Appendix for "Program analysis for Adaptivity Analysis"

In this appendix, we present three languages: high level loop language, low level loop language, SSA language. In the paper, we choose to only use high level loop language and SSA language. The use of low level language helps us to divide control dependency and data dependency, which is useful in designing the framework. So we keep the low level language in the appendix.

Contents

1 System Overview

In adaptive data analysis, a data analysis can depend on the results of previous analysis over the same data. This dependency may affect the *generalization properties of the data analysis*. To study this phenomenon in a formal way, we consider the *statistical query model*. In this model, a dataset X consisting of d attributes (columns) and n individuals' data (rows) can be accessed only through an interface to which one can submit statistical queries. More precisely, suppose that the type of a row is R (as an example, a row with d binary attributes would have type $R = \{0,1\}^d$. Then, in the statistical query model one can access the dataset only by submitting a query to the interface, in the form of a function $p : D \to [0,1]$ where D represents dataset. The collected answer of the asked query is the average result of p on each row in the dataset D. For example, the result is the value $\frac{1}{n}\sum_{i=1}^{n} p(X_i)$ where X_i is the row of index i in X. While this model is rather simple, in fact it supports sufficient statistics one may be interested.

We are interested in the adaptivity of mechanisms in the model, which is straightforward supported by a high level language. In this language, queries are allowed to carry arguments to simulate the process of submitting a query to the interface in the model, for example, the expression $q(e)$ tells us the argument e is consumed to construct the query. To be precise, one submitted query who needs the average of answers of previous queries is expressed as $q(x)$, where the variable x stores the expected average results.This makes these mechanisms quite straightforward to express in the high level language. However, this convenience pays at the price that the adaptivity A of a mechanism P becomes quite tricky to estimate because the definition of dependency between two queries becomes vague in the high level language.

```
$x \leftarrow q_1()$;
if $(x_1 > 0)$
$y \leftarrow q_2(x)$
```

The dependency between two query submissions is the essential of the adaptivity of a mechanism. To study the dependency, we first study its dual, independence between two queries, which is defined to be: one query q_1 does not depend on another query q_2 when the result of q_1 remains the same regardless of the modification of the result of q_2. Hence, it becomes hard to distinguish whether the variance of result of q_1 comes from the control flow or the argument of queries. Since we know that the result of one query from a specific D may vary under different contexts in the high level language.

To resolve the dilemma, we translate any program(mechanism) into its counterpart in a low level language, which mimics the high level one except its only allowing atomic queries, – q –. That is to say, given a data base D, the result of the query from D becomes deterministic. We need to show the two programs P and P^* are observably equivalent over the translation. In this way, we can define the adaptivity of a program under this model only based on the control flow. To be specific, the adaptivity A^* of a low level program P^* is defined by graphs, called dependency graph, which comes from the semantics of the low level program. The dependency graph is constructed using a trace of queries generated along with the semantics: The queries in the trace consists of the nodes in the graph while the edge represents dependency. If there is no dependency between two node(queries), there will be no edge. Intuitively, we want to give an over-approximate of A^* by static analysis. To this end, we propose AdaptFun, which estimates a reasonable upper bound on the arbitrary low level program.

The adaptivity A of arbitrary high level program P is defined to be the minimal of the adaptivity A^* of all the possible P^* via various valid translations. Being valid means the programs before and after the translation are observably equivalent. Naturally, following this definition, the upper bound estimated by AdaptFun is sound with respect to its low level adaptivity A^*, hence the high level one A.

Finally we extend the language to support the probabilistic program and extend the adaptivity definition accordingly.

The key component of the system is an program analysis procedure, which provides an upper bound on the adaptivity of the program.

2 While Language

2.1 Syntax and Semantics

Arithmatic Operators	$\oplus_a$	::=	$+ \mid - \mid \times \mid \div$
Boolean Operators	$\oplus_b$	::=	$\vee \mid \wedge \mid \neg$
Relational Operators	$\sim$	::=	$< \mid \leq \mid ==$
Arithmetic Expressions	a	::=	$n \mid x \mid a \oplus_a a \mid$
Boolean Expressions	b	::=	$\texttt{true} \mid \texttt{false} \mid \neg b \mid b \oplus_b b \mid a \sim a$
Expressions	e	::=	$[JL: a \mid b \mid [] \mid [e, \dots, e]]$
Values	v	::=	$[JL: n \mid \texttt{true} \mid \texttt{false} \mid [] \mid [v, \dots, v]]$
Query expressions	ψ	::=	$[JL: a \mid \chi \mid \chi[a] \mid \psi \oplus_a \psi]$
Query Values	v_q	::=	$[JL: n \mid \chi \mid \chi[n] \mid v_q \oplus_a v_q]$
commands	c	::=	$x \leftarrow e \mid x \leftarrow q(\psi) \mid$
			$\texttt{loop}\ a\ \texttt{do}\ c\ [JL: \texttt{while}\ b\ \texttt{do}\ c] \mid c; c \mid \texttt{if}(b, c, c) \mid \texttt{skip}$
Memory	m	::=	$[] \mid m[x \to v]$

2.2 Rewriting from High Level Program into Low Level Program

The transformation $\llparenthesis e^h \rrparenthesis = e^l$ transfers the expression e^h in the high level language to an expression e_l in the low language. Let us look at the special cases: the query.
In the first transition, if a query in high level language isn't atomic, i.e., $q(e)$ depends on e with free variables, then it will be rewrite into a switch command. This rewriting will switch on the possible values v_i of e and convert the $q(e)$ into a series of atomic queries q_i.
In the second transition, if a query in high level language is atomic, $q()$ only depends on data base D and some constant values, then it will be rewrite into identity in our low level language.
Another special case is the sampling command in high level language. To exclude the dependency caused by the randomness, we will rewrite the sampling into an assignment command in low level language. This will assign a constant value to the corresponding variable.
The resting commands will be rewrote identically.

$$\begin{array}{lcl}
\llparenthesis [x \leftarrow q\ (e)]^l \rrparenthesis & \Rightarrow & \left[\ \texttt{switch}\ \left(e, x, \begin{pmatrix} v_1 \to q_1, \\ \cdots, \\ v_m \to q_m \end{pmatrix}\right)\ \right]^l \\
\llparenthesis x \leftarrow q() \rrparenthesis & \Rightarrow & x \leftarrow q \\
\llparenthesis [x \leftarrow e]^l \rrparenthesis & \Rightarrow & [x \leftarrow e]^l \\
\llparenthesis c_1 ; c_2 \rrparenthesis & \Rightarrow & \llparenthesis c_1 \rrparenthesis ; \llparenthesis c_2 \rrparenthesis \\
\llparenthesis \texttt{if}([b]^l, c_1, c_2) \rrparenthesis & \Rightarrow & \texttt{if}([b]^l, \llparenthesis c_1 \rrparenthesis, \llparenthesis c_2 \rrparenthesis) \\
\llparenthesis \texttt{loop}\ [v_N]^l\ (c_1)\ \texttt{do}\ c_2 \rrparenthesis & \Rightarrow & \texttt{loop}\ [v_N]^l\ (\llparenthesis c_1 \rrparenthesis)\ \texttt{do}\ \llparenthesis c_2 \rrparenthesis
\end{array}$$

3 Low Level Language Based on Control Flow

We first consider a low level language where the queries are atomic and the dependency relations are caused only by control flow.

3.1 Syntax and Semantics

Syntax.

Arithmatic Operators	$\oplus_a$	$::=$	$+ \mid - \mid \times \mid \div$
Boolean Operators	$\oplus_b$	$::=$	$\vee \mid \wedge \mid \neg$
Relational Operators	$\sim$	$::=$	$< \mid \leq \mid ==$
Label	l	$:=$	$\mathbb{N}$
Loop Maps	w	$\in$	$\text{Label} \times \mathbb{N}$
Arithmetic Expressions	a	$::=$	$n \mid x \mid a \oplus_a a$
Boolean Expressions	b	$::=$	$\mathtt{true} \mid \mathtt{false} \mid \neg b \mid b \oplus_b b \mid a \sim a$
Expressions	e	$::=$	$a \mid b \mid [] \mid [e, \dots, e]$
Labelled commands	c	$::=$	$[x \leftarrow e]^l \mid [x \leftarrow q]^l \mid [\mathtt{switch}\,(e, x, v_i \to q_i)]^l \mid$
			$[\mathtt{loop}\ a\ \mathtt{do}\ c]^l$ $[JL:\mathtt{while}\ [b]^l\ \mathtt{do}\ c]$ $\mid c; c \mid [\mathtt{if}(b, c, c)]^l \mid [\mathtt{skip}]^l$
Memory	m	$::=$	$\emptyset \mid (x^l \to v) :: t$
Trace	t	$::=$	$[] \mid ((q^{(l,w)}, v)) :: t$
Annotated Query	$\mathcal{AQ}$	$::=$	$\{q^{(l,w)}\}$

Expressions can be either arithmetic expressions or boolean expressions. An arithmetic expression can be a constant n denoting integer, a variable x from some countable set Var, the empty list $[]$, a list $[a_1, \dots, a_k]$ of arithmetic expressions, A boolean expression can be as usual `true` or `false`, the negation of a boolean expression, or a combination of boolean expressions by means of an operation $\oplus_b$ or the result of a comparison $\sim$ between arithmetic expressions.

Commands in the low-level language are labelled — we assume that labels are unique and corresponds to the line of code where they appear, implicitly this gives us a control flow graph representation for the program. A command can be either a labelled assignment $[x \leftarrow e]^l$, or a labelled query request $[x \leftarrow q]^l$, a labelled skip $[\mathtt{skip}]^l$, the composition of two labelled commands $[c_1; c_2]^l$, a labelled if statement $\mathtt{if}([b, c_1, c_2)]^l$, a labelled loop statement $[\mathtt{loop}\ (a, c_1\ \mathtt{do}\ c_2]^l$ or a labelled switch statement $[\mathtt{switch}\,(e, x, (v_i \to q_i))]^l$.

A memory is a partial map from the labelled variables to values.

A trace is a list which only tracks the queries called and the results of the queries.

For memory and traces we will use notation: we will write $++$ for concatenation of lists in traces, and we will use the standard notation $m[x^l \to v]$ for the update of the memory.

Operational Semantics. Small step operational semantics of expressions are standard and omitted.

$$\boxed{\langle m, a \rangle \to_a a' : Memory \times AExpr \Rightarrow AExpr}$$

$$\frac{\langle m, a \rangle \to_a a'}{\langle m, \chi(a) \rangle \to_a \chi(a')} \qquad \boxed{\langle m, b \rangle \to_b b' : Memory \times BExpr \Rightarrow BExpr}$$

$$\boxed{\langle m, c, t, w\rangle \to \langle m', c', t', w'\rangle}$$

$$Memory \times Com \times Trace \times loopmaps \Rightarrow Memory \times Com \times Trace \times loopmaps$$

$$\frac{q(D) = v}{\langle m, [x \leftarrow q]^{l}, t, w\rangle \to \langle m[v/x], \texttt{skip}, t +\!+ [q^{(l,w)}], w\rangle}\ \textbf{low-query}$$

$$\frac{}{\langle m, [x \leftarrow v]^{l}, t, w\rangle \to \langle m[v/x], [\texttt{skip}]^{l}, t, w\rangle}\ \textbf{low-assn}$$

$$\frac{\langle m, c_1, t, w\rangle \to \langle m', c_1', t', w'\rangle}{\langle m, c_1; c_2, t, w\rangle \to \langle m', c_1'; c_2, t', w'\rangle}\ \textbf{low-seq1} \qquad \frac{}{\langle m, [\texttt{skip}]^{l}; c_2, t, w\rangle \to \langle m, c_2, t, w\rangle}\ \textbf{low-seq2}$$

$$\frac{\langle m, b\rangle \to_b b'}{\langle m, [\texttt{if}(b, c_1, c_2)]^{l}, t, w\rangle \to \langle m, [\texttt{if}(b', c_1, c_2)]^{l}, t, w\rangle}\ \textbf{low-if}$$

$$\frac{}{\langle m, [\texttt{if}(\texttt{true}, c_1, c_2)]^{l}, t, w\rangle \to \langle m, c_1, t, w\rangle}\ \textbf{low-if-t} \qquad \frac{}{\langle m, [\texttt{if}(\texttt{false}, c_1, c_2)]^{l}, t, w\rangle \to \langle m, c_2, t, w\rangle}\ \textbf{low-if-f}$$

$$\frac{\langle m, e\rangle \to e'}{\langle m, [\texttt{switch}\,(e, x, (v_i \to q_i))]^{l}, t, w\rangle \to \langle m, [\texttt{switch}\,(e', x, (v_i \to q_i))]^{l}, t, w\rangle}\ \textbf{switch}$$

$$\frac{}{\langle m, [\texttt{switch}\,(v_k, x, (v_i \to q_i))]^{l}, t, w\rangle \to \langle m, [x \leftarrow q_k]^{l}, t, w\rangle}\ \textbf{low-switch-v}$$

$$\frac{v_N > 0}{\langle m, [\texttt{loop}\ v_N\ \texttt{do}\ c]^{l}, t, w\rangle \to \langle m, c; [\texttt{loop}\ (v_N - 1)\ \texttt{do}\ c]^{l}, t, (w + l)\rangle}\ \textbf{low-loop}$$

$$\frac{v_N = 0}{\langle m, [\texttt{loop}\ v_N\ \texttt{do}\ c]^{l}, t, w\rangle \to \langle m, [\texttt{skip}]^{l}, t, (w \setminus l)\rangle}\ \textbf{low-loop-exit}$$

$$[JL: \frac{}{\langle m, \texttt{while}\ [b]^{l}\ \texttt{do}\ c, t, w\rangle \to \langle m, c; \texttt{if}_w(b, c; \texttt{while}\ [b]^{l}\ \texttt{do}\ c, \texttt{skip}), t, (w + l)\rangle}\ \textbf{low-while-b}]$$

$$[JL: \frac{m, b \to b'}{\langle m, \texttt{if}_w(b, c, \texttt{skip}), t, w\rangle \to \langle m, c; \texttt{if}_w(b', c, \texttt{skip}), t, w\rangle}\ \textbf{low-ifw-b}]$$

$$[JL: \frac{}{\langle m, \texttt{if}_w(\texttt{true}, c, \texttt{skip}), t, w\rangle \to \langle m, c, t, (w + l)\rangle}\ \textbf{low-ifw-true}]$$

$$[JL: \frac{}{\langle m, \texttt{if}_w(\texttt{false}, c, \texttt{skip}), t, w\rangle \to \langle m, \texttt{skip}, t, (w \setminus l)\rangle}\ \textbf{low-ifw-false}]$$

where w_l refers to a map w without the key l.

$$\begin{array}{lll} w \setminus l & = w & l \not\in Keys(w) \\ & = w_l & Otherwise \\ w + l & = w[l \to 0] & l \not\in Keys(w) \\ & w_l[l \to w(l) + 1] & Otherwise \end{array}$$

3.2 Adaptivity of Programs in Low level language

Definition 1 (Label Order). $<_w$ *and* $=_w$.

$$\begin{array}{lll} w_1 =_w w_2 & \triangleq & Keys(w_1) = Keys(w_2) \land \forall k \in Keys(w_1). w_1(k) = w_2(k) \\ \emptyset =_w \emptyset & & \end{array}$$

$mk(w_i) = MinKey(w_i)$

$$\begin{array}{llll} w_1 <_w w_2 & \triangleq & & w_1 = \emptyset \\ & \triangleq & mk(w_1) < mk(w_2) & w_1, w_2 != \emptyset \\ & \triangleq & w_1(mk(w_1)) < w_2(mk(w_2)) & mk(w_1) = mk(w_2) \\ & \triangleq & (w_1 \setminus mk(w_1)) <_w (w_2 \setminus mk(w_2)) & Otherwise \end{array}$$

Definition 2 (Query Direction). *Direction between two queries.*
$\forall q_1, q_2, l_1, l_2, w_1, w_2$. *the execution of query* q_1 *appears before query* q_2*, denoted as*

$$\mathsf{To}(q_1^{l_1, w_1}, q_2^{(l_2, w_2)}) = (q_1^{(l_1, w_1)}) <_q (q_2^{(l_2, w_2)})$$

where

$$(q_1^{(l_1, w_1)}) <_q (q_2^{(l_2, w_2)}) \triangleq \begin{array}{ll} l_1 < l_2 & w_1 = \emptyset \lor w_2 = \emptyset \lor w_1 =_w w_2 \\ w_1 <_w w_2 & \text{Otherwise} \end{array}$$

Definition 3 (Query may dependency). *One query* $q_1(v_1)$ *may depend on another query* $q_2(v_2)$ *in a program c, with a starting loop maps w, denoted as* $\mathsf{DEP}(q_1(v_1)^{(l_1, w_1)}, q_2(v_2)^{(l_2, w_2)}, c, w)$ *is defined as below.*

$$\begin{array}{l} \forall t. \exists m_1, m_3, t_1, t_3. \langle m, c, t, w \rangle \to^* \langle m_1, [x \leftarrow q_1(v_1)]^{l_1}; c_2, t_1, w_1 \rangle \to \\ \langle m_1[q_1(v_1)(D)/x], c_2, t_1 ++ [q_1(v_1)^{(l_1, w_1)}], w_1 \rangle \to^* \langle m_3, \texttt{skip}, t_3, w_3 \rangle \\ \land \Big(q_1(v_1)^{(l_1, w_1)} \in (t_3 - t) \land q_2(v_2)^{(l_2, w_2)} \in (t_3 - t_1) \implies \exists v \in \texttt{codom}(q_1(v_1)), m_3', t_3', w_3'. \\ \langle m_1[v/x], c_2, t_1 ++ [q_1(v_1)^{(l_1, w_1)}], w_1 \rangle \to^* \langle m_3', \texttt{skip}, t_3', w_3' \rangle \land (q_2(v_2)^{(l_2, w_2)}) \not\in (t_3' - t_1) \Big) \\ \land \Big(q_1(v_1)^{(l_1, w_1)} \in (t_3 - t) \land q_2(v_2)^{(l_2, w_2)} \not\in (t_3 - t_1) \implies \exists v \in \texttt{codom}(q_1(v_1)), m_3', t_3', w_3'. \\ \langle m_1[v/x], c_2, t_1 ++ [q_1(v_1)^{(l_1, w_1)}], w_1 \rangle \to^* \langle m_3', \texttt{skip}, t_3', w_3' \rangle \land (q_2(v_2)^{(l_2, w_2)}) \in (t_3' - t_1) \Big) \end{array}$$

Definition 4. *Query may dependency.*
[JL: (Updated Definition) One query $q(v_{q_1})$ *may depend on another query* $q(v_{q_2})$ *in a program c, with a starting loop maps w, a starting memory m, hidden database D, denoted as*

$$\boxed{\langle m, c, t, w\rangle \to \langle m', c', t', w'\rangle}$$

$$\frac{\langle m, e\rangle \to \langle m, e'\rangle}{\langle m, [x \leftarrow e]^l, t, w\rangle \to \langle m, [x \leftarrow e']^l, t, w\rangle}\ \textbf{l-assn1} \qquad \frac{}{\langle m, [x \leftarrow v]^l, t, w\rangle \to \langle m[v/x], [\mathtt{skip}]^l, t, w\rangle}\ \textbf{l-assn2}$$

$$\frac{\langle m, a\rangle \to_a \langle m, a'\rangle}{\langle m, \mathtt{loop}\ [a]^l\ \mathtt{do}\ c, t, w\rangle \to \langle m, \mathtt{loop}\ [a']^l\ \mathtt{do}\ c, t, (w + l)\rangle}\ \textbf{l-loop-a}$$

$$\frac{v_N > 0}{\langle m, \mathtt{loop}\ [v_N]^l\ \mathtt{do}\ c, t, w\rangle \to \langle m, c; \mathtt{loop}\ [(v_N - 1)]^l\ \mathtt{do}\ c, t, (w + l)\rangle}\ \textbf{l-loop}$$

$$\frac{v_N = 0}{\langle m, \mathtt{loop}\ [v_N]^l\ \mathtt{do}\ c, t, w\rangle \to \langle m, [\mathtt{skip}]^l, t, (w \setminus l)\rangle}\ \textbf{l-loop-exit}$$

$$[JL: \frac{}{\langle m, \mathtt{while}\ [b]^l\ \mathtt{do}[c]^{l+1}, t, w\rangle \to \langle m, c; \mathtt{if}_w(b, c; \mathtt{while}\ [b]^l\ \mathtt{do}[c]^{l+1}, \mathtt{skip}), t, (w + l)\rangle}\ \textbf{low-while-b}]$$

$$[JL: \frac{m, b \to b'}{\langle m, \mathtt{if}_w(b, c, \mathtt{skip}), t, w\rangle \to \langle m, \mathtt{if}_w(b', c, \mathtt{skip}), t, w\rangle}\ \textbf{low-ifw-b}]$$

$$[JL: \frac{}{\langle m, \mathtt{if}_w(\mathtt{true}, c, \mathtt{skip}), t, w\rangle \to \langle m, c t, (w + l)\rangle}\ \textbf{low-ifw-true}]$$

$$[JL: \frac{}{\langle m, \mathtt{if}_w(\mathtt{false}, c, \mathtt{skip}), t, w\rangle \to \langle m, \mathtt{skip}, t, (w \setminus l)\rangle}\ \textbf{low-ifw-false}]$$

$$[JL: \frac{\langle m, \psi\rangle \to_q \langle m, \psi'\rangle}{\langle m, [x \leftarrow q(\psi)]^l, t, w\rangle \to \langle m, [x \leftarrow q(\psi')]^l, t, w\rangle}\ \textbf{l-query-e}]$$

$$[JL: \frac{q(v_q) = v}{\langle m, [x \leftarrow q(v_q)]^l, t, w\rangle \to \langle m[v/x], \mathtt{skip}, t \mathrel{++} [q(v_q)^{(l,w)}], w\rangle}\ \textbf{l-query-v}]$$

$$\frac{\langle m, c_1, t, w\rangle \to \langle m', c_1', t', w'\rangle}{\langle m, c_1; c_2, t, w\rangle \to \langle m', c_1'; c_2, t', w'\rangle}\ \textbf{l-seq1} \qquad \frac{}{\langle m, [\mathtt{skip}]^l; c_2, t, w\rangle \to \langle m, c_2, t, w\rangle}\ \textbf{l-seq2}$$

$$\frac{\langle m, b\rangle \to_b b'}{\langle m, \mathtt{if}([b]^l, c_1, c_2), t, w\rangle \to \langle m, \mathtt{if}([b']^l, c_1, c_2), t, w\rangle}\ \textbf{l-if}$$

$$\frac{}{\langle m, \mathtt{if}([\mathtt{true}]^l, c_1, c_2), t, w\rangle \to \langle m, c_1, t, w\rangle}\ \textbf{l-if-t} \qquad \frac{}{\langle m, \mathtt{if}([\mathtt{false}]^l, c_1, c_2), t, w\rangle \to \langle m, c_2, t, w\rangle}\ \textbf{l-if-f}$$

Figure 1: Trace-based operational semantics of loop language.

