Procedure Summarization via Vector Addition Systems and Inductive Linear Bounds

General Exam

Nikhil Pimpalkhare

October 2023

Introduction

Background

Introduction

Background

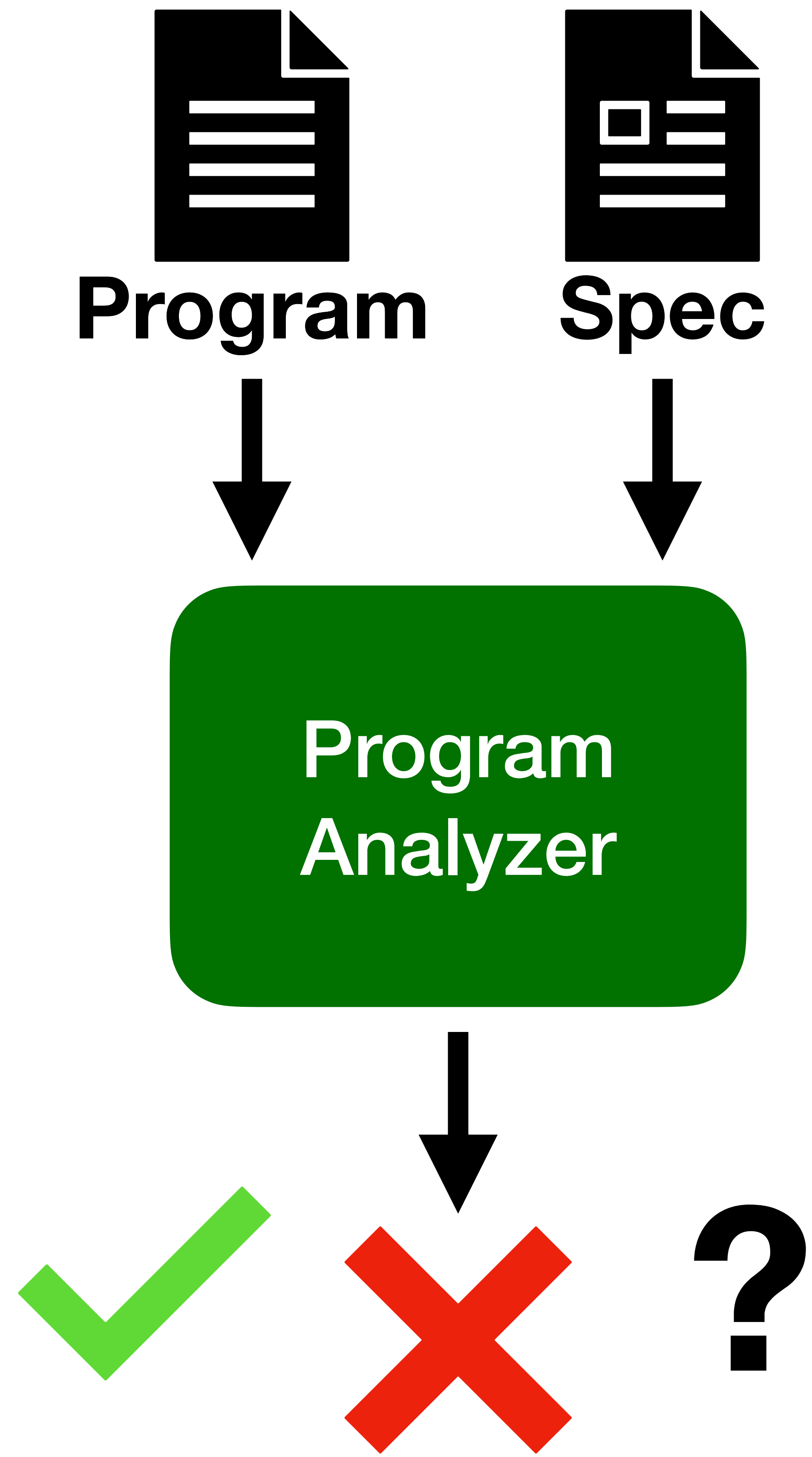
- Program Analysis: how much can we understand about the *runtime* behavior of a program from its *static* representation

Introduction

Background

- Program Analysis: how much can we understand about the *runtime* behavior of a program from its *static* representation

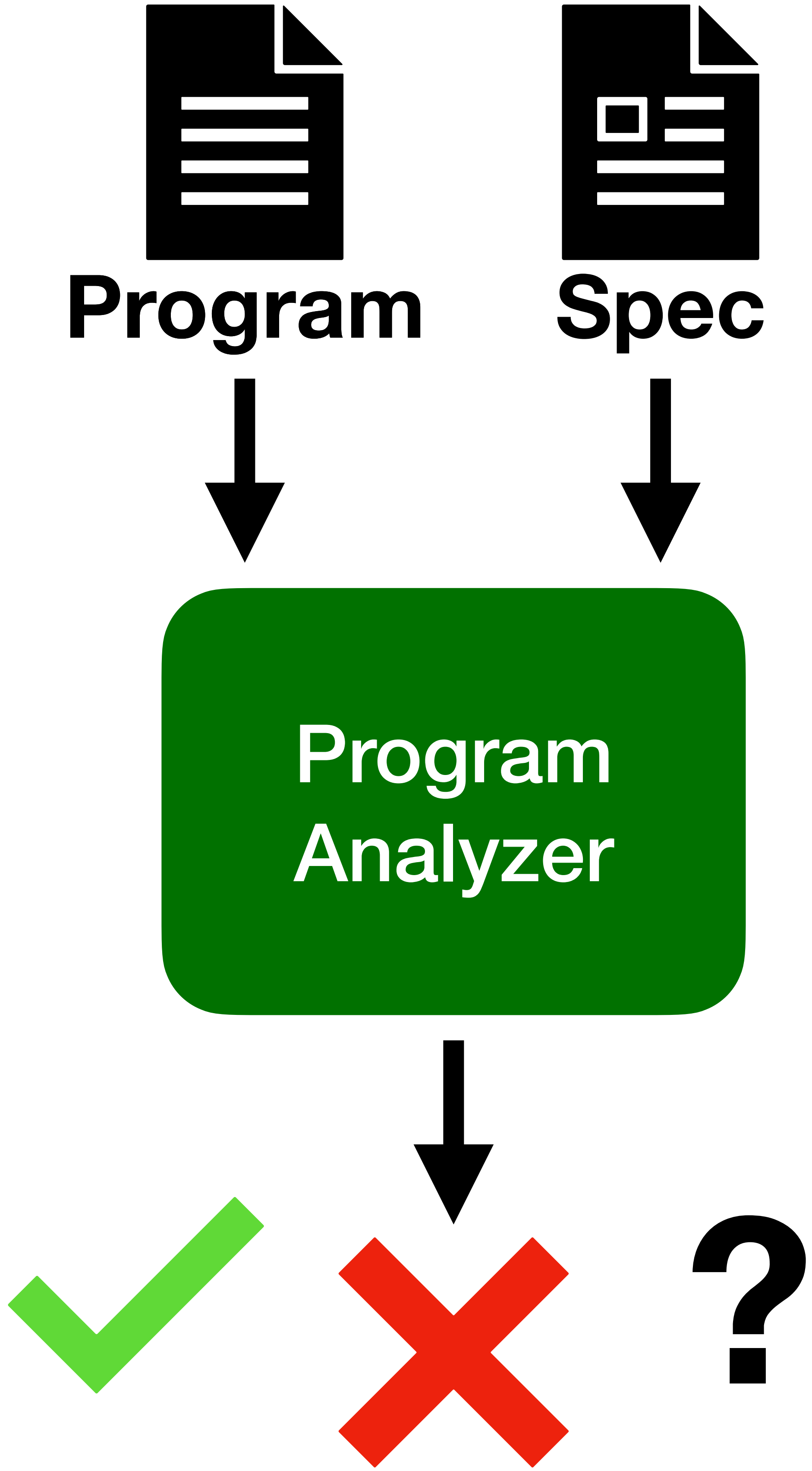
- A query: a program and a logical specification of desired behavior



Introduction

Background

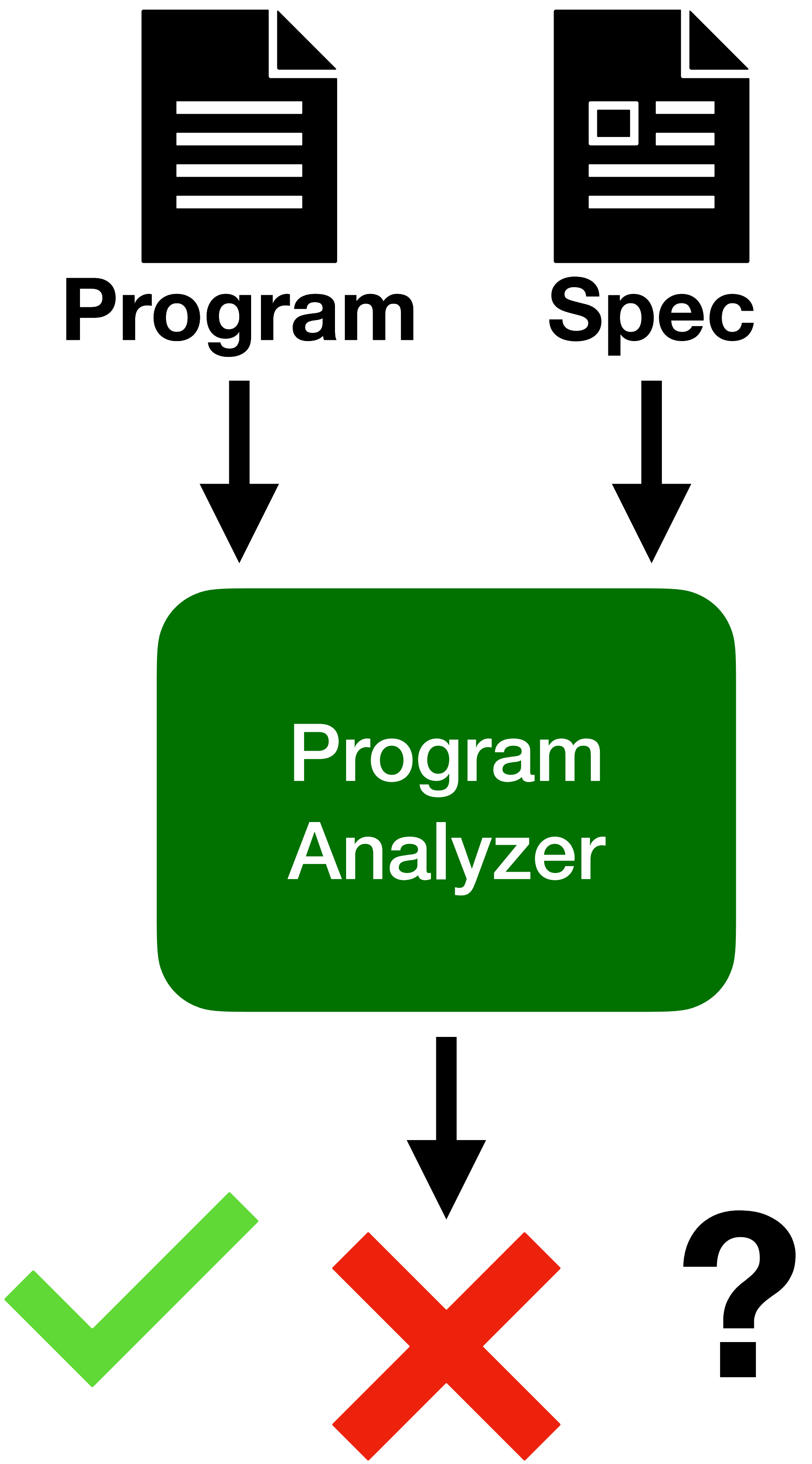
- Program Analysis: how much can we understand about the *runtime* behavior of a program from its *static* representation
- A query: a program and a logical specification of desired behavior
- “Necessary” Property: Soundness



Introduction

Background

- Program Analysis: how much can we understand about the *runtime* behavior of a program from its *static* representation
- A query: a program and a logical specification of desired behavior
- "Necessary" Property: Soundness
- Desirable Property: *Predictability*
    - Changes to a program should have a predictable impact on its analysis

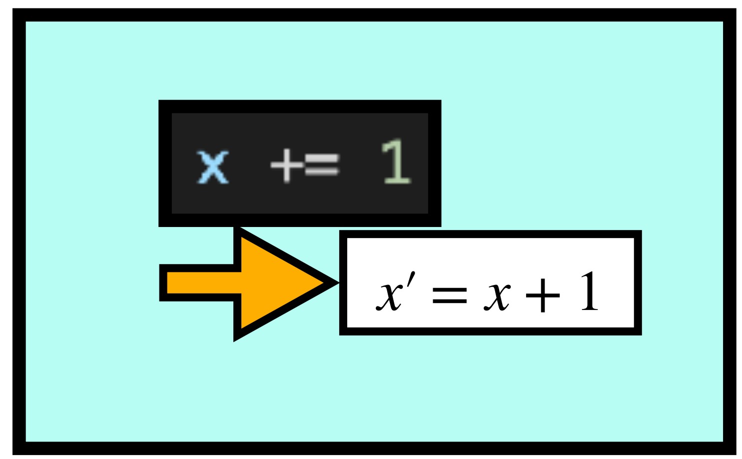


Introduction

Background

- Intra-procedural Analysis: Perform program analysis in single procedures
    - Core Algorithmic Challenge: Loop Invariants
- **Inter-procedural Analysis:** Perform program analysis in the presence of recursive calls
    - Core Algorithmic Challenge: **Procedure Summarization**
- Summaries + Intra = Inter

Transition Formula: a (LIRA) formula over a set of program variables X and primed copies X' describing the pre and post state of a transition respectively



An Example Analysis Task

```
int mem_ops, buf;
void save_tree(int size) {
    buf += 1;
    if (size <= 1) {
        mem_ops += buf;
        buf = 0;
    } else {
        save_tree((size - 1) / 2);
        save_tree((size - 1) / 2);
    }
}
```

```
void main() {
    mem_ops = 0; buf = 0;
    int size = nondet_int();
    assume(size >= 1);
    save_tree(size);
    assert(mem_ops <= size);
}
```

What is this task?

- Integer model of a program that recurses over a balanced binary tree, saving each node's value in an array and writing the array to disk whenever a leaf is reached

Why does the assertion hold?

- mem_ops + buf is incremented by one for each recursive call made
- buf must be set to zero at the end of any terminating execution
- There are at most *size* recursive calls made

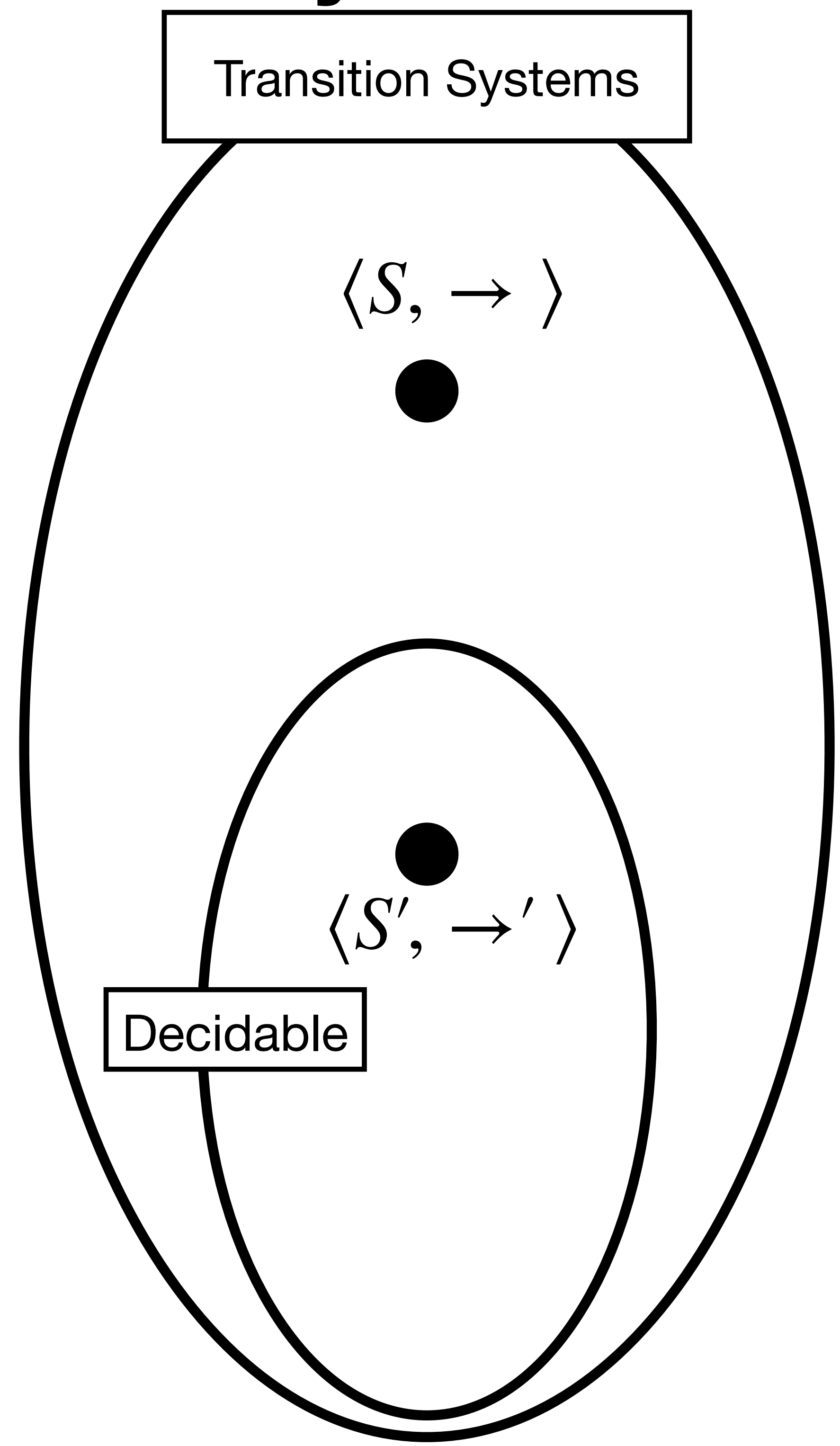
The Best Abstraction Recipe

How do we analyze programs predictably?

The Best Abstraction Recipe

How do we analyze programs predictably?

- A program can be viewed as a transition system $\langle S, \rightarrow \rangle$ where S is the state space and $\rightarrow \subseteq S \times S$ describes transitions

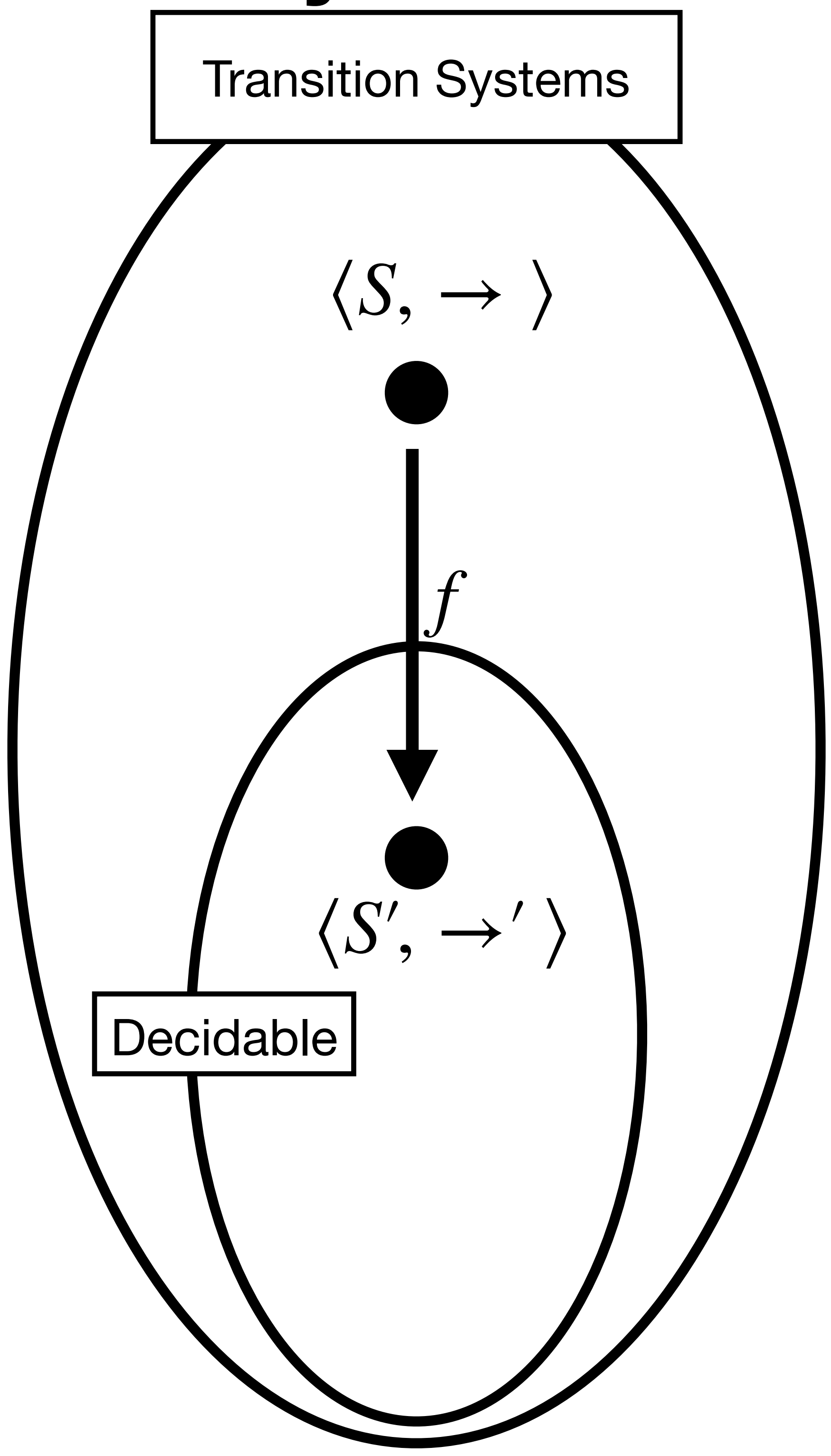


The Best Abstraction Recipe

How do we analyze programs predictably?

