Q** (the last component is a combination of g and τ)

Synthesising MΦRFs

- The set of all candidates g that satisfy $g(\vec{x}) \geq 0$ for all $(\vec{x},\vec{x}') \in Q$ is a polyhedral cone, and thus it is finitely generated by some function $g_1,\ldots,g_k$.
- Any such g can be written as $\Sigma a_i * g_i$ for some $a_i \geq 0$.
- If $g(\vec{x})-g(\vec{x}') > 0$ holds for some $(\vec{x},\vec{x}') \in Q$ then $g_i(\vec{x})-g_i(\vec{x}') > 0$ must hold for some g_i.
- $Q_1 \equiv Q \wedge g_1(\vec{x})-g_1(\vec{x}') \leq 0 \wedge \ldots \wedge g_k(\vec{x})-g_k(\vec{x}') \leq 0$.
- If we succeed to build a MΦRF τ of length k for Q_1, then we can use $g_1,\ldots,g_k$ and τ to build one of length $k+1$ for Q.

(semi-)Deciding Existence MΦRFs

decideMΦRF(Q) {
- if Q is empty, **return YES**
- Compute the generators $g_1,\ldots,g_k$ of the cone of non-negative function (over the enabled states)
- $Q' = Q \wedge g_1(\vec{x})-g_1(\vec{x}') \leq 0 \wedge \ldots \wedge g_k(\vec{x})-g_k(\vec{x}') \leq 0$
- **return** decideMΦRF(Q')

}

- If Q has a **MΦRF** of optimal length **d** the algorithm will make exactly **d** recursive calls.
- The algorithm diverges if Q has no **MΦRF**.

No Progress and Infinite Progress

decideMΦRF(Q) {

- if Q is empty, **return YES**
- Compute the generators $g_1,\ldots,g_k$ of the cone of non-negative function (over the enabled states)
- $Q' = Q \wedge g_1(\vec{x})-g_1(\vec{x}') \leq 0 \wedge \ldots \wedge g_k(\vec{x})-g_k(\vec{x}') \leq 0$
- if Q' == Q, **return Q is a recurrent set**
- **return** decideMΦRF(Q')

}

The algorithm can also make infinite progress, when Q is terminating and when Q is non-terminating

$$Q = Q_0 \supset Q_1 \supset Q_2 \supset Q_3 \supset \ldots$$

Better understanding of MΦRFs

We have an algorithm that does not completely solve the **MΦRF** problem we wanted to solve, but …

- Reveals an interesting relation between seeking **MΦRFs** and seeking (monotonic) **recurrent sets**.
- We used its properties to find classes of programs for which **MΦRF** are enough, e.g., octagonal relations.
- If Q_d is not empty, it explains why the loop does not have a **MΦRF** of length **d** (useful for conditional termination).
- Left us with several new research directions and open problems …

What about General Programs?

- All what we have seen so far works only for the case of SLC loops.
- The **d-MΦRFs** problem for general linear-constraint programs is decidable, and is at least NP-Hard, unlike PTIME for SLC loops.
- **NLRFs** can still be used for general programs, but for such case they are weaker than **MΦRFs** — there are programs that have **MΦRFs** but not **NLRFs**.
- How we can prove termination of programs that need **MΦRFs** (or have multiphase behaviour)?

Encoding to SMT Formulas

- Leike and Heizmann [LMCS'15] encode the **d-MΦRF** conditions as non-linear SMT formula:
  - The first component decreases for all transitions, and if it is negative the second decreases, etc.
  - Satisfiability implies existence of **MΦRF** of length d (the models define the components).
  - Complete for real variables
- Brockschmidt et al. [TACAS'17] do it incrementally
  - Infer the components of the **MΦRF** one at a time using conditional termination and safety analyser
  - Non-linear SMT formulas
  - Not complete

Simplifying the Control-Flow

```
while (x ≥ 1) {
  if ( y < z ) {
    y := y+1
  } else {
    x' := x-1
  }
}                LLRF <z-y-1,x-1>
```

```
while (x ≥ 1 ∧ y < z) {
    y := y+1
}                LRF: z-y-1

while (x ≥ 1 ∧ y ≥ z) {
    x := x-1
}                LRF: x-1
```

It is easier to prove termination of the one on the right, and also prove that its runtime is linear