$\mathsf{DEP}(q(v_{q_1})^{(l_1,w_1)}, q(v_{q_2})^{(l_2,w_2)}, c, w, m, D)$ *is defined below.] [JL:*

$$
\begin{array}{l}
\forall t. \exists m_1, m_3, t_1, t_3, c_2. \\
\left(
\begin{array}{l}
\langle m, c, t, w\rangle \to^* \langle m_1, [x \leftarrow q(v_{q_1})]^{l_1}; c_2, t_1, w_1\rangle \to \\
\langle m_1[q(v_{q_1})(D)/x], c_2, t_1 ++ [q(v_{q_1})^{(l_1,w_1)}], w_1\rangle \to^* \langle m_3, \mathtt{skip}, t_3, w_3\rangle \\
\land \\
\Big(q(v_{q_1})^{(l_1,w_1)} \in_q (t_3 - t) \land q(v_{q_2})^{(l_2,w_2)} \in_q (t_3 - t_1) \\
\implies \exists v \in \mathtt{codom}(q(v_{q_1})), m_3', t_3', w_3'. \\
\langle m_1[v/x], c_2, t_1 ++ [q(v_{q_1})^{(l_1,w_1)}], w_1\rangle \to^* \langle m_3', \mathtt{skip}, t_3', w_3'\rangle \\
\land (q(v_{q_2})^{(l_2,w_2)}) \notin_q (t_3' - t_1)\Big) \\
\land \\
\Big(q(v_{q_1})^{(l_1,w_1)} \in_q (t_3 - t) \land q(v_{q_2})^{(l_2,w_2)} \notin_q (t_3 - t_1) \\
\implies \exists v \in \mathtt{codom}(q(v_{q_1})), m_3', t_3', w_3'. \\
\langle m_1[v/x], c_2, t_1 ++ [q(v_{q_1})^{(l_1,w_1)}], w_1\rangle \to^* \langle m_3', \mathtt{skip}, t_3', w_3'\rangle \\
\land (q(v_{q_2})^{(l_2,w_2)}) \in_q (t_3' - t_1)\Big)
\end{array}
\right)
\end{array}
$$

]

Definition 5. *Dependency Graph.*
Given a program c, a database D, a starting memory m, an initial loop map w, the dependency graph $G_{low}(c, D, m, w) = (V, E)$ is defined as:
$V = \{q(v)^{l,w} \in \mathcal{AQ} \mid \forall t. \exists m', w', t'. \langle m, c, t, w\rangle \to^* \langle m', \mathtt{skip}, t', w'\rangle \land q(v)^{l,w} \in (t' - t)\}$.
$E = \left\{(q(v)^{(l,w)}, q'(v')^{(l',w')}) \in \mathcal{AQ} \times \mathcal{AQ} \;\middle|\; \mathsf{DEP}(q(v)^{(l,w)}, q'(v')^{(l',w')}, c, w, m, D) \land \mathsf{To}(q(v)^{(l,w)}, q(v')^{(l',w')}\right\}$.

Definition 6. *Query-based Dependency Graph*
[JL: (Updated Definition).
Given a program c, a database D, a starting memory m, an initial loop map w, the query-based dependency graph $G(c, D, m, w) = (V, E)$ is defined as:
$V = \{q(v_q)^{l,w} \in \mathcal{AQ} \mid \forall t. \exists m', w', t'. \langle m, c, t, w\rangle \to^* \langle m', \mathtt{skip}, t', w'\rangle \land q(v_q)^{l,w} \in (t' - t)\}$.
$E = \left\{(q(v_q)^{(l,w)}, q(v_q{}')^{(l',w')}) \in \mathcal{AQ} \times \mathcal{AQ} \;\middle|\; \mathsf{DEP}(q(v_q{}')^{(l',w')}, q(v_q)^{(l,w)}, c, w, m, D)\right\}$. *]*

The edge is directed, when an annotated query $q(v_q)^{(l,w)}$ may depend on its previous query $q(v_q{}')^{(l',w')}$, we have the directed edge $(q(v_q)^{(l,w)}, q(v_q{}')^{(l'.w')})$, from $q(v_q)^{(l,w)}$ to $q(v_q{}')^{(l'.w')}$.

Definition 7. *Adaptivity from a dependency graph.*
Given a program c, and a meory m, a database D, the adaptivity of the dependency graph $G_{low}(c, D, m, w) = (V, E)$ is the length of the longest path in the graph. We denote the path from $q_i^{(l,w)}$ to $q_j^{(l',w')}$ as $p(q_i^{(l,w)}, q_j^{(l',w')})$.

$$A^*(c, D, m, w) = \max_{q_i^{(l,w)}, q_j^{(l',w')} \in V} \{|p(q_i^{(l,w)}, q_j^{(l',w')})|\}$$

4 SSA-form language

Syntax of the SSA-form

Arithmatic Operators	$\oplus_a$	::=	$+ \mid - \mid \times \mid \div$
Boolean Operators	$\oplus_b$	::=	$\vee \mid \wedge \mid \neg$
Relational Operators	$\sim$	::=	$< \mid \leq \mid ==$
Label	l	:=	$\mathbb{N}$
Loop Maps	w	$\in$	$\text{Label} \times \mathbb{N}$
Arithmetic Expressions	a	::=	$\mathrm{n} \mid \mathrm{x} \mid \mathrm{a} \oplus_a \mathrm{a}$
Boolean Expressions	b	::=	$\mathrm{true} \mid \mathrm{false} \mid \neg \mathrm{b} \mid \mathrm{b} \oplus_b \mathrm{b} \mid \mathrm{a} \sim \mathrm{a}$
Query expressions	ψ	::=	$[JL: a \mid \chi \mid \chi[a] \mid \psi \oplus_a \psi]$
Query Values	v_q	::=	$[JL: n \mid \chi \mid \chi[n] \mid v_q \oplus_a v_q]$
Expressions	e	::=	$\mathrm{a} \mid \mathrm{b} \mid [] \mid [e, \ldots, e]$
Labelled commands	**c**	::=	$[\mathbf{x} \leftarrow \mathbf{e}]^l \mid [\mathbf{x} \leftarrow \mathbf{q}(\psi)]^l \mid ifvar(\bar{\mathbf{x}}, \bar{\mathbf{x}}')$
			$[\mathtt{loop}\ \mathrm{a}, n, [\bar{\mathbf{x}}, \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}]\ \mathtt{do}\ \mathbf{c}]^l$ $[JL: \mathtt{while}\ [b]^l, n, [\bar{\mathbf{x}}, \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}]\ \mathtt{do}\ c]$
			$\mid \mathbf{c}; \mathbf{c} \mid [\mathtt{if}(\mathrm{b}, [\bar{\mathbf{x}}, \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}], \mathbf{c}, \mathbf{c})]^l \mid [\mathtt{skip}]^l$
Memory	m	::=	$\emptyset \mid (\mathbf{x}^{\mathbf{l}} \to \mathbf{v}) :: \mathbf{m}$
Trace	t	::=	$[] \mid (\mathbf{q}(\mathbf{v_q})^{(\mathbf{l},\mathbf{w})}) :: \mathbf{t}$
Annotated Query	$\mathcal{AQ}$	::=	$[JL: \{q(\psi)^{(l,w)}\}]$
SSA Variables	$\mathcal{SV}$	::=	$\{\mathbf{x}\}$
Labelled SSA Variables	$\mathcal{LV}$	::=	$\{\mathbf{x}^{(l,w)}\}$

4.1 SSA Transformation Rules

We use a translation environment δ, to map variables x in the low level language to those $\mathbf{x}$ in ssa-form language. We use a name environment denoted as Σ, a set of ssa variables so we can get a fresh variable by $fresh(\Sigma)$. We define $\delta_1 \bowtie \delta_2$ in a similar way as [3].

$$\delta_1 \bowtie \delta_2 = \{(x, \mathbf{x_1}, \mathbf{x_2}) \in \mathcal{VAR} \times \mathcal{SV} \times \mathcal{SV} \mid x \mapsto \mathbf{x_1} \in \delta_1, x \mapsto \mathbf{x_2} \in \delta_2, \mathbf{x_1} \neq \mathbf{x_2}\}$$

$$\delta_1 \bowtie \delta_2 / \bar{x} = \{(x, \mathbf{x_1}, \mathbf{x_2}) \in \mathcal{VAR} \times \mathcal{SV} \times \mathcal{SV} \mid x \not\in \bar{x} \wedge x \mapsto \mathbf{x_1} \in \delta_1, x \mapsto \mathbf{x_2} \in \delta_2, \mathbf{x_1} \neq \mathbf{x_2}\}$$

We call a list of variables $\bar{x}$.

$$[\bar{x}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] = \{(x, x_1, x_2) | \forall 0 \leq i < |\bar{x}|, x = \bar{x}[i] \wedge x_1 = \bar{x_1}[i] \wedge x_2 = \bar{x_2}[i] \wedge |\bar{x}| = |\bar{x_1}| = |\bar{x_2}|\}$$

$$\boxed{\delta; e \hookrightarrow \mathbf{e}}$$

$$\frac{}{\delta; x \hookrightarrow \delta(x)}\ \textbf{S-VAR}$$

$$\boxed{\Sigma;\delta;c \hookrightarrow \mathbf{c};\delta';\Sigma'}$$

$$\frac{\begin{array}{c}\delta; b \hookrightarrow \mathbf{b} \qquad \Sigma;\delta;c_1 \hookrightarrow \mathbf{c_1};\delta_1;\Sigma_1 \qquad \Sigma_1;\delta;c_2 \hookrightarrow \mathbf{c_2};\delta_2;\Sigma_2 \\ [\bar{x},\bar{\mathbf{x}}_\mathbf{1},\bar{\mathbf{x}}_\mathbf{2}] = \delta_1 \bowtie \delta_2 \qquad [\bar{y},\bar{\mathbf{y}}_\mathbf{1},\bar{\mathbf{y}}_\mathbf{2}] = \delta \bowtie \delta_1/\bar{x} \qquad [\bar{z},\bar{\mathbf{z}}_\mathbf{1},\bar{\mathbf{z}}_\mathbf{2}] = \delta \bowtie \delta_2/\bar{x} \\ \delta' = \delta[\bar{x} \mapsto \bar{\mathbf{x}}'][\bar{y} \mapsto \bar{\mathbf{y}}'][\bar{z} \mapsto \bar{\mathbf{z}}'] \qquad \bar{\mathbf{x}}',\bar{\mathbf{y}}',\bar{\mathbf{z}}'\ fresh(\Sigma_2) \qquad \Sigma' = \Sigma_2 \cup \{\bar{\mathbf{x}}',\bar{\mathbf{y}}',\bar{\mathbf{z}}'\}\end{array}}{\Sigma;\delta;[\texttt{if}(b,c_1,c_2)]^l \hookrightarrow [\texttt{if}(\mathbf{b},[\bar{\mathbf{x}}',\bar{\mathbf{x}}_\mathbf{1},\bar{\mathbf{x}}_\mathbf{2}],[\bar{\mathbf{y}}',\bar{\mathbf{y}}_\mathbf{1},\bar{\mathbf{y}}_\mathbf{2}],[\bar{\mathbf{z}}',\bar{\mathbf{z}}_\mathbf{1},\bar{\mathbf{z}}_\mathbf{2}],\mathbf{c_1},\mathbf{c_2})]^l;\delta';\Sigma'}\ \textbf{S-IF}$$

$$\frac{\delta; e \hookrightarrow \mathbf{e} \qquad \delta' = \delta[x \mapsto \mathbf{x}] \qquad \mathbf{x}\ fresh(\Sigma) \qquad \Sigma' = \Sigma \cup \{\mathbf{x}\}}{\Sigma;\delta;[x \leftarrow e]^l \hookrightarrow [\mathbf{x} \leftarrow \mathbf{e}]^l;\delta';\Sigma'}\ \textbf{S-ASSN}$$

$$\frac{\delta; q \hookrightarrow \mathbf{q} \qquad \delta; e \hookrightarrow \mathbf{e} \qquad \delta' = \delta[x \mapsto \mathbf{x}] \qquad \mathbf{x}\ fresh(\Sigma) \qquad \Sigma' = \Sigma \cup \{\mathbf{x}\}}{\Sigma;\delta;[x \leftarrow q(e)]^l \hookrightarrow [\mathbf{x} \leftarrow \mathbf{q}(\mathbf{e})]^l;\delta';\Sigma'}\ \textbf{S-QUERY}$$

$$\frac{\delta; q_i \hookrightarrow \mathbf{q_i} \qquad \delta; v_k \hookrightarrow \mathbf{v_k} \qquad \delta; v_i \hookrightarrow \mathbf{v_i} \qquad \delta' = \delta[x \mapsto \mathbf{x}] \qquad \mathbf{x}\ fresh(\Sigma) \qquad \Sigma' = \Sigma \cup \{\mathbf{x}\}}{\Sigma;\delta;[\texttt{switch}\,(v_k,x,(v_i \to q_i))]^l \hookrightarrow [\texttt{switch}\,(\mathbf{v_k},\mathbf{x},(\mathbf{v_i} \to \mathbf{q_i}))]^l;\delta';\Sigma'}\ \textbf{S-SWITCH}$$

$$[[\frac{\begin{array}{c}\delta; a \hookrightarrow \mathbf{a} \qquad \Sigma;\delta;c \hookrightarrow \mathbf{c_1};\delta_1;\Sigma_1 \qquad \Sigma;\delta_1;c \hookrightarrow \mathbf{c_2};\delta_1;\Sigma_1 \\ [\bar{x},\bar{\mathbf{x}}_\mathbf{1},\bar{\mathbf{x}}_\mathbf{2}] = \delta \bowtie \delta_1 \qquad \delta' = \delta[\bar{x} \mapsto \bar{\mathbf{x}}'] \qquad \bar{\mathbf{x}}'\ fresh(\Sigma_1) \qquad \mathbf{c}' = \mathbf{c_1}[\bar{\mathbf{x}}'/\bar{\mathbf{x}}_\mathbf{1}] = \mathbf{c_2}[\bar{\mathbf{x}}'/\bar{\mathbf{x}}_\mathbf{2}]\end{array}}{\Sigma;\delta;[\texttt{loop}\ a\ \texttt{do}\ c]^l \hookrightarrow [\texttt{loop}\ \mathbf{a},\mathbf{0},[\bar{\mathbf{x}}',\bar{\mathbf{x}}_\mathbf{1},\bar{\mathbf{x}}_\mathbf{2}]\ \texttt{do}\ \mathbf{c}']^l;\delta_1[\bar{x} \to \bar{\mathbf{x}}'];\Sigma \cup \{\bar{\mathbf{x}}'\}}\ \textbf{S-LOOP}]]$$

$$[WQ: \frac{\begin{array}{c}\Sigma;\delta;c \hookrightarrow \mathbf{c_1};\delta_1;\Sigma_1 \qquad [\bar{x},\bar{\mathbf{x}}_\mathbf{1},\bar{\mathbf{x}}_\mathbf{2}] = \delta \bowtie \delta_1 \\ \bar{\mathbf{x}}'\ fresh(\Sigma_1) \qquad \delta' = \delta[\bar{x} \mapsto \bar{\mathbf{x}}'] \qquad \delta'; b \hookrightarrow \mathbf{b} \qquad \mathbf{c} = \mathbf{c_1}[\bar{\mathbf{x}}'/\bar{\mathbf{x}}_\mathbf{1}]\end{array}}{\Sigma;\delta;\texttt{while}\ [b]^l\ \texttt{do}\ c \hookrightarrow \texttt{while}\ [\mathbf{b}]^\mathbf{l},\mathbf{0},[\bar{\mathbf{x}}',\bar{\mathbf{x}}_\mathbf{1},\bar{\mathbf{x}}_\mathbf{2}]\ \texttt{do}\ \mathbf{c};\delta';\Sigma_1 \cup \{\bar{\mathbf{x}}'\}}\ \textbf{S-WHILE}]$$

$$\frac{\Sigma;\delta;c_1 \hookrightarrow \mathbf{c_1};\delta_1;\Sigma_1 \qquad \Sigma_1;\delta_1;c_2 \hookrightarrow \mathbf{c_2};\delta';\Sigma'}{\Sigma;\delta;c_1;c_2 \hookrightarrow \mathbf{c_1};\mathbf{c_2}\ ;\delta';\Sigma'}\ \textbf{S-SEQ}$$

Concrete examples.

$$c_1 \triangleq \begin{array}{l} [x \leftarrow q_1]^1; \\ \mathsf{if}\ (x == 0)^2 \\ \mathsf{then}\ [y \leftarrow q_2]^3 \\ \mathsf{else}\ [y \leftarrow 0]^4; \\ \mathsf{if}\ (x == 1)^5 \\ \mathsf{then}\ [y \leftarrow 0]^6 \\ \mathsf{else}\ [y \leftarrow q_2]^7 \end{array} \qquad \hookrightarrow \qquad \begin{array}{l} [\mathbf{x_1} \leftarrow \mathbf{q_1}]^1; \\ \mathsf{if}\ (\mathbf{x_1} == \mathbf{0})^2, [\mathbf{y_3},\mathbf{y_1},\mathbf{y_2}], [], [] \\ \mathsf{then}\ [\mathbf{y_1} \leftarrow \mathbf{q_2}]^3 \\ \mathsf{else}\ [\mathbf{y_2} \leftarrow \mathbf{0}]^4; \\ \mathsf{if}\ (\mathbf{x_1} == \mathbf{1})^5, [\mathbf{y_6},\mathbf{y_4},\mathbf{y_5}] \\ \mathsf{then}\ [\mathbf{y_4} \leftarrow \mathbf{0}]^6 \\ \mathsf{else}\ [\mathbf{y_5} \leftarrow \mathbf{q_2}]^7 \end{array}$$

$$c_2 \triangleq \begin{array}{l} [x \leftarrow q_1]^1; \\ [y \leftarrow q_2]^2; \\ \mathtt{if}\ (x+y==5)^3 \\ \mathtt{then}\ [z \leftarrow q_3]^4 \\ \mathtt{else}\ [\mathtt{skip}]^5; \\ [w \leftarrow q_4]^6; \end{array} \quad \hookrightarrow \quad \begin{array}{l} [\mathbf{x_1} \leftarrow \mathbf{q_1}]^1; \\ [\mathbf{y_1} \leftarrow \mathbf{q_2}]^2; \\ \mathtt{if}\ (\mathbf{x_1}+\mathbf{y_1}==\mathbf{5})^3, [], [], [] \\ \mathtt{then}\ [\mathbf{z_1} \leftarrow \mathbf{q_3}]^4 \\ \mathtt{else}\ [\mathtt{skip}]^5; \\ [\mathbf{w_1} \leftarrow \mathbf{q_4}]^6; \end{array}$$

[JL:

$$c_3 \triangleq \begin{array}{l} [x \leftarrow q_1]^1; \\ [i \leftarrow 0]^2; \\ \mathtt{while}\ [i < 100]^3\ \mathtt{do} \\ \Big([z \leftarrow q_3]^4; \\ [x \leftarrow z+x]^5; \\ [i \leftarrow i+1]^6\Big); \end{array} \quad \hookrightarrow \quad \begin{array}{l} [\mathbf{x_1} \leftarrow \mathbf{q_1}]^1; \\ [\mathbf{i_1} \leftarrow 0]^2; \\ \mathtt{while}\ [\mathbf{i_1} < 100]^3, 0,\ [\mathbf{x_3}, \mathbf{x_1}, \mathbf{x_2}], [\mathbf{i_3}, \mathbf{i_1}, \mathbf{i_2}]\ \mathtt{do} \\ \Big([\mathbf{z_1} \leftarrow \mathbf{q_3}]^4; \\ [\mathbf{x_2} \leftarrow \mathbf{z_1}+\mathbf{x_3}]^5; \\ [\mathbf{i_2} \leftarrow \mathbf{i_3}+1]^6\Big); \end{array}$$

]