- A program can be viewed as a transition system $\langle S, \rightarrow \rangle$ where S is the state space and $\rightarrow \subseteq S \times S$ describes transitions

- $f : S \rightarrow S'$ is a simulation between $\langle S, \rightarrow \rangle$ and $\langle S', \rightarrow' \rangle$ if for all $u, u' \in S$, if $u \rightarrow u'$ then $f(u) \rightarrow' f(u')$



The Best Abstraction Recipe

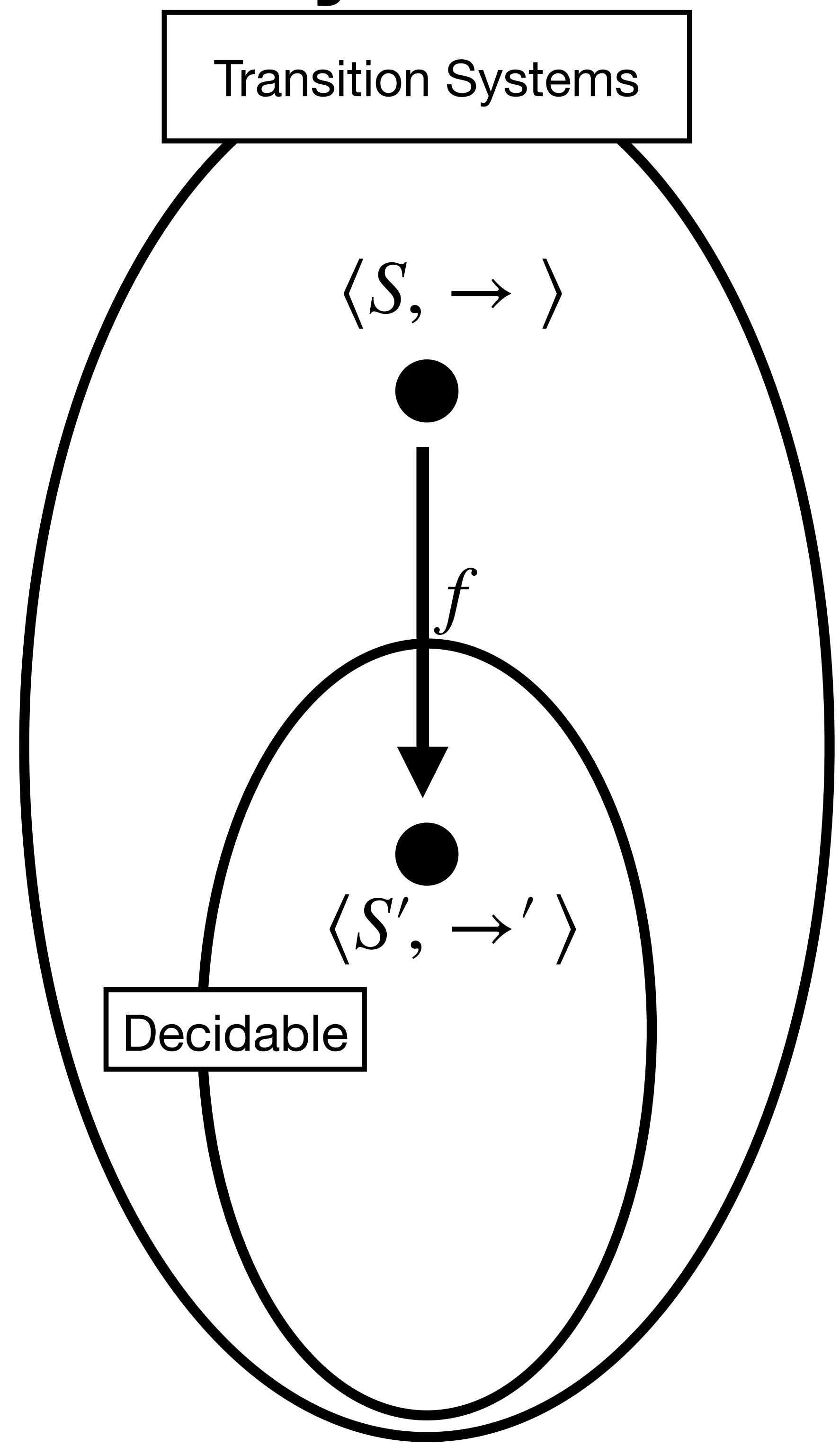
How do we analyze programs predictably?

- A program can be viewed as a transition system $\langle S, \to \rangle$ where S is the state space and $\to \subseteq S \times S$ describes transitions

- $f : S \to S'$ is a simulation between $\langle S, \to \rangle$ and $\langle S', \to' \rangle$ if for all $u, u' \in S$, if $u \to u'$ then $f(u) \to' f(u')$

- A simulation implies that an algorithm for the reachability of $\langle S', \to' \rangle$ can be used to over-approximate the reachability of $\langle S, \to \rangle$, as:

$$\{u, u' : u \to u'\} \subseteq \{u, u' : f(u) \to' f(u')\}$$

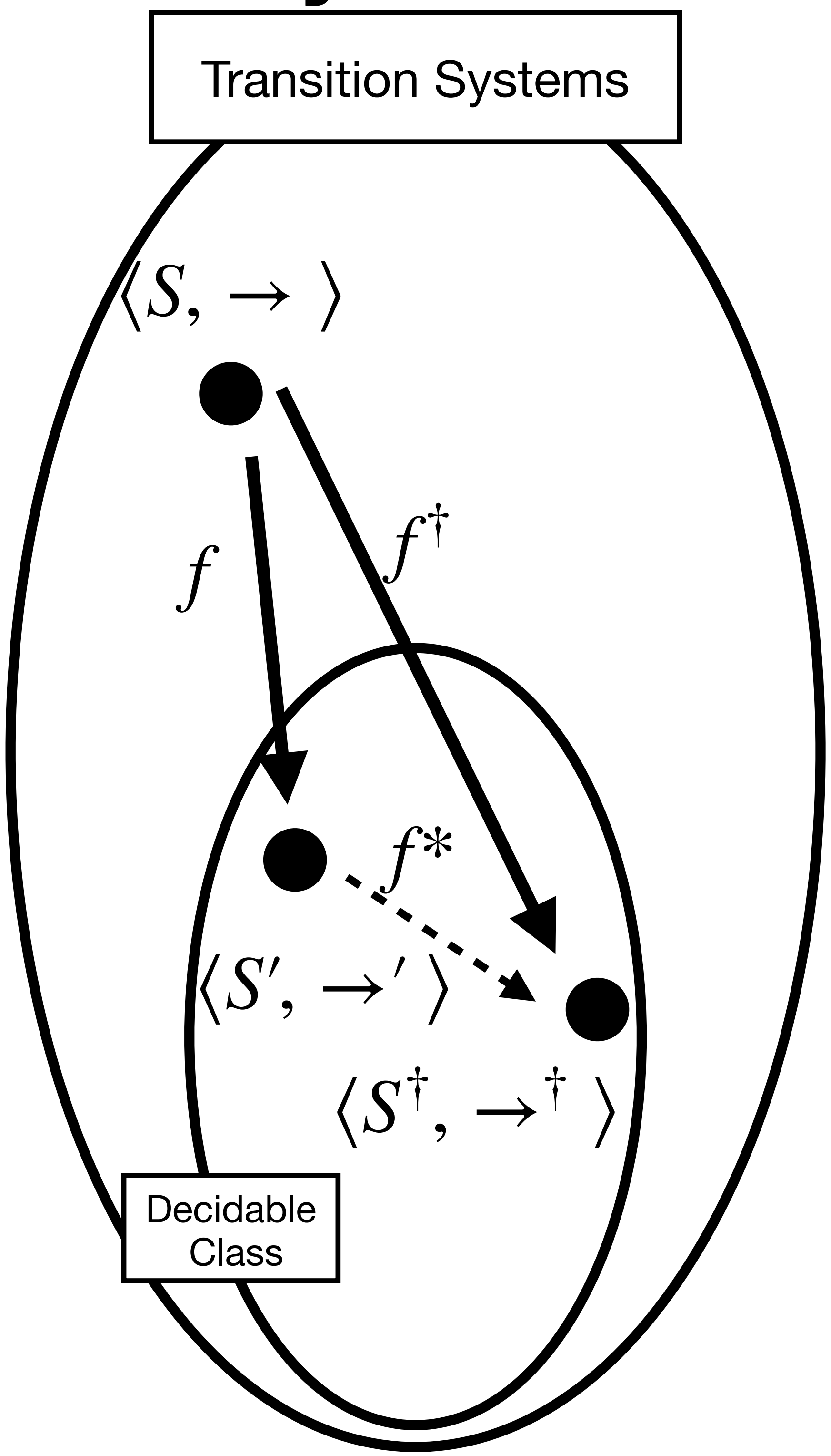


The Best Abstraction Recipe

How do we analyze programs predictably?

- An abstraction of $\langle S, \rightarrow \rangle$ is another transition system $\langle S', \rightarrow' \rangle$ and a simulation f to it

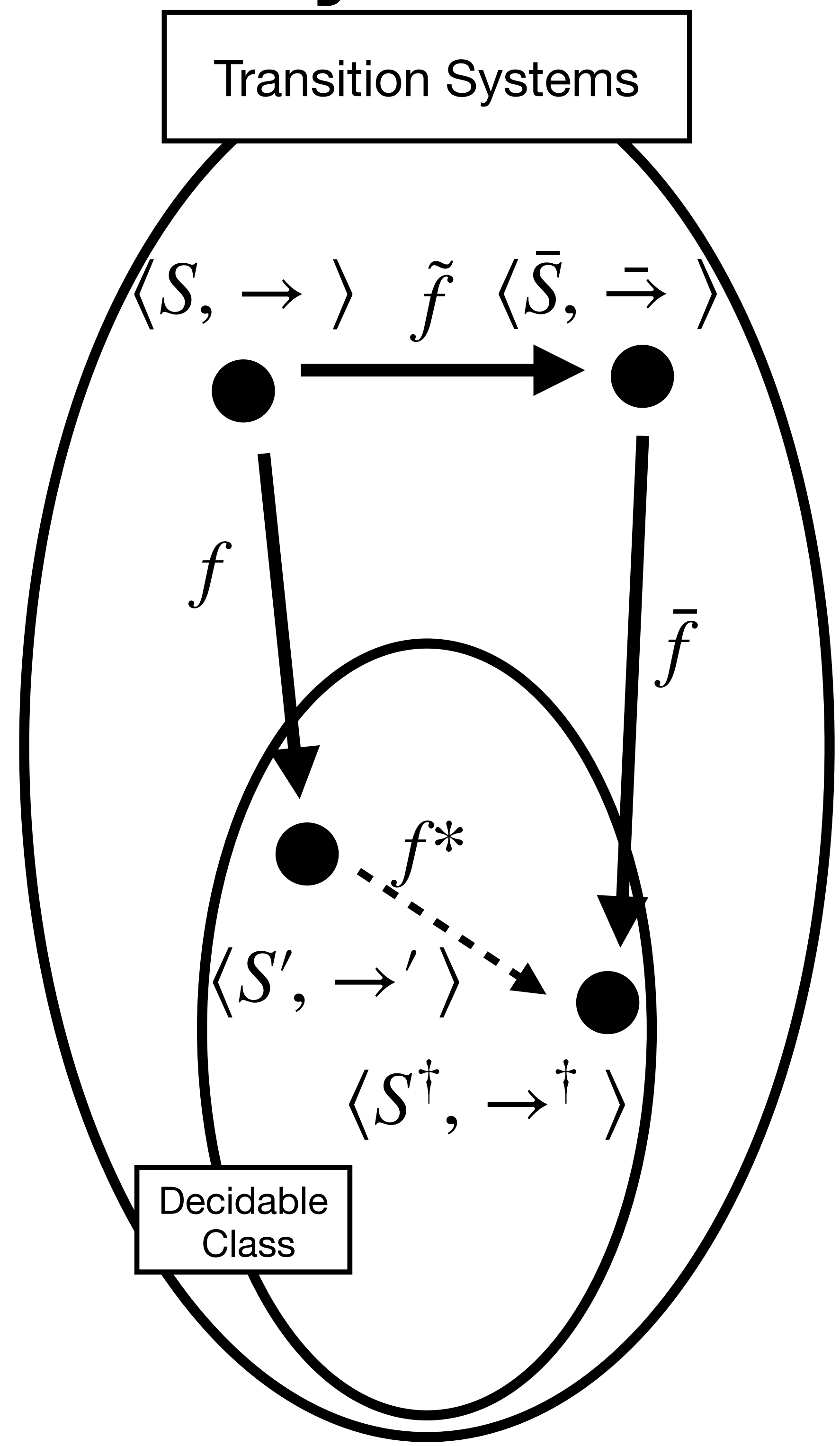
- An abstraction is **best** if for any other abstraction in the same class $\langle S^\dagger, \rightarrow^\dagger \rangle$ and $f^\dagger$, there is a simulation f^* from $\langle S', \rightarrow' \rangle$ to $\langle S^\dagger, \rightarrow^\dagger \rangle$



The Best Abstraction Recipe

How do we analyze programs predictably?

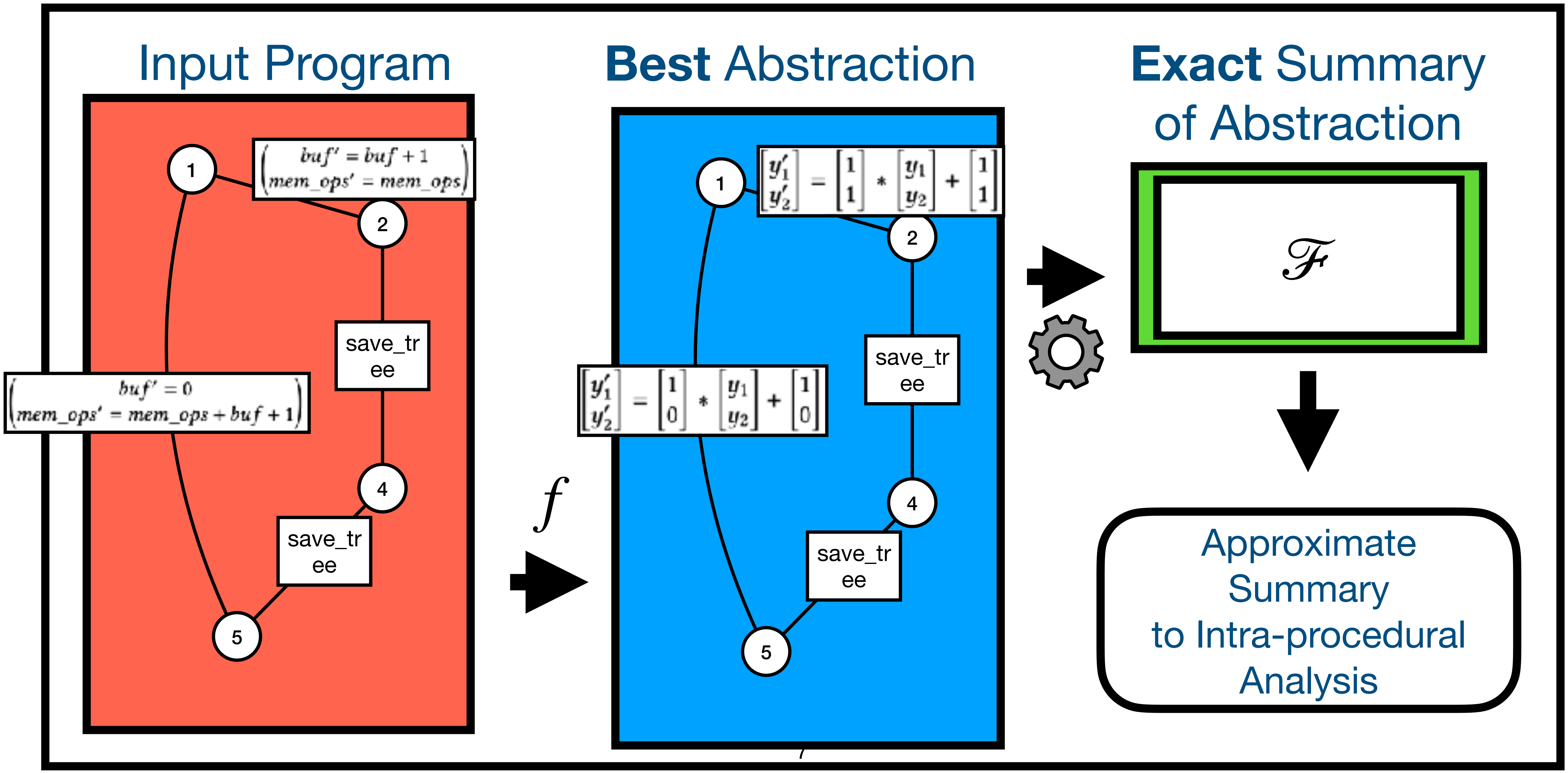
- An abstraction of $\langle S, \rightarrow \rangle$ is another transition system $\langle S', \rightarrow' \rangle$ and a simulation f to it
- An abstraction is **best** if for any other abstraction in the same class $\langle S^{\dagger}, \rightarrow^{\dagger} \rangle$ and $f^{\dagger}$, there is a simulation f^{*} from $\langle S', \rightarrow' \rangle$ to $\langle S^{\dagger}, \rightarrow^{\dagger} \rangle$
- Best abstractions lead to **monotone** over-approximations: if there is a simulation $\tilde{f}$ from $\langle S, \rightarrow \rangle$ to $\langle \bar{S}, \bar{\rightarrow} \rangle$, there will be a simulation between their best abstractions



The Best Abstraction Recipe

How do we analyze programs predictably?

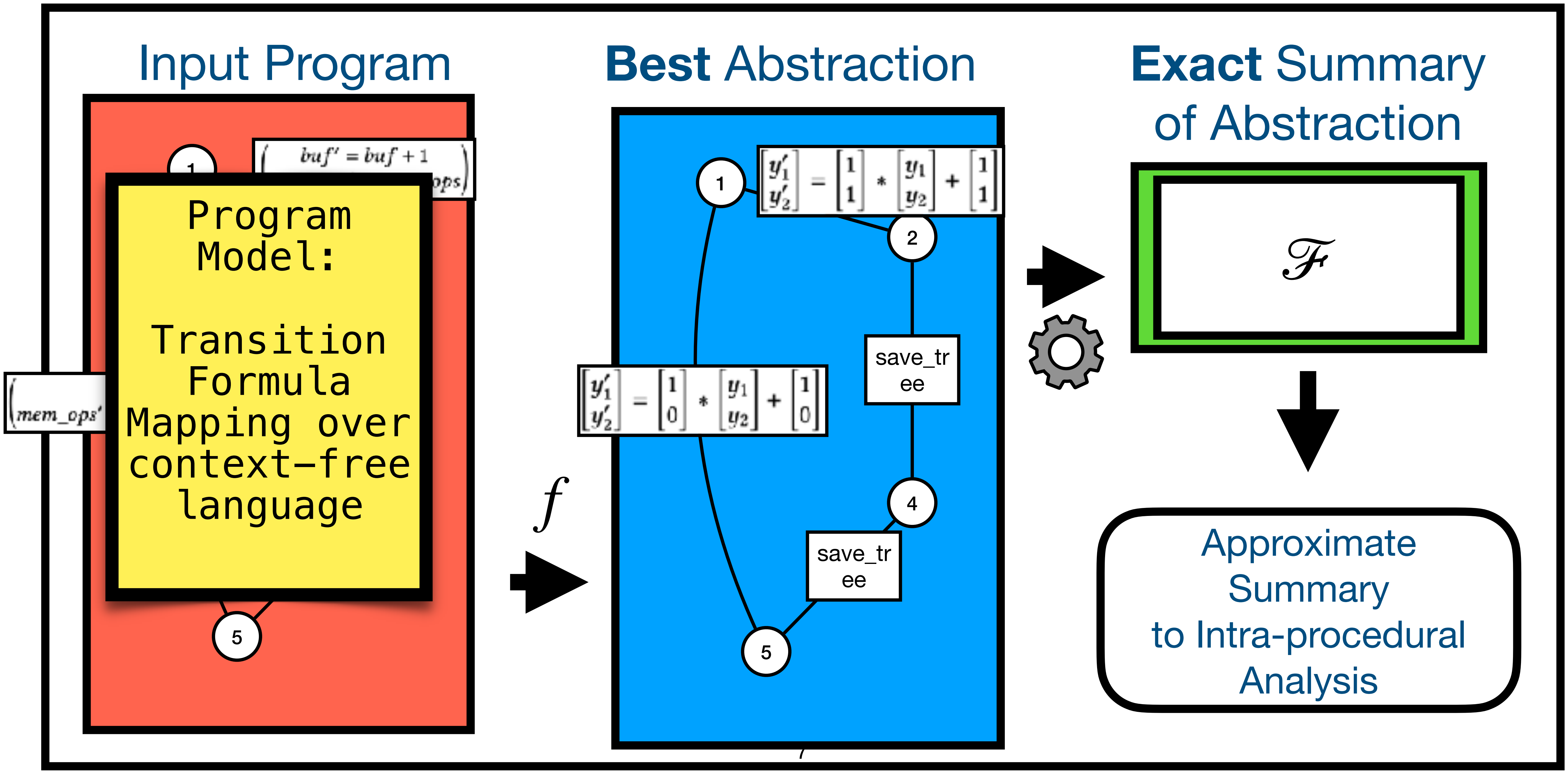
- Ensure “predictability” through *Monotonicity:* a more specific program always results in a more specific summary



The Best Abstraction Recipe

How do we analyze programs predictably?