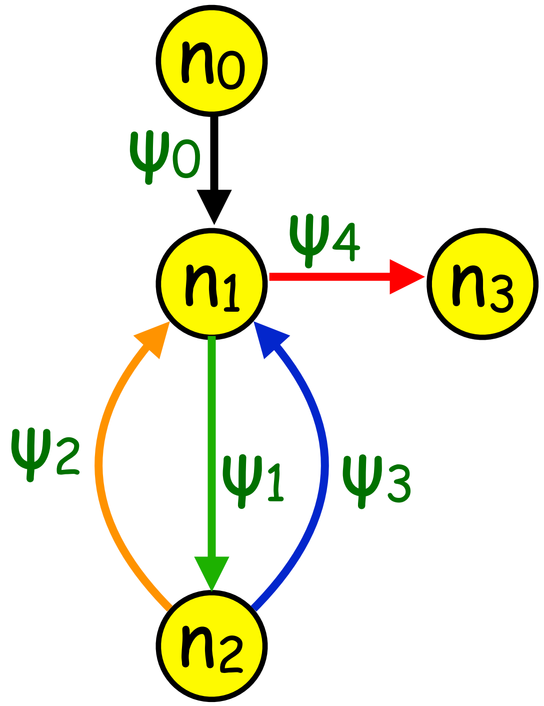
Control-Flow Refinement (CFR)

- Control-Flow refinement was already used, e.g., for cost analysis and invariants generation
  - Gulwani et al. [PLDI'09]
  - Sharma et al. [CAV'11]
- These techniques develop program transformations from scratch, and tailored to the very specific application (cost, invariants, etc)
- We wanted to explore the use of a general purpose program transformation techniques to refine the control-flow in multiphase programs [Domenech, Gallagher, and Genaim, TPLP'19]

CFR via Partial Evaluation

- We started from a partial evaluator for horn-clause programs [John P. Gallagher [VPT 2019]]
- It is based on performing unfolding and abstraction
  - Unfolding is like executing parts of the program
  - Abstraction is applied to loop head predicates, using a finte set of abstract properties, to guarantee termination of the process
- Our linear-constraint programs can be translated to (linear) horn-clause programs (and back)

Example




```
while (x ≥ 1) {
   if ( y < z ) {
      y := y+1
   } else {
      x' := x-1
   }
}      LLRF <z-y-1,x-1>
```

Properties for n_1:

$\phi_1=\{x\geq 1\}$

$\phi_2=\{y\geq z\}$

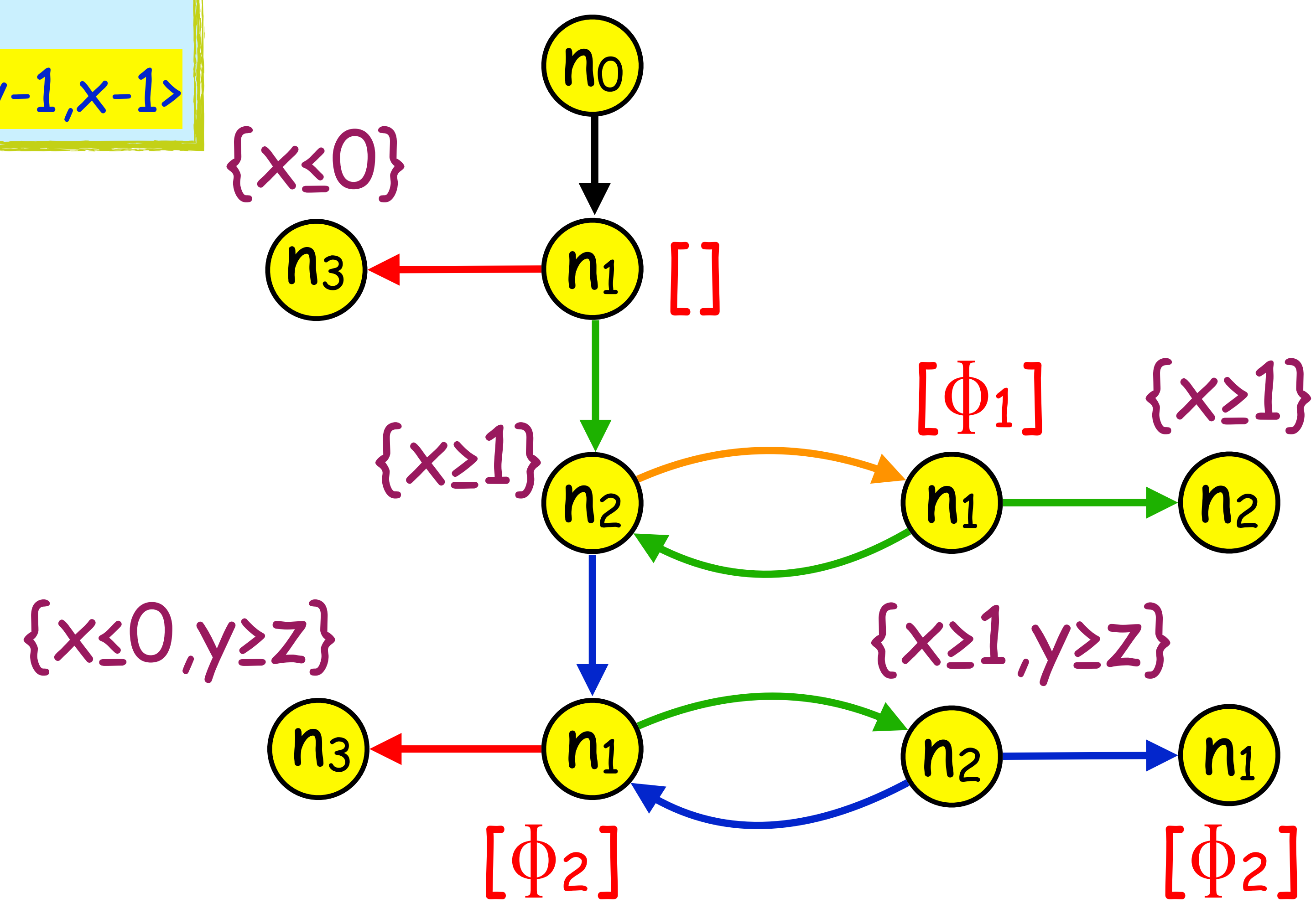
$\psi_0=\{x'=x,y'=y,z'=z\}$

$\psi_1=\{x\geq 1,x'=x,y'=y,z'=z\}$

$\psi_2=\{y<z,x'=x,y'=y+1,z'=z\}$

$\psi_3=\{y\geq z,x'=x-1,y'=y,z'=z\}$

$\psi_4=\{x\geq 1,x'=x,y'=y,z'=z\}$



Inference of Properties

- We use several heuristics/schemes
  - Extract them from constraints on outgoing/ incoming edges of loop heads
  - Propagate conditions backwards/forwards from loop bodies to loop heads
  - Use concrete intervals for variables, such as x>=1, y<=100,… taken from outgoing/incoming edges of loop heads

Granularity of CFR

- Applying CFR to the whole program is not practical for large programs.
- We have incorporated CFR in a termination analyser with different levels of granularity
  - Apply to the whole program
  - Apply at the level of SCCs
  - Apply only to parts that we could not prove terminating, etc.

Benefits of using CFR (experimentally)

- More precise termination analysis
  - our tool could prove termination of programs in the last termination competition, due to the use of CFR, that no one could handle
  - simpler ranking functions due to phase splitting
  - more precise invariants due to case splitting
- More precise cost analysis (for the same reasons)
  - We use off-the-shelf cost analyser, applied after CFR of the whole program
- More precise assertions checker
  - due to more precise invariants

iRankFinder

- All techniques are implemented in a termination analyser that supports:
  - **LRF**, different kinds of **LLRFs**, **MΦRFs**, tuples of **NLRFs** (similar to polyranking of Bradley et al.)
  - Non-termination using the **MΦRFs** algorithm, but applied to closed-walks instead of SLC loops
  - Includes a CFR component
- The CFR component can be used independently, so other tools can take advantage of it.
- Assertions checking, invariants generation, ...
- All available at: http://loopkiller.com

Concluding Remarks

- Multiphase ranking functions (**MΦRFs**)
  - For SLC loops: algorithms, complexity, relation to non-termination, witnesses, etc.
  - For general linear-constraint programs we know vert little, and further research is needed.

- Control-Flow Refinement of Multiphase programs
  - A proof of concept that general purpose program transformations can be use for CFR.
  - Not only for termination.
  - Future work should explore other applications, and also the use of CFR for programs with non-numerical variables.

Thank You!