Definition 8 (Written variables in a program)**.** *We defined the assigned variables in a low-level program c as* $\mathsf{wr}(c)$*,the assigned variables in the ssa-form program* $\mathbf{c}$ *as* $\mathsf{wr_s}(\mathbf{c})$ *defined as follows.*

$$\begin{array}{lcl} \mathsf{wr}(x \leftarrow e) & = & \{x\} \\ \mathsf{wr}(x \leftarrow q) & = & \{x\} \\ \mathsf{wr}(c_1; c_2) & = & \mathsf{wr}(c_1) \cup \mathsf{wr}(c_2) \\ \mathsf{wr}(\mathtt{loop}\ a\ \mathtt{do}\ c) & = & \mathsf{wr}(c) \\ \mathsf{wr}(\mathtt{if}(b, c_1, c_2)) & = & \mathsf{wr}(c_1) \cup \mathsf{wr}(c_2) \end{array} \qquad \begin{array}{lcl} \mathsf{wr_s}(\mathbf{x} \leftarrow \mathbf{e}) & = & \{x\} \\ \mathsf{wr_s}(\mathbf{x} \leftarrow \mathbf{q}) & = & \{x\} \\ \mathsf{wr_s}(\mathbf{c_1}; \mathbf{c_2}) & = & \mathsf{wr_s}(c_1) \cup \mathsf{wr_s}(c_2) \\ \mathsf{wr_s}(\mathtt{loop}\ \mathbf{a}, \mathbf{n}, [\bar{\mathbf{x}}, \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2]\ \mathtt{do}\ \mathbf{c}) & = & \{\bar{\mathbf{x}}\} \cup \mathsf{wr_s}(\mathbf{c}) \\ \mathsf{wr_s}(\mathtt{if}(\mathbf{b}, [\bar{\mathbf{x}}, \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2], [\bar{\mathbf{y}}, \bar{\mathbf{y}}_1, \bar{\mathbf{y}}_2], [\bar{\mathbf{z}}, \bar{\mathbf{z}}_1, \bar{\mathbf{z}}_2], \mathbf{c_1}, \mathbf{c_2})) & = & \{\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}\} \\ & & \cup \mathsf{wr_s}(\mathbf{c_1}) \cup \mathsf{wr_s}(\mathbf{c_2}) \end{array}$$

The variables read in the low-level programs c as $\mathsf{rd}(c)$, *variables used in ssa-form program* $\mathbf{c}$ *:*

$$\begin{array}{lcl} \mathsf{rd}(x \leftarrow e) & = & \mathsf{rd}(e) \\ \mathsf{rd}(x \leftarrow q) & = & \{\} \\ \mathsf{rd}(c_1; c_2) & = & \mathsf{rd}(c_1) \cup \mathsf{rd}(c_2) \\ \mathsf{rd}(\mathtt{loop}\ a\ \mathtt{do}\ c) & = & \mathsf{rd}(a) \cup \mathsf{rd}(c) \\ \mathsf{rd}(\mathtt{if}(b, c_1, c_2)) & = & \mathsf{rd}(b) \cup \mathsf{rd}(c_1) \cup \mathsf{rd}(c_2) \end{array}$$

$$\begin{array}{lcl} \mathsf{rd_s}(\mathbf{x} \leftarrow \mathbf{e}) & = & \mathsf{rd_s}(\mathbf{e}) \\ \mathsf{rd_s}(\mathbf{x} \leftarrow \mathbf{q}) & = & \{\} \\ \mathsf{rd_s}(\mathbf{c_1}; \mathbf{c_2}) & = & \mathsf{rd_s}(\mathbf{c_1}) \cup \mathsf{rd_s}(\mathbf{c_2}) \\ \mathsf{rd_s}(\mathtt{loop}\ \mathbf{a}, \mathbf{n}, [\bar{\mathbf{x}}, \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2]\ \mathtt{do}\ \mathbf{c}) & = & \mathsf{rd_s}(\mathbf{a}) \cup \mathsf{rd_s}(\mathbf{c}) \cup \{\bar{\mathbf{x}}_1\} \cup \{\bar{\mathbf{x}}_2\} \\ \mathsf{rd_s}(\mathtt{if}(\mathbf{b}, [\bar{\mathbf{x}}, \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2], [\bar{\mathbf{y}}, \bar{\mathbf{y}}_1, \bar{\mathbf{y}}_2], [\bar{\mathbf{z}}, \bar{\mathbf{z}}_1, \bar{\mathbf{z}}_2], \mathbf{c_1}, \mathbf{c_2})) & = & \mathsf{rd_s}(\mathbf{b}) \cup \mathsf{rd_s}(\mathbf{c_1}) \cup \mathsf{rd_s}(\mathbf{c_2}) \cup \{\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2, \bar{\mathbf{y}}_1, \bar{\mathbf{y}}_2, \bar{\mathbf{z}}_1, \bar{\mathbf{z}}_2\} \end{array}$$

We call the variables needed to be assigned before executing the program c as necessary variables $\mathsf{FV}(c)$. *Its ssa form is :* $\mathsf{nv_s}(\mathbf{c})$.

$$\begin{array}{lcl} nv(x \leftarrow e) & = & \mathsf{rd}(e) \\ nv(x \leftarrow q) & = & \{\} \\ nv(\mathtt{loop}\ a\ \mathtt{do}\ c) & = & \mathsf{rd}(a) \cup nv(c) \\ nv(\mathtt{if}(b, c_1, c_2)) & = & \mathsf{rd}(b) \cup nv(c_1) \cup nv(c_2) \\ nv(c_1; c_2) & = & nv(c_1) \cup (nv(c_2) - \mathsf{wr}(c_1)) \end{array}$$

$$
\begin{array}{lcl}
\mathsf{nv_s}(\mathbf{x} \leftarrow \mathbf{e}) & = & \mathsf{rd_s}(\mathbf{e}) \\
\mathsf{nv_s}(\mathbf{x} \leftarrow \mathbf{q}) & = & \{\} \\
\mathsf{nv_s}(\texttt{loop}\ \mathbf{a}, \mathbf{n}, [\bar{\mathbf{x}}, \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2]\ \texttt{do}\ \mathbf{c}) & = & \mathsf{rd_s}(\mathbf{a}) \cup \mathsf{nv_s}(\mathbf{c})[\bar{\mathbf{x}}_1 / \bar{\mathbf{x}}] \\
\mathsf{nv_s}(\texttt{if}(\mathbf{b}, [\bar{\mathbf{x}}, \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2], [\bar{\mathbf{y}}, \bar{\mathbf{y}}_1, \bar{\mathbf{y}}_2], [\bar{\mathbf{z}}, \bar{\mathbf{z}}_1, \bar{\mathbf{z}}_2], \mathbf{c}_1, \mathbf{c}_2)) & = & \mathsf{rd_s}(\mathbf{b}) \cup \mathsf{nv_s}(\mathbf{c}_1) \cup \mathsf{nv_s}(\mathbf{c}_2) \\
\mathsf{nv_s}(\mathbf{c}_1; \mathbf{c}_2) & = & \mathsf{nv_s}(\mathbf{c}_1) \cup (\mathsf{nv_s}(\mathbf{c}_2) - \mathsf{wr_s}(\mathbf{c}_1))
\end{array}
$$

Lemma 1. *If* $\Sigma; \delta; c \hookrightarrow \mathbf{c}; \delta'; \Sigma'$, $\mathsf{nv_s}(\mathbf{c}) = \delta(nv(c))$.

Proof. By induction on the transformation.

- **Case** $\dfrac{\begin{array}{c}\delta; b \hookrightarrow \mathbf{b} \qquad \delta; c_1 \hookrightarrow \mathbf{c_1}; \delta_1 \qquad \delta; c_2 \hookrightarrow \mathbf{c_2}; \delta_2 \\ [\bar{x}, \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2] = \delta_1 \bowtie \delta_2 \qquad [\bar{y}, \bar{\mathbf{y}}_1, \bar{\mathbf{y}}_2] = \delta \bowtie \delta_1 / \bar{x} \qquad [\bar{z}, \bar{\mathbf{z}}_1, \bar{\mathbf{z}}_2] = \delta \bowtie \delta_2 / \bar{x} \\ \delta' = \delta[\bar{x} \mapsto \bar{\mathbf{x}}'] \qquad \bar{\mathbf{x}}', \bar{\mathbf{y}}', \bar{\mathbf{z}}'\ fresh \end{array}}{\delta; [\texttt{if}(b, c_1, c_2)]^l \hookrightarrow [\texttt{if}(\mathbf{b}, [\bar{\mathbf{x}}', \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2], [\bar{\mathbf{y}}', \bar{\mathbf{y}}_1, \bar{\mathbf{y}}_2], [\bar{\mathbf{z}}', \bar{\mathbf{z}}_1, \bar{\mathbf{z}}_2], \mathbf{c_1}, \mathbf{c_2})]^l; \delta'}$ **S-IF**

  From the definition of $\mathsf{nv_s}([\texttt{if}(\mathrm{b}, [\bar{\mathbf{x}}', \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2], \mathbf{c_1}, \mathbf{c_2})]^l) = \mathsf{rd_s}(\mathbf{b}) \cup \mathsf{nv_s}(\mathbf{c_1}) \cup \mathsf{nv_s}(\mathbf{c_2})$. We want to show:

  $$\mathsf{rd_s}(\mathbf{b})) \cup \mathsf{nv_s}(\mathbf{c_1}) \cup \mathsf{nv_s}(\mathbf{c_2}) = \delta(\mathsf{rd}(b)) \cup \delta(\mathsf{FV}(c_1)) \cup \delta(\mathsf{FV}(c_2))$$

  By induction hypothesis on the second and third premise, we know that : $\mathsf{nv_s}(\mathbf{c_1}) = \delta(\mathsf{FV}(c_1))$ and $\mathsf{nv_s}(\mathbf{c_2}) = \delta(\mathsf{FV}(c_2))$. We still need to show that:

  $$\mathsf{rd_s}(\mathbf{b}) = \delta(\mathsf{rd}(b))$$

  From the first premise, we know that $\mathsf{rd}(b) \subseteq \mathtt{dom}(\delta)$. This is goal is proved by the rule **S-VAR** on all the variables in b.

- **Case** [[$\dfrac{\begin{array}{c}\delta; a \hookrightarrow \mathbf{a} \qquad \Sigma; \delta; c \hookrightarrow \mathbf{c_1}; \delta_1; \Sigma_1 \qquad \Sigma; \delta_1; c \hookrightarrow \mathbf{c_2}; \delta_1; \Sigma_1 \\ [\bar{x}, \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2] = \delta \bowtie \delta_1 \qquad \delta' = \delta[\bar{x} \mapsto \bar{\mathbf{x}}'] \qquad \bar{\mathbf{x}}'\ fresh(\Sigma_1) \qquad \mathbf{c}' = \mathbf{c_1}[\bar{\mathbf{x}}' / \bar{\mathbf{x}}_1] = \mathbf{c_2}[\bar{\mathbf{x}}' / \bar{\mathbf{x}}_2] \end{array}}{\Sigma; \delta; [\texttt{loop}\ a\ \texttt{do}\ c]^l \hookrightarrow [\texttt{loop}\ \mathbf{a}, \mathbf{0}, [\bar{\mathbf{x}}', \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2]\ \texttt{do}\ \mathbf{c}']^l; \delta_1[\bar{x} \to \bar{\mathbf{x}}']; \Sigma \cup \{\bar{\mathbf{x}}'\}}$ **S-LOOP**]]

  Unfolding the definition, we need to show:

  $$\mathsf{rd_s}(\mathbf{a}) \cup \mathsf{nv_s}(\mathbf{c}')[\bar{\mathbf{x}}_1 / \bar{\mathbf{x}}] = \delta(\mathsf{rd}(a)) \cup \delta(\mathsf{FV}(c))$$

  We can similarly show that $\mathsf{rd_s}(\mathbf{a}) = \delta(\mathsf{rd}(a))$ as in the if case. We still need to show that:

  $$\mathsf{nv_s}(\mathbf{c_1}[\bar{\mathbf{x}}' / \bar{\mathbf{x}}_1])[\bar{\mathbf{x}}_1 / \bar{\mathbf{x}}'] = \delta(\mathsf{FV}(c))$$

  It is proved by induction hypothesis on $\Sigma; \delta; c \hookrightarrow \mathbf{c_1}; \delta_1; \Sigma_1$.

- **Case** $\dfrac{\Sigma; \delta; c_1 \hookrightarrow \mathbf{c_1}; \delta_1; \Sigma_1 \qquad \Sigma_1; \delta_1; c_2 \hookrightarrow \mathbf{c_2}; \delta'; \Sigma'}{\Sigma; \delta; c_1; c_2 \hookrightarrow \mathbf{c_1}; \mathbf{c_2}\ ; \delta'; \Sigma'}$ **S-SEQ**

  To show:

  $$\mathsf{nv_s}(\mathbf{c_1}) \cup (\mathsf{nv_s}(\mathbf{c_2}) - \mathsf{wr_s}(\mathbf{c_1})) = \delta(\mathsf{FV}(c_1)) \cup \delta(\mathsf{FV}(c_2) - \mathsf{wr}(c_1))$$

  By induction hypothesis on the first premise, we know that : $\mathsf{nv_s}(\mathbf{c_1}) = \delta(\mathsf{FV}(c_1))$, still to show:

  $$(\mathsf{nv_s}(\mathbf{c_2}) - \mathsf{wr_s}(\mathbf{c_1})) = \delta(\mathsf{FV}(c_2) - \mathsf{wr}(c_1))$$

  We know that $\delta_1 = \delta[\mathsf{wr}(c_1) \mapsto \mathsf{wr_s}(\mathbf{c_1})]$, so by induction hypothesis, we know: $\mathsf{nv_s}(\mathbf{c_2}) = \delta[\mathsf{wr}(c_1) \mapsto \mathsf{wr_s}(\mathbf{c_1})](\mathsf{FV}(c_2)) = \delta(\mathsf{FV}(c_2)) \cup \mathsf{wr_s}(\mathbf{c_1}) - \delta(\mathsf{wr}(c_1))$.

  This case is proved.

□

Definition 9 (Well defined memory). *1.* $m \vDash \delta \triangleq \forall x \in \texttt{dom}(\delta), \exists v, (x, v) \in m.$

2. $\mathbf{m} \vDash_{ssa} \delta \triangleq \forall \mathbf{x} \in \texttt{codom}(\delta), \exists v, (\mathbf{x}, v) \in \mathbf{m}.$

Definition 10 (Inverse of transformed memory). $m = \delta^{-1}(\mathbf{m}) \triangleq \forall x \in \texttt{dom}(\delta), (\delta(x), m(x)) \in \mathbf{m}.$

Lemma 2 (value remains during transformation). *Given* $\delta; e \hookrightarrow \mathbf{e}$, $\forall m.m \vDash \delta.\forall \mathbf{m}, \mathbf{m} \vDash_{ssa} \delta \wedge m = \delta^{-1}(\mathbf{m})$, $\forall m.m \vDash \delta.\forall \mathbf{m}, \mathbf{m} \vDash_{ssa} \delta \wedge m = \delta^{-1}(\mathbf{m})$, *then* $\langle m, e \rangle \to v$ *and* $\langle \mathbf{m}, \mathbf{e} \rangle \to \mathbf{v}$ *and* $v = \mathbf{v}$.

Lemma 3 (Value remains the same during transformation). *[JL: Updated Definition Given* $\delta; e \hookrightarrow \mathbf{e}$, $\forall m.m \vDash \delta.\forall \mathbf{m}, \mathbf{m} \vDash_{ssa} \delta \wedge m = \delta^{-1}(\mathbf{m})$, *then* $\langle m, e \rangle \to v$ *and* $\langle \mathbf{m}, \mathbf{e} \rangle \to v$. *]*

Theorem 4.1 (Soundness of transformation). *Given* $\Sigma; \delta; c \hookrightarrow \mathbf{c}; \delta'; \Sigma'$, $\forall m.m \vDash \delta.\forall \mathbf{m}, \mathbf{m} \vDash_{ssa} \delta \wedge m = \delta^{-1}(\mathbf{m})$, *if there exist an execution of* c *in the loop language, starting with a trace* t *and loop maps* w, $\langle m, c, t, w \rangle \to^* \langle m', \texttt{skip}, t', w' \rangle$, *then there also exists a corresponding execution of* $\mathbf{c}$ *in the ssa language so that* $\langle \mathbf{m}, \mathbf{c}, t, w \rangle \to^* \langle \mathbf{m}', \texttt{skip}, t', w' \rangle$ *and* $m' = \delta'^{-1}(\mathbf{m}')$.

Proof. We assume that q is the same when transformed to $\mathbf{q}$, as the primitive in both languages. And the value remains the same during the transformation. It is proved by induction on the transformation rules.

- **Case** $\dfrac{\Sigma; \delta; c_1 \hookrightarrow \mathbf{c_1}; \delta_1; \Sigma_1 \qquad \Sigma_1; \delta_1; c_2 \hookrightarrow \mathbf{c_2}; \delta'; \Sigma'}{\Sigma; \delta; c_1; c_2 \hookrightarrow \mathbf{c_1}; \mathbf{c_2}\ ; \delta'; \Sigma'}$ **S-SEQ**

  We choose an arbitrary memory m so that $m \vDash \delta$, we choose a trace t and a loop maps w.

  $$\frac{\langle m, c_1, t, w \rangle \to^* \langle m_1, \texttt{skip}, t_1, w_1 \rangle \qquad \langle m_1, c_2, t_1, w_1 \rangle \to^* \langle m', \texttt{skip}, t', w' \rangle}{\langle m, c_1; c_2, t, w \rangle \to^* \langle m', \texttt{skip}, t', w' \rangle}$$

  We choose the transformed memory $\mathbf{m}$ so that $m = \delta^{-1}(\mathbf{m})$.

  To show: $\langle \mathbf{m}, \mathbf{c_1}; \mathbf{c_2}, t, w \rangle \to^* \langle \mathbf{m}', \texttt{skip}, t'.w' \rangle$ and $m' = \delta'^{-1}(\mathbf{m}')$.

  By induction hypothesis on the first premise, we have:

  $$\langle \mathbf{m}, \mathbf{c_1}, t, w \rangle \to^* \langle \mathbf{m_1}, \texttt{skip}, t_1, w_1 \rangle \wedge m_1 = \delta_1^{-1}(\mathbf{m_1})$$

  By induction hypothesis on the second premise, using the conclusion $m_1 = \delta_1^{-1}(\mathbf{m_1})$. We have:

  $$\langle \mathbf{m_1}, \mathbf{c_2}, t_1, w_1 \rangle \to^* \langle \mathbf{m}', \texttt{skip}, t', w' \rangle \wedge m' = \delta'^{-1}(\mathbf{m}')$$

  So we know that

  $$\frac{\langle \mathbf{m}, \mathbf{c_1}, t, w \rangle \to^* \langle \mathbf{m_1}, \texttt{skip}, t_1, w_1 \rangle \qquad \langle \mathbf{m_1}, \mathbf{c_2}, t_1, w_1 \rangle \to^* \langle \mathbf{m}', \texttt{skip}, t', w' \rangle}{\langle \mathbf{m}, \mathbf{c_1}; \mathbf{c_2}, t, w \rangle \to^* \langle \mathbf{m}', \texttt{skip}, t', w' \rangle}$$

- **Case** $\dfrac{\delta; e \hookrightarrow \mathbf{e} \qquad \delta' = \delta[x \mapsto \mathbf{x}] \qquad \mathbf{x}\ fresh(\Sigma) \qquad \Sigma' = \Sigma \cup \{x\}}{\Sigma; \delta; [x \leftarrow e]^l \hookrightarrow [\mathbf{x} \leftarrow \mathbf{e}]^l; \delta'; \Sigma'}$ **S-ASSN**

We choose an arbitrary memory m so that $m \vDash \delta$, we choose a trace t and a loop maps w, we know that the resulting trace is still t from its evaluation rule **low-assn** when we suppose $m, e \to v$.

$$\frac{}{\langle m, [x \leftarrow v]^l, t, w\rangle \to \langle m[v/x], [\texttt{skip}]^l, t, w\rangle}\ \textbf{low-assn}$$

We choose the transformed memory $\mathbf{m}$ so that $m = \delta^{-1}(\mathbf{m})$.

To show: $\langle \mathbf{m}, [\mathbf{x} \leftarrow \mathbf{e}]^l, t, w\rangle \to^* \langle \mathbf{m}', \texttt{skip}, t, w\rangle$ and $m' = \delta'^{-1}(\mathbf{m}')$.

From the rule **ssa-assn**, we assume $\mathbf{m}, \mathbf{e} \to \mathbf{v}$, we know that

$$\frac{}{\langle \mathbf{m}, [\mathbf{x} \leftarrow \mathbf{v}]^{\mathbf{l}}, t, w\rangle \to \langle \mathbf{m}[\mathbf{x} \mapsto \mathbf{v}], [\texttt{skip}]^{\mathbf{l}}, t, w\rangle}\ \textbf{ssa-assn}$$

We also know that $\delta' = \delta[x \mapsto \mathbf{x}]$ and $m = \delta^{-1}(\mathbf{m})$, $m' = m[v/x]$. We can show that $m[v/x] = \delta[x \mapsto \mathbf{x}]^{-1}(\mathbf{m}[\mathbf{x} \mapsto v])$.

- **Case** $\dfrac{\delta; q \hookrightarrow \mathbf{q} \qquad \delta; e \hookrightarrow \mathbf{e} \qquad \delta' = \delta[x \mapsto \mathbf{x}] \qquad \mathbf{x}\ fresh(\Sigma) \qquad \Sigma' = \Sigma \cup \{x\}}{\Sigma; \delta; [x \leftarrow q(e)]^l \hookrightarrow [\mathbf{x} \leftarrow \mathbf{q}(\mathbf{e})]^l; \delta'}$ **S-QUERY**

  We choose an arbitrary memory m so that $m \vDash \delta$, we choose a trace t and a loop maps w, we know that when we suppose $\langle m, e\rangle \to v$.

$$\frac{q(v)(D) = v_q}{\langle m, [x \leftarrow q(v)]^l, t, w\rangle \to \langle m[v_q/x], \texttt{skip}, t \mathbin{++} [q(v)^{(l,w)}], w\rangle}\ \textbf{low-query}$$

  We choose the transformed memory $\mathbf{m}$ so that $m = \delta^{-1}(\mathbf{m})$.