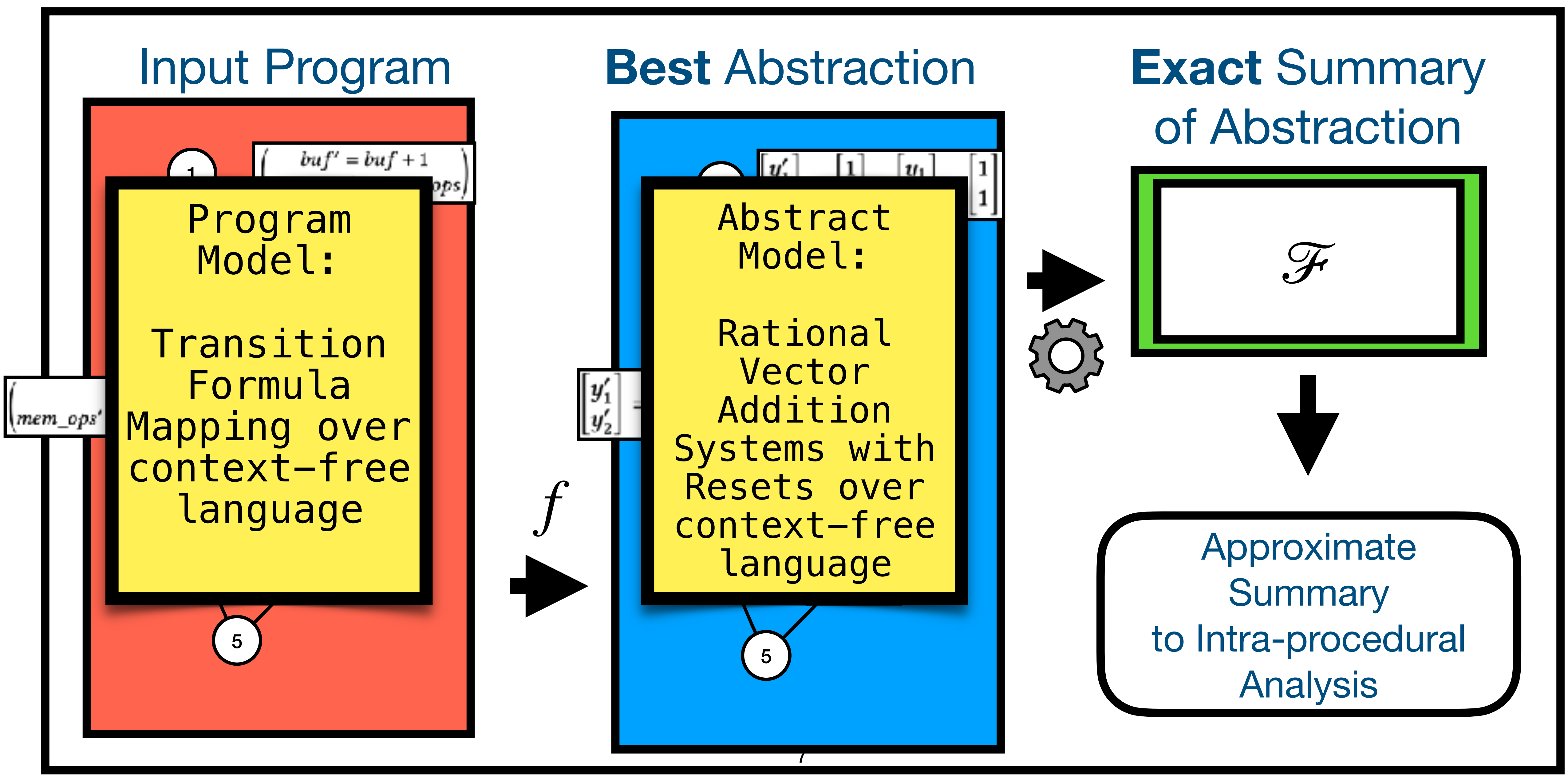
- Ensure “predictability” through *Monotonicity:* a more specific program always results in a more specific summary



The Best Abstraction Recipe

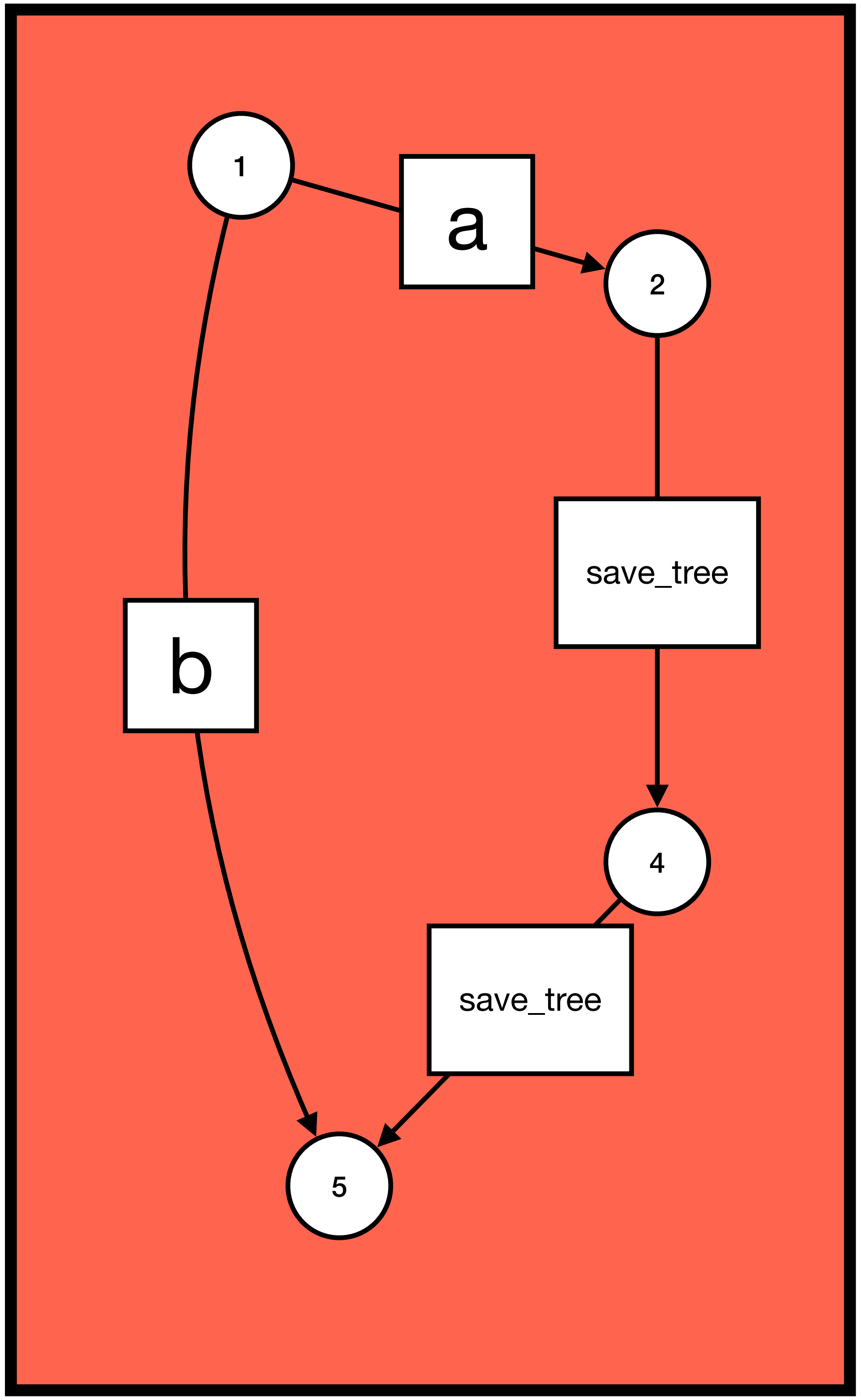
How do we analyze programs predictably?

- Ensure "predictability" through *Monotonicity:* a more specific program always results in a more specific summary



Model of Input Program

- A **program graph** is a directed graph in which nodes represent control locations
  - Every edge carries a label from:
    - a set of standard edges Σ
    - a set of procedures P
  - A program graph is additionally equipped with two functions $\mathit{in} : P \to V$ and $\mathit{out} : P \to V$ which map procedures to their entry and exit vertices respectively

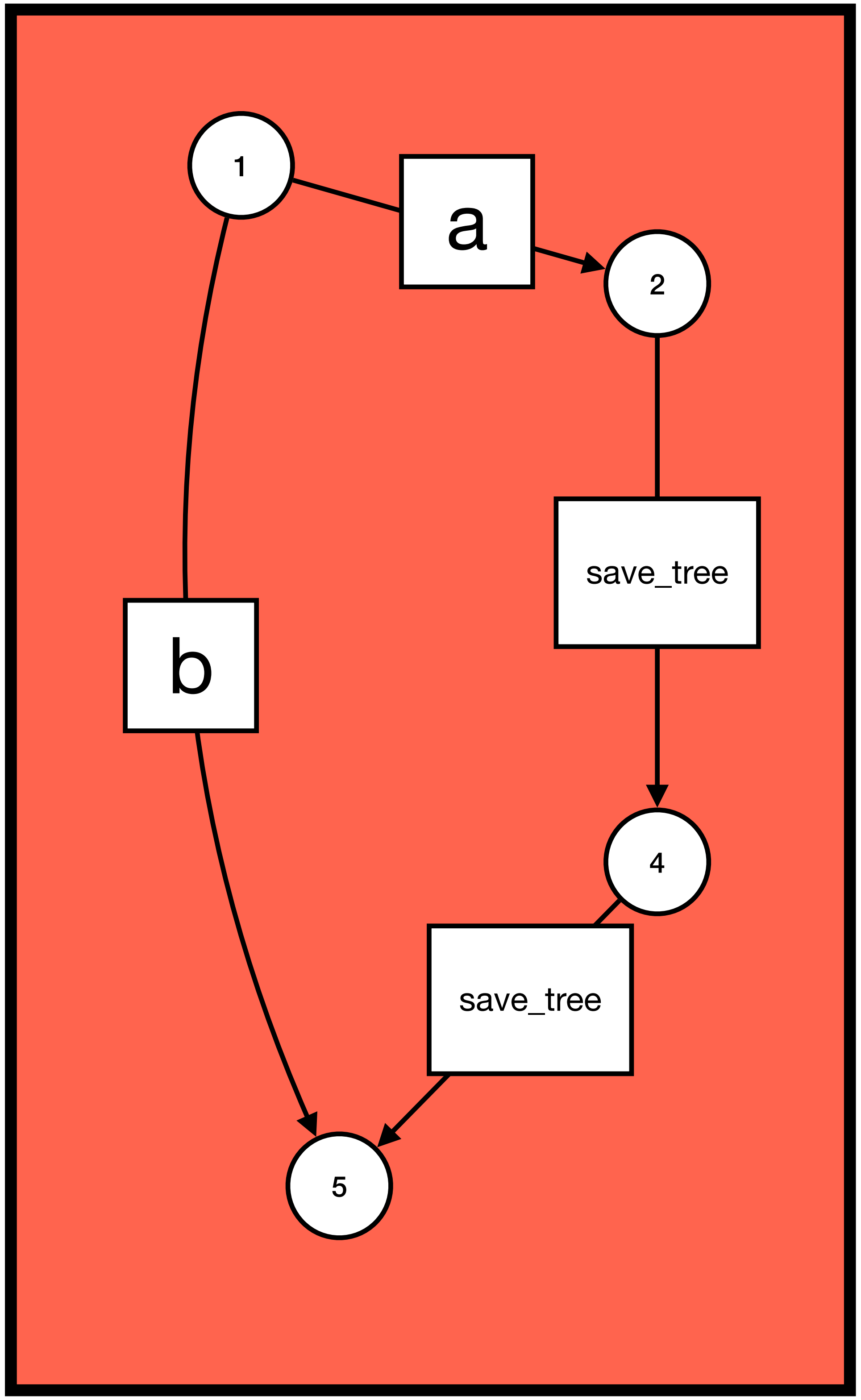


Model of Input Program

- A **trajectory** through a procedure p in a program graph is a sequence in Σ^* corresponding to a sequence of edges in $(\Sigma \cup P)^*$ forming a path from $in(p)$ to $out(p)$ in which every element $p' \in P$ has been replaced with a trajectory through p'

- Programs are understood as a program graph and a transition formula mapping $tf : \Sigma \rightarrow TF(X)$ representing the state transformation

- The semantic meaning of a trajectory can be computed by composing the transition formulas of each edge in order.



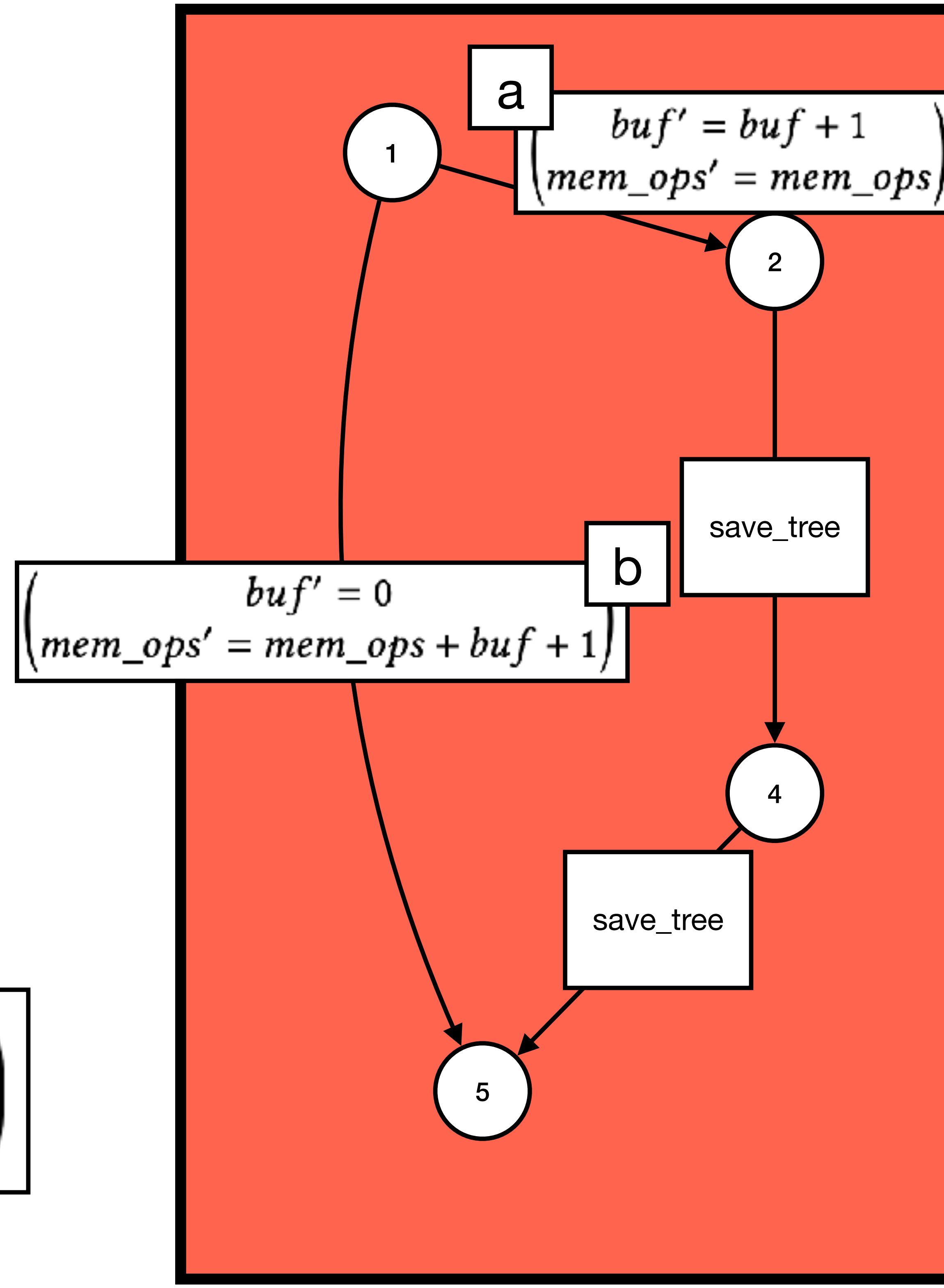
Model of Input Program

Example Execution

- An example trajectory: abb

$$\begin{pmatrix} buf' = 0 \\ mem\_ops' = mem\_ops + buf + 1 \end{pmatrix} \circ \begin{pmatrix} buf' = 0 \\ mem\_ops' = mem\_ops + buf + 1 \end{pmatrix} \circ \begin{pmatrix} buf' = buf + 1 \\ mem\_ops' = mem\_ops \end{pmatrix} =$$

$$\begin{pmatrix} buf' = 0 \\ mem\_ops' = mem\_ops + buf + 3 \end{pmatrix}$$



Vector Addition System with Resets

- Vector Addition Systems [Karp, Miller 1969] are classically used to model parallel computing/distributed systems

- Rational Vector Addition Systems with Resets (VASR) transformations are the restricted subclass of transition formulas which can be written as:

$$\overrightarrow{X'} = \vec{r} * \overrightarrow{X} + \vec{a}$$

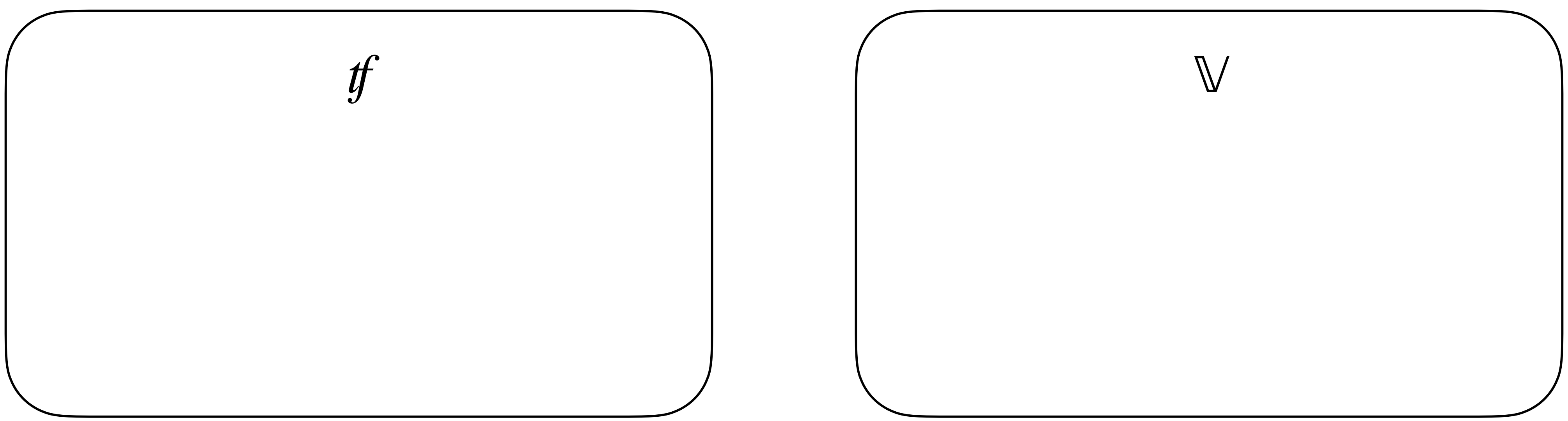
where $\vec{r} \in \{0,1\}^{|X|}$, $\vec{a} \in \mathbb{Q}^{|X|}$, and $*$ is elementwise product

Ex: $x' = 1 * x + 3 \wedge y' = 0 * y + 0$

- We consider VASRs over rational numbers instead of over naturals as the reachability of the latter is Ackermann-complete [Czerwiński 2021]

Vector Addition System with Resets

- A VASR $\mathbb{V}$ is a transition formula mapping where each formula is a VASR transformation
- Letting ρ denote valuations over X, $\mathbb{V}$ simulates tf according to f if…



Vector Addition System with Resets

- A VASR $\mathbb{V}$ is a transition formula mapping where each formula is a VASR transformation

- Letting ρ denote valuations over X, $\mathbb{V}$ simulates $t\!f$ according to f if…

$t\!f$	$\mathbb{V}$
$[\rho, \rho'] \vDash t\!f(s)$	

Vector Addition System with Resets

- A VASR $\mathbb{V}$ is a transition formula mapping where each formula is a VASR transformation
- Letting ρ denote valuations over X, $\mathbb{V}$ simulates tf according to f if…

tf

$$[\rho, \rho'] \models \mathit{tf}(s)$$

→

$\mathbb{V}$

$$[f(\rho), f(\rho')] \models \mathbb{V}(s)$$

Vector Addition System with Resets

Vector Addition System with Resets

If $\mathcal{F}$ holds iff y transitions to y' along some trajectory w in the language of our program graph according to $\mathbb{V}$,

Vector Addition System with Resets

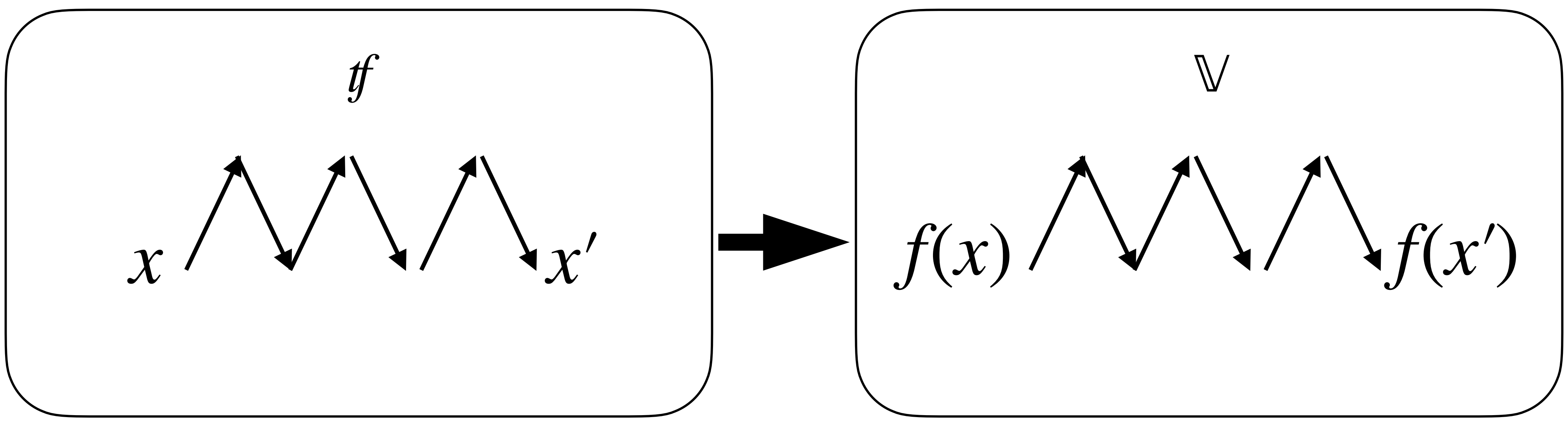
If $\mathscr{F}$ holds iff y transitions to y' along some trajectory w in the language of our program graph according to $\mathbb{V}$,