  To show: $\langle \mathbf{m}, [\mathbf{x} \leftarrow \mathbf{q}(\mathbf{e})]^l, t, w\rangle \to^* \langle \mathbf{m}', \texttt{skip}, t, w\rangle$ and $m' = \delta'^{-1}(\mathbf{m}')$.

  From the rule **ssa-query**, we know that

$$\frac{\mathbf{q}(\mathbf{v})(\mathbf{D}) = \mathbf{v_q}}{\langle \mathbf{m}, [\mathbf{x} \leftarrow \mathbf{q}(\mathbf{e})]^{\mathbf{l}}, t, w\rangle \to \langle \mathbf{m}[\mathbf{x} \mapsto \mathbf{v}], \texttt{skip}, t \mathbin{++} [(q^{(l,w)}, v)], w\rangle}\ \textbf{ssa-query}$$

  We also know that $\delta' = \delta[x \mapsto \mathbf{x}]$ and $m = \delta^{-1}(\mathbf{m})$, $m' = m[v/x]$. We can show that $m[v/x] = \delta[x \mapsto \mathbf{x}]^{-1}(\mathbf{m}[\mathbf{x} \mapsto v])$.

- **Case** $\dfrac{\begin{array}{c}\delta; b \hookrightarrow \mathbf{b} \qquad \Sigma; \delta; c_1 \hookrightarrow \mathbf{c_1}; \delta_1; \Sigma_1 \qquad \Sigma_1; \delta; c_2 \hookrightarrow \mathbf{c_2}; \delta_2; \Sigma_2 \\ [\bar{x}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] = \delta_1 \bowtie \delta_2 \qquad [\bar{y}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}] = \delta \bowtie \delta_1/\bar{x} \qquad [\bar{z}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}] = \delta \bowtie \delta_2/\bar{x} \\ \delta' = \delta[\bar{x} \mapsto \bar{\mathbf{x}}'][\bar{y} \mapsto \bar{\mathbf{y}}'][\bar{z} \mapsto \bar{\mathbf{z}}'] \qquad \bar{\mathbf{x}}', \bar{\mathbf{y}}', \bar{\mathbf{z}}'\ fresh(\Sigma_2) \qquad \Sigma' = \Sigma_2 \cup \{\bar{\mathbf{x}}', \bar{\mathbf{y}}', \bar{\mathbf{z}}'\}\end{array}}{\Sigma; \delta; [\texttt{if}(b, c_1, c_2)]^l \hookrightarrow [\texttt{if}(\mathbf{b}, [\bar{\mathbf{x}}', \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}', \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}', \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2})]^l; \delta'; \Sigma'}$ **S-IF**

  We choose an arbitrary memory m so that $m \vDash \delta$, we choose a trace t and a loop maps w. There are two possible evaluation rules depending on the the condition b, we choose the case when $b = \texttt{true}$, we know there is an execution in ssa language so that $\mathbf{b} = \texttt{true}$, we use the rule **low-if-t**.

$$\frac{}{\langle m, [\texttt{if}(\texttt{true}, c_1, c_2)]^l, t, w\rangle \to \langle m, c_1, t, w\rangle \to^* \langle m', \texttt{skip}, t', w'\rangle}$$

  We choose the transformed memory $\mathbf{m}$ so that $m = \delta^{-1}(\mathbf{m})$.

  To show: $\langle \mathbf{m}, [\texttt{if}(\texttt{true}, [\bar{\mathbf{x}}', \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}', \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}', \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], c_1, c_2)]^l, t, w\rangle \to^* \langle \mathbf{m}', \texttt{skip}, t', w'\rangle$ and $m' = \delta'^{-1}(\mathbf{m}')$.

We use the corresponding rule **SSA-IF-T**.

$$\frac{}{\begin{array}{l}\langle \mathbf{m}, [\mathtt{if}(\mathtt{true}, [\bar{\mathbf{x}}', \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}], [\bar{\mathbf{y}}', \bar{\mathbf{y}}_1, \bar{\mathbf{y}_2}], [\bar{\mathbf{z}}', \bar{\mathbf{z}_1}, \bar{\mathbf{z}_2}], \mathbf{c_1}, \mathbf{c_2})]^{\mathbf{l}}, t, w\rangle \to \\ \quad \langle \mathbf{m}, \mathbf{c_1}; ifvar(\bar{\mathbf{x}}', \bar{\mathbf{x}_1}); ifvar(\bar{\mathbf{y}}', \bar{\mathbf{y}_2}); ifvar(\bar{\mathbf{z}}', \bar{\mathbf{z}_1}), t, w\rangle\end{array}} \textbf{ ssa-if-t}$$

By induction hypothesis on $\Sigma; \delta; c_1 \hookrightarrow \mathbf{c_1}; \delta_1; \Sigma_1$, and we know that $\langle m, c_1, t, w\rangle \to^* \langle m', \mathtt{skip}, t', w'\rangle$, from our assumption that $m = \delta^{-1}(\mathbf{m})$, we know that

$$\langle \mathbf{m}, \mathbf{c_1}, t, w\rangle \to^* \langle \mathbf{m_1}, \mathtt{skip}, t', w'\rangle \wedge m' = \delta_1^{-1}(\mathbf{m_1})$$

and we then have:

$$\frac{\langle \mathbf{m}, \mathbf{c_1}, t, w\rangle \to^* \langle \mathbf{m_1}, \mathtt{skip}, t', w'\rangle}{\langle \mathbf{m}, \mathbf{c_1}; ifvar(\bar{\mathbf{x}}', \bar{\mathbf{x}_1}); ifvar(\bar{\mathbf{y}}', \bar{\mathbf{y}_1}); ifvar(\bar{\mathbf{z}}', \bar{\mathbf{z}_1}), t, w\rangle \to^* \langle \mathbf{m_1}[\bar{\mathbf{x}}' \mapsto \mathbf{m_1}(\bar{\mathbf{x}_1}), \bar{\mathbf{y}}' \mapsto \mathbf{m_1}(\bar{\mathbf{y}_2}), \bar{\mathbf{z}}' \mapsto \mathbf{m_1}(\bar{\mathbf{z}_1})], \mathtt{skip}, t',}$$

Now, we want to show that $m' = \delta[\bar{x} \mapsto \bar{\mathbf{x}}', \bar{y} \mapsto \bar{\mathbf{y}}', \bar{z} \mapsto \bar{\mathbf{z}}']^{-1}(\mathbf{m_1}[\bar{\mathbf{x}}' \mapsto \mathbf{m_1}(\bar{\mathbf{x}_1}), \bar{\mathbf{y}}' \mapsto \mathbf{m_1}(\bar{\mathbf{y}_2}), \bar{\mathbf{z}}' \mapsto \mathbf{m_1}(\bar{\mathbf{z}_1})])$.

Unfold the definition, we want to show that

$\forall x \in (\mathtt{dom}(\delta) \cup \bar{x} \cup \bar{y} \cup \bar{z}), (\delta[\bar{x} \mapsto \bar{\mathbf{x}}', \bar{y} \mapsto \bar{\mathbf{y}}', \bar{z} \mapsto \bar{\mathbf{z}}'](x), m'(x)) \in \mathbf{m_1}[\bar{\mathbf{x}}' \mapsto \mathbf{m_1}(\bar{\mathbf{x}_1}), \bar{\mathbf{y}}' \mapsto \mathbf{m_1}(\bar{\mathbf{y}_2}), \bar{\mathbf{z}}' \mapsto \mathbf{m_1}(\bar{\mathbf{z}_1})]$.

1. For variable x in $\bar{x}$, we can find a corresponding ssa variable $\mathbf{x} \in \bar{\mathbf{x}}'$, so that $(\mathbf{x}, m'(x)) \in \mathbf{m_1}[\bar{\mathbf{x}}' \mapsto \mathbf{m_1}(\bar{\mathbf{x}_1})]$. It is because we know $[x \mapsto \mathbf{x_1}]$ for certain $\mathbf{x_1} \in \bar{\mathbf{x}_1}$ in δ_1, then by unfolding $m' = \delta_1^{-1}(\mathbf{m_1})$ and $\bar{\mathbf{x}_1} \in \mathtt{codom}(\delta_1)$, we know $(\mathbf{x_1}, m'(x)) \in \mathbf{m_1}$ so that $m'(x) = \mathbf{m_1}(\mathbf{x_1})$.
2. For variable $y \in \bar{y}$, we know that $y \in \mathtt{dom}(\delta_1)$, then $[y \mapsto \mathbf{y_2}]$ for certain $\mathbf{y_2} \in \bar{\mathbf{y}_2}$ in δ_1. So we know that $(\delta_1(y), m'(y)) \in \mathbf{m_1}$, and then $m'(y) = \mathbf{m_1}(\mathbf{y_2})$. We can show $(\mathbf{y}, m'(y)) \in \mathbf{m_1}[\bar{\mathbf{y}}' \mapsto \mathbf{m_1}(\bar{\mathbf{y}_2})]$.
3. For variable $z \in \bar{z}$, we know that $z \notin \mathtt{dom}(\delta_1)$ by the definition (otherwise z will appear in $\bar{x}$), then $[z \mapsto \mathbf{z_1}]$ for certain $\mathbf{z_1} \in \bar{\mathbf{z}_1}$ in δ. We know $(\delta(z), m(z)) \in \mathbf{m}$ from our assumption, so we have $m(z) = \mathbf{m}(\mathbf{z_1})$. Because z is not modified in c_1, so that $m(z) = m'(z)$. Also $\mathbf{m}$ will not shrink during execution and $\mathbf{z_1}$ will not be written in $\mathbf{c_1}$, so $(\mathbf{z_1}, m'(z)) \in \mathbf{m_1}$. Then we can show that $(\mathbf{z}, m'(z)) \in \mathbf{m_1}[\bar{\mathbf{z}}' \mapsto \mathbf{m_1}(\bar{\mathbf{z}_1})]$.
4. For variable $k \in \mathtt{dom}(\delta) - \bar{x} - \bar{y} - \bar{z}$. From our assumption $m = \delta^{-1}(\mathbf{m})$, we can show $(\delta(k), m(k)) \in \mathbf{m}$. We know that k is not written in either branch from our definition, so $(\delta(k), m'(k)) \in \mathbf{m_1}$.

- **Case** [[$\dfrac{\begin{array}{c}\delta; a \hookrightarrow \mathbf{a} \qquad \Sigma; \delta; c \hookrightarrow \mathbf{c_1}; \delta_1; \Sigma_1 \qquad \Sigma; \delta_1; c \hookrightarrow \mathbf{c_2}; \delta_1; \Sigma_1 \\ [\bar{x}, \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}] = \delta \bowtie \delta_1 \qquad \delta' = \delta[\bar{x} \mapsto \bar{\mathbf{x}}'] \qquad \bar{\mathbf{x}}'\ fresh(\Sigma_1) \qquad \mathbf{c}' = \mathbf{c_1}[\bar{\mathbf{x}}'/\bar{\mathbf{x}_1}] = \mathbf{c_2}[\bar{\mathbf{x}}'/\bar{\mathbf{x}_2}]\end{array}}{\delta; [\mathtt{loop}\ a\ \mathtt{do}\ c]^l \hookrightarrow [\mathtt{loop}\ \mathbf{a}, \mathbf{0}, [\bar{\mathbf{x}}', \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}]\ \mathtt{do}\ \mathbf{c}']^l; \delta[\bar{x} \to \bar{\mathbf{x}}']; \Sigma_1 \cup \{\bar{\mathbf{x}}'\}}$ **S-LOOP**]]
  We choose an arbitrary memory m so that $m \vDash \delta$, we choose a trace t and a loop maps w. Suppose $\langle m, a\rangle \to v_N$. There are two cases, when $v_N = 0$, the loop body is not executed so we can easily show that the trace is not modified.

  When the loop is still running ($v_N > 0$), we have the following low-level evaluation:

$$\frac{v_N > 0}{\langle m, [\mathtt{loop}\ v_N\ \mathtt{do}\ c]^l, t, w\rangle \to \langle m, c; [\mathtt{loop}\ (v_N - 1)\ \mathtt{do}\ c]^l, t, (w + l)\rangle} \textbf{ low-loop}$$

We then have the following evaluation:

$$\frac{\langle m, c, t, (w+l)\rangle \to^* \langle m', \texttt{skip}, t'_1, w'\rangle}{\langle m, c; [\texttt{loop}\ (v_N - 1)\ \texttt{do}\ c]^l, t, (w+l)\rangle \to^* \langle m', [\texttt{loop}\ (v_N - 1)\ \texttt{do}\ c]^l, t'_1, w'\rangle}$$

We can have the following evaluation of the ssa-form using the rule **ssa-loop**, when we assume , $m = \delta^{-1}(\mathbf{m})$.

$$\frac{\mathbf{v_N} > \mathbf{0} \qquad \mathbf{n} = 0 \implies i = 1 \qquad \mathbf{n} > 0 \implies i = 2}{\langle \mathbf{m}, [\texttt{loop}\ \mathbf{v_N}, \mathbf{n}, [\bar{\mathbf{x}}', \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}']^{\mathbf{l}}, \mathbf{t}, \mathbf{w}\rangle \to \langle \mathbf{m}, \mathbf{c}'[\bar{\mathbf{x_i}}/\bar{\mathbf{x}}']; [\texttt{loop}\ (\mathbf{v_N} - \mathbf{1}), \mathbf{n} + \mathbf{1}, [\bar{\mathbf{x}}', \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}', \mathbf{t}, (\mathbf{w} + \mathbf{l})\rangle} \textbf{ ssa-loop}$$

Depending on if the counter n is equal to 0 or not, there are two possible execution paths (the variables $\bar{\mathbf{x}}$ is replaced by the $\bar{\mathbf{x_1}}$ or $\bar{\mathbf{x_2}}$). We start from the first iteration (when $n = 0$) when $v_N > 0$.

By induction hypothsis on the premise $\Sigma; \delta; c \hookrightarrow \mathbf{c_1}; \delta_1; \Sigma_1$, we know that

$$\langle \mathbf{m}, \mathbf{c}'[\bar{\mathbf{x_1}}/\bar{\mathbf{x}}'], t, (w+l)\rangle \to^* \langle \mathbf{m}', \texttt{skip}, t'_i, w'\rangle \wedge m' = \delta_1^{-1}(\mathbf{m}')$$

Hence we can conclude that:

$$\frac{\langle \mathbf{m}, \mathbf{c}'[\bar{\mathbf{x_1}}/\bar{\mathbf{x}}'], t, (w+l)\rangle \to^* \langle \mathbf{m}', \texttt{skip}, t'_1, w'\rangle}{\begin{array}{c}\langle \mathbf{m}, \mathbf{c}'[\bar{\mathbf{x_1}}/\bar{\mathbf{x}}']; [\texttt{loop}\ (\mathbf{v_N} - \mathbf{1}), \mathbf{n} + \mathbf{1}, [\bar{\mathbf{x}}', \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}']^{\mathbf{l}}, t, (w+l)\rangle \to^* \\ \langle \mathbf{m}', [\texttt{loop}\ (\mathbf{v_N} - \mathbf{1}), \mathbf{n} + \mathbf{1}, [\bar{\mathbf{x}}', \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}']^{\mathbf{l}}, t'_1, w'\rangle\end{array}}$$

Then there are two cases,

1. when $v_N - 1 = 0$ so that the loop exits. The low-level execution as follows by the evaluation rule **low-loop-exit**.

   $$\frac{v_N - 1 = 0}{\langle m', [\texttt{loop}\ v_N - 1\ \texttt{do}\ c]^l, t'_i, w'\rangle \to \langle m', [\texttt{skip}]^l, t'_1, (w' \setminus l)\rangle} \textbf{ low-loop-exit}$$

   The corresponding ssa-form evaluation as follows:

   $$\frac{\mathbf{v_N} - \mathbf{1} = 0}{\langle \mathbf{m}', [\texttt{loop}\ \mathbf{v_N} - \mathbf{1}, \mathbf{n}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^{\mathbf{l}}, t'_1, w'\rangle \to \langle \mathbf{m}'[\bar{\mathbf{x}} \mapsto \mathbf{m}'(\bar{\mathbf{x_2}})], [\texttt{skip}]^{\mathbf{l}}, t'_1, (w' \setminus l)\rangle} \textbf{ ssa-loop-exit}$$

   We can see that both traces are not changed during the exit of the loop. We need to show that $m' = \delta^{-1}(\mathbf{m}'[\bar{\mathbf{x}} \mapsto \mathbf{m}'(\bar{\mathbf{x_2}})])$. We know that $[\bar{x} \mapsto \bar{x_2}]$ in δ_1 from the definition, so we can show that for any variable $\mathbf{x_2} \in \bar{x_2}$, $(\mathbf{x_2}, m'(x)) \in \mathbf{m}'$. For variables $x \in \texttt{dom}(\delta) - \bar{x}$, the variable is not modified during the execution of c so that we know $m(x) = m'(x)$, and then we can show that $(\delta(x), m'(x)) \in \mathbf{m}'$ because $\delta(x)$ is not written in $\mathbf{c}'[\bar{\mathbf{x_1}}/\bar{\mathbf{x}}']$.

2. when $v_n - 1 > 0$ so that the loop is still running.
   We want to show that : assuming in the $i-th$ ($i < v_N$) iteration, starting with t_i and w_i and $m_i = \delta_1^{-1}(\mathbf{m_i})$ so that the low level evaluation as follows $\langle m_i, [\texttt{loop}\ v_N - i\ \texttt{do}\ c]^l, t_i, w_i\rangle \to^* \langle m_{i+1}, [\texttt{loop}\ (v_N - i - 1)\ \texttt{do}\ c]^l, t_{i+1}, w_{i+1}\rangle$. Then the corresponding ssa form evaluation as follows :

   $$\begin{array}{c}\langle \mathbf{m_i}, [\texttt{loop}\ (\mathbf{v_N} - \mathbf{i}), \mathbf{n}, [\bar{\mathbf{x}}', \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}']^{\mathbf{l}}, t_i, (w_i + l)\rangle \to^* \\ \langle \mathbf{m_{i+1}}, [\texttt{loop}\ (\mathbf{v_N} - \mathbf{i} - \mathbf{1}), \mathbf{n} + \mathbf{1}, [\bar{\mathbf{x}}', \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}']^{\mathbf{l}}, t_{i+1}, w_{i+1}\rangle\end{array}$$

and $m_i = \delta^{-1}(\mathbf{m_i})$.

We then have the low level evauation:

$$\frac{v_N - i > 0}{\langle m_i, [\texttt{loop}\ v_N - i\ \texttt{do}\ c]^l, t_i, w_i\rangle \to \langle m_i, c; [\texttt{loop}\ (v_N - i - 1)\ \texttt{do}\ c]^l, t_i, (w_i + l)\rangle}\ \textbf{low-loop}$$

We then have the following evaluation:

$$\frac{\langle m_i, c, t_i, (w_i + l)\rangle \to^* \langle m_{i+1}, \texttt{skip}, t_{i+1}, w_{i+1}\rangle}{\langle m_i, c; [\texttt{loop}\ (v_N - i - 1)\ \texttt{do}\ c]^l, t_i, (w_i + l)\rangle \to^* \langle m_{i+1}, [\texttt{loop}\ (v_N - i - 1)\ \texttt{do}\ c]^l, t_{i+1}, w_{i+1}\rangle}$$

By induction hypothsis on the premise $\Sigma; \delta_1; c \hookrightarrow \mathbf{c_2}; \delta_1; \Sigma_1$, we know that

$$\langle \mathbf{m_i}, \mathbf{c}'[\bar{\mathbf{x}}_2/\bar{\mathbf{x}}'], t_i, (w_i + l)\rangle \to^* \langle \mathbf{m_{i+1}}, \texttt{skip}, t_{i+1}, w_{i+1}\rangle \land m_{i+1} = \delta_1^{-1}(\mathbf{m_{i+1}})$$

Hence we can conclude that:

$$\frac{\langle \mathbf{m_i}, \mathbf{c}'[\bar{\mathbf{x}}_2/\bar{\mathbf{x}}'], t_i, (w_i + l)\rangle \to^* \langle \mathbf{m_{i+1}}, \texttt{skip}, t_{i+1}, w_{i+1}\rangle}{\begin{array}{c}\langle \mathbf{m_i}, \mathbf{c}'[\bar{\mathbf{x}}_2/\bar{\mathbf{x}}']; [\texttt{loop}\ (\mathbf{v_N} - \mathbf{i} - \mathbf{1}), \mathbf{n} + \mathbf{1}, [\bar{\mathbf{x}}', \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2]\ \texttt{do}\ \mathbf{c}']^{\mathbf{l}}, t_i, (w_i + l)\rangle \to^* \\ \langle \mathbf{m_{i+1}}, [\texttt{loop}\ (\mathbf{v_N} - \mathbf{i} - \mathbf{1}), \mathbf{n} + \mathbf{1}, [\bar{\mathbf{x}}', \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2]\ \texttt{do}\ \mathbf{c}']^{\mathbf{l}}, t_{i+1}, w_{i+1}\rangle\end{array}}$$

So we can show that before the exit of the loop after $(v_N = n)$ iterations, we have $t_n = t_n$ and $m_n = \delta_1^{-1}(\mathbf{m_n})$. This proof is similar when it comes to the exit as in case 1.