Then $\mathscr{F}[y \to f(x), y' \to f(x')]$ holds if x can transition to x' along some trajectory w according to $\mathbb{f}$

Vector Addition System with Resets

If $\mathscr{F}$ holds iff y transitions to y' along some trajectory w in the language of our program graph according to $\mathbb{V}$,

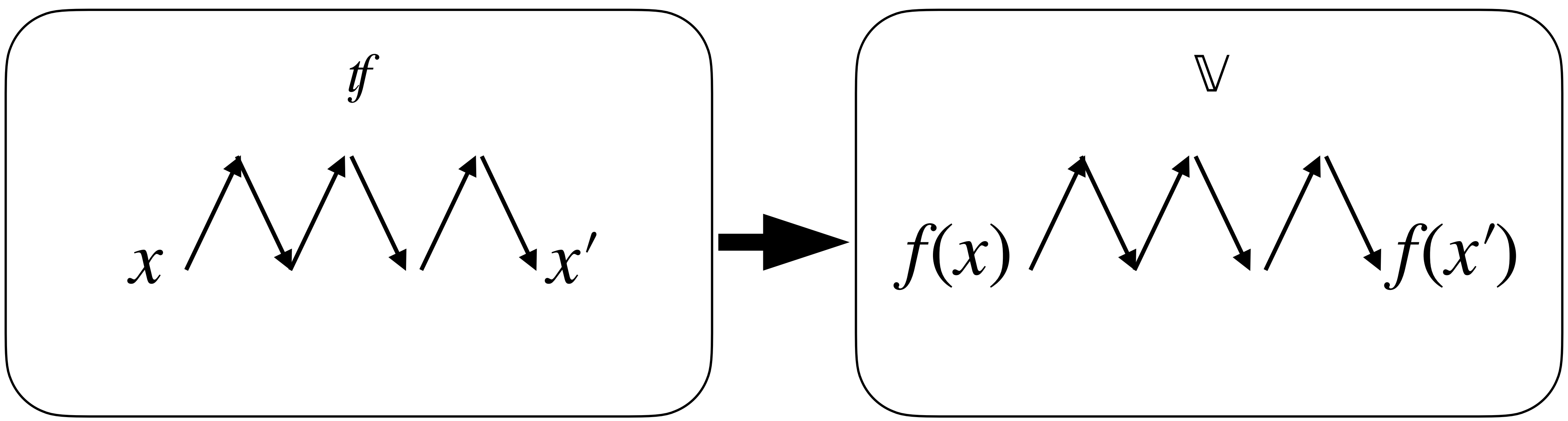
Then $\mathscr{F}[y \to f(x), y' \to f(x')]$ holds if x can transition to x' along some trajectory w according to tf



Vector Addition System with Resets

If $\mathcal{F}$ holds iff y transitions to y' along some trajectory w in the language of our program graph according to $\mathbb{V}$,

Then $\mathcal{F}[y \to f(x), y' \to f(x')]$ holds if x can transition to x' along some trajectory w according to tf

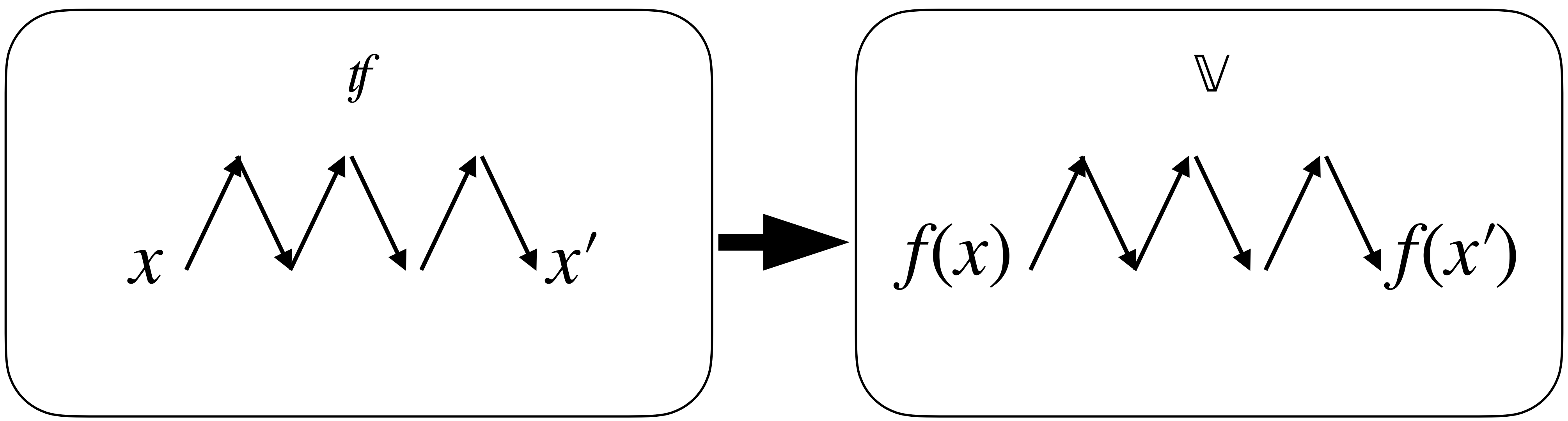


So $\mathcal{F}[y \to f(x), y' \to f(x')]$ can be used as an over-approximate summary

Vector Addition System with Resets

If $\mathscr{F}$ holds iff y transitions to y' along some trajectory w in the language of our program graph according to $\mathbb{V}$,

Then $\mathscr{F}[y \to f(x), y' \to f(x')]$ holds if x can transition to x' along some trajectory w according to tf

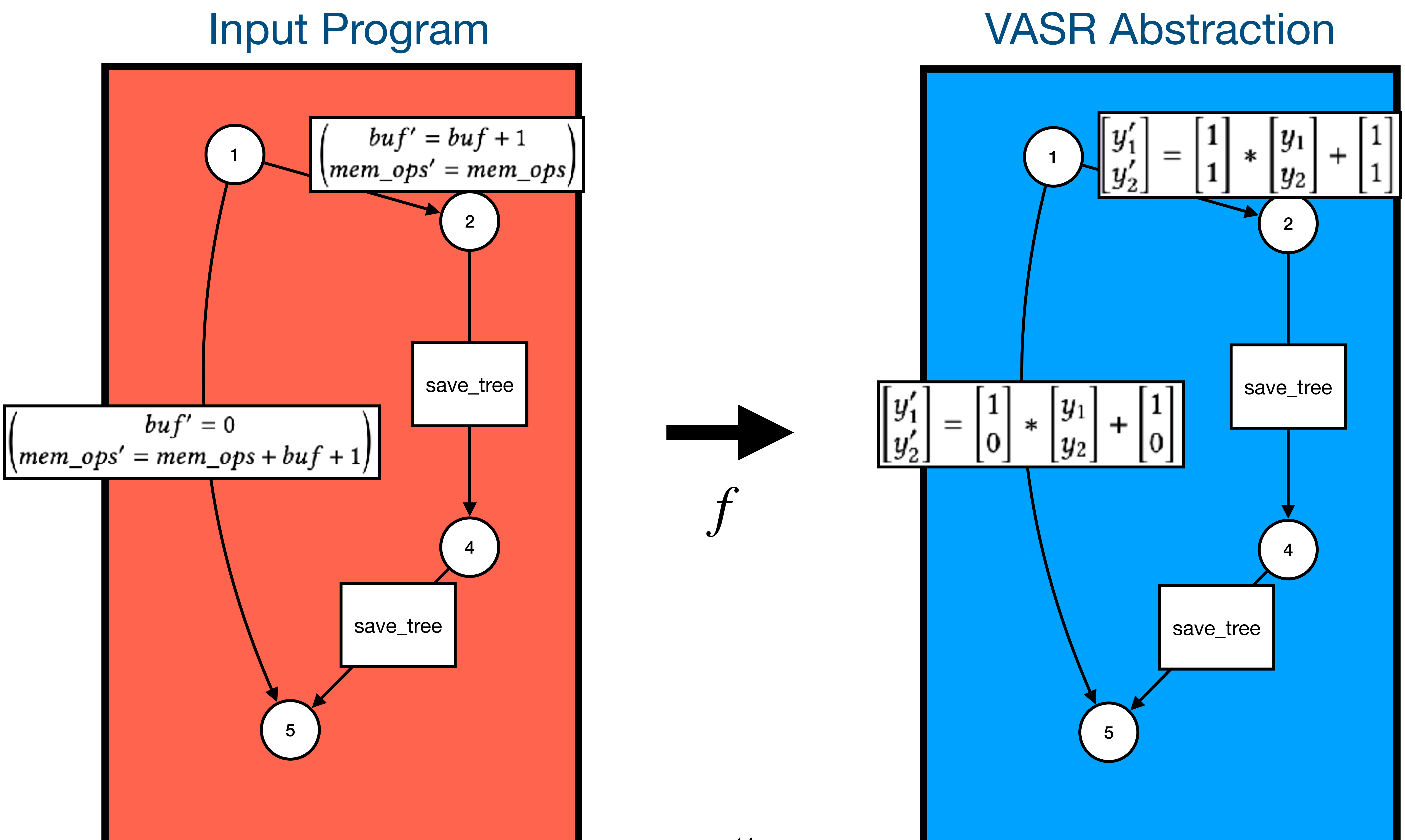


So $\mathscr{F}[y \to f(x), y' \to f(x')]$ can be used as an over-approximate summary

- We restrict our attention to linear simulations

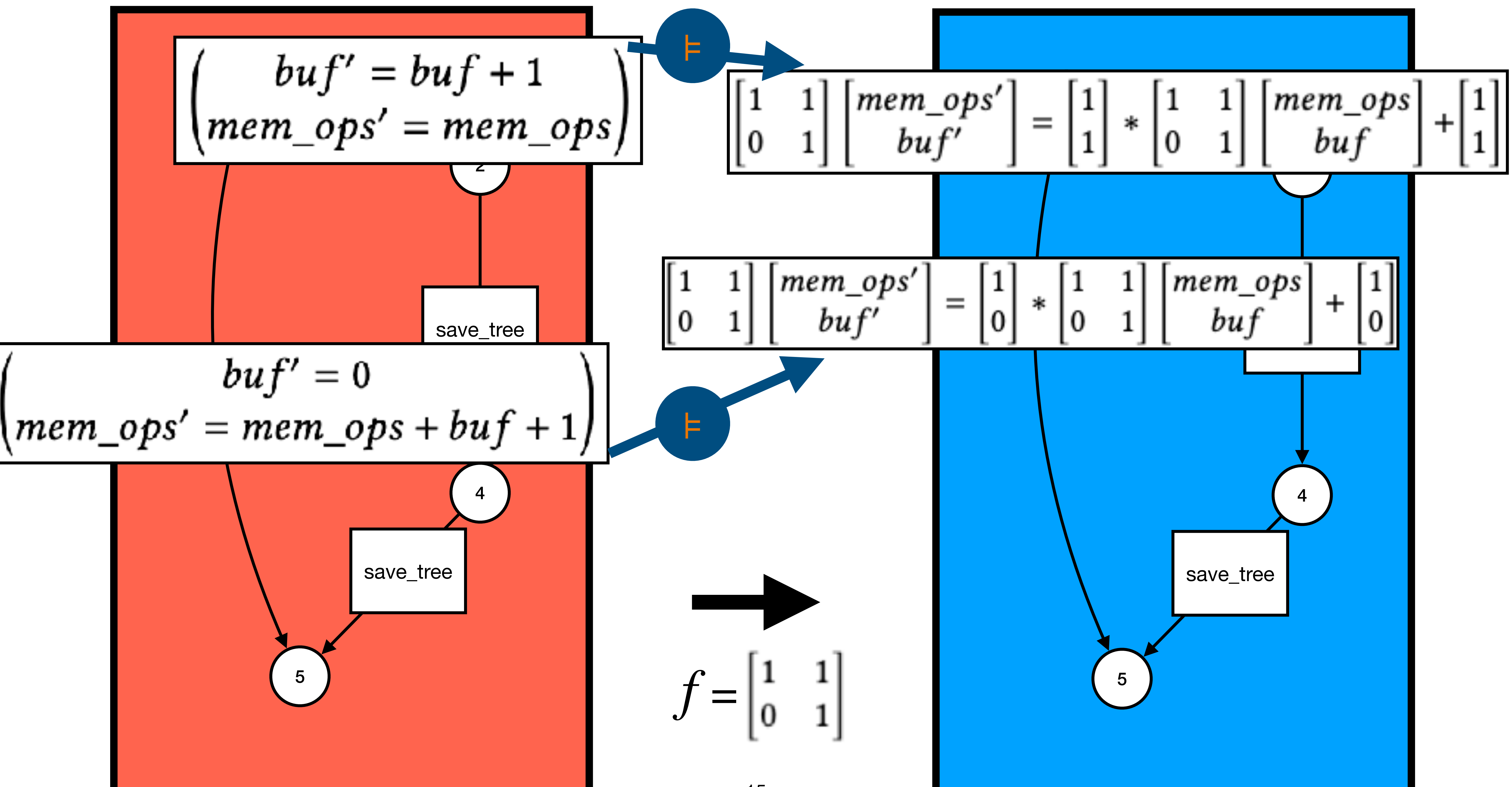
Vector Addition System with Resets

Example VASR Abstraction



Vector Addition System with Resets

Example VASR Abstraction



Vector Addition System with Resets

Example Abstract Execution

- An example trajectory: abb

$$\begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} * \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \circ$$

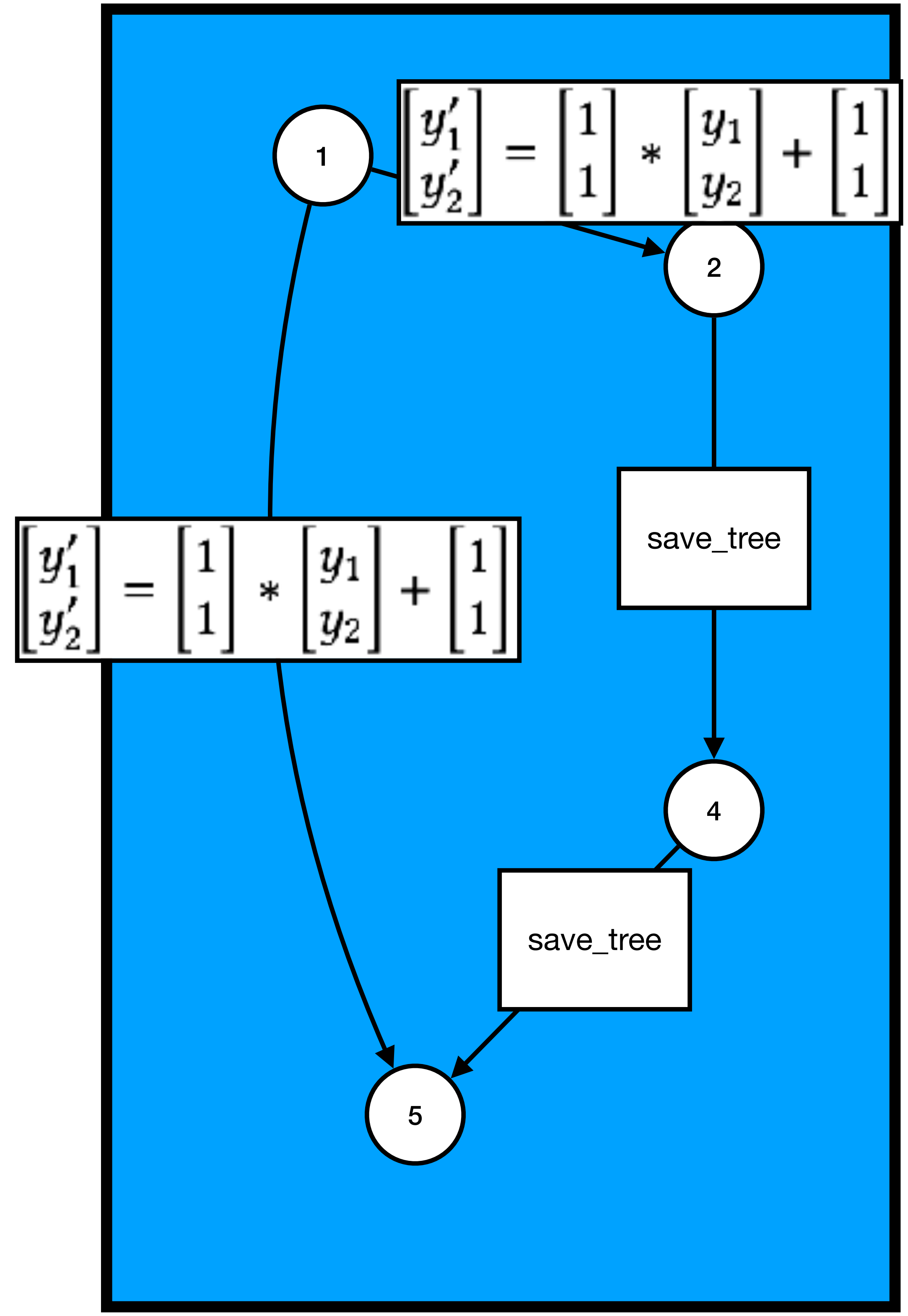
$$\begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} * \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \circ$$

$$\begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} * \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} * \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

f

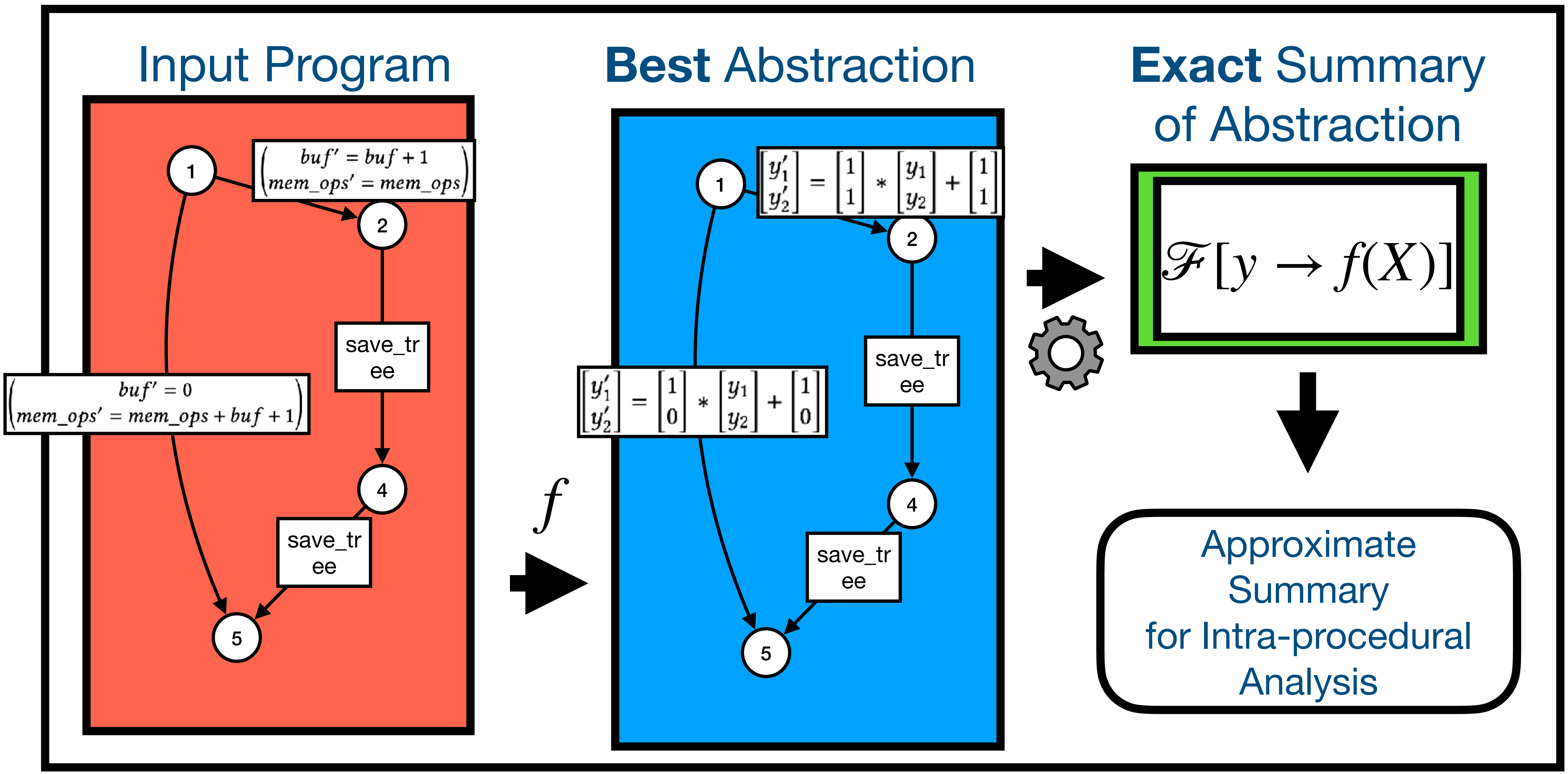
$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} mem\_ops' \\ buf' \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} mem\_ops \\ buf \end{bmatrix} + \begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

over-approximate summary!