$\square$

Definition 11 (Query may dependency in SSA). *One query* $q(v_2)$ may depend *on another query* $q(v_1)$ *in a program* $\mathbf{c}$*, with a starting loop maps* w*, denoted as* $\mathsf{DEP_{ssa}}(q(v_1)^{(l_1,w_1)}, q(v_2)^{(l_2,w_2)}, \mathbf{c}, w, \mathbf{m}, D)$. *is defined as below.*

$$\begin{array}{l}
\forall t. \exists \mathbf{m_1}, \mathbf{m_3}, t_1, t_3. \langle \mathbf{m}, \mathbf{c}, t, w\rangle \to^* \langle \mathbf{m_1}, [\mathbf{x} \leftarrow q(v_1)]^{l_1}; \mathbf{c_2}, t_1, w_1\rangle \to \\
\langle \mathbf{m_1}[q(v_1)(D)/\mathbf{x}], \mathbf{c_2}, t_1 + +[q(v_1)^{(l_1,w_1)}], w_1\rangle \to^* \langle \mathbf{m_3}, \texttt{skip}, t_3, w_3\rangle \\
\land \Big(q(v_1)^{(l_1,w_1)} \in (t_3 - t) \land q(v_2)^{(l_2,w_2)} \in (t_3 - t_1) \implies \exists v \in \mathtt{codom}(q(v_1)), \mathbf{m}_3', t_3', w_3'. \\
\langle \mathbf{m_1}[v/\mathbf{x}], \mathbf{c_2}, t_1 + +[(q(v_1)^{(l_1,w_1)})], w_1\rangle \to^* \langle \mathbf{m}_3', \texttt{skip}, t_3', w_3'\rangle \land (q(v_2)^{(l_2,w_2)}) \not\in (t_3' - t_1) \Big) \\
\land \Big(q(v_1)^{(l_1,w_1)} \in (t_3 - t) \land q(v_2)^{(l_2,w_2)} \not\in (t_3 - t_1) \implies \exists v \in \mathtt{codom}(q(v_1)), \mathbf{m}_3', t_3', w_3'. \\
\langle \mathbf{m_1}[v/\mathbf{x}], c_2, t_1 + +[(q(v_1)^{(l_1,w_1)})], w_1\rangle \to^* \langle \mathbf{m}_3', \texttt{skip}, t_3', w_3'\rangle \land (q(v_2)^{(l_2,w_2)}) \in (t_3' - t_1) \Big)
\end{array}$$

Definition 12. *Query may dependency in SSA.*

[JL: Updated Definition One query $q(v_{q_2})$ may depend *on another query* $q(v_{q_1})$ *in a program* $\mathbf{c}$*, with a starting loop maps* w*, a starting memory* $\mathbf{m}$*, hidden database* D*, denoted as*

$\mathsf{DEP}_{\mathsf{ssa}}(q(v_{q_1})^{(l_1,w_1)}, q(v_{q_2})^{(l_2,w_2)}, \mathbf{c}, w, \mathbf{m}, D).$

$$
\begin{array}{l}
\forall t.\exists \mathbf{m_1}, \mathbf{m_3}, t_1, t_3, c_2. \\
\left(\begin{array}{l}
\langle \mathbf{m}, c, t, w\rangle \to^* \langle \mathbf{m_1}, [\mathbf{x} \leftarrow q(v_{q_1})]^{l_1}; \mathbf{c_2}, t_1, w_1\rangle \to \\
\langle \mathbf{m_1}[q(v_{q_1})(D)/\mathbf{x}], \mathbf{c_2}, t_1 ++ [q(v_{q_1})^{(l_1,w_1)}], w_1\rangle \to^* \langle \mathbf{m_3}, \mathtt{skip}, t_3, w_3\rangle \\
\land \\
\Big(q(v_{q_1})^{(l_1,w_1)} \in_q (t_3 - t) \land q(v_{q_2})^{(l_2,w_2)} \in_q (t_3 - t_1) \\
\implies \exists v \in \mathtt{codom}(q(v_{q_1})), \mathbf{m_3'}, t_3', w_3'. \\
\langle \mathbf{m_1}[v/\mathbf{x}], \mathbf{c_2}, t_1 ++ [q(v_{q_1})^{(l_1,w_1)}], w_1\rangle \to^* \langle \mathbf{m_3'}, \mathtt{skip}, t_3', w_3'\rangle \\
\land (q(v_{q_2})^{(l_2,w_2)}) \not\in_q (t_3' - t_1)\Big) \\
\land \\
\Big(q(v_{q_1})^{(l_1,w_1)} \in_q (t_3 - t) \land q(v_{q_2})^{(l_2,w_2)} \not\in_q (t_3 - t_1) \\
\implies \exists v \in \mathtt{codom}(q(v_{q_1})), \mathbf{m_3'}, t_3', w_3'. \\
\langle \mathbf{m_1}[v/\mathbf{x}], \mathbf{c_2}, t_1 ++ [q(v_{q_1})^{(l_1,w_1)}], w_1\rangle \to^* \langle \mathbf{m_3'}, \mathtt{skip}, t_3', w_3'\rangle \\
\land (q(v_{q_2})^{(l_2,w_2)}) \in_q (t_3' - t_1)\Big)
\end{array}\right)
\end{array}
$$

]

Definition 13 (Dependency Graph in SSA)**.** .
Given a program $\mathbf{c}$*, a database* D*, a starting memory* $\mathbf{m}$*, an initial loop maps* w*, the dependency graph* $G_s(\mathbf{c}, D, \mathbf{m}, w) = (V, E)$ *is defined as:*
$V = \{q(v)^{l,w} \in \mathcal{AQ} \mid \forall t.\exists \mathbf{m}', w', t'.\langle \mathbf{m}, \mathbf{c}, t, w\rangle \to^* \langle \mathbf{m}', \mathtt{skip}, t', w'\rangle \land q(v)^{l,w} \in (t' - t)\}.$
$E = \left\{(q(v)^{(l,w)}, q(v')^{(l',w')}) \in \mathcal{AQ} \times \mathcal{AQ} \;\middle|\; \mathsf{DEP}_{\mathsf{ssa}}(q(v')^{(l',w')}, q(v)^{(l,w)}, \mathbf{c}, w, \mathbf{m}, D) \right\}.$

Definition 14. *Dependency Graph in SSA.*
[JL: Updated Definition
Given a program $\mathbf{c}$*, a database* D*, a starting memory* $\mathbf{m}$*, an initial loop maps* w*, the dependency graph* $G_s(\mathbf{c}, D, \mathbf{m}, w) = (V, E)$ *is defined as:*
$V = \{q(v_q)^{l,w} \in \mathcal{AQ} \mid \forall t.\exists \mathbf{m}', w', t'.\langle \mathbf{m}, \mathbf{c}, t, w\rangle \to^* \langle \mathbf{m}', \mathtt{skip}, t', w'\rangle \land q(v_q)^{l,w} \in (t' - t)\}.$
$E = \left\{(q(v_q)^{(l,w)}, q({v_q}')^{(l',w')}) \in \mathcal{AQ} \times \mathcal{AQ} \;\middle|\; \mathsf{DEP}_{\mathsf{ssa}}(q({v_q}')^{(l',w')}, q(v_q)^{(l,w)}, \mathbf{c}, w, \mathbf{m}, D) \right\}.$ *]*

Definition 15 (Adaptivity in SSA language)**.** *Given a program* $\mathbf{c}$*, and a meory* $\mathbf{m}$*, a database* D*, a starting loop maps* w*, the adaptivity of the dependency graph* $G_s(\mathbf{c}, D, \mathbf{m}, w) = (V, E)$ *is the length of the longest path in this graph. We denote the path from* $q(v)^{(l,w)}$ *to* $q(v')^{(l',w')}$ *as* $p(q(v)^{(l,w)}, q(v')^{(l',w')})$*. The adaptivity denoted as* $A_s(\mathbf{c}, D, \mathbf{m}, w)$*.*

$$A_s(\mathbf{c}, D, \mathbf{m}, w) = \max_{q(v)^{(l,w)}, q(v')^{(l',w')} \in V} \{|p(q(v)^{(l,w)}, q(v')^{(l',w')})|\}$$

Definition 16. *Adaptivity in SSA.*
[JL: Updated Definition Given a program $\mathbf{c}$*, and a meory* $\mathbf{m}$*, a database* D*, a starting loop maps* w*, the adaptivity of the dependency graph* $G_s(\mathbf{c}, D, \mathbf{m}, w) = (V, E)$ *is the length of the longest path in this graph. We denote the path from* $q(v_q)^{(l,w)}$ *to* $q({v_q}')^{(l',w')}$ *as* $p_s(q(v_q)^{(l,w)}, q({v_q}')^{(l',w')})$*. The adaptivity denoted as* $A_s(\mathbf{c}, D, \mathbf{m}, w)$*.*

$$A_s(\mathbf{c}, D, \mathbf{m}, w) = \max_{q(v_q)^{(l,w)}, q({v_q}')^{(l',w')} \in V} |p_s(q(v_q)^{(l,w)}, q({v_q}')^{(l',w')})|$$

]

Theorem 4.2 (Same graph)**.** *Given* $\Sigma;\delta;c \hookrightarrow \mathbf{c};\delta';\Sigma'$, $\forall m.m \vDash \delta.\forall \mathbf{m}, \mathbf{m} \vDash_{ssa} \delta \wedge m = \delta^{-1}(\mathbf{m})$, *starting with a trace* t *and loop maps* w*, then* $G_{low}(c, D, m, w) = G_{ls}(\mathbf{c}, D, \mathbf{m}, w)$.

Proof. The nodes in the two graphs are the same according to soundness of the transformation theorem. We will show that the edges are the same. Unfold the definition of DEP, when we find some v in the codomain of one query that may change the appearance of another query, we can always show that $v = \mathbf{v}$ works for $\mathsf{DEP_s}$ as well. In the other direction, for any $\mathbf{v}$ that may change the appearance of another query and lead to a different trace t_3' in the ssa execution, there is the same $v = \mathbf{v}$ so that the trace t_3' is generated in the loop language execution, according to the transformation theorem. □

Corollary 4.2.1. *Given* $\Sigma;\delta;c \hookrightarrow \mathbf{c};\delta';\Sigma'$, $\forall m.m \vDash \delta.\forall \mathbf{m}, \mathbf{m} \vDash_{ssa} \delta \wedge m = \delta^{-1}(\mathbf{m})$, *starting with a trace* t *and loop maps* w*, then* $A(c, D, m, w) = A_s(\mathbf{c}, D, \mathbf{m}, w)$.

4.2 ssa form operational semantics

the command $\mathsf{ifvar}(\bar{x}, \bar{x}')$ stores the variable map during the run time, which is a map from ssa variable $\mathbf{x}$ to variable $\mathbf{x_i}$. This map is designed for if condition, when the variable may comes from two branches and this command records which branch the variable comes from.

$$\boxed{\langle \mathbf{m}, \mathbf{a}\rangle \to \langle \mathbf{a}\rangle} \quad \frac{\mathbf{m}(\mathbf{x}) = \mathbf{v}}{\langle \mathbf{m}, \mathbf{x}\rangle \to \mathbf{v}}\ \textbf{SSA-VAR} \qquad \boxed{\langle \mathbf{m}, \mathbf{c}, \mathbf{t}, \mathbf{w}\rangle \to \langle \mathbf{m}', \mathbf{c}', \mathbf{t}', \mathbf{w}'\rangle}$$

$$Memory \times Com \times Trace \times loopmaps \Rightarrow Memory \times Com \times Trace \times loopmaps$$

$$\frac{\mathbf{q}(\mathbf{v})(\mathbf{D}) = \mathbf{v_q}}{\langle \mathbf{m}, [\mathbf{x} \leftarrow \mathbf{q}(\mathbf{v})]^{\mathbf{l}}, \mathbf{t}, \mathbf{w}\rangle \to \langle \mathbf{m}[\mathbf{x} \mapsto \mathbf{v_q}], \mathtt{skip}, \mathbf{t} +\!+ [\mathbf{q}(\mathbf{v})^{(\mathbf{l},\mathbf{w})}], \mathbf{w}\rangle}\ \textbf{ssa-query}$$

$$\frac{}{\langle \mathbf{m}, [\mathbf{x} \leftarrow \mathbf{v}]^{\mathbf{l}}, \mathbf{t}, \mathbf{w}\rangle \to \langle \mathbf{m}[\mathbf{x} \mapsto \mathbf{v}], [\mathtt{skip}]^{\mathbf{l}}, \mathbf{t}, \mathbf{w}\rangle}\ \textbf{ssa-assn}$$

$$\frac{\langle \mathbf{m}, \mathbf{b}\rangle \to_b \mathbf{b}'}{\langle \mathbf{m}, [\mathtt{if}(\mathbf{b}, [\bar{\mathbf{x}}, \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2], [\bar{\mathbf{y}}, \bar{\mathbf{y}}_1, \bar{\mathbf{y}}_2], [\bar{\mathbf{z}}, \bar{\mathbf{z}}_1, \bar{\mathbf{z}}_2], \mathbf{c}_1, \mathbf{c}_2)]^{\mathbf{l}}, \mathbf{t}, \mathbf{w}\rangle \to \langle \mathbf{m}, [\mathtt{if}(\mathrm{b}', [\bar{\mathbf{x}}, \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2], [\bar{\mathbf{y}}, \bar{\mathbf{y}}_1, \bar{\mathbf{y}}_2], [\bar{\mathbf{z}}, \bar{\mathbf{z}}_1, \bar{\mathbf{z}}_2], \mathbf{c}_1, \mathbf{c}_2)]^{\mathbf{l}}, \mathbf{t}, \mathbf{w}\rangle}\ \textbf{ssa-if}$$

$$\frac{}{\langle \mathbf{m}, [\mathtt{if}(\mathtt{true}, [\bar{\mathbf{x}}, \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2], [\bar{\mathbf{y}}, \bar{\mathbf{y}}_1, \bar{\mathbf{y}}_2], [\bar{\mathbf{z}}, \bar{\mathbf{z}}_1, \bar{\mathbf{z}}_2], \mathbf{c}_1, \mathbf{c}_2)]^{\mathbf{l}}, \mathbf{t}, \mathbf{w}\rangle \to \langle m, c_1;\ ifvar(\bar{\mathbf{x}}, \bar{\mathbf{x}}_1); ifvar(\bar{\mathbf{y}}, \bar{\mathbf{y}}_2); ifvar(\bar{\mathbf{z}}, \bar{\mathbf{z}}_1), t, w\rangle}\ \textbf{ssa-i}$$

$$\frac{}{\langle \mathbf{m}, [\mathtt{if}(\mathtt{false}, [\bar{\mathbf{x}}, \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2], [\bar{\mathbf{y}}, \bar{\mathbf{y}}_1, \bar{\mathbf{y}}_2], [\bar{\mathbf{z}}, \bar{\mathbf{z}}_1, \bar{\mathbf{z}}_2], \mathbf{c}_1, \mathbf{c}_2)]^{\mathbf{l}}, \mathbf{t}, \mathbf{w}\rangle \to \langle m, c_2; ifvar(\bar{\mathbf{x}}, \bar{\mathbf{x}}_2); ifvar(\bar{\mathbf{y}}, \bar{\mathbf{y}}_1); ifvar(\bar{\mathbf{z}}, \bar{\mathbf{z}}_2), t, w\rangle}\ \textbf{ssa-}$$

$$\frac{}{\langle \mathbf{m}, ifvar(\bar{\mathbf{x}}, \bar{\mathbf{x}}'), \mathbf{t}, \mathbf{w}\rangle \to \langle \mathbf{m}[\bar{\mathbf{x}} \to \mathbf{m}(\bar{\mathbf{x}}')], \mathtt{skip}, \mathbf{t}, \mathbf{w}\rangle}\ \textbf{ssa-ifvar}$$

$$[[\frac{\mathbf{v_N} > \mathbf{0} \qquad \mathbf{n} = 0 \implies i = 1 \qquad \mathbf{n} > 0 \implies i = 2}{\langle \mathbf{m}, [\mathtt{loop}\ \mathbf{v_N}, \mathbf{n}, [\bar{\mathbf{x}}, \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2]\ \mathtt{do}\ \mathbf{c}]^{\mathbf{l}}, \mathbf{t}, \mathbf{w}\rangle \to \langle \mathbf{m}, \mathbf{c}[\bar{\mathbf{x}}_\mathbf{i}/\bar{\mathbf{x}}]; [\mathtt{loop}\ (\mathbf{v_N} - \mathbf{1}), \mathbf{n} + \mathbf{1}, [\bar{\mathbf{x}}, \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2]\ \mathtt{do}\ \mathbf{c}]^{\mathbf{l}}, \mathbf{t}, (\mathbf{w} + \mathbf{l})\rangle}\ \textbf{ssa-loop}]]$$

$$\frac{v_N = 0 \qquad \mathbf{n} = 0 \implies i = 1 \qquad \mathbf{n} > 0 \implies i = 2}{\langle \mathbf{m}, [\mathtt{loop}\ \mathbf{v_N}, \mathbf{n}, [\bar{\mathbf{x}}, \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2]\ \mathtt{do}\ \mathbf{c}]^{\mathbf{l}}, \mathbf{t}, \mathbf{w}\rangle \to \langle \mathbf{m}[\bar{\mathbf{x}} \mapsto \mathbf{m}(\bar{\mathbf{x}}_\mathbf{i})], [\mathtt{skip}]^{\mathbf{l}}, \mathbf{t}, (\mathbf{w} \setminus \mathbf{l})\rangle}\ \textbf{ssa-loop-exit}$$

$$[WQ: \frac{n = 0 \to i = 1 \qquad n > 0 \to i = 2}{\begin{array}{c}\langle \mathbf{m}, \mathtt{while}\ [\mathbf{b}]^{\mathbf{l}}, \mathbf{n}, [\bar{\mathbf{x}}', \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2]\ \mathtt{do}\ \mathbf{c}, t, w\rangle \to \\ \langle \mathbf{m}, \mathtt{if}_w(\mathbf{b}[\bar{\mathbf{x}}_\mathbf{i}/\bar{\mathbf{x}}'], [\bar{\mathbf{x}}', \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2], \mathbf{n}, \mathbf{c}[\bar{\mathbf{x}}_\mathbf{i}/\bar{\mathbf{x}}']; \mathtt{while}\ [\mathbf{b}]^{\mathbf{l}}, \mathbf{n} + \mathbf{1}, [\bar{\mathbf{x}}', \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2]\ \mathtt{do}\ \mathbf{c}, \mathtt{skip}), t, (w + l)\rangle\end{array}}\ \textbf{ssa-while-b}]$$

$$[WQ: \frac{\mathbf{m}, \mathbf{b} \to \mathbf{b}'}{\langle \mathbf{m}, \mathtt{if}_\mathbf{w}(\mathbf{b}, [\bar{\mathbf{x}}', \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2], \mathbf{n}, \mathbf{c}_1, \mathbf{c}_2), t, w\rangle \to \langle \mathbf{m}, \mathtt{if}_\mathbf{w}(\mathbf{b}', [\bar{\mathbf{x}}', \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2], \mathbf{n}, \mathbf{c}_1, \mathbf{c}_2), t, w\rangle}\ \textbf{ssa-ifw-b}]$$

$$[WQ: \frac{}{\langle \mathbf{m}, \mathtt{if}_\mathbf{w}(\mathtt{true}, [\bar{\mathbf{x}}', \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2], \mathbf{n}, \mathbf{c}_1, \mathbf{c}_2), t, w\rangle \to \langle \mathbf{m}, \mathbf{c}_1\, t, (w + l)\rangle}\ \textbf{ssa-ifw-true}]$$

$$[WQ: \frac{n = 0 \to i = 1 \qquad n > 0 \to i = 2}{\langle \mathbf{m}, \mathtt{if}_\mathbf{w}(\mathtt{false}, [\bar{\mathbf{x}}', \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2], \mathbf{n}, \mathbf{c}_1, \mathbf{c}_2), t, w\rangle \to \langle \mathbf{m}, \mathbf{c}_2; \mathbf{ifvar}(\bar{\mathbf{x}}', \bar{\mathbf{x}}_\mathbf{i}), t, (w \setminus l)\rangle}\ \textbf{ssa-ifw-false}]$$

5 AdaptFun

5.1 Analysis Rules/Algorithms of AdaptFun

There are two steps for our algorithm to get the estimation of the adaptivity of a program $\mathbf{c}$ in the ssa form.

1. Estimate the variables that are new generated (via assignment) and store these variables in a global list G. We have the algorithm of the form : $\mathsf{VetxEst}(G; w; \mathbf{c}) \to (G'; w')$.
2. We start to track the dependence between variables in a matrix M, whose size is $|G| \times |G|$, and track whether arbitrary variable is assigned with a query result in a vector V with size $|G|$. The algorithm to fill in the matrix is of the form: $\mathsf{GraphGen}(\Gamma; \mathbf{c}; i_1) \to (M; V; i_2)$. Γ is a vector records the variables the current program $\mathbf{c}$ depends on, the index i_1 is a pointer which refers to the position of the first new-generated variable in $\mathbf{c}$ in the global list G, and i_2 points to the first new variable that is not in $\mathbf{c}$ (if exists).