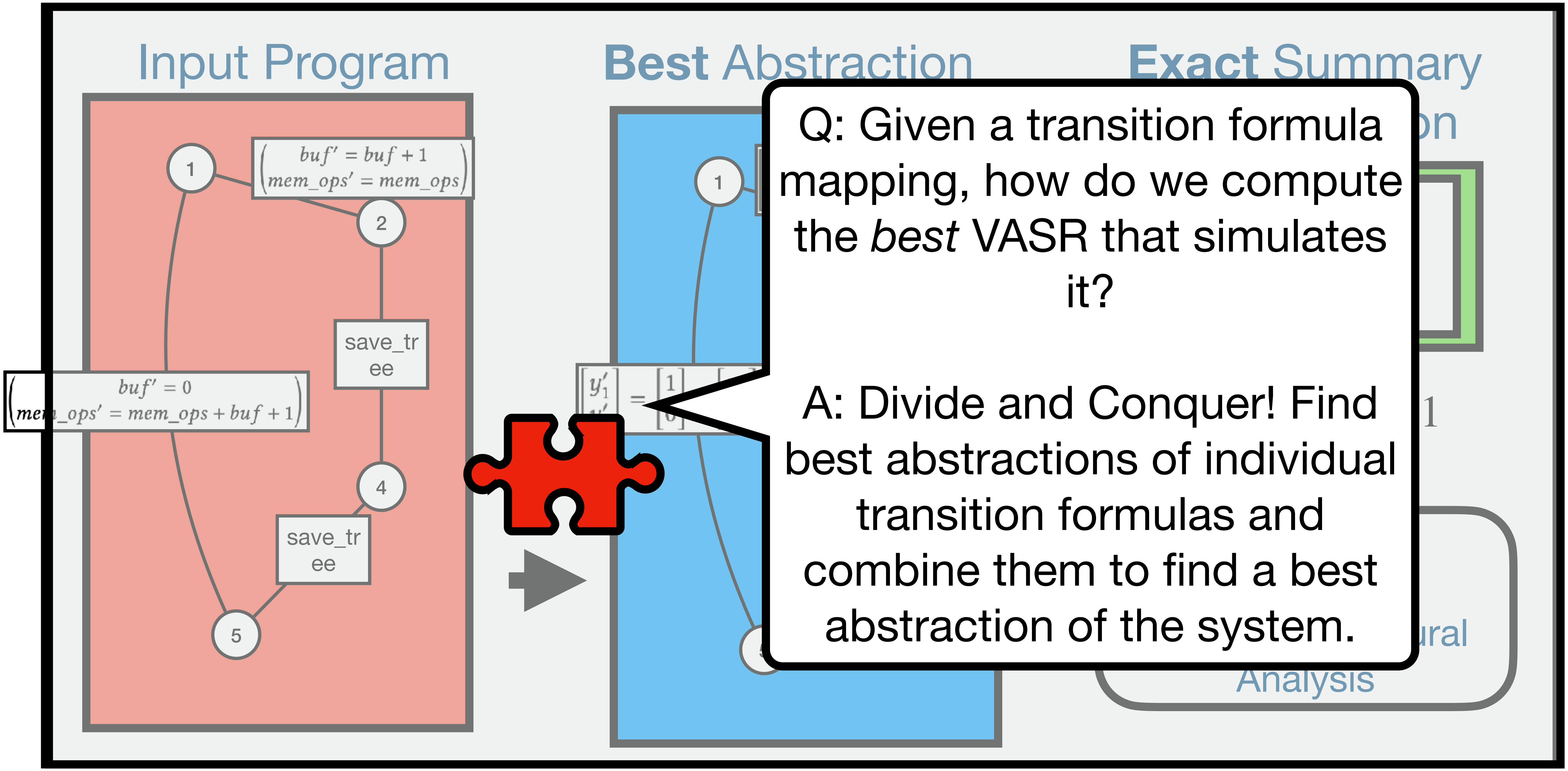
Overview

Any questions?



Overview

Which step are we covering?



Best VASR Abstractions of $t\!f$

Abstracting $t\!f(s)$

Best VASR Abstractions of tf

Abstracting $\mathit{tf}(s)$

Examples

$$\begin{pmatrix} buf' = buf + 1 \\ mem\_ops' = mem\_ops \end{pmatrix}$$

$\mathit{tf}(b)$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} mem\_ops' \\ buf' \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} mem\_ops \\ buf \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\begin{pmatrix} buf' = 0 \\ mem\_ops' = mem\_ops + buf + 1 \end{pmatrix}$$

$\mathit{tf}(c)$

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} mem\_ops' \\ buf' \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} mem\_ops \\ buf \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Best VASR Abstractions of tf

Abstracting $\mathit{tf}(s)$

- Consider the problem of abstracting a single transition formula $\mathit{tf}(s)$

$$\mathrm{Reset}(\mathit{tf}(s)) = \left\{[\vec{a}_r, o_r] \in \mathbb{Q}^{|X_G|+1} : \mathit{tf}(s) \models (\vec{a}_r^T \vec{X}') = o_r\right\}$$

$$\mathrm{Incr}(\mathit{tf}(s)) = \left\{[\vec{a}_a, o_a] \in \mathbb{Q}^{|X_G|+1} : \mathit{tf}(s) \models (\vec{a}^T \vec{X}') = (\vec{a}_a^T \vec{X}) + o_a\right\}$$

Examples

$$\begin{pmatrix} buf' = buf + 1 \\ mem\_ops' = mem\_ops \end{pmatrix} \quad \mathit{tf}(b)$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} mem\_ops' \\ buf' \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} mem\_ops \\ buf \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\begin{pmatrix} buf' = 0 \\ mem\_ops' = mem\_ops + buf + 1 \end{pmatrix} \quad \mathit{tf}(c)$$

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} mem\_ops' \\ buf' \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} mem\_ops \\ buf \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Best VASR Abstractions of tf

Abstracting $tf(s)$

- Consider the problem of abstracting a single transition formula $tf(s)$

$$\text{Reset}(tf(s)) = \left\{[\vec{a}_r, o_r] \in \mathbb{Q}^{|X_G|+1} : tf(s) \models (\vec{a}_r^T \vec{X}') = o_r\right\}$$

$$\text{Incr}(tf(s)) = \left\{[\vec{a}_a, o_a] \in \mathbb{Q}^{|X_G|+1} : tf(s) \models (\vec{a}^T \vec{X}') = (\vec{a}_a^T \vec{X}) + o_a\right\}$$

- These are linear spaces; using tools from the literature [Reps, Sagiv, Yorsh 2004], we can generate bases

$$\left\{\left\langle \vec{a}_r^1, o_r^1 \right\rangle, \ldots, \left\langle \vec{a}_r^n, o_r^n \right\rangle\right\} \qquad \left\{\left\langle \vec{a}_a^1, o_a^1 \right\rangle, \ldots, \left\langle \vec{a}_a^m, o_a^m \right\rangle\right\}$$

Examples

$$\begin{pmatrix} buf' = buf + 1 \\ mem\_ops' = mem\_ops \end{pmatrix} \quad tf(b)$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} mem\_ops' \\ buf' \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} mem\_ops \\ buf \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\begin{pmatrix} buf' = 0 \\ mem\_ops' = mem\_ops + buf + 1 \end{pmatrix} \quad tf(c)$$

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} mem\_ops' \\ buf' \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} mem\_ops \\ buf \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Best VASR Abstractions of $t\!f$

Abstracting $t\!f(s)$

- Consider the problem of abstracting a single transition formula $t\!f(s)$

$$\mathrm{Reset}(t\!f(s)) = \left\{[\vec{a}_r, o_r] \in \mathbb{Q}^{|X_G|+1} : t\!f(s) \models (\vec{a}_r^T \vec{X}') = o_r\right\}$$

$$\mathrm{Incr}(t\!f(s)) = \left\{[\vec{a}_a, o_a] \in \mathbb{Q}^{|X_G|+1} : t\!f(s) \models (\vec{a}^T \vec{X}') = (\vec{a}_a^T \vec{X}) + o_a\right\}$$

- These are linear spaces; using tools from the literature [Reps, Sagiv, Yorsh 2004], we can generate bases

$$\left\{\langle \vec{a}_r^1, o_r^1\rangle, \ldots, \langle \vec{a}_r^n, o_r^n\rangle\right\} \qquad \left\{\langle \vec{a}_a^1, o_a^1\rangle, \ldots, \langle \vec{a}_a^m, o_a^m\rangle\right\}$$

- These bases form **best** abstractions

$$\begin{bmatrix} (\vec{a}_r^1)^T \\ \cdots \\ (\vec{a}_r^n)^T \\ (\vec{a}_a^1)^T \\ \cdots \\ (\vec{a}_a^m)^T \end{bmatrix} X' = \begin{bmatrix} 0 \\ \cdots \\ 0 \\ 1 \\ \cdots \\ 1 \end{bmatrix} * \begin{bmatrix} (\vec{a}_r^1)^T \\ \cdots \\ (\vec{a}_r^n)^T \\ (\vec{a}_a^1)^T \\ \cdots \\ (\vec{a}_a^m)^T \end{bmatrix} X + \begin{bmatrix} o_r^1 \\ \cdots \\ o_r^n \\ o_a^1 \\ \cdots \\ o_a^m \end{bmatrix}$$

Examples

$t\!f(b)$

$$\begin{pmatrix} buf' = buf + 1 \\ mem\_ops' = mem\_ops \end{pmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} mem\_ops' \\ buf' \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} mem\_ops \\ buf \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$t\!f(c)$

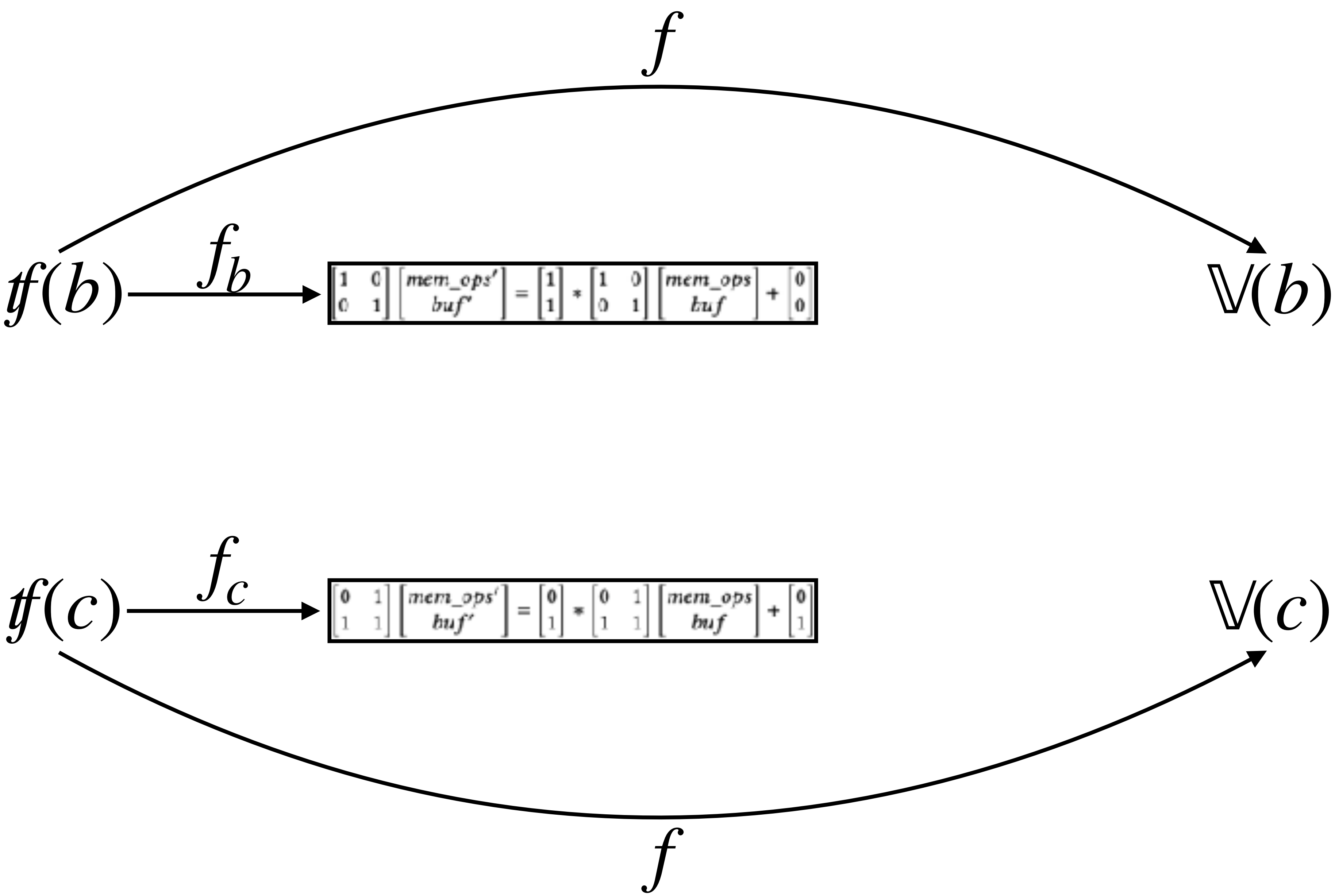
$$\begin{pmatrix} buf' = 0 \\ mem\_ops' = mem\_ops + buf + 1 \end{pmatrix}$$

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} mem\_ops' \\ buf' \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} mem\_ops \\ buf \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Best VASR Abstractions of tf

The Combination Step

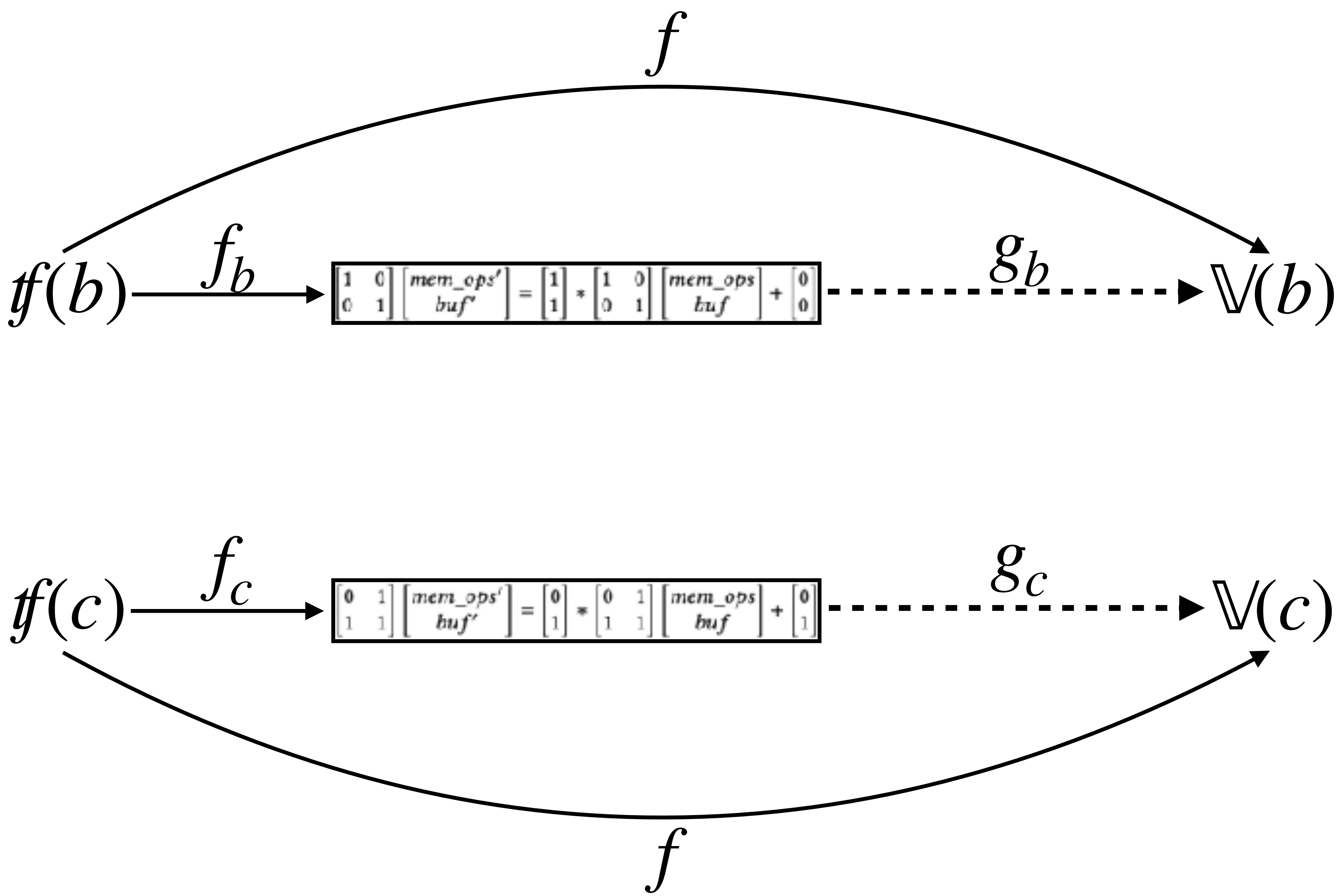
- If $\mathbb{V}$ is a VASR abstraction of tf…



Best VASR Abstractions of *tf*

The Combination Step

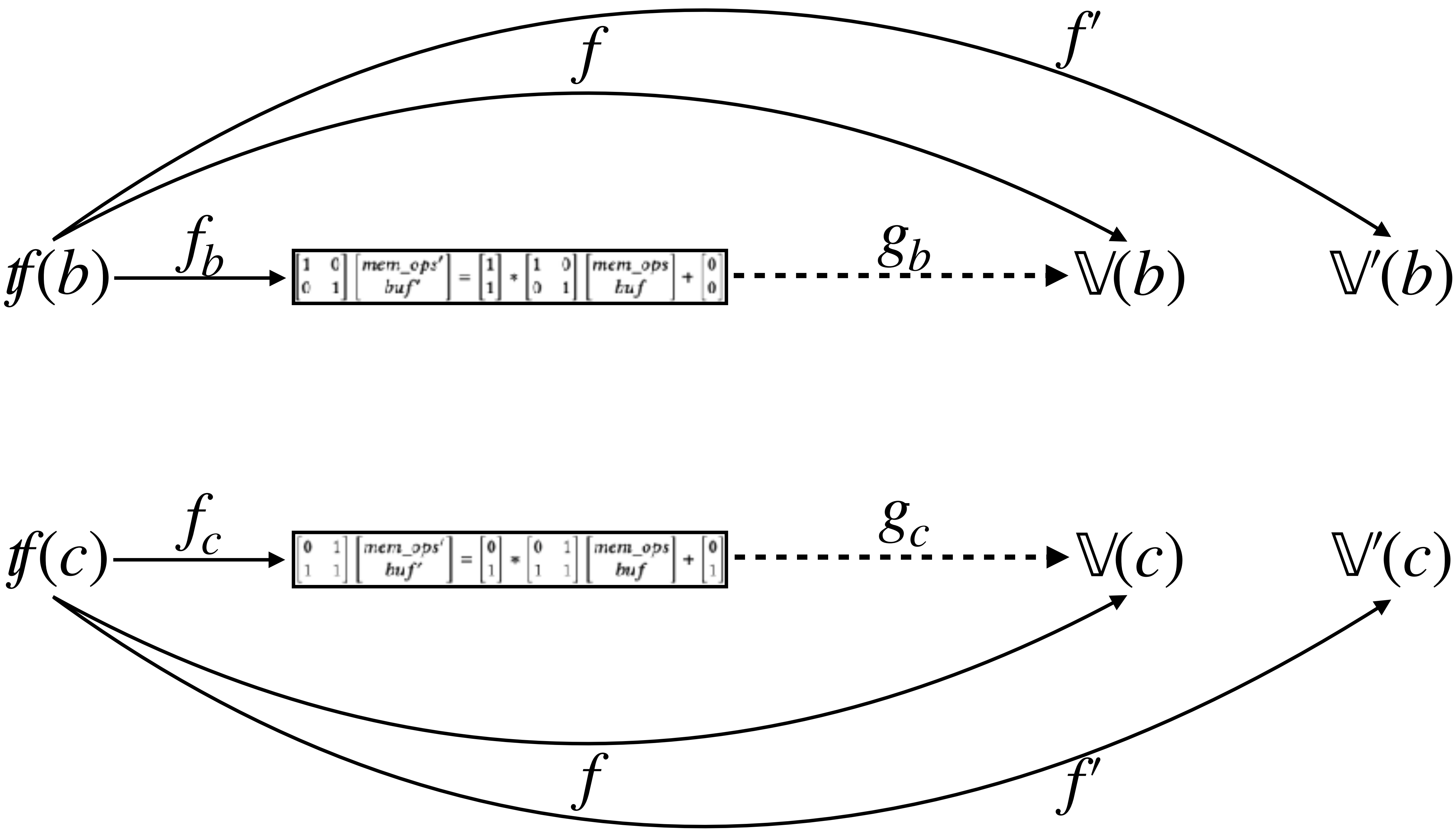
- If $\mathbb{V}$ is a VASR abstraction of *tf*…



Best VASR Abstractions of tf

The Combination Step

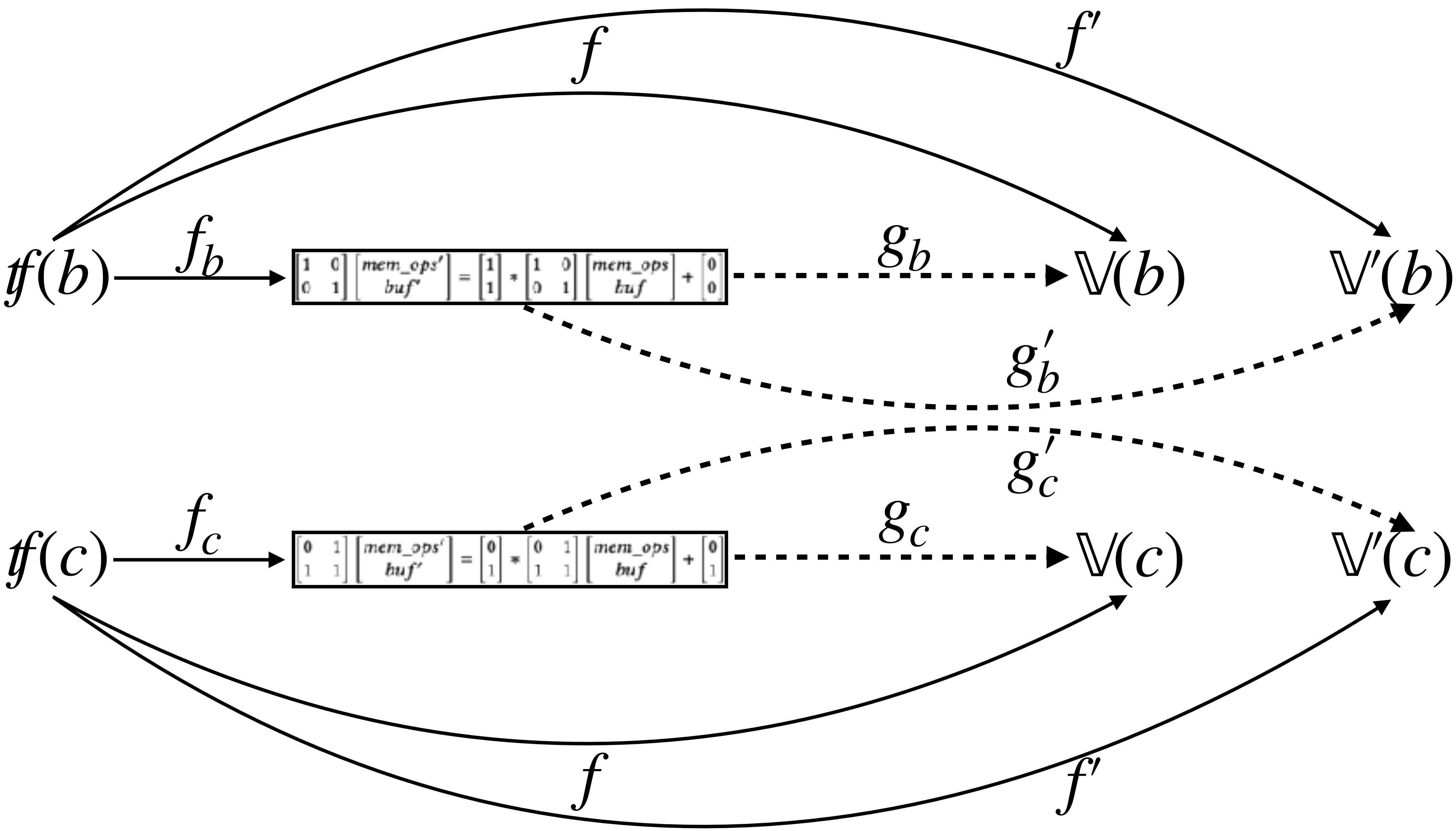
If $\mathbb{V}$ is the best VASR abstraction of tf…



Best VASR Abstractions of tf

The Combination Step

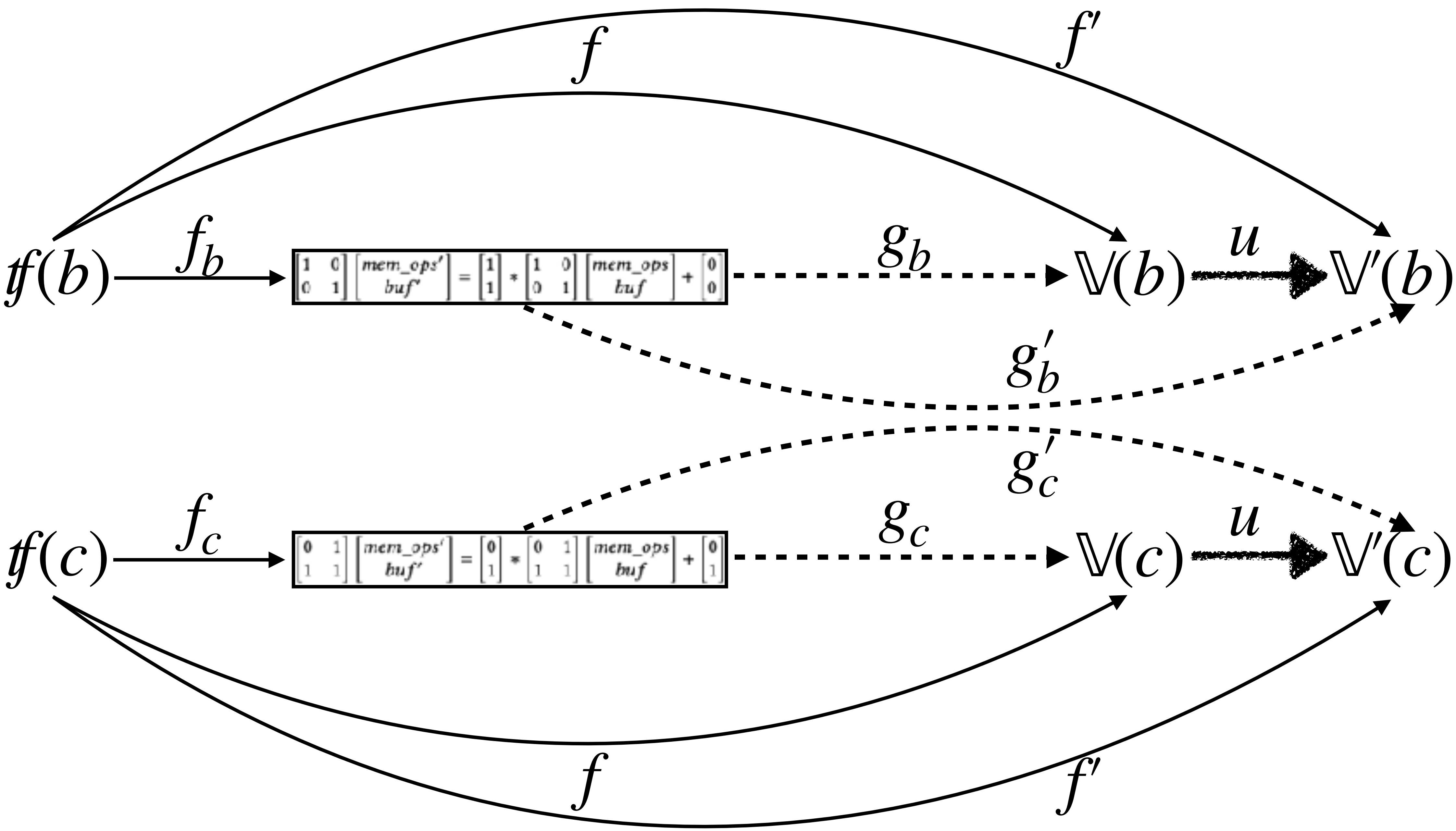
If $\mathbb{V}$ is the best VASR abstraction of tf…



Best VASR Abstractions of tf

The Combination Step

If $\mathbb{V}$ is the best VASR abstraction of tf…



Best VASR Abstractions of tf

Insights from the Combination Step

- For g to be a simulation between VASRs, each dimension of the output must only be dependent on either reset or incremented dimensions of the input

- The state space of a VASR is well represented by a *separated space*, a linear space S along with a canonical decomposition as a direct sum $S = \bigoplus H$

- The combination step can cause a potentially exponential blowup in the state space of the resulting VASR to ensure best abstraction

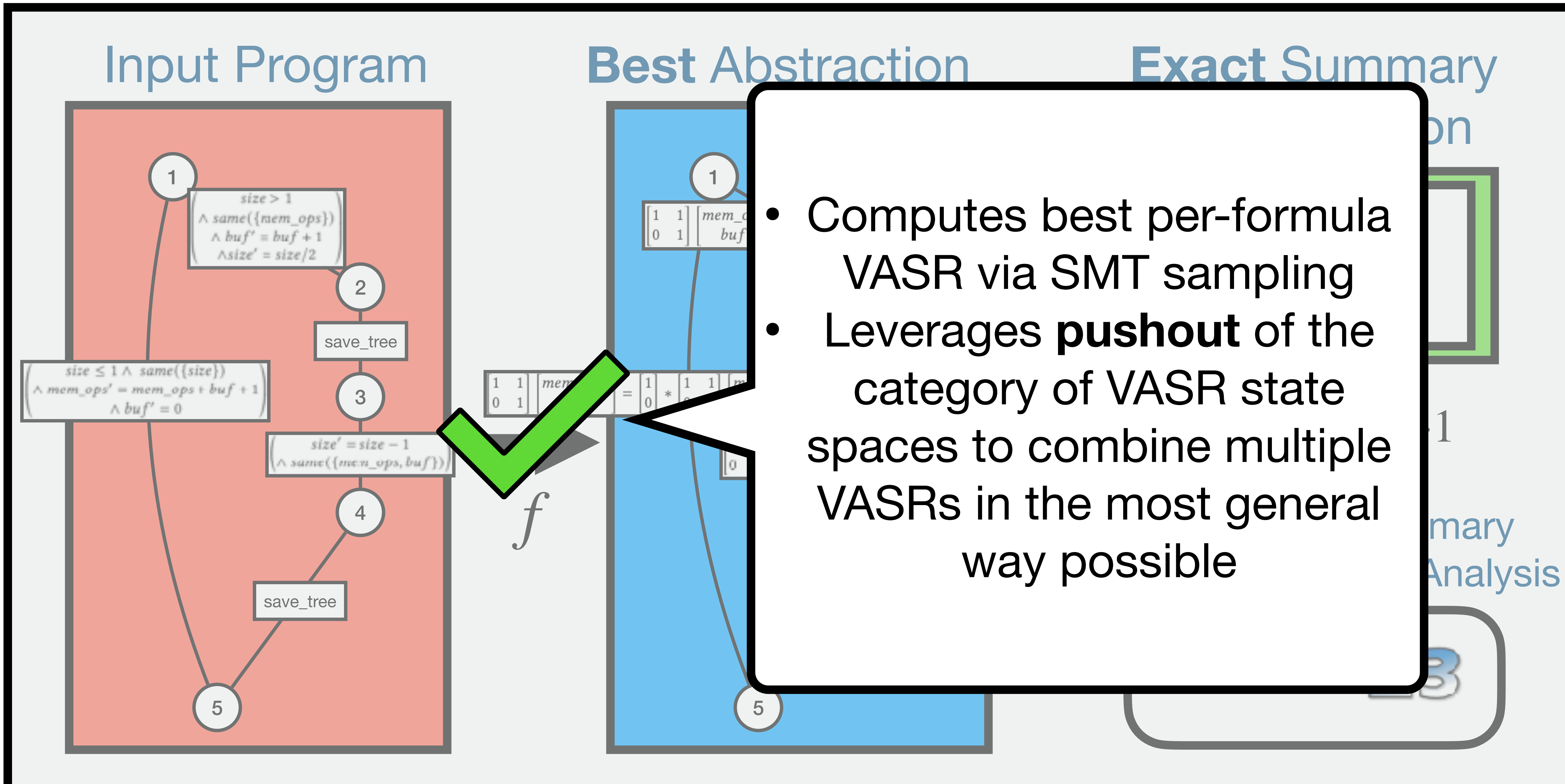
Best VASR Abstractions of tf

Related Work: Silverman & Kincaid 2019

- Extracts a set of VASR transformations simulating a single transition formula representing the body of a loop
- Uses reachability relation of the resulting VASR as an over-approximate summary for the loop
- Limitation: Extraction process relies on the convexity of the underlying theory. While it extracts best abstractions for Linear Rational Arithmetic, does not extract best abstractions for Linear Integer/Rational Arithmetic.
- **Gap Filled:** Our work is able to compute best VASR abstractions for LIRA transition formula systems

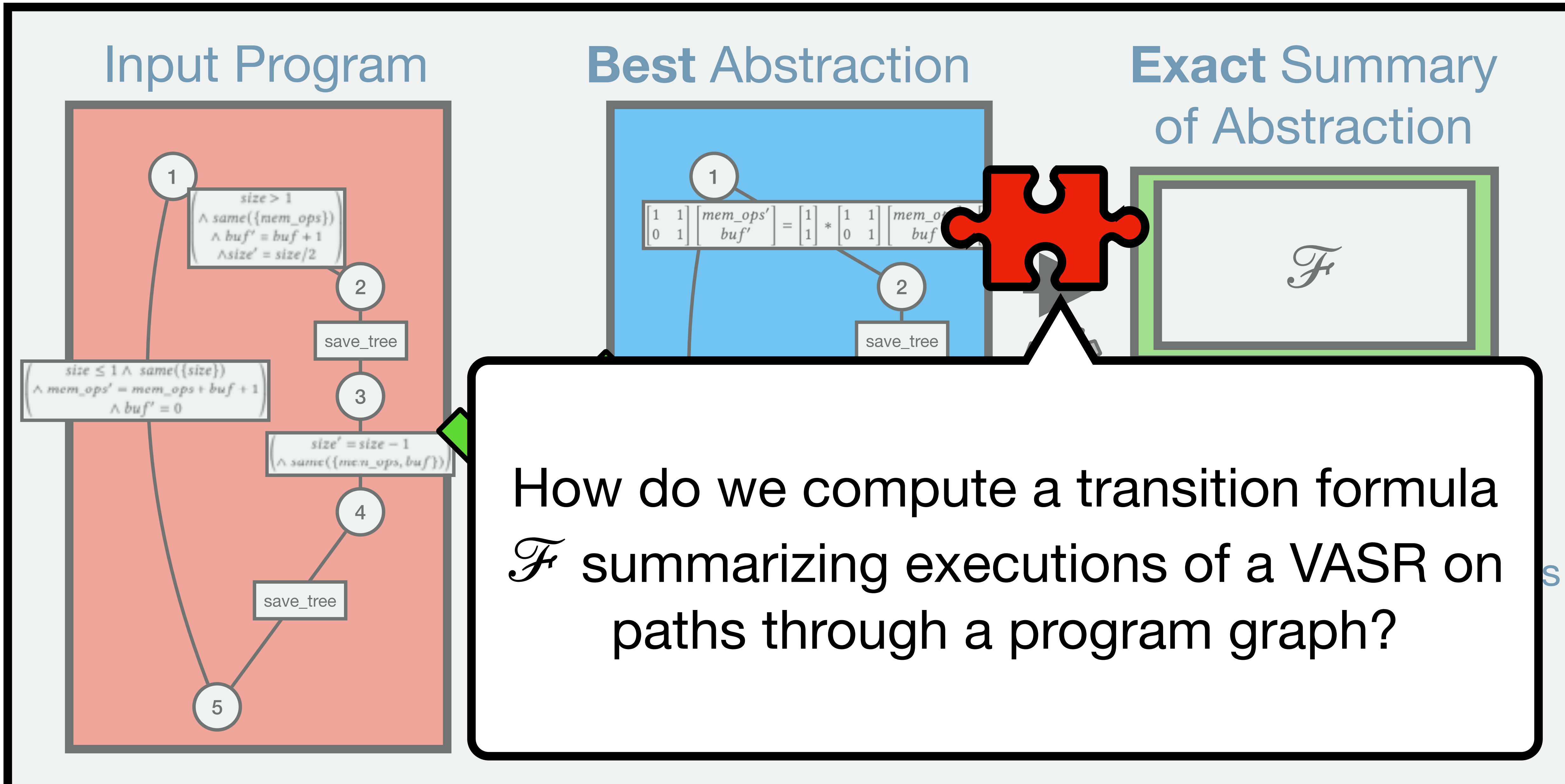
Overview

Any Questions?



Overview

Which step are we covering?



Background

What do we need to know?

- Context Free Grammar:
  - Formalism for describing a set of strings over some alphabet
  - Production Rules: consume one nonterminal and produce any string of terminals and nonterminals
- The set of all trajectories through a program graph is context free

Grammar

Alphabet: $\Sigma = \{a, b, c\}$

Nonterminals: $N = \{X, Y\}$

Production Rules: $P = \{X \rightarrow aY, Y \rightarrow bX, X \rightarrow c\}$

start: X

Example Derivations

$$X \rightarrow aY \rightarrow abX \rightarrow abc$$

$$X \rightarrow aY \rightarrow abX \rightarrow abaY \rightarrow ababX \rightarrow ababac$$

$$X \rightarrow aY \rightarrow abX \rightarrow \ldots \rightarrow (ab)^n c$$

Background

What do we need to know?

- The **Parikh image** of a word w in Σ^* is a function $\pi : \Sigma \to \mathbb{N}$ mapping each character to its number of occurrences in w
- The Parikh image of a language is the set of Parikh images of all words in the language
- [Verma, Seidl, Schwentick 2005] Given a grammar G, we can compute in linear time a logical formula $\mathscr{P}_G(\pi)$ which holds iff π is the Parikh image of some word in the language of G

VASR CFL-Reachability

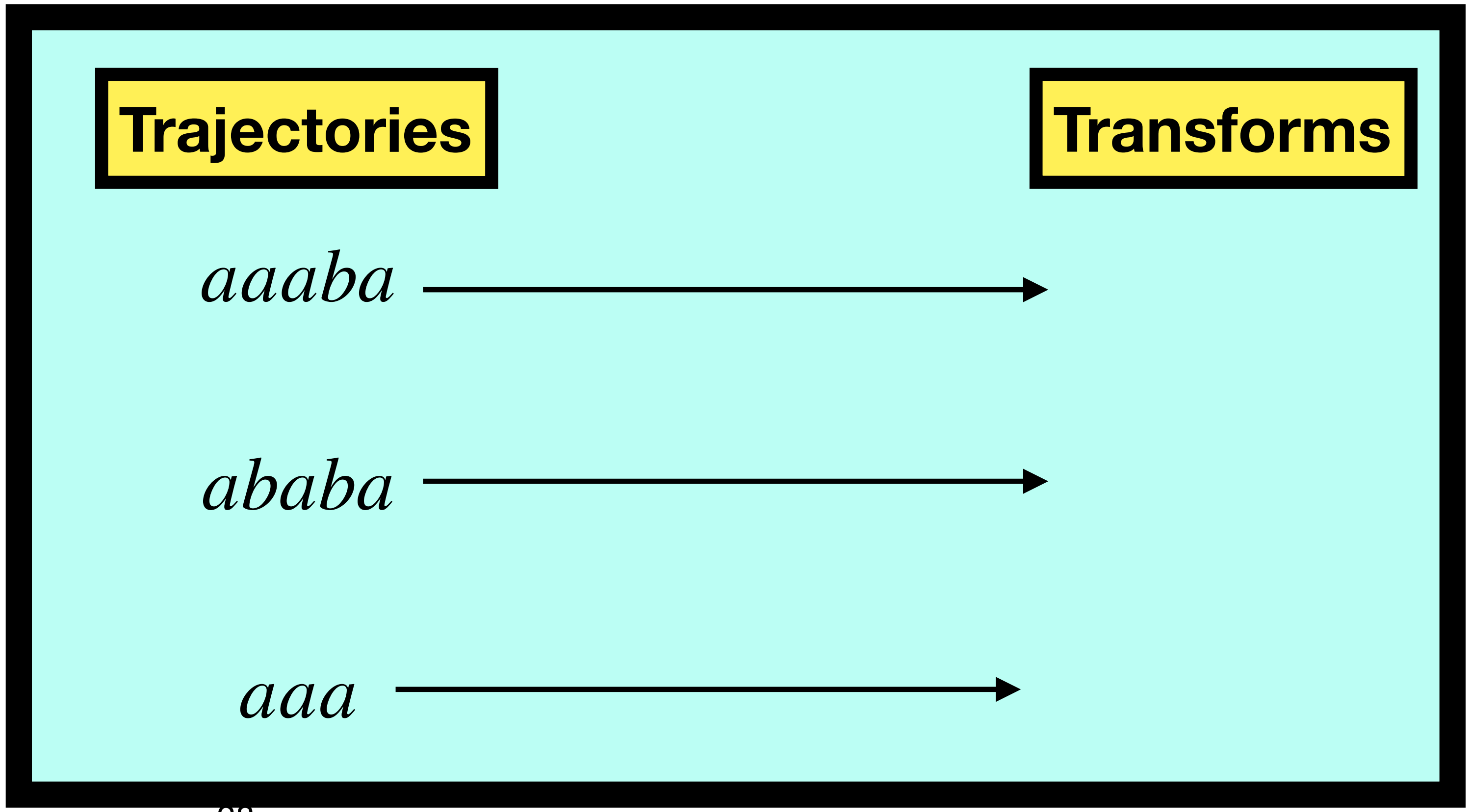
Analyzing the Single Dimension Case

- Without resets, the Parikh image is sufficient to compute the composition of VASR transformations because they commute