We give an example of M and V of the program c.

$$c = \begin{array}{c} \mathbf{x_1} \leftarrow \mathbf{q}; \\ \mathbf{x_2} \leftarrow \mathbf{x_1} + \mathbf{1}; \\ \mathbf{x_3} \leftarrow \mathbf{x_2} + \mathbf{2} \end{array} \qquad M = \left[\begin{array}{cccc} & (x_1) & (x_2) & (x_3) \\ (x_1) & 0 & 0 & 0 \\ (x_2) & 1 & 0 & 0 \\ (x_3) & 1 & 1 & 0 \end{array}\right], V = \left[\begin{array}{cc} (x_1) & 1 \\ (x_2) & 0 \\ (x_3) & 0 \end{array}\right]$$

Still use the program c as the example, the global list G is now : $[x_1, x_2, x_3]$. The function Left and Right is used to generate the corresponding vector of the left side and right side of an assignment. Take $x_2 \leftarrow x_1 + 1$ as an example, the result is shown as follows.

$$\mathsf{L}(1) = \left[\begin{array}{cc} 0 & (x_1) \\ 1 & (x_2) \\ 0 & (x_3) \end{array}\right] \qquad \mathsf{R}(\mathsf{x_1} + 1, 1) = \left[\begin{array}{ccc} 1 & 0 & 0 \\ (x_1) & (x_2) & (x_3) \end{array}\right]$$

Now let us think about the loop.

$$[JL : \mathbf{c_3} \triangleq \begin{array}{l} [\mathbf{x_1} \leftarrow \mathbf{q_1}]^1; \\ [\mathbf{i_1} \leftarrow 0]^2; \\ \mathtt{while}\ [\mathbf{i_1} < 2]^3 \\ \quad [\mathbf{x_3}, \mathbf{x_1}, \mathbf{x_2}], [\mathbf{i_3}, \mathbf{i_1}, \mathbf{i_2}]\ \mathtt{do} \\ \Big([\mathbf{y_1} \leftarrow \mathbf{q_2}]^4; \\ [\mathbf{x_2} \leftarrow \mathbf{y_1} + \mathbf{x_3}]^5; \\ [\mathbf{i_2} \leftarrow \mathbf{1} + \mathbf{i_3}]^6\Big); \\ [z_1 \leftarrow x_3 + 2]^7 \end{array}, \quad M = \left[\begin{array}{ccccccc} & (x_1) & (x_3) & (y_1) & (x_2) & (x_3^f) & (z_1) \\ (x_1) & 0 & 0 & 0 & 0 & 0 & 0 \\ (x_3) & 1 & 0 & 0 & 0 & 0 & 0 \\ (y_1) & 0 & 0 & 0 & 0 & 0 & 0 \\ (x_2) & 0 & 1 & 1 & 0 & 0 & 0 \\ (x_3) & 1 & 0 & 0 & 1 & 0 & 0 \\ (z_1) & 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right], \quad V = \left[\begin{array}{cc} (x_1) & 1 \\ (x_3) & 0 \\ (y_1) & 1 \\ (x_2) & 0 \\ (x_3) & 0 \\ (z_1) & 0 \end{array}\right]]$$

$$\mathbf{c_3} \triangleq \begin{array}{l} [\mathbf{x_1} \leftarrow \mathbf{q_1}]^1; \\ \mathtt{loop}\ [2]^2, 0, \\ [\mathbf{x_3}, \mathbf{x_1}, \mathbf{x_2}]\ \mathtt{do} \\ \Big([\mathbf{y_1} \leftarrow \mathbf{q_2}]^4; \\ [\mathbf{x_2} \leftarrow \mathbf{y_1} + \mathbf{x_3}]^5; \\ [\mathbf{i_2} \leftarrow \mathbf{1} + \mathbf{i_3}]^6\Big); \\ [z_1 \leftarrow x_3 + 2]^7 \end{array} \quad M = \left[\begin{array}{c|ccccccccc} & (x_1) & (x_3^1) & (y_1^1) & (x_3^1) & (x_3^2) & (y_1^2) & (x_2^2) & (x_3^f) & (z_1) \\ (x_1) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (x_3^1) & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (y_1^1) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (x_2^1) & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ (x_3^2) & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ (y_1^2) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (x_2^2) & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ (x_3^f) & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ (z_1) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right], V = \left[\begin{array}{cc} (x_1) & 1 \\ (x_3^1) & 0 \\ (y_1^1) & 1 \\ (x_2^1) & 0 \\ (x_3^2) & 0 \\ (y_1^2) & 1 \\ (x_2^2) & 0 \\ (x_3^f) & 0 \\ (z_1) & 0 \end{array}\right]$$

$$[JL : \mathbf{c_3'} \triangleq \begin{array}{l} [\mathbf{x_1} \leftarrow \mathbf{q_1}]^1; \\ [\mathbf{i_1} \leftarrow 1]^2; \\ \mathtt{while}\ [i < 0]^3, \\ [\mathbf{x_3}, \mathbf{x_1}, \mathbf{x_2}]\ \mathtt{do} \\ \Big([\mathbf{y_1} \leftarrow \mathbf{q_2}]^3; \\ [\mathbf{x_2} \leftarrow \mathbf{y_1} + \mathbf{x_3}]^5\Big); \\ [z_1 \leftarrow x_3 + 2]^6 \end{array} , \quad M = \left[\begin{array}{c|ccccc} & (x_1) & i_1 & (x_3^f) & (i_3^f) & (z_1) \\ (x_1) & 0 & 0 & 0 & 0 & 0 \\ (i_1) & 0 & 0 & 0 & 0 & 0 \\ (x_3^f) & 1 & 0 & 0 & 0 & 0 \\ (i_3^f) & 0 & 1 & 0 & 0 & 0 \\ (z_1^2) & 0 & 0 & 1 & 0 & 0 \end{array}\right], \quad V = \left[\begin{array}{cc} (x_1) & 1 \\ (i_1) & 0 \\ (x_3^f) & 0 \\ (i_3^f) & 0 \\ (z_1) & 0 \end{array}\right]]$$

$$\mathbf{c_3'} \triangleq \begin{array}{l} [\mathbf{x_1} \leftarrow \mathbf{q_1}]^1; \\ \mathtt{loop}\ [0]^2, 0, \\ [\mathbf{x_3}, \mathbf{x_1}, \mathbf{x_2}]\ \mathtt{do} \\ \Big([\mathbf{y_1} \leftarrow \mathbf{q_2}]^3; \\ [\mathbf{x_2} \leftarrow \mathbf{y_1} + \mathbf{x_3}]^5\Big); \\ [z_1 \leftarrow x_3 + 2]^6 \end{array} \quad M = \left[\begin{array}{c|ccc} & (x_1) & (x_3^f) & (z_1) \\ (x_1) & 0 & 0 & 0 \\ (x_3^f) & 1 & 0 & 0 \\ (z_1^2) & 0 & 1 & 0 \end{array}\right], V = \left[\begin{array}{cc} (x_1) & 1 \\ (x_3^f) & 0 \\ (z_1) & 0 \end{array}\right]$$

We can now look at the if statement.

$$c_4 \triangleq \begin{array}{l} [x \leftarrow q_1]^1; \\ [y \leftarrow q_2]^2; \\ \mathtt{if}\ (x + y == 5)^3 \\ \mathtt{then}\ [x \leftarrow q_3]^4 \\ \mathtt{else}(\ [x \leftarrow q_4]^5; \\ y \leftarrow 2); \\ [z \leftarrow x + y]^6; \end{array} \quad \hookrightarrow \quad \mathbf{c_4} \triangleq \begin{array}{l} [\mathbf{x_1} \leftarrow \mathbf{q_1}]^1; \\ [\mathbf{y_1} \leftarrow \mathbf{q_2}]^2; \\ \mathtt{if}\ (\mathbf{x_1} + \mathbf{y_1} == \mathbf{5})^3, [x_4, x_2, x_3], [], [y_3, y_1, y_2] \\ \mathtt{then}\ [\mathbf{x_2} \leftarrow \mathbf{q_3}]^4 \\ \mathtt{else}(\ [\mathbf{x_3} \leftarrow \mathbf{q_4}]^5; \\ \mathbf{y_2} \leftarrow \mathbf{2}) \\ [\mathbf{z_1} \leftarrow \mathbf{x_4} + \mathbf{y_3}]^6; \end{array}$$

$$M_{c4} = \left[\begin{array}{c|cccccccc} & (x_1) & (y_1) & (x_2) & (x_3) & (y_2) & (x_4) & (y_3) & (z_1) \\ (x_1) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (y_1) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (x_2) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (x_3) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (y_2) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (x_4) & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ (y_3) & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ (z_1) & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}\right], V_{c4} = \left[\begin{array}{cc} (x_1) & 1 \\ (y_1) & 1 \\ (x_2) & 1 \\ (x_3) & 1 \\ (y_2) & 0 \\ (x_4) & 0 \\ (y_3) & 0 \\ (z_1) & 0 \end{array}\right]$$

5.2 The algorithm to estimate the matrix and vector

We first generate a list of variables G that will be assigned with values (via the command $x \leftarrow e$ or $x \leftarrow q$).

$$\frac{}{\mathsf{VetxEst}(G; w; [\mathbf{x} \leftarrow \mathbf{e}]^{\mathbf{l}}) \rightarrow (G++[\mathbf{x}^{(l,w)}]; w)}\ \textbf{ag-asgn}$$

$$\frac{}{\mathsf{VetxEst}(G; w; [\mathbf{x} \leftarrow q(\mathbf{e})]^{l}) \rightarrow (G++[\mathbf{x}^{(l,w)}]; w)}\ \textbf{ag-query}$$

$$\frac{\mathsf{VetxEst}(G; w; \mathbf{c_1}) \rightarrow (G_1; w_1) \qquad \mathsf{VetxEst}(G_1; w; \mathbf{c_2}) \rightarrow (G_2; w_2) \\ G_3 = G_2 ++[\bar{\mathbf{x}}^{(\mathbf{l,w})}] ++[\bar{\mathbf{y}}^{(\mathbf{l,w})}] ++[\bar{\mathbf{z}}^{(\mathbf{l,w})}]}{\mathsf{VetxEst}(G; w; [\mathtt{if}(\mathbf{b}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2})]^{l}) \rightarrow (G_3; w)}\ \textbf{ag-if}$$

$$\frac{\mathsf{VetxEst}(G; w; \mathbf{c_1}) \rightarrow (G_1; w_1) \qquad \mathsf{VetxEst}(G_1; w_1; \mathbf{c_2}) \rightarrow (G_2; w_2)}{\mathsf{VetxEst}(G; w; (\mathbf{c_1}; \mathbf{c_2})) \rightarrow (G_2; w_2)}\ \textbf{ag-seq}$$

$$[[\ \frac{G_0 = G \quad w_0 = w \qquad \forall 0 \leq z < N. \mathsf{VetxEst}(G_z ++[\bar{\mathbf{x}}^{(\mathbf{l,w_z+l})}]; (w_z + l); \mathbf{c}) \rightarrow (G_{z+1}; w_{z+1}) \\ G_f = G_N ++[\bar{\mathbf{x}}^{(\mathbf{l,w_N \setminus l})}] \qquad \mathbf{a} = N}{\mathsf{VetxEst}(G; w; [\mathtt{loop}\ \mathbf{a}, n, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \mathtt{do}\ \mathbf{c}]^{l}) \rightarrow (G_f; w_N \setminus l)}\ \textbf{ag-loop}]]$$

$$[JL: \frac{\mathsf{VetxEst}(G++[\bar{\mathbf{x}}^{(\mathbf{l,w+l})}]; (w + l); \mathbf{c}) \rightarrow (G'; w') \qquad G_f = G' ++[\bar{\mathbf{x}}^{(\mathbf{l,w' \setminus l})}]}{\mathsf{VetxEst}(G; w; \mathtt{while}\ [b]^{l}\ \mathbf{n}\ [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \mathtt{do}\ c) \rightarrow (G_f; w \setminus l)}\ \textbf{ag-while}]$$

Analysis Logic Rules. Γ is a matrix of one row and N columns, $N = |G| = |V|$.

$\mathsf{L(i)}$ generates a matrix of one column, N rows, where the $i - th$ row is 1, all the other rows are 0.

$\mathsf{R(e,i)}$ generates a matrix of one row and N columns, where the locations of variables in e is marked as 1. To handle loop, for instance, the variable y appears many times in G, the argument i helps to find the location of the current living variable y in the expression e, which is the latest y with the largest location $i_y < i$ in our global variable list G.

$$\forall 0 \leq z < |\bar{x}|. \bar{x}(z) = x_z, \bar{x_1}(z) = x_{1z}, \bar{x_2}(z) = x_{2z}$$

$$\Gamma \vdash^{i, i+|\bar{\mathbf{x}}|}_{M, v_{\emptyset}} [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \triangleq \forall 0 \leq z < |\bar{x}|. \Gamma \vdash^{i+z, i+z+1}_{M_{x_z}, V_{\emptyset}} x_z \leftarrow x_{1z} + x_{2z} \text{ where } M = \sum_{z \in [|\bar{x}|]} M_{x_z}$$

$\boxed{\Gamma \vdash^{i_1,i_2}_{M,V} c}$

$$\frac{M = \mathsf{L}(i) * (\mathsf{R}(\mathbf{e}, i) + \Gamma)}{\mathsf{GraphGen}(\Gamma; [\mathbf{x} \leftarrow \mathbf{e}]^{l}; i) \to (M; V_{\emptyset}; i+1)}\ \textbf{ad-asgn}$$

$$\frac{M = \mathsf{L}(i) * (\mathsf{R}(\mathbf{e}, i) + \Gamma) \qquad V = \mathsf{L}(i)}{\mathsf{GraphGen}(\Gamma; [\mathbf{x} \leftarrow q(\mathbf{e})]^{l}; i) \to (M; V; i+1)}\ \textbf{ad-query}$$

$$\frac{\begin{array}{c}\mathsf{GraphGen}(\Gamma + \mathsf{R}(\mathbf{b}, i_1); \mathbf{c_1}; i_1) \to (M_1; V_1; i_2) \qquad \mathsf{GraphGen}(\Gamma + \mathsf{R}(\mathbf{b}, i_1); \mathbf{c_2}; i_2) \to (M_2; V_2; i_3) \\ \mathsf{GraphGen}(\Gamma; [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]; i_3) \to (M_x; V_{\emptyset}; i_3 + |\bar{\mathbf{x}}|) \qquad \mathsf{GraphGen}(\Gamma; [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}]; i_3 + |\bar{\mathbf{x}}|) \to (M_y; V_{\emptyset}; i_3 + |\bar{\mathbf{x}}| + |\bar{\mathbf{y}}|) \\ \mathsf{GraphGen}(\Gamma; [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}]; i_3 + |\bar{\mathbf{x}}| + |\bar{\mathbf{y}}|) \to (M_y; V_{\emptyset}; i_3 + |\bar{\mathbf{x}}| + |\bar{\mathbf{y}}| + |\bar{\mathbf{z}}|) \\ M = (M_1 + M_2) + M_x + M_y + M_z\end{array}}{\mathsf{GraphGen}(\Gamma; \mathtt{if}([\mathbf{b}]^{l}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2}); i_1) \to (M; V_1 \uplus V_2; i_3 + |\bar{x}| + |\bar{y}| + |\bar{z}|)}\ \textbf{ad-if}$$

$$\frac{\mathsf{GraphGen}(\Gamma; \mathbf{c_1}; i_1) \to (M_1; V_1; i_2) \qquad \mathsf{GraphGen}(\Gamma; \mathbf{c_2}; i_2) \to (M_2; V_2; i_3)}{\mathsf{GraphGen}(\Gamma; (\mathbf{c_1}; \mathbf{c_2}); i_1) \to ((M_1;M_2); V_1 \uplus V_2; i_3)}\ \textbf{ad-seq}$$

$$[[\frac{\begin{array}{c} B = |\bar{\mathbf{x}}| \qquad A = |\mathbf{c}| \\ \forall 0 \leq j < N. \mathsf{GraphGen}(\Gamma; [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]; i + j * (B + A)) \to (M_{1j}; V_{1j}; i + B + j * (B + A)) \\ \mathsf{GraphGen}(\Gamma; \mathbf{c}; i + B + j * (B + A)) \to (M_{2j}; V_{2j}; i + B + A + j * (B + A)) \\ \mathsf{GraphGen}(\Gamma; [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]; i + N * (B + A)) \to (M; V; i + N * (B + A) + B) \\ \mathbf{a} = N \qquad M' = M + \sum_{0 \leq j < N} (M_{1j} + M_{2j}) \qquad V' = V \uplus \sum_{0 \leq j < N} (V_{1j} \uplus V_{2j}) \end{array}}{\mathsf{GraphGen}(\Gamma; \mathtt{loop}\ [\mathbf{a}]^{l},\ 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \mathtt{do}\ \mathbf{c}, i) \to (M'; V'; i + N * (B + A) + B)}\ \textbf{ad-loop}]]$$

$$[JL: \frac{\begin{array}{c} B = |\bar{\mathbf{x}}| \qquad A = |\mathbf{c}| \qquad \mathsf{GraphGen}(\Gamma; [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]; i + (B + A)) \to (M_1; V_1; i + B + (B + A)) \\ \mathsf{GraphGen}(\Gamma; \mathbf{c}; i + B + (B + A)) \to (M_2; V_2; i + B + A + (B + A)) \\ \mathsf{GraphGen}(\Gamma; [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]; i + (B + A)) \to (M; V; i + (B + A) + B) \\ M' = M + \sum_{0 \leq j < N} (M_1 + M_2) \qquad V' = V \uplus \sum_{0 \leq j < N} (V_1 \uplus V_2) \end{array}}{\mathsf{GraphGen}(\Gamma; \mathtt{while}\ [b]^{l}\ \mathbf{n}\ [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \mathtt{do}\ c; i) \to (M'; V'; i + (B + A) + B)}\ \textbf{ad-while}]$$

$$\begin{array}{rcl} M_1; M_2 & := & M_2 \cdot M_1 + M_1 + M_2 \\ V_1 \uplus V_2 & := & \begin{cases} 1 & (V_1[i] = 1 \lor V_2[i] = 1) \land i = 1, \cdots, N \land |V_1| = |V_2| \\ 0 & o.w. \end{cases} \end{array}$$

Definition 17 (Valid matrix)**.** *For a global list G, $G \vDash (M, V)$ iff the cardinality of G equals to the one of V, $|G| = |V|$ and the matrix M is of size $|V| \times |V|$.*

Definition 18 (Valid index)**.** *For a global list G, a loop maps w, $G; w \vDash (\mathbf{c}, i_1, i_2)$ iff $G' = G[0, \ldots, i_1 - 1], G'; w; \mathbf{c} \to G''; w' \land G'' = G[0, \ldots, i_2 - 1]$.*

Definition 19 (Valid gamma)**.** $\Gamma \vDash i_1$ *iff* $\forall i \geq i_1, \Gamma(i_1) = 0$.

Definition 20 (Approximated SSA-Based Graph)**.** ***[[Given a program $\mathbf{c}$, the global list G, s.t. $\Gamma \vdash^{i_1,i_2}_{M,V} \mathbf{c}$, the Graph $G_{ssa}(M, V, G, i_1, i_2) = (V_{ssa}, E_{ssa}, V)$ is. defined as:***

$$
\begin{array}{lcl}
|[\mathtt{switch}\,(\mathbf{e},\mathbf{x},(v_j \to q_j)]^l|_{low} & = & [\mathtt{switch}\,(|\mathbf{e}|_{low},|x|_{low},(v_j \to q_j)]^l \\
|\mathtt{while}\ [\mathbf{b}]^l, n, [\bar{\mathbf{x}},\bar{\mathbf{x_1}},\bar{\mathbf{x_2}}]\ \mathtt{do}\ c|_{low} & = & [JL\!:\mathtt{while}\ [|\mathbf{b}|_{low}]^l,\ \mathtt{do}\ |\mathbf{c}|_{low}] \\
|\mathbf{c_1};\mathbf{c_2}|_{low} & = & |\mathbf{c_1}|_{low};|\mathbf{c_2}|_{low} \\
|[\mathtt{if}(\mathrm{b},[\bar{\mathbf{x}},\bar{\mathbf{x_1}},\bar{\mathbf{x_2}}],[\bar{\mathbf{y}},\bar{\mathbf{y_1}},\bar{\mathbf{y_2}}],[\bar{\mathbf{z}},\bar{\mathbf{z_1}},\bar{\mathbf{z_2}}],\mathbf{c_1},\mathbf{c_2})]^l|_{low} & = & [\mathtt{if}(|\mathrm{b}|_{low},|\mathbf{c_1}|_{low},|\mathbf{c_2}|_{low})]^l \\
|[\mathbf{x} \leftarrow \mathbf{e}]^l|_{low} & = & [|\mathbf{x}|_{low} \leftarrow |\mathbf{e}|_{low}]^l \\
|[\mathbf{x} \leftarrow q]^l|_{low} & = & [|\mathbf{x}|_{low} \leftarrow q]^l \\
|x_i|_{low} & = & x \\
|n|_{low} & = & n \\
|\mathbf{a_1} \oplus_a \mathbf{a_2}|_{low} & = & |\mathbf{a_1}|_{low} \oplus_a |\mathbf{a_2}|_{low} \\
|\mathbf{b_1} \oplus_b \mathbf{b_2}|_{low} & = & |\mathbf{b_1}|_{low} \oplus_b |\mathbf{b_2}|_{low}
\end{array}
$$

Figure 2: The erasure of SSA

Nodes $V_{ssa} = \{G(j) \in \mathcal{LV} \mid i_1 \leq j < i_2\}$
Edges $E_{ssa} = \{(G(j_1), G(j_2)) \in \mathcal{LV} \times \mathcal{LV} \mid M[j_1][j_2] \geq 1 \land i_1 \leq j_1, j_2 < i_2\}$
$V = V$ *]]*

5.3 Weighting Algorithm

[JL: We use the algorithm from paper [2] to estimate the reachability bound for each node on the graph.]

Definition 21 (Approximated SSA-Based Weighted Graph)**.** .
[JL: Given a program **c** *and its Approximated SSA-Based Graph* $G_{ssa} = \{V_{ssa}, E_{ssa}, V\}$*, we added weights on every vertice in this graph according to the reachibility bound algorithm* RechBound*. Then we will have an updated graph Approximated SSA-Based Weighted Graph* $G_{aprox} = \{V_{aprox}, E_{aprox}, W_{aprox}\}$ *defined as:]*
$V_{aprox} = V_{ssa}$
$E_{aprox} = E_{ssa}$
$W_{aprox} = \{(v, w) | \forall\ v \in V_{aprox} s.t. V(v) > 0, w = \mathsf{RechBound}(v, c)\} \cup \{(v, \infty) | \forall\ v \in V_{aprox}\ s.t.\ V(v) = 0\}$

Definition 22 (Adaptivity of piece of program)**.** . *[JL: Given a program* **c***, and its approximated SSA-Based Weighted Graph* $G_{aprox} = \{V_{aprox}, E_{aprox}, W_{aprox}\}$ *Adaptivity of the program defined according to the logic is as:]*

$$[JL\!: Adapt(c) := \max_{v_1, v_2 \in \{v | v \in V_{aprox} \land W_{aprox}(v) < \infty\}} \{\mathsf{AdaptPathSearch}(G_{aprox}, v_1, v_2)\}]$$

[JL: The algorithm of computing the adaptivity is dfined in the algorithm 1.]

5.4 [[Soundness of the **AdaptFun**]]

Definition 23 (Subgraph)**.** *Given two graphs* $G_{low} = (V_1, E_1)$*,* $G_{ssa} = (V_2, E_2)$*,* $G_{low} \subseteq G_{ssa}$ *iff:*
$\exists f, g$ *so that*
1. for every $v \in V_1$*,* $f(v) \in V_2$*.*
2. $\forall e = (v_i, v_j) \in E_1$*, there exists a path* $g(e)$ *from* $f(v_i)$ *to* $f(v_2)$ *in* G_2*.*

Algorithm 1 [JL: Aptivity Path Search Algorithm (AdaptPathSearch)]

```
Require: Weighted Directed Graph $G = (V, E, W)$ with a start vertex $s$ and destination vertex $t$ .
  $dfs(c, \textbf{curr\_min\_flow}, \textbf{nested\_max\_flows})$:
      if $c == t$:
          return curr_min_flow $* \prod_{r \in \text{nested\_max\_flows}} \{r\}$.
      for .
      let $a_j = \mathcal{M}(q_j)$
      {In the line above, $\mathcal{M}$ computes approx. the exp. value of $q_j$ over $X$. So, $a_j \in [-1, +1]$.}
  define $q_{k+1}(x) = \text{sign}\left(\sum_{i \in [k]} x(i) \times \ln \frac{1+a_i}{1-a_i}\right)$ where $x \in \{-1, +1\}^{k+1}$.
  {In the line above, $\text{sign}(y) = \begin{cases} +1 & \text{if } y \geq 0 \\ -1 & \text{otherwise} \end{cases}$ .}
  let $a_{k+1} = \mathcal{M}(q_{k+1})$
  {In the line above, $\mathcal{M}$ computes approx. the exp. value of $q_{k+1}$ over $X$. So, $a_{k+1} \in [-1, +1]$.}
  return $a_{k+1}$.
Ensure: $a_{k+1} \in [-1, +1]$
```

Definition 24 (Mapping of vertices from G_s to G_{ssa} graph)**.** *Let us define a map $f : \mathcal{AQ} -> \mathcal{LV}$ as follow: $f(q^{l,w}) = \mathbf{x}^{l,w}$*

Lemma 4. *$\forall t, m, w, c, \langle m, c, t, w \rangle \to^* \langle m', \texttt{skip}, t', w' \rangle$, every node in $t' - t$ has unique label.*

Proof. By induction on the evaluation of program c.

case: $\langle m, [\texttt{loop}\ v_N\ \texttt{do}\ c]^l, t, w \rangle \to^* \langle m_N, \texttt{skip}, t_N, w_N \rangle$
We want to show every node in $t_N - t$ has unique label. Suppose $v_N = N$,so we have N iteration. When $N = 0$, there is no new generated trace, it is trivially true. When $N > 0$, we have the following evaluation.

$$[[\frac{v_N > 0}{\langle m, [\texttt{loop}\ v_N\ \texttt{do}\ c]^l, t, w \rangle \to \langle m, c; [\texttt{loop}\ (v_N - 1)\ \texttt{do}\ c]^l, t, (w + l) \rangle}\ \textbf{low-loop}]]$$

$$\frac{\langle m, c, t, (w + l) \rangle \to^* \langle m_1, \texttt{skip}, t_1, w_1 \rangle}{\langle m, c; [\texttt{loop}\ (v_N - 1)\ \texttt{do}\ c]^l, t, (w + l) \rangle \to^* \langle m_1, [\texttt{loop}\ (v_N - 1)\ \texttt{do}\ c]^l, t_1, w_1 \rangle}$$

By induction hypothesis on $\langle m, c, t, (w + l) \rangle \to^* \langle m_1, \texttt{skip}, t_1, w_1 \rangle$, we know that for every node in $t_1 - t$ has unique label.

Unfold the loop, for the $2nd$, $3rd$, until the $N - th$ iteration, we can also conclude that every node in $t_2 - t_1$, $t_3 - t_2$, until $t_N - t_{N-1}$ has unique label.

Because $w \neq w + l \neq w_1 + l \neq w_2 + l \neq \ldots, \neq w_{N-1} + l$, we know that node from different iteration $(i \neq j)$ $t_i - t_{i-1}$ and $t_j - t_{j-1}$ has different loop maps.

case: $\langle m, c_1; c_2, t, w \rangle \to^* \langle m'', \texttt{skip}, t'', w'' \rangle$

$$\frac{\langle m, c_1, t, w \rangle \to^* \langle m', \texttt{skip}, t', w' \rangle \qquad \langle m', c_2, t', w' \rangle \to^* \langle m'', \texttt{skip}, t'', w'' \rangle}{\langle m, c_1; c_2, t, w \rangle \to^* \langle m', \texttt{skip}, t', w' \rangle}$$

By induction on premises, we know that : every node in $t' - t$ and $t'' - t'$ has unique label. Also, all the program lines $[l_1, l_2, \ldots, l_n]$ in c_1 is smaller than the minimal program line in c_2, so every node in $t' - t$ has different line number with node in $t'' - t'$. □

Lemma 5 (Label consistency in low level graph). *Given a low level program c, for a database D, a memory m, a loop maps w, every node $q^{(l,w)}$ in the low level graph $(V_1, E_1) = G_{low}(c, D, m, w)$ has unique label (l, w).*

Proof. From the definition of $G_{low}(c, D, m, w)$, the vertices comes from the trace generated in the low level evaluation $\langle m, c, [], w\rangle \to^* \langle m', \texttt{skip}, t, w'\rangle$. This can be proved using Lemma 2. □

Lemma 6. *Given $\Gamma \vdash^{(i_1,i_2)}_{M,V} \mathbf{c}$, for any global list G and a loop maps w, such that $G; w \vDash (\mathbf{c}, i_1, i_2) \wedge G \vDash (M, V)$. Every node $\mathbf{x}^{(l,w)} \in \mathcal{LV}$ in G has unique label.*

Lemma 7. *Given $\Gamma \vdash^{(i_1,i_2)}_{M,V} \mathbf{c}$, for any global list G and a loop maps w, such that $G; w \vDash (\mathbf{c}, i_1, i_2) \wedge G \vDash (M, V)$. Assume $G' = G[0, \dots, i_1 - 1]$, so that $G'; w; \mathbf{c} \hookrightarrow G''; w'$. For any database D and memory m and trace t, $\langle m, |\mathbf{c}|_{low}, t, w\rangle \to^* \langle m', \texttt{skip}, t', w''\rangle$. Then $w' = w''$.*

Lemma 8 ($f_{D,m}$ is well defined). *Given $\Gamma \vdash^{(i_1,i_2)}_{M,V} \mathbf{c}$, for any global list G and a loop maps w, such that $G; w \vDash (\mathbf{c}, i_1, i_2) \wedge G \vDash (M, V)$ and $\Gamma \vdash i_1$. For any database D and memory $\mathbf{m}$, the function $f_{D,m} : \mathcal{AQ} \to \mathcal{LV}$ maps all the nodes in the low level graph $(V_1, E_1) = G_s(\mathbf{c}, D, \mathbf{m}, w)$ to a node in the ssa graph $(V_2, E_2) = G_{ssa}(M, V, G, i_1, i_2)$, defined as :*
$f_{D,m}(q^{l,w}) = \mathbf{x}^{(l,w)}$ where $q^{(l,w)} \in V_1$ and $\mathbf{x}^{(l,w)} \in V_2$ is injective.

Proof. To show $f_{D,m}$ is injective.

We want to show that:

for every node $q^{(l,w)}$ in V_1(in the new generated trace by executing c), there exist only one corresponding labelled variable $\mathbf{x}^{(l,w)} \in \mathcal{LV}$ in V_2(also know as $G[i_1, \dots, (i_2 - 1)]$) with the same label (l, w).

By induction on $\Gamma \vdash^{(i_1,i_2)}_{M,V} \mathbf{c}$.

Case: $\dfrac{M = \mathsf{L}(i) * \Gamma \qquad V = \mathsf{L}(i)}{\Gamma \vdash^{(i,i+1)}_{M,V} \mathbf{x} \leftarrow q}$ **query**

We assume $G; w \vDash ([\mathbf{x} \leftarrow q]^l, i, i+1)$ and $G \vDash (M, V)$. We have the low level program $|[\mathbf{x} \leftarrow q]^l|_{low} = [x \leftarrow q]^l$. We choose the database D and memory m, a starting trace t, from the definition of $(V_1, E_1) = G_{low}([x \leftarrow q]^l, D, m, w)$, we know v_1 comes from the trace $t' - t$ generated in the following ssa evaluation.

$$\frac{q(v)(D) = v_q}{\langle \mathbf{m}, [\mathbf{x} \leftarrow \mathbf{q(v)}]^{\mathbf{l}}\rangle, t, w \to^* \langle \mathbf{m}[\mathbf{v_q}/\mathbf{x}], \texttt{skip}, t + +[q(v)^{(l,w)}], w\rangle}$$

We unfold $G; w \vDash ([\mathbf{x} \leftarrow q(e)]^l, i, i+1)$ so we know that $G' = G[0, \dots, (i-1)] \wedge G'; w; [\mathbf{x} \leftarrow q(e)]^l \to G' + +[\mathbf{x}^{(l,w)}]; w$

Considering every node in V_2 has unique label. we know that there is a unique labelled variable $\mathbf{x}^{(l,w)}$ in V_2 with the same label as the one $q^{(l,w)}$ in V_1.

Case: $\dfrac{\Gamma \vdash^{(i_1,i_2)}_{M_1,V_1} \mathbf{c_1} \qquad \Gamma \vdash^{(i_2,i_3)}_{M_2,V_2} \mathbf{c_2}}{\Gamma \vdash^{(i_1,i_3)}_{M_1;M_2,V_1 \uplus V_2} \mathbf{c_1}; \mathbf{c_2}}$ **seq**

We assume $G; w \vDash (\mathbf{c_1}; \mathbf{c_2}, i_1, i_3)$ and denote $c_1 = |\mathbf{c_1}|_{low}$ and $c_2 = |\mathbf{c_2}|_{low}$. We choose the memory m, trace t and hidden database D, we have the ssa evaluation:

$$\frac{\langle m, c_1, t, w\rangle \to^* \langle m_1, \texttt{skip}, t_1, w_1\rangle \qquad \langle m_1, c_2, t_1, w_1\rangle \to^* \langle m_2, \texttt{skip}, t_2, w_2\rangle}{\langle m, c_1; c_2, t, w\rangle \to \langle m_2, \texttt{skip}, t_2, w_2\rangle}$$

From our assumption $G; w \vDash (\mathbf{c_1}; \mathbf{c_2}, i_1, i_3)$, we denote $G' = G[0, \ldots, (i_1 - 1)]$. We have the following :

$$\frac{G'; w; \mathbf{c_1} \to G_1; w_1' \qquad G_1; w_1'; \mathbf{c_2} \to G_2; w_2'}{G'; w; \mathbf{c_1}; \mathbf{c_2} \to G_2; w_2'}$$

We know that $\mathbf{c_1}, \mathbf{c_2}$ are the subterm of $\mathbf{c_1}; \mathbf{c_2}$ so there exist i_2 such that $G; w \vDash (\mathbf{c_1}, i_1, i_2)$ and $G; w_1' \vDash (\mathbf{c_2}, i_2, i_3)$. By Lemma 7, we conclude that $w_1 = w_1'$.

By induction hypothesis on the first premise, we know : every node $q^{(l,w)} \in \mathcal{AQ}$ in $t_1 - t$ has a unique mapping $\mathbf{x}^{(l,w)}$ with the same label in $G[i_1, \ldots, (i_2 - 1)]$.

By induction hypothesis on the second premise, we conclude that : every node $q^{(l,w)} \in \mathcal{AQ}$ in $t_2 - t_1$ has a unique mapping $\mathbf{x}^{(l,w)}$ with the same label in $G[i_2, \ldots, (i_3 - 1)]$.

Now we can conclude that : every node $q^{(l,w)} \in \mathcal{AQ}$ in $t_2 - t$ has a unique mapping $\mathbf{x}^{(l,w)}$ with the same label in $G[i_1, \ldots, (i_3 - 1)]$

Case:
$$\frac{\begin{array}{c} B = |\bar{\mathbf{x}}| \qquad \Gamma \vdash^{(i, i+B)}_{M_{10}, V_{10}} \qquad \Gamma \vdash^{(i+B, i+B+A)}_{M_{20}, V_{20}} \mathbf{c} \\ \forall 1 \leq j < N. \Gamma \vdash^{(i+j*(B+A), i+B+j*(B+A))}_{M_{1j}, V_{1j}} [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \qquad \Gamma \vdash^{(i+B+j*(B+A), i+B+A+j*(B+A))}_{M_{2j}, V_{2j}} \mathbf{c} \\ \Gamma \vdash^{(i+N*(B+A), i+N*(B+A)+B)}_{M, V} [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \qquad \mathbf{a} = \lceil N \rceil \\ M' = M + \sum_{0 \leq j < N} M_{1j} + M_{2j} \qquad V' = V \uplus \sum_{0 \leq j < N} V_{1j} \uplus V_{2j} \end{array}}{\Gamma \vdash^{(i, i+N*(B+A)+B)}_{M', V'} [\texttt{loop}\ \mathbf{a}, n, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^l} \textbf{loop}$$

We assume $G; w \vDash ([\texttt{loop}\ \mathbf{a}, n, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^l, i, i + N * (B + A) + B)$ and denote $c = |\mathbf{c}|_{low}$ and $a = |\mathbf{a}|_{low}$. We choose the memory m, trace t and hidden database D, we have the ssa evaluation supposing $\langle m, a \rangle \to v_N$:

$$\frac{v_N > 0}{\langle m, [\texttt{loop}\ v_N\ \texttt{do}\ c]^l, t, w \rangle \to \langle m, c; [\texttt{loop}\ (v_N - 1)\ \texttt{do}\ c]^l, t, (w + l) \rangle} \textbf{low-loop}$$

$$\frac{\langle m, c, t, (w + l) \rangle \to^* \langle m_1, \texttt{skip}, t_1, w_1 \rangle}{\langle m, c; [\texttt{loop}\ (v_N - 1)\ \texttt{do}\ c]^l, t, (w + l) \rangle \to^* \langle m_1, [\texttt{loop}\ (v_N - 1)\ \texttt{do}\ c]^l, t_1, w_1 \rangle}$$

We assume $G' = G[0, \ldots, i - 1]$, from our assumption, we have:

$$\frac{G_0 = G' \quad w_0 = w \qquad \forall 0 \leq z < N. G_z + + [\bar{\mathbf{x}}^{(\mathbf{l}, \mathbf{w_z} + \mathbf{l})}]; (w_z + l); \mathbf{c} \to G_{z+1}; w_{z+1} \qquad G_f = G_N + + [\bar{\mathbf{x}}^{(\mathbf{l}, \mathbf{w_N} \setminus \mathbf{l})}] \qquad \mathbf{e} = \lceil \mathbf{N} \rceil}{G'; w; [\texttt{loop}\ (e), n, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^l \to G_f; w_N \setminus l} \textbf{loo}$$

By induction hypothesis on $\Gamma \vdash^{(i+B, i+B+A)}_{M_{20}, V_{20}} \mathbf{c}$ using the conclusion $\langle m, c, t, (w+l) \rangle \to^* \langle m_1, \texttt{skip}, t_1, w_1 \rangle$ and $G_z + + [\bar{\mathbf{x}}^{(\mathbf{l}, \mathbf{w_z} + \mathbf{l})}]; (w_z + l); \mathbf{c} \to G_{z+1}; w_{z+1}$ where $z = 0$. We can conclude the following about the first iteration.

Every node $q^{(l,w)}$ in $t_1 - t$, there is a unique labelled variable $\mathbf{x}^{(l,w)}$ with the same label in. $G[i + B, \ldots, i + B + A - 1]$.

For other iteration $1 \leq j < N$, we can show similarly by induction hypothesis on $\Gamma \vdash^{(i+B+j*(B+A), i+B+A+j*(B+A))}_{M_{2j}, V_{2j}} \mathbf{c}$.

$\square$

Theorem 5.1 (Soundness of AdaptFun). *Given* $\Gamma \vdash^{(i_1, i_2)}_{M, V} \mathbf{c}$, *for any global list G, loop maps w such that* $G; w \vDash (\mathbf{c}, i_1, i_2) \land G \vDash (M, V)$, $\Gamma \vdash i_1$. *K is the number of queries inquired during the execution of the piece of program* $\mathbf{c}$ *and* $|V|$ *gives the number of non-zeros in V. Then,*

$$K \leq |V| \land \forall D, \mathbf{m}. G_s(\mathbf{c}, D, \mathbf{m}, w) \subseteq G_{ssa}(M, V, G, i_1, i_2)$$

Proof. By induction on the judgment $\Gamma \vdash^{i_1,i_2}_{M,V} \mathbf{c}$.

- **Case:** $\dfrac{M = \mathsf{L}(i) * (\mathsf{R}(e,i) + \Gamma)}{\Gamma \vdash^{(i,i+1)}_{M,V_\emptyset} \mathbf{x} \leftarrow \mathbf{e}}$ **asgn**

  Given a memory m, database D, trace t, loop maps w, then from the following operational semantics:

  $$\frac{}{\langle \mathbf{m}, [\mathbf{x} \leftarrow \mathbf{e}]^l, t, w\rangle \to^* \langle \mathbf{m}, [\mathbf{x} \leftarrow v]^l, t, w\rangle \to \langle \mathbf{m}[\mathbf{x} \mapsto \mathbf{v}], [\mathtt{skip}]^l, t, w\rangle}$$

  We need to show:

  1. $G_s(\mathbf{x} \leftarrow \mathbf{e}, D, m, w) \subseteq G_{ssa}(M, V_\emptyset, i, i+1)$.
  2. $K \leq |V_\emptyset|$.

  The first goal is shown because the new generated trace t' is empty, based on the definition of G_s, we know that G_s has no vertices.
  The second goal is proved because $K = 0$.

- **Case:** $\dfrac{M = \mathsf{L}(i) * (\Gamma + \mathsf{R}(e,i)) \qquad V = \mathsf{L}(i)}{\Gamma \vdash^{(i,i+1)}_{M,V} \mathbf{x} \leftarrow \mathbf{q}(\mathbf{e})}$ **query**

  Given a memory m, database D, trace t, loop maps w, then from the following operational semantics:

  $$\frac{q(v)(D) = v_q}{\langle \mathbf{m}, [\mathbf{x} \leftarrow \mathbf{q}(\mathbf{e})]^l, t, w\rangle \to \langle \mathbf{m}[\mathbf{v_q}/\mathbf{x}], \mathtt{skip}, t +\!+ [(q(v)^{(l,w)})], w\rangle}$$

  We need to show:

  1. $G_s([\mathbf{x} \leftarrow \mathbf{q}(\mathbf{e})]^l, D, m, w) \subseteq G_{ssa}(M, V, i, i+1)$.
  2. $K \leq |V|$.

  The first goal is shown as follows: because the new generated trace $t' = [q(v)^{(l,w)}]$ only has one element. By the definition of $G_s = (V_s, E_s)$, we know there is only one vertex vx in V and no edge in E, i.e., $V_s = \{q(v)^{(l,w)}\}$ and $E_s = \{\}$.
  From Lemma 8,we know that there exists a function f so that $f(vx)$ (equivalent to $q(v)^{(l,w)}$ in G_s) exists in the vertices V_{ssa} of $G_{ssa}(M, v, i, i+1) = (V_{ssa}, E_{ssa})$. Since $V_s \subseteq V_{ssa}$ and $E_s \subseteq E_{ssa}$, it is proved that $G_s([\mathbf{x} \leftarrow \mathbf{q}(\mathbf{e})]^l, D, m, w) \subseteq G_{ssa}(M, V, i, i+1)$. So we have the first goal proved.

  The second goal is proved because $V = \mathsf{L}(i)$ is not empty so that $|V| \geq 1$, and $K = 1$.