a: $X' = 1 * X + 1$

b: $X' = 1 * X + 2$

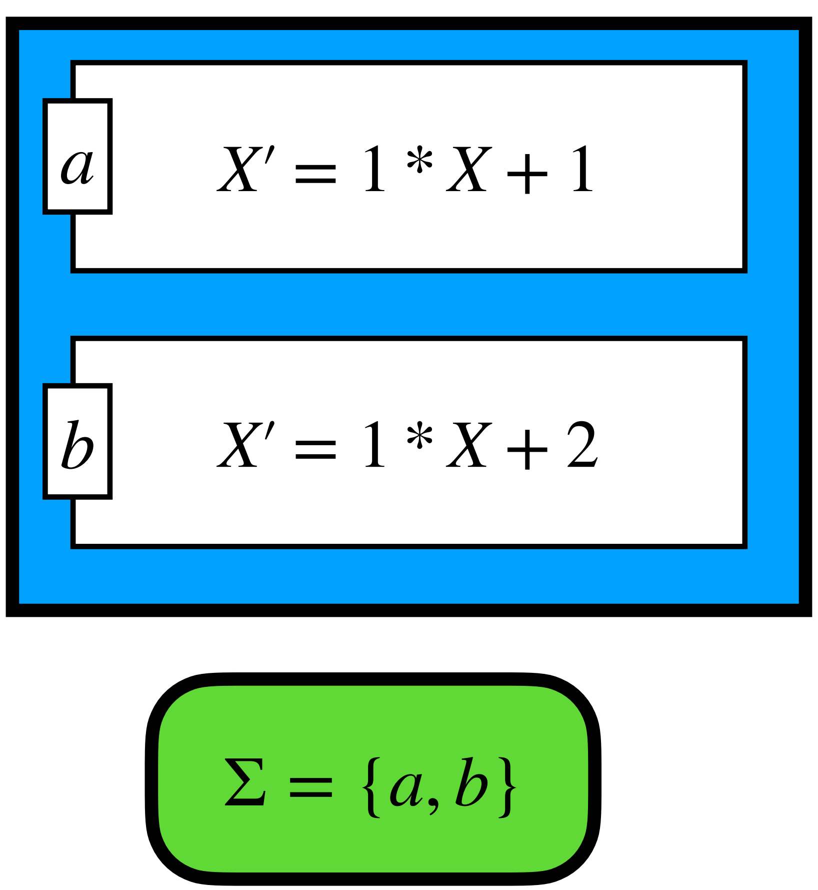
$\Sigma = \{a, b\}$

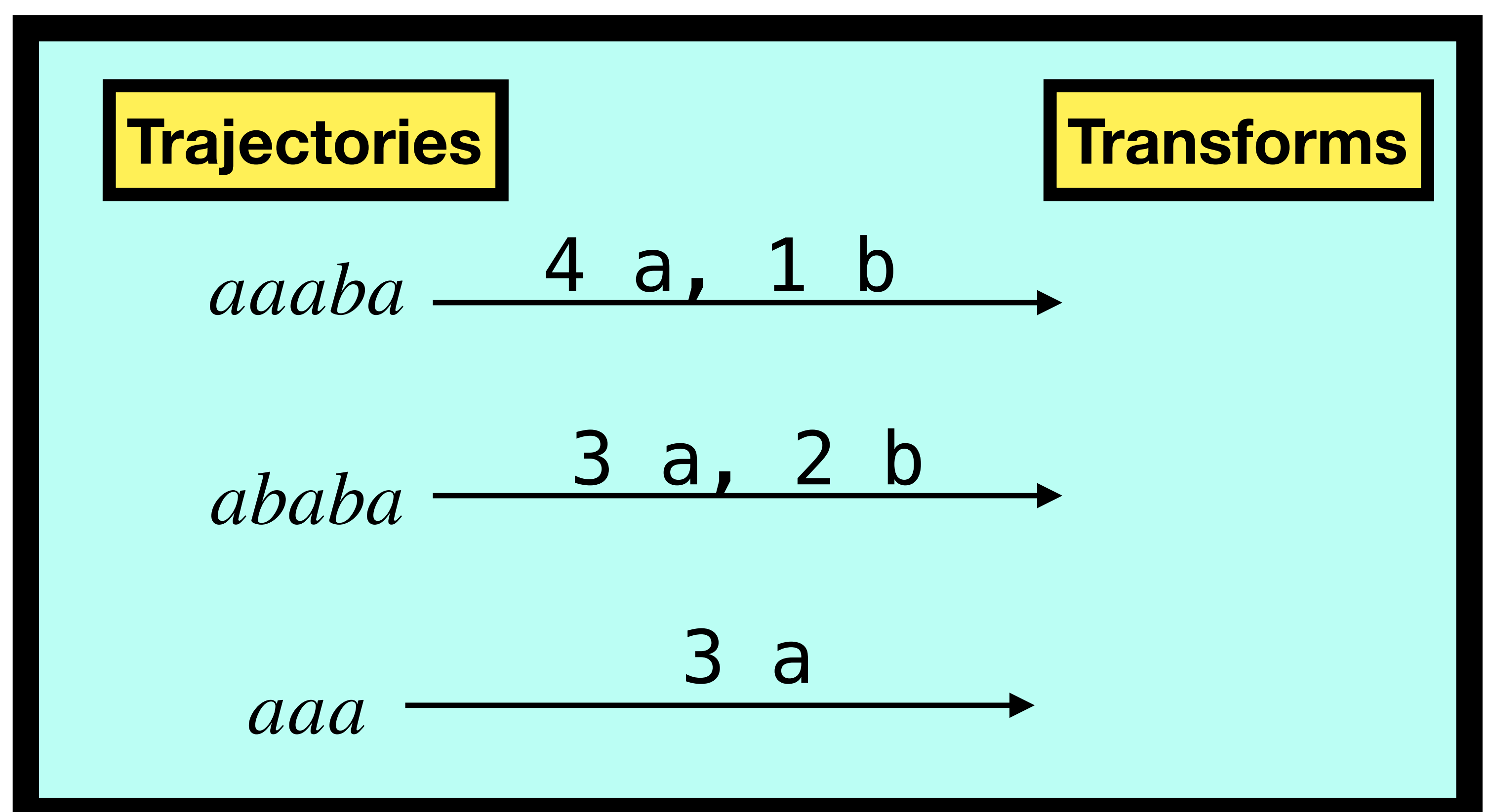


VASR CFL-Reachability

Analyzing the Single Dimension Case

- Without resets, the Parikh image is sufficient to compute the composition of VASR transformations because they commute

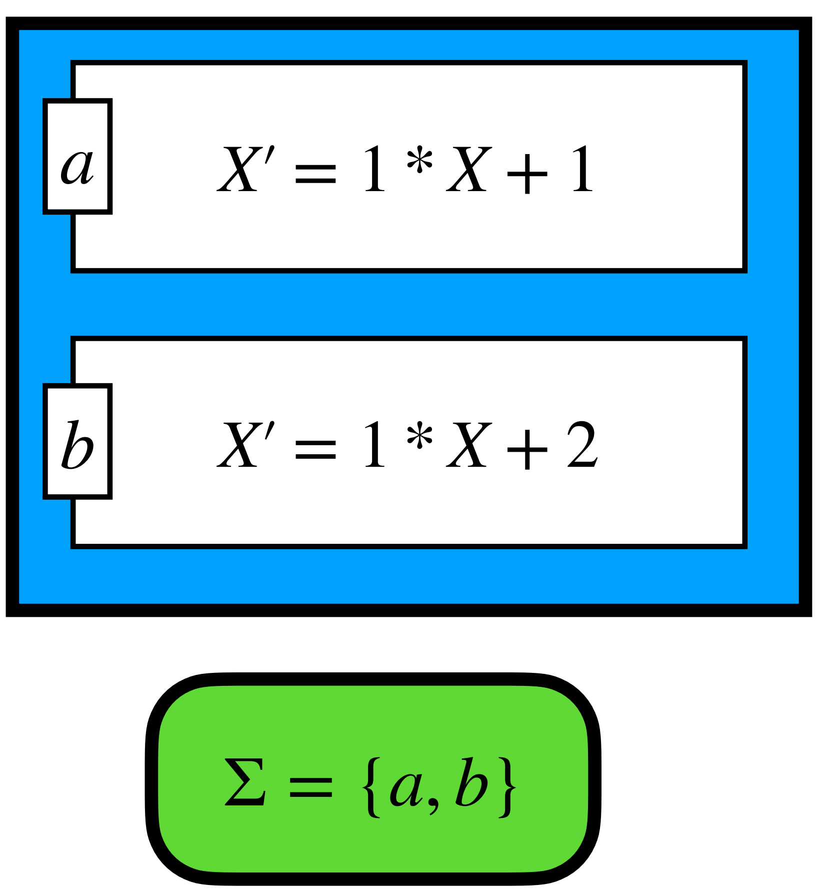


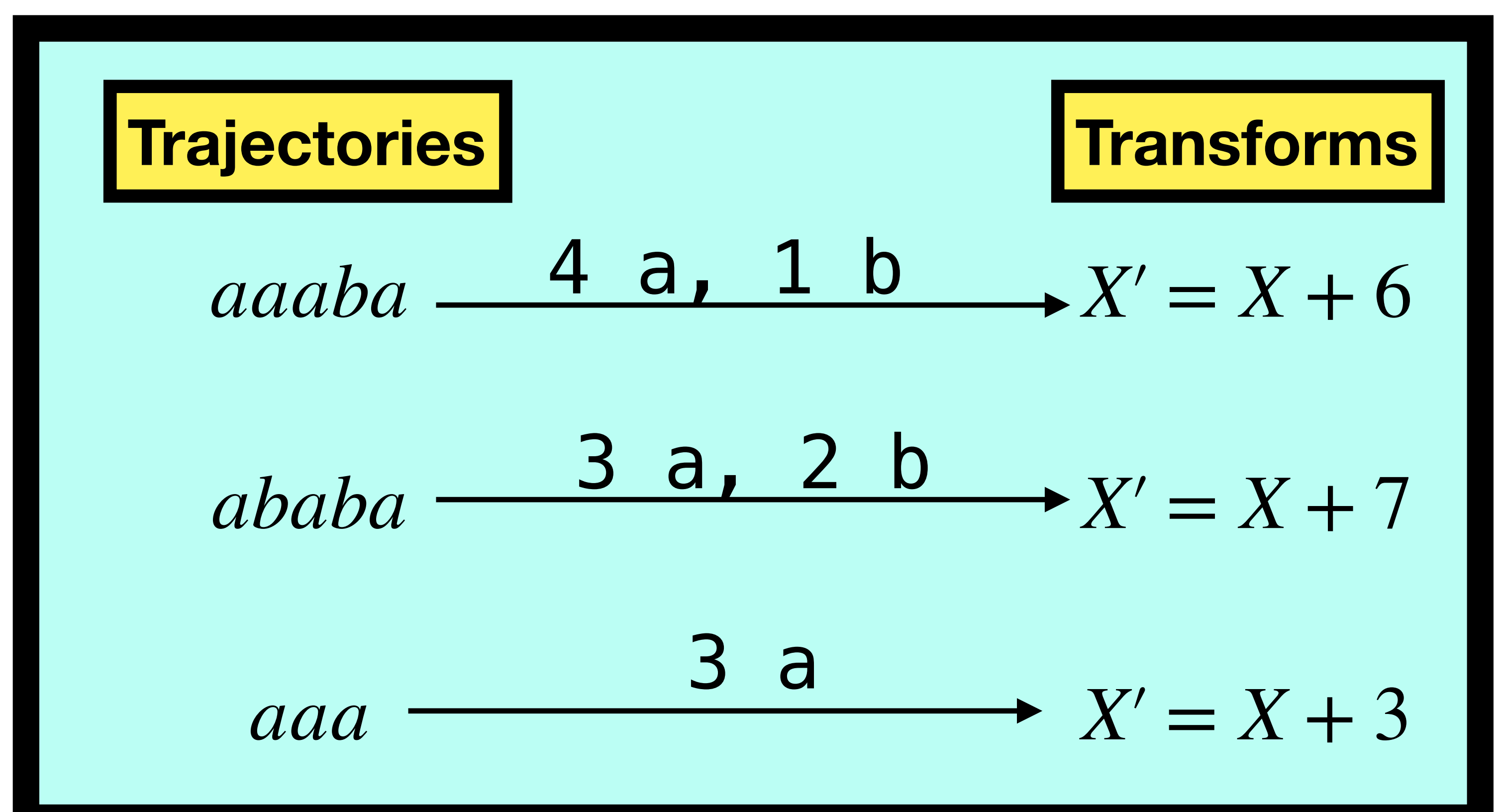


VASR CFL-Reachability

Analyzing the Single Dimension Case

- Without resets, the Parikh image is sufficient to compute the composition of VASR transformations because they commute





VASR CFL-Reachability

Analyzing the Single Dimension Case

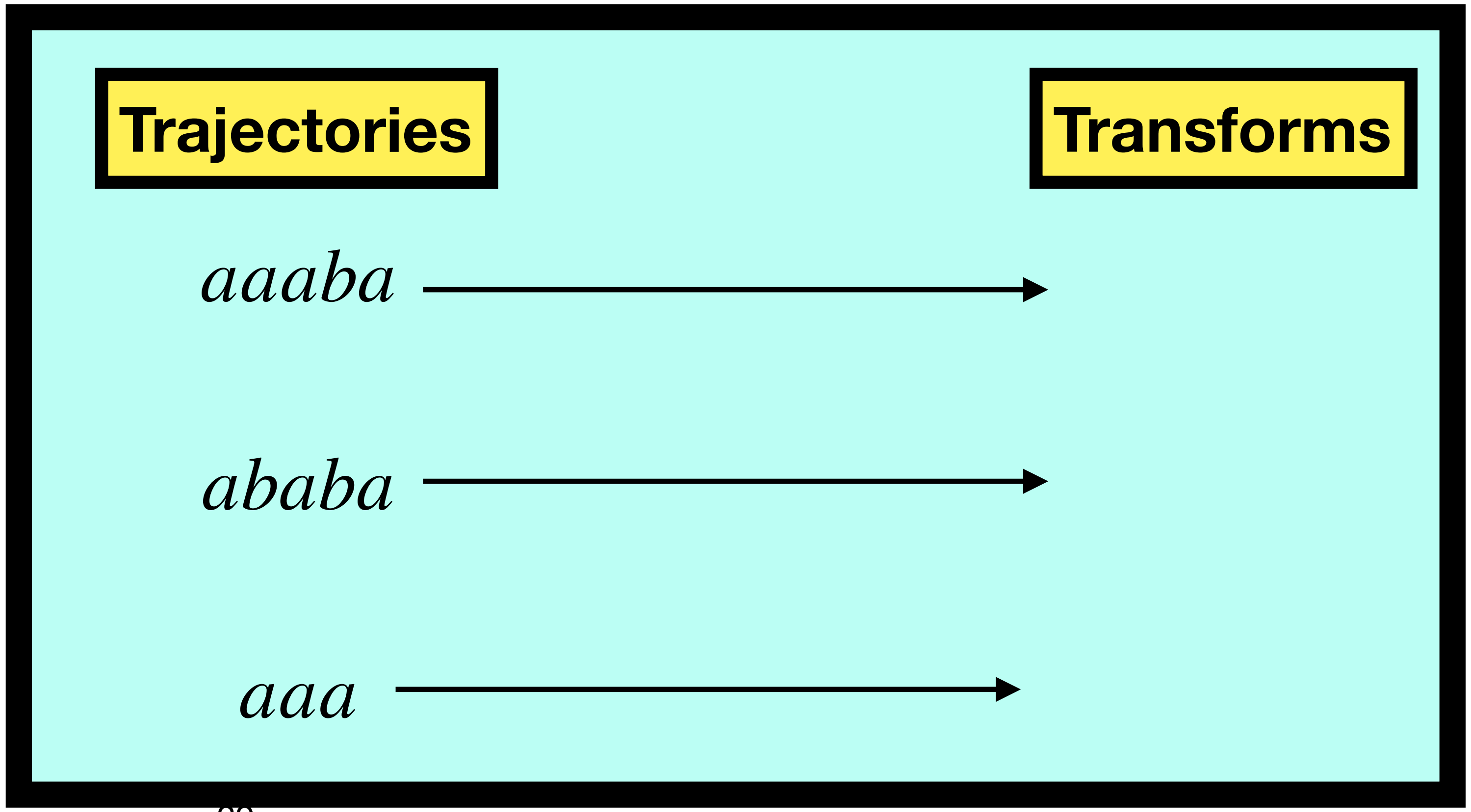
- Without resets, the Parikh image is sufficient to compute the composition of VASR transformations because they commute

- Intuition for resets: it is sufficient to identify the **final reset** in a word and the Parikh image of the sub-word after because all remaining operations commute

a: $X' = 1 * X + 1$

b: $X' = 0 * X + 2$

$\Sigma = \{a, b\}$



VASR CFL-Reachability

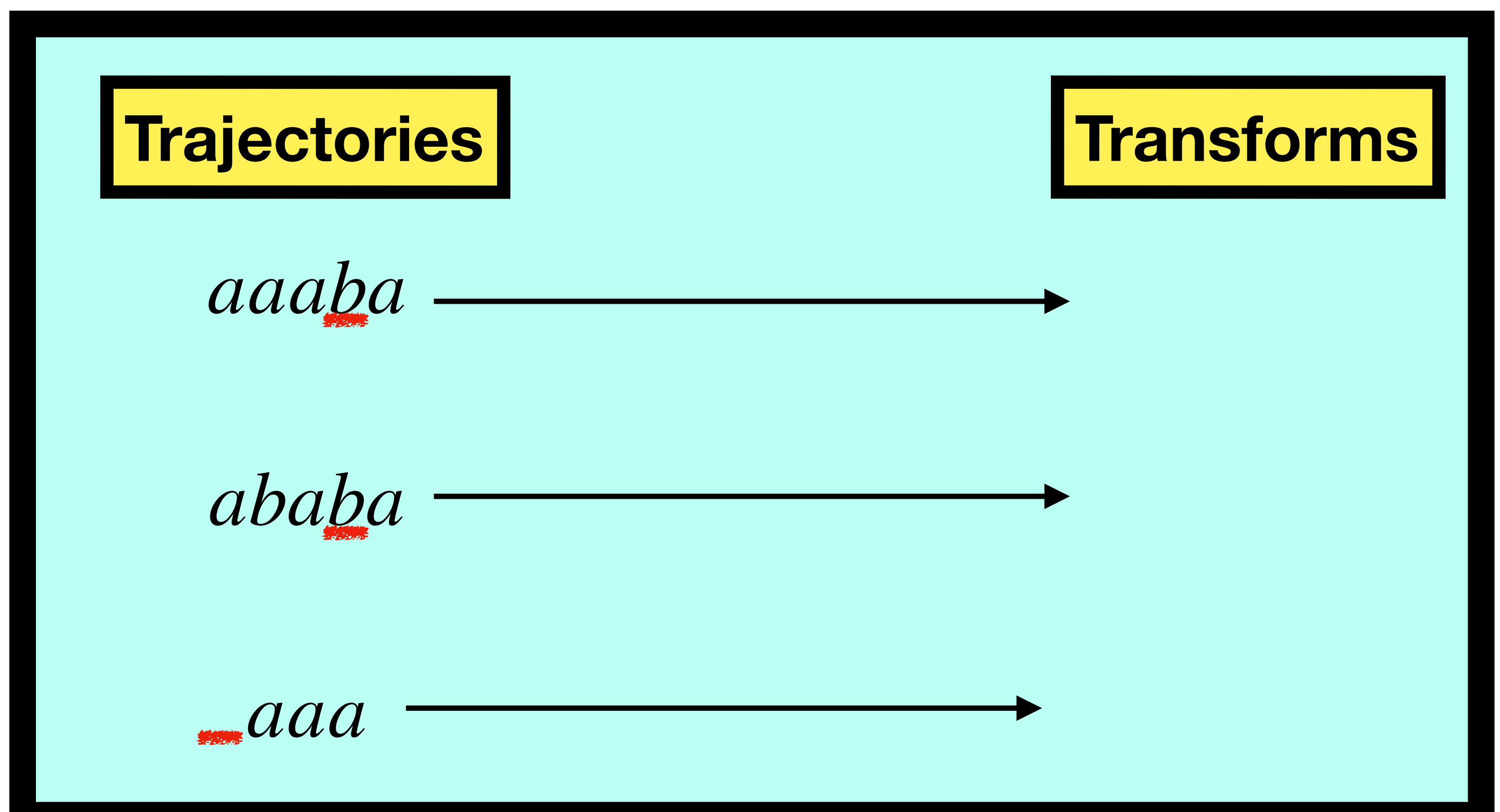
Analyzing the Single Dimension Case

- Without resets, the Parikh image is sufficient to compute the composition of VASR transformations because they commute
- Intuition for resets: it is sufficient to identify the **final reset** in a word and the Parikh image of the sub-word after because all remaining operations commute

a: $X' = 1 * X + 1$

b: $X' = 0 * X + 2$

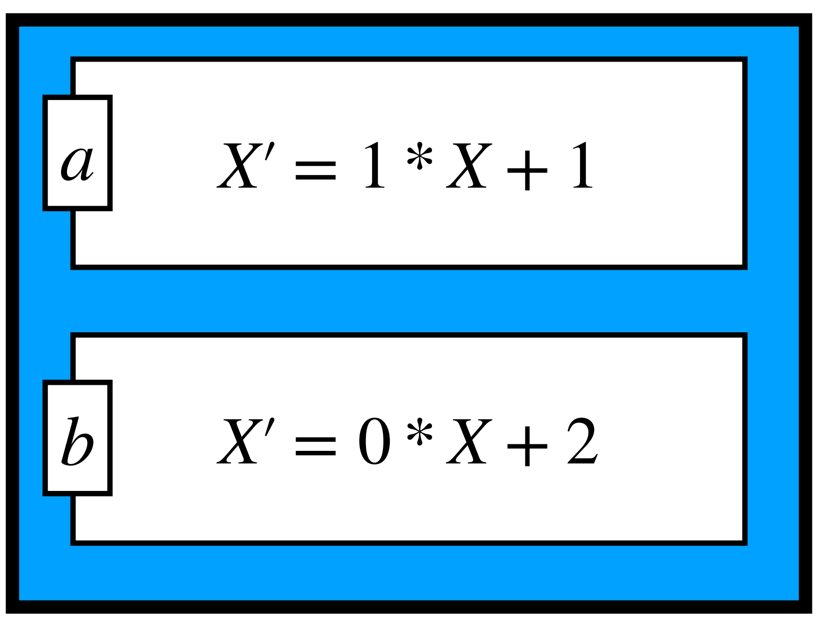
$\Sigma = \{a, b\}$



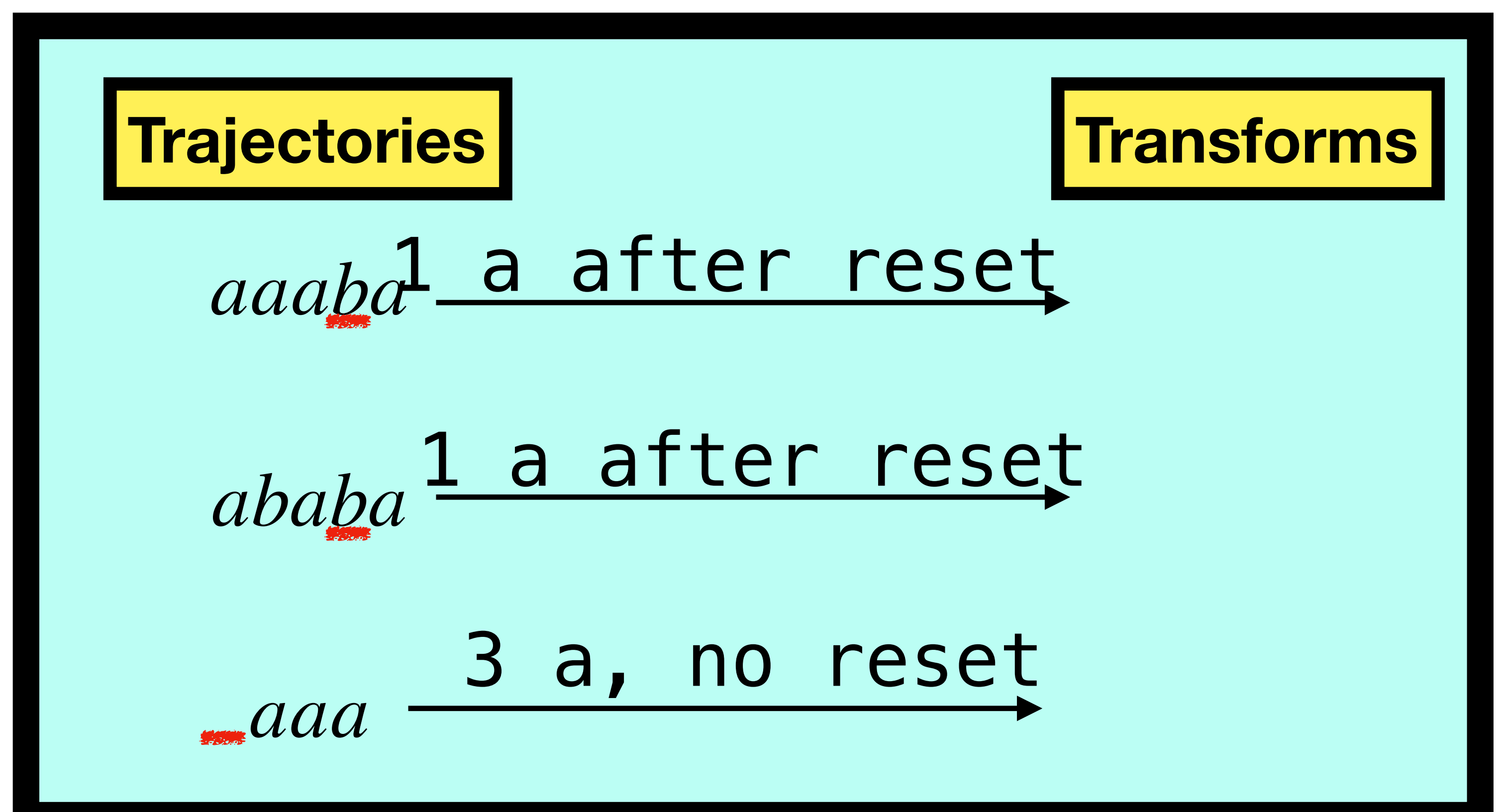
VASR CFL-Reachability

Analyzing the Single Dimension Case

- Without resets, the Parikh image is sufficient to compute the composition of VASR transformations because they commute
- Intuition for resets: it is sufficient to identify the **final reset** in a word and the Parikh image of the sub-word after because all remaining operations commute



$\Sigma = \{a, b\}$



VASR CFL-Reachability

Analyzing the Single Dimension Case

- Without resets, the Parikh image is sufficient to compute the composition of VASR transformations because they commute
- Intuition for resets: it is sufficient to identify the **final reset** in a word and the Parikh image of the sub-word after because all remaining operations commute

a: $X' = 1 * X + 1$

b: $X' = 0 * X + 2$

$\Sigma = \{a, b\}$

Trajectories		Transforms
$aaaba$	1 a after reset	$X' = 3$
$ababa$	1 a after reset	$X' = 3$
aaa	3 a, no reset	$X' = X + 3$

VASR CFL-Reachability

Formalizing “Final Resets”

VASR CFL-Reachability

Formalizing "Final Resets"

- Let d be the dimension of our VASR $\mathbb{V}$. To compute the transformation associated with a trajectory w we need:

  1. The locations of the final resets of each dimension

  2. The Parikh Images of the subwords between these final resets

VASR CFL-Reachability

Formalizing “Final Resets”

- Let d be the dimension of our VASR $\mathbb{V}$. To compute the transformation associated with a trajectory w we need:
  1. The locations of the final resets of each dimension
  2. The Parikh Images of the subwords between these final resets
- **Abstract Trajectory $\pi : (\Sigma \times [2d+1]) \to \mathbb{N}$:** a formalization of the necessary information of a trajectory to compute its transition
  - For any even i, $\sum_{s \in \Sigma} \pi(s, i) \leq 1$ (High level: even symbols identify the final resets)

VASR CFL-Reachability

Formalizing "Final Resets"

- Let d be the dimension of our VASR $\mathbb{V}$. To compute the transformation associated with a trajectory w we need:
  1. The locations of the final resets of each dimension
  2. The Parikh Images of the subwords between these final resets
- **Abstract Trajectory $\pi : (\Sigma \times [2d+1]) \to \mathbb{N}$:** a formalization of the necessary information of a trajectory to compute its transition
  - For any even i, $\sum_{s \in \Sigma} \pi(s, i) \leq 1$ (High level: even symbols identify the final resets)
- An abstract trajectory is **well-formed** according to $\mathbb{V}$ if the final reset of each dimension from left to right is at an even symbol

VASR CFL-Reachability

Formalizing “Final Resets”

- Let d be the dimension of our VASR $\mathbb{V}$. To compute the transformation associated with a trajectory w we need:

  1. The locations of the final resets of each dimension
  2. The Parikh Images of the subwords between these final resets

- **Abstract Trajectory $\pi : (\Sigma \times [2d+1]) \rightarrow \mathbb{N}$:** a formalization of the necessary information of a trajectory to compute its transition

  - For any even i, $\sum_{s \in \Sigma} \pi(s, i) \leq 1$ (High level: even symbols identify the final resets)

- An abstract trajectory is **well-formed** according to $\mathbb{V}$ if the final reset of each dimension from left to right is at an even symbol

- An abstract trajectory π represents a trajectory w if there is some decomposition $w = w_1 \ldots w_{2d+1}$ such that the character count of symbol s in w_i is $\pi(s, i)$

VASR CFL-Reachability

Formalizing "Final Resets"

- Let's look at some examples! $w = aabba$, b resets

Abstract Trajectories that Represent w

$a_1 a_1 b_2 b_3 a_3$

$a_1 a_2 b_3 b_3 a_3$

$a_3 a_3 b_3 b_3 a_3$

$a_1 a_1 b_3 b_3 a_3$

$a_1 a_1 b_1 b_2 a_3$

Not Abstract Trajectories

$a_1 a_1 b_2 b_2 a_3$

$a_1 a_2 b_3 b_2 a_2$

VASR CFL-Reachability

Formalizing “Final Resets”

- Let’s look at some examples! $w = aabba$, b resets

Abstract Trajectories that Represent w

$a_1a_1b_2b_3a_3$

$a_1a_2b_3b_3a_3$

$a_3a_3b_3b_3a_3$

$a_1a_1b_3b_3a_3$

well-formed

$a_1a_1b_1b_2a_3$

Not Abstract Trajectories

$a_1a_1b_2b_2a_3$

$a_1a_2b_3b_2a_2$

VASR CFL-Reachability

Can we compute abstract trajectories of a CFL?

VASR CFL-Reachability

Can we compute abstract trajectories of a CFL?

- Consider the regular language:

$$O \triangleq \Sigma_1^*(\Sigma_2 + \epsilon)\Sigma_3^* \ldots \Sigma_{2d-1}^*(\Sigma_{2d} + \epsilon)\Sigma_{2d+1}^*$$

where $\Sigma_i = \langle i, s \rangle$ for all $s \in \Sigma$

VASR CFL-Reachability

Can we compute abstract trajectories of a CFL?

- Consider the regular language:

$$\mathcal{O} \triangleq \Sigma_1^*(\Sigma_2 + \epsilon)\Sigma_3^* \ldots \Sigma_{2d-1}^*(\Sigma_{2d} + \epsilon)\Sigma_{2d+1}^*$$

where $\Sigma_i = \langle i, s \rangle$ for all $s \in \Sigma$

- If h is the homomorphism sending characters in Σ_i to their corresponding character in Σ, then the Parikh Image of the language $h^{-1}(\mathcal{L}(G)) \cap \mathcal{O}$ is the set of all abstract trajectories of trajectories in the language of G

VASR CFL-Reachability

Can we compute abstract trajectories of a CFL?

- Consider the regular language:

$$\mathcal{O} \triangleq \Sigma_1^*(\Sigma_2 + \epsilon)\Sigma_3^* \ldots \Sigma_{2d-1}^*(\Sigma_{2d} + \epsilon)\Sigma_{2d+1}^*$$

  where $\Sigma_i = \langle i, s \rangle$ for all $s \in \Sigma$

- If h is the homomorphism sending characters in Σ_i to their corresponding character in Σ, then the Parikh Image of the language $h^{-1}(\mathcal{L}(G)) \cap \mathcal{O}$ is the set of all abstract trajectories of trajectories in the language of G

- Since context-free languages are closed under intersection with regular languages and inverse homomorphism, this language is context-free

VASR CFL-Reachability

Can we compute abstract trajectories of a CFL?

- Consider the regular language:

$$\mathcal{O} \triangleq \Sigma_1^*(\Sigma_2 + \epsilon)\Sigma_3^* \ldots \Sigma_{2d-1}^*(\Sigma_{2d} + \epsilon)\Sigma_{2d+1}^*$$

  where $\Sigma_i = \langle i, s \rangle$ for all $s \in \Sigma$

- If h is the homomorphism sending characters in Σ_i to their corresponding character in Σ, then the Parikh Image of the language $h^{-1}(\mathcal{L}(G)) \cap \mathcal{O}$ is the set of all abstract trajectories of trajectories in the language of G

- Since context-free languages are closed under intersection with regular languages and inverse homomorphism, this language is context-free

- Let $I(G)$ be a grammar generating this language

VASR CFL-Reachability

What is our logical summary?

$$\mathscr{F}(y, y') := \exists \pi \,.\, \mathscr{P}_{I(G)}(\pi) \wedge \mathscr{F}_{wf}(\pi) \wedge \mathscr{T}_{V}(\pi)$$

$\mathscr{P}_{I(G)}(\pi)$

π is the abstract trajectory of a trajectory in $\mathscr{L}(G)$

$\mathscr{F}_{wf}(\pi)$

Even symbols mark final resets

$\mathscr{T}_{V}(\pi)$

encodes transform associated with π on y and y'

$\mathscr{F}(y, y')$ holds iff y steps to y' along some program path!

VASR CFL-Reachability

Related Work: Haase and Halfon 2014

- Identified the *generalized Parikh image*, similar to our abstract trajectories, to be sufficient to compute the VASR transformation associated with a word

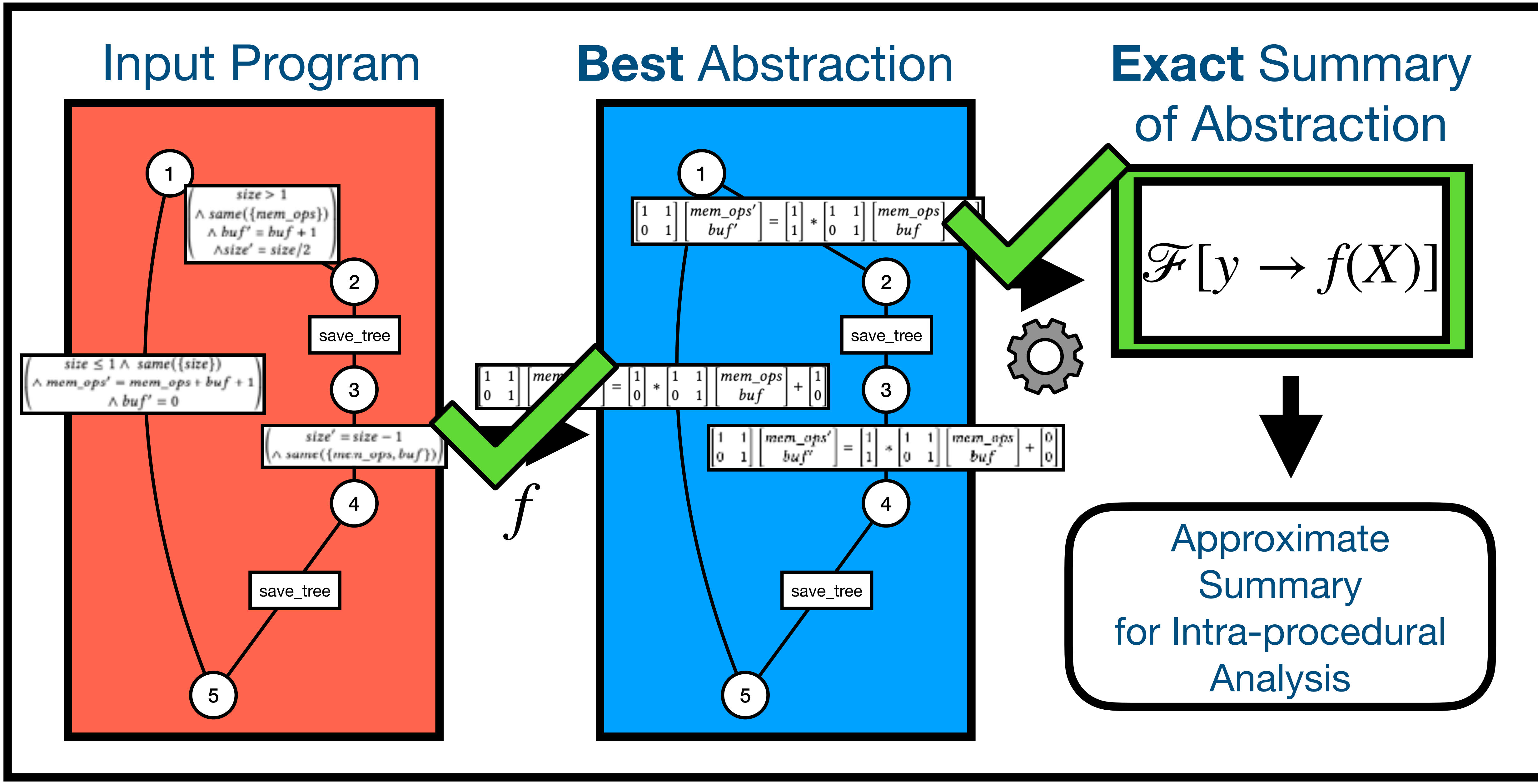
- Showed that the reachability relation of a VASR along Σ^* and regular languages is computable

- [Chistikov 2015] showed that the reachability relation of a VASR along communication-free Petri-net languages is computable

- **Gap Filled:** Our work shows that the reachability relation of a VASR along context-free languages is computable

Context-Free VASR Reachability

What is our logical summary?



We have a monotone inter-procedural analyzer!

Evaluation

What can we use this to analyze?

```c
int end;
int start;
char EOF;

char lexer(char* s, int slen) {
    if (slen <= 0) {return EOF;}
    char c = s[0];
    if (c == '\0') {
        end += 1;
        start = end;
    } else {
        end += 1;
    }
    lexer(s + 1, slen - 1);
}

int main() {
        start = __VERIFIER_nondet_int();
        end = start;
        lexer(0, __VERIFIER_nondet_int());
        __VERIFIER_assert(start <= end);
        return 0;
}
```

```c
int leafs;
int internal_nodes;

void tree_count() {
    if (__VERIFIER_nondet_int()) {
        leafs += 1;
    } else {
        internal_nodes += 1;
        tree_count();
        tree_count();
    }
    return;
}

int main() {
    leafs = 0; internal_nodes = 0;
    tree_count();
    __VERIFIER_assert(internal_nodes +
        1 == leafs);
    return 0;
}
```

Evaluation

What can’t we use this to analyze?

```c
int id (int x) {
    if (x <= 0) {
        return 0;
    } else {
        return id(x - 1) + 1;
    }
}


int main() {
    int number = __VERIFIER_nondet_int();
    int result = id(number);
    __VERIFIER_assert(
        (number < 0 && result == 0) ||
        (result == number));
}
```

```c
int call_count;

void quicksort (int left, int right) {
    call_count += 1;
    if (right - left <= 1) {
        return;
    } else {
        int pivot = __VERIFIER_nondet_int();
        __VERIFIER_assume (left <= pivot &&
            pivot < right);
        quicksort(left, pivot);
        quicksort(pivot + 1, right);
    }
}

int main() {
    call_count = 0;
    int size = __VERIFIER_nondet_int();
    __VERIFIER_assume (1 <= size);
    quicksort(0, size);
    __VERIFIER_assert(call_count <= 2 *
        size + 1
}
```

Q: How can we refine the language considered by our summary?

Q: How can we refine the language considered by our summary?

Q: How can we refine the language considered by our summary?

A: Our summary has variables representing the number of times each edge appears in an execution - we can synthesize bounds on recursive depth and use them to constrain these symbols.

Inductive Linear Bounds

Introduction

- Related to the *potential method* [Tarjan 1985] used in amortized complexity analysis
- Goal is to find a function $\nu_q : (P \times S) \to \mathbb{Z}$ where $\nu_q(p, \rho)$ is a resource bound on the number of times procedure q can be called in any execution of procedure p starting in state ρ
- Potential for example: $\nu_{\texttt{save\_tree}}(\texttt{save\_tree}, \rho)$ $= \max(0, \rho(\texttt{size}))$

```
int mem_ops, buf;
void save_tree(int size) {
    buf += 1;
    if (size <= 1) {
        mem_ops += buf;
        buf = 0;
    } else {
        save_tree((size - 1) / 2);
        save_tree((size - 1) / 2);
    }
}
```

```
void main() {
    mem_ops = 0; buf = 0;
    int size = nondet_int();
    assume(size >= 1);
    save_tree(size);
    assert(mem_ops <= size);
}
```

Inductive Linear Bounds

Inductiveness

- A sufficient condition for being a potential function is *inductiveness:* the potential of any state is ≥ the resource cost and the sub-potentials of any child calls in any execution beginning from that state

```
int mem_ops, buf;
void save_tree(int size) {
    buf += 1;
    if (size <= 1) {
        mem_ops += buf;
        buf = 0;
    } else {
        save_tree((size − 1) / 2);
        save_tree((size − 1) / 2);
    }
}
```

```
void main() {
    mem_ops = 0; buf = 0;
    int size = nondet_int();
    assume(size >= 1);
    save_tree(size);
    assert(mem_ops <= size);
}
```

1

1

Inductive Linear Bounds

Inductiveness

- A sufficient condition for being a potential function is *inductiveness:* the potential of any state is ≥ the resource cost and the sub-potentials of any child calls in any execution beginning from that state

```
int mem_ops, buf;
void save_tree(int size) {
    buf += 1;
    if (size <= 1) {
        mem_ops += buf;
        buf = 0;
    } else {
        save_tree((size - 1) / 2);
        save_tree((size - 1) / 2);
    }
}
```

$$\nu(\text{save\_tree}, \rho) \geq 0 \qquad \text{if } size \leq 1$$

$$\nu(\text{save\_tree}, \rho) \geq 2 + \nu\left(\begin{array}{c}\text{save\_tree},\\ \rho\begin{bmatrix} size \mapsto (\rho(size)-1)/2 \\ buf \mapsto \rho(buf)+1 \end{bmatrix}\end{array}\right) + \nu\left(\begin{array}{c}\text{save\_tree},\\ \rho\begin{bmatrix} size \mapsto (\rho(size)-1)/2 \\ buf \mapsto \rho(buf)+1 \end{bmatrix}\end{array}\right) \qquad \text{if } size > 1$$

Inductive Linear Bounds

Method Overview

- Search for potential functions of the template $\nu(X) = \max(0, \vec{a}^T \overrightarrow{X})$
- Use a black-box intra-procedural analysis over a transformed program to form a constraint system encoding *inductiveness* for a symbolic $\vec{a}$ vector of coefficients
- Leverage polyhedral techniques to solve constraint system
- Construct finite formula which holds iff a variable (Parikh variable representing the number of function calls) is less than a (potentially infinite) set of potential functions
- Bound extraction and application is **monotone** (assuming helper intra-procedural analysis routine is monotone)

Inductive Linear Bounds

Related Work: Carbonneaux, Hoffman, Shao 2015

- Automatically derives linear resource bounds by generating a constraint system via a set of Hoare-logic style inference rules and solving the resulting system with a Linear Programming solver

- Limitation: The Hoare-style inference rules, while sound, do not ensure monotonicity of the resulting constraint system. In particular, the inference rules use a heuristic weakening rule. This can lead to unpredictable effects on the resource bound computed for related programs

- **Gap Filled:** By using a monotone intraprocedural analysis routine, our work is able to synthesize linear bounds matching a similar template in a monotone way

Evaluation

What can we use this to analyze?

```c
int id (int x) {
    if (x <= 0) {
        return 0;
    } else {
        return id(x - 1) + 1;
    }
}


int main() {
    int number = __VERIFIER_nondet_int();
    int result = id(number);
    __VERIFIER_assert(
        (number < 0 && result == 0) ||
        (result == number));
}
```

```c
int call_count;

void quicksort (int left, int right) {
    call_count += 1;
    if (right - left <= 1) {
        return;
    } else {
        int pivot = __VERIFIER_nondet_int();
        __VERIFIER_assume (left <= pivot &&
            pivot < right);
        quicksort(left, pivot);
        quicksort(pivot + 1, right);
    }
}

int main() {
    call_count = 0;
    int size = __VERIFIER_nondet_int();
    __VERIFIER_assume (1 <= size);
    quicksort(0, size);
    __VERIFIER_assert(call_count <= 2 *
        size + 1 );
}
```

Evaluation

	#tasks	VSB #correct	VSB time	Korn #correct	Korn time	UAutomizer #correct	UAutomizer time
Recursive-Safe	17	4	27.6	14	1825.7	12	3366.3
RecursiveSimple-Safe	35	20	102.7	35	67.1	28	5872.7
cfg-crafted	12	12	20.7	4	4202.9	9	1914.4
Total	64	36	151.1	53	6095.7	49	11153.4

	#tasks	VSB #correct	VSB time	VS #correct	VS time	CRA #correct	CRA time
Recursive-Safe	17	4	27.6	4	27.1	3	22.0
RecursiveSimple-Safe	35	20	102.7	19	86.4	13	39.7
cfg-crafted	12	12	20.7	7	20.5	6	14.4
Total	64	36	151.1	30	134.0	22	76.1

Conclusion

What did we achieve?

- **Best Labeled VASR Abstractions** of LIRA transition formula mappings
- **VASR Reachability** along context free languages
- **Inductive Linear Bounds** which are synthesized and applied in a *monotone* way
- An implementation of the end-to-end summarization routine that is comparable to the state of the art on standard benchmarks and outperforms the SOTA in some domains

Future Work

What's next?

- **Extending the VASR Model:** How can we modify the VASR model to better capture program behavior?
- **Develop Abstract Trajectory Analysis:** What are the algebraic qualities of VASRs that allow us to compute its reachability using abstract trajectories? Are there other useful classes of transition systems which meet these conditions?
- **CHC Solving:** How can we apply similar techniques to those found in this work to solve nonlinear Constrained Horn Clause problems?

Procedure Summarization via Vector Addition Systems and Inductive Linear Bounds

General Exam

Nikhil Pimpalkhare

October 2023