- **Case:** $$\frac{\begin{array}{c} \Gamma + \mathsf{R}(b, i_1) \vdash^{(i_1,i_2)}_{M_1,V_1} \mathbf{c_1} \qquad \Gamma + \mathsf{R}(b, i_1) \vdash^{(i_2,i_3)}_{M_2,V_2} \mathbf{c_2} \\ \forall 0 \leq j < |\bar{x}|. \bar{x}(j) = x_j, \bar{x_1}(j) = x_{1j}, \bar{x_2}(j) = x_{2j} \\ \forall 0 \leq j < |\bar{x}|. \Gamma \vdash^{i_3+j, i_3+j+1}_{M_{x_j}, V_\emptyset} x_j \leftarrow x_{1j} + x_{2j} \qquad \forall 0 \leq j < |\bar{y}|. \Gamma \vdash^{i_3+|\bar{x}|+j, i_3+|\bar{x}|+j+1}_{M_{y_j}, V_\emptyset} y_j \leftarrow y_{1j} + y_{2j} \\ \forall 0 \leq j < |\bar{z}|. \Gamma \vdash^{i_3+|\bar{x}|+|\bar{y}|+j, i_3+|\bar{x}|+|\bar{y}|+j+1}_{M_{z_j}, V_\emptyset} z_j \leftarrow z_{1j} + z_{2j} \\ M = (M_1 + M_2) + \sum_{j \in [|\bar{x}|]} M_{x_j} + \sum_{j \in [|\bar{y}|]} M_{y_j} + \sum_{j \in [|\bar{z}|]} M_{z_j} \end{array}}{\Gamma \vdash^{(i_1, i_3+|\bar{x}|+|\bar{y}|+|\bar{z}|)}_{M, V_1 \uplus V_2} [\mathtt{if}(\mathbf{b}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2})]^l}\ \textbf{if}$$

  Given a memory $\mathbf{m}$, database D, trace t, loop maps w and a global list G. We assume that

$G \vDash M, V_1 \uplus V_2$, $G; w \vDash (\mathtt{if}(\mathrm{b}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2}), i_1, i_3 + |\bar{x}| + |\bar{y}| + |\bar{z}|)$. There are two possible executions. We choose one branch and the other will be similar.

From the assumption, denote $G' = G[0, \ldots, (i_1 - 1)]$ we know that

$$\frac{G'; w; \mathbf{c_1} \to G_1; w_1 \qquad G_1; w; \mathbf{c_2} \to G_2; w_2 \qquad G_3 = G_2 + +[\bar{\mathbf{x}}^{(\mathbf{l},\mathbf{w})}] + +[\bar{\mathbf{y}}^{(\mathbf{l},\mathbf{w})}] + +[\bar{\mathbf{z}}^{(\mathbf{l},\mathbf{w})}]}{G'; w; \mathtt{if}(\mathrm{b}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2})]^{l} \to G_3; w} \textbf{ if}$$

$\mathbf{c_1}$ is the sub term of $\mathtt{if}(\mathrm{b}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2})$. so we have: $G \vDash M_1, V_1$ and $G; w \vDash (\mathbf{c_1}, i_1, i_2)$, and $\Gamma + \mathsf{R}(b, i_1) \vDash i_1$.

By induction hypothesis on the first premise $\Gamma + \mathsf{R}(b, i_1) \vdash^{(i_1,i_2)}_{M,V} \mathbf{c_1}$, we can conclude that :

$$K_{\mathbf{c_1}} \le |V_1| \wedge G_s(\mathbf{c_1}, D, m, w) \subseteq G_{ssa}(M_1, V_1, i_1, i_2).$$

By the definition of K, we have $K_{\mathtt{if}(\mathrm{b}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2})} = K_{\mathbf{c_1}}$ when $b = \mathtt{true}$.

Next, we want to show :

$$G_s(\mathtt{if}(\mathrm{b}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2}), D, \mathbf{m}, w) \subseteq G_{ssa}(M, V_1 \uplus V_2, i_1, i_3 + |\bar{x}| + |\bar{y}| + |\bar{z}).$$

By the definition of G_s, we know that $G_s(\mathtt{if}(\mathrm{b}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2}), D, \mathbf{m}, w)$ is the same graph as $G_s(\mathbf{c_1}, D, \mathbf{m}, w)$. We can also show that $G_{ssa}(M_1, V_1, G, i_1, i_2)$ is a subgraph of $G_{ssa}(M, V_1 \uplus V_2, i_1, i_3 + |\bar{x}| + |\bar{y}| + |\bar{z})$ because all the vertices in the former also appears in the later($[i_1, i_2]$ is a smaller range of $[i_1, i_3 + i_3 + |\bar{x}| + |\bar{y}| + |\bar{z}|]$) and same for the edges(M contains M_1).

- **Case:** $\dfrac{\Gamma \vdash^{(i_1,i_2)}_{M_1,V_1} \mathbf{c_1} \qquad \Gamma \vdash^{(i_2,i_3)}_{M_2,V_2} \mathbf{c_2}}{\Gamma \vdash^{(i_1,i_3)}_{M_1;M_2,V_1 \uplus V_2} \mathbf{c_1};\mathbf{c_2}}$ **seq**

  Given a memory $\mathbf{m}$, database D, trace t, loop maps w ,a global list G. We assume $G; w \vDash (\mathbf{c_1};\mathbf{c_2}, i_1, i_3)$, $G \vDash M_1; M_2, V_1 \uplus V_2$. From the following ssa operational semantics:

$$\frac{\langle \mathbf{m}, \mathbf{c_1}, t, w \rangle \to^{*} \langle \mathbf{m_1}, \mathtt{skip}, t_1, w_1 \rangle \qquad \langle \mathbf{m_1}, \mathbf{c_2}, t_1, w_1 \rangle \to^{*} \langle \mathbf{m_2}, \mathtt{skip}, t_2, w_2 \rangle}{\langle \mathbf{m}, \mathbf{c_1}; \mathbf{c_2}, t, w \rangle \to \langle \mathbf{m_2}, \mathtt{skip}, t_2, w_2 \rangle}$$

  We need to show:

  1. $G_s(\mathbf{c_1};\mathbf{c_2}, D, \mathbf{m}, w) \subseteq G_{ssa}(M_1; M_2, V_1 \uplus V_2, i_1, i_3)$.
  2. $K_{\mathbf{c_1};\mathbf{c_2}} \le |V_1 \uplus V_2|$.

  From our assumption, we denote $G' = G[0, \ldots, (i_1 - 1)]$. We have the following :

$$\frac{G'; w; \mathbf{c_1} \to G_1; w_1' \qquad G_1; w_1'; \mathbf{c_2} \to G_2; w_2'}{G'; w; \mathbf{c_1}; \mathbf{c_2} \to G_2; w_2'}$$

  Because $\mathbf{c_1}, \mathbf{c_2}$ are subterms of $\mathbf{c_1};\mathbf{c_2}$, we know that there exists an index $i_1 \le i_2 \le i_3$ so that $G, w \vDash (\mathbf{c_1}, i_1, i_2)$ and $G, w_1' \vDash (\mathbf{c_2}, i_2, i_3)$.

By induction hypothesis, we have:

$$K_{\mathbf{c_1}} \leq |V_1| \wedge G_s(\mathbf{c_1}, D, \mathbf{m}, w) \subseteq G_{ssa}(M_1, V_1, G, i_1, i_2)$$

$$K_{\mathbf{c_2}} \leq |V_2| \wedge G_s(\mathbf{c_2}, D, \mathbf{m_1}, w_1') \subseteq G_{ssa}(M_2, V_2, G, i_2, i_3)$$

We can easily show that $K_{\mathbf{c_1};\mathbf{c_2}} \leq |V_1 \uplus V_2|$ by the definition of $K_{\mathbf{c_1};\mathbf{c_2}} = K_{\mathbf{c_1}} + K_{\mathbf{c_2}}$, and $|V_1 \uplus V_2| = |V_1| + |V_2|$ because the non-zeros range in $V_1([i_1, i_2))$ is disjoint with that in $V_2([i_2, i_3))$.

By the definition of G_s, we know that vertices of $(VX_1, E_1) = G_s(\mathbf{c_1}, D, \mathbf{m}, w)$ come from the trace $t_1 - t$ and $(VX_2, E_2) = G_{low}(c_2)$ from $t_2 - t_1$.

Unfold the definition of subgraph $\subseteq$, we need to show two cases.

1. The vertices in $G_s(c_1; c_2)$ can be mapped to the vertices in $G_{ssa}(M_1; M_2, V_1 \uplus V_2, i_1, i_3)$ by $f_{D,m}$.
   By induction hypothesis, we know that there exist functions f_{D,m,c_1} for $G_s(\mathbf{c_1}, D, \mathbf{m}, w)$ and f_{D,m_1,c_2} for $G_s(\mathbf{c_2}, D, \mathbf{m}, w_1')$. Then, we construct the mapping function

$$f_{D,m,\mathbf{c_1};\mathbf{c_2}}(vx) = \begin{cases} f_{D,m,c_1}(vx) & (vx) \in (t_1 - t) \\ f_{D,m_1,c_2}(vx) & (vx) \in (t_2 - t_1) \end{cases},$$

   which maps the vertices in $G_s(c_1; c_2)$ to the vertices in $G_{ssa}(M_1; M_2, V_1 \uplus V_2, i_1, i_3)$.
2. Every edge in $G_s(\mathbf{c_1}; \mathbf{c_2}, D, \mathbf{m}, w)$ can be mapped to a corresponding path in $G_{ssa}(M_1; M_2, G, V_1 \uplus V_2, i_1, i_3)$. The edges in $G_s(\mathbf{c_1}; \mathbf{c_2}, D, \mathbf{m}, w)$ comes from three sources:
   (a) Edges in $G_s(\mathbf{c_1}, D, \mathbf{m}, w)$. We can find a path in $G_{ssa}(M_1; M_2, V_1 \uplus V_2, i_1, i_3)$ for edge of this kind because we can find one path in a smaller graph $G_{ssa}(M_1 V_1, i_1, i_2)$
   (b) Edges in $G_s(\mathbf{c_2}, D, \mathbf{m_1}, w_1')$. This can be proved in a similar way as in previous case.
   (c) Edges (vx_1, vx_2) where $vx_1 = q_1(v_1)^{(l_1, w_1)}$ comes from $G_s(\mathbf{c_1}, D, \mathbf{m}, w)$ and $vx_2 = q_2^{(l_2, w_2)}$ comes from $G_s(\mathbf{c_2}, D, \mathbf{m_1}, w_1')$. We unfold the definition of one edge so that we have:

$$\mathsf{DEP}(vx_1, vx_2, \mathbf{c_1}; \mathbf{c_2}, w, \mathbf{m}) \wedge \mathsf{To}(vx_1, vx_2)$$

   Since we know that $q_1(v_1)^{(l_1, w_1)}$ comes from $t_1 - t$, so we can assume that $\mathbf{c_1} = \mathbf{c_1''}; \mathbf{c_1'}$. The last command of c_1'' is $[\mathbf{x} \leftarrow \mathbf{q_1}(\mathbf{e})]^{l_1}$. We have the following ssa evaluation:

$$\frac{\langle \mathbf{m}, \mathbf{c_1}, t, w \rangle \to^* \langle \mathbf{m_1'}, [\mathbf{x} \leftarrow \mathbf{q_1}(\mathbf{e})]^{l_1}; \mathbf{c_1'}, t_1', w_1' \rangle \to \langle \mathbf{m_1}, \mathtt{skip}, t_1, w_1 \rangle \quad \langle \mathbf{m_1}, \mathbf{c_2}, t_1, w_1 \rangle \to^* \langle \mathbf{m_2}, \mathtt{skip}, t_2, w_2 \rangle}{\langle \mathbf{m}, \mathbf{c_1}; \mathbf{c_2}, t, w \rangle \to \langle \mathbf{m_2}, \mathtt{skip}, t_2, w_2 \rangle}$$

   Then unfold the DEP:

$$\begin{array}{l}
\forall \mathbf{m_1}, \mathbf{m_3}, D, t, t_1, t_3. \langle \mathbf{m}, \mathbf{c}, t, w \rangle \to^* \langle \mathbf{m_1}, [\mathbf{x} \leftarrow \mathbf{q_1}(\mathbf{v_1})]^{\mathbf{l_1}}; \mathbf{c_2}, t_1, w_1 \rangle \to \\
\langle \mathbf{m_1}[\mathbf{v_q}/\mathbf{x}], \mathbf{c_2}, t_1 + +[q_1(v_1)^{(l_1, w_1)}], w_1 \rangle \to^* \langle \mathbf{m_3}, \mathtt{skip}, t_3, w_3 \rangle \\
\wedge \Big(q_1(v_1)^{(l_1, w_1)} \in (t_3 - t) \wedge q_2(v_2)^{(l_2, w_2)} \in (t_3 - t_1) \implies \exists v \in \mathtt{codom}(\mathbf{q_1}(\mathbf{v_1})(D)). \\
\langle \mathbf{m_1}[\mathbf{v}/\mathbf{x}], \mathbf{c_2}, t_1 + +[q_1(v_1)^{(l_1, w_1)}], w_1 \rangle \to^* \langle \mathbf{m_3'}, \mathtt{skip}, t_3', w_3' \rangle \wedge (q_2(v_2)^{(l_2, w_2)}) \not\in (t_3' - t_1) \Big) \\
\wedge \Big(q_1(v_1)^{(l_1, w_1)} \in (t_3 - t) \wedge q_2(v_2)^{(l_2, w_2)} \not\in (t_3 - t_1) \implies \exists v \in \mathtt{codom}(\mathbf{q_1}(\mathbf{v_1})(D)). \\
\langle \mathbf{m_1}[\mathbf{v}/\mathbf{x}], \mathbf{c_2}, t_1 + +[q_1(v_1)^{(l_1, w_1)}], w_1 \rangle \to^* \langle \mathbf{m_3'}, \mathtt{skip}, t_3', w_3' \rangle \wedge q_2(v_2)^{(l_2, w_2)} \in (t_3' - t_1) \Big)
\end{array}$$

We already know that $q_1(v_1)^{(l_1,w_1)} \in (t_1 - t), q_2(v_2)^{(l_2,w_2)} \in (t_2 - t_1)$. So from DEP, we know that: $\exists v \in \texttt{codom}(q_1(v_1)(D)).\langle \mathbf{m_1'}[\mathbf{v}/\mathbf{x}], \mathbf{c_1'}; \mathbf{c_2}, t_1' ++ [(q_1(v_1)^{(l_1,w_1)})], w_1' \rangle \to^* \langle \mathbf{m_2'}, \texttt{skip}, t_2', w_2' \rangle \wedge (q_2(v_2)^{(l_2,w_2)}) \not\in (t_2' - t)$.

We have:

$$\frac{\langle \mathbf{m}, \mathbf{c_1}; \mathbf{c_2}, t, w \rangle \to^* \langle \mathbf{m_1'}, [\mathbf{x} \leftarrow \mathbf{q_1}(\mathbf{v_1})]^{l_1}; \mathbf{c_1'}; \mathbf{c_2}, t_1', w_1' \rangle \to^* \langle \mathbf{m_1}, \texttt{skip}, t_1, w_1 \rangle \quad \langle \mathbf{m_1}, \mathbf{c_2}, t_1, w_1 \rangle \to^* \langle \mathbf{m_2}, \texttt{skip}, t_2, w_2 \rangle}{\langle \mathbf{m}, \mathbf{c_1}; \mathbf{c_2}, t, w \rangle \to^* \langle \mathbf{m_2}, \texttt{skip}, t_2, w_2 \rangle}$$

We know there exists a value v of $q_1(D)$, assigned to variable $\mathbf{x}^{(l_1,w_1')}$, such that $q_2(v_2)^{(l_2,w_2)} \not\in t_2 - t_1$.

Since $q_2(v_2)^{(l_2,w_2)}$ comes from some certain query request $c_2' = [\mathbf{y} \leftarrow \mathbf{q_2}(\mathbf{e_2})]^{l_2}$. We know that the execution of $\mathbf{c_2'}$ depends on $\mathbf{x}^{(l_1,w_1')}$. We want to show there is a path from $y^{(l_2,w_2)}$ to $x^{(l_1,w_1)}$ in G_{ssa}.

There are two situations that $\mathbf{c_2'}$ depends on $\mathbf{x}^{(l_1,w_1)}$.

i. $\mathbf{c_2'}$ directly depends on the variable $\mathbf{x}$. In this case, there are two possible cases:

   A. $\mathbf{c_2'}$ in either branch c_a, c_b of $[\texttt{if}(\mathbf{b}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_a}, \mathbf{c_b})]^l$. And $\mathbf{x}$ is used in the conditional $\mathbf{b}$.

   B. $c_2' = [\mathbf{y} \leftarrow \mathbf{q_2}(\mathbf{e_2})]^{l_2}$ and variable $\mathbf{x}$ is used in $\mathbf{e_2}$.

   In both situations, $M[\mathbf{y}^{(l_2,w_2)}][\mathbf{x}^{(l_1,w_1)}] = 1$ so we know there is a direct edge in G_{ssa}.

ii. $\mathbf{c_2'}$ indirectly depends on $\mathbf{x}^{(l_1,w_1)}$. Then there exists a sequence of labelled variables $\mathbf{z_1}^{(l^1,w^1)}, \mathbf{z_2}^{(l^2,w^2)}, \ldots, \mathbf{z_n}^{(l^n,w^n)}$ for some $l_1 < l^1, l^2 \ldots, l^n < l_2$ and some $w^1, w^2, \ldots, w^n$ such that :1. $\mathbf{z_1}^{(l^1,w^1)}$ directly depends on $\mathbf{x}$. 2. $\mathbf{z_i}^{(l^i,w^i)}$ directly depends on $\mathbf{z_j}^{(l^j,w^j)}$ when $i = j+1 \wedge 1 < i < n$. 3. $\mathbf{c_2'}$ directly depends on $\mathbf{z_n}^{(l^n,w^n)}$.

   We now show that direct dependence between labelled variables $\mathbf{y}^{(l,w)}$ and $\mathbf{x}^{(l',w')}$ in ssa language is tracked by our algorithm.

   There are three possible situations:

   A. The assignment statement $[\mathbf{y} \leftarrow \mathbf{e}]$, and $\mathbf{x}$ appears in $\mathbf{e}$.

   B. The assignment of $\mathbf{y}$ (either $[\mathbf{y} \leftarrow \mathbf{e}]^l$ or $[\mathbf{y} \leftarrow q(e)]^l$) appears in either branch of an if statement. $\mathbf{x}$ is used in b.

   C. $[\mathbf{y} \leftarrow \mathbf{q}(\mathbf{e})]^l$ and the variable $\mathbf{x}$ is used in $\mathbf{e}$.

   In the above three situations, our algorithm shows that $M[\mathbf{y}^{(l,w)}][\mathbf{x}^{(l',w')}] = 1$ so that there is an edge from $\mathbf{z}^{(l,w)}$ to $\mathbf{x}^{(l',w')}$.

So we can show that there is path from the labelled variable in $\mathbf{c_2'}$ to $\mathbf{x}^{(l_1,w_1)}$ when $\mathbf{c_2'}$ indirectly depends on $\mathbf{x}^{(l_1,w_1)}$.

- **[[Case:**
$$\frac{\begin{array}{c} B = |\bar{\mathbf{x}}| \qquad \Gamma \vdash^{(i,i+B)}_{M_{10},V_{10}} [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \qquad \Gamma \vdash^{(i+B,i+B+A)}_{M_{20},V_{20}} \mathbf{c} \\ \forall 1 \leq j < N. \Gamma \vdash^{(i+j*(B+A),i+B+j*(B+A))}_{M_{1j},V_{1j}} [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \qquad \Gamma \vdash^{(i+B+j*(B+A),i+B+A+j*(B+A))}_{M_{2j},V_{2j}} \mathbf{c} \\ \Gamma \vdash^{(i+N*(B+A),i+N*(B+A)+B)}_{M,V} [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \qquad \mathbf{a} = \lceil N \rceil \qquad M' = M + \sum_{0 \leq j < N} M_{1j} + M_{2j} \\ V' = V \uplus \sum_{0 \leq j < N} V_{1j} \uplus V_{2j} \end{array}}{\Gamma \vdash^{(i,i+N*(B+A)+B)}_{M',V'} [\texttt{loop}\ \mathbf{a}, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^l}$$
loop]]

  Given a memory $\mathbf{m}$, database D, trace t, loop maps w, a global list G. Suppose we have $G \vDash M', V'$ and $G; w \vDash (\texttt{loop}\ \mathbf{a}, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}), i, i + N * (B + A) + B$, suppose $\langle \mathbf{m}, \mathbf{a} \rangle \to v_N$, then from the following ssa operational semantics:

$$\frac{\mathbf{v_N} > \mathbf{0} \qquad \mathbf{n} = 0 \implies i = 1 \qquad \mathbf{n} > 0 \implies i = 2}{\langle \mathbf{m}, [\texttt{loop}\ \mathbf{v_N}, \mathbf{n}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^{\mathbf{l}}, t, w\rangle \to \langle \mathbf{m}, \mathbf{c}[\bar{\mathbf{x_i}}/\bar{\mathbf{x}}]; [\texttt{loop}\ (\mathbf{v_N} - \mathbf{1}), \mathbf{n} + \mathbf{1}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^{\mathbf{l}}, t, (w + l)\rangle}\ \textbf{ssa-loop}$$

$$\frac{v_N = 0 \qquad \mathbf{n} = 0 \implies i = 1 \qquad \mathbf{n} > 0 \implies i = 2}{\langle \mathbf{m}, [\texttt{loop}\ \mathbf{v_N}, \mathbf{n}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^{\mathbf{l}}, t, w\rangle \to \langle \mathbf{m}[\bar{\mathbf{x}} \mapsto \mathbf{m}(\bar{\mathbf{x_i}})], [\texttt{skip}]^{\mathbf{l}}, t, (w \setminus l)\rangle}\ \textbf{ssa-loop-exit}$$

When $v_N = 0$, the body is not executed and the new generated trace is empty, this case is trivially proved.

When $e_N = N$ for a positive integer N, there will be N iterations. The execution as follows :

$$\langle \mathbf{m}, [\texttt{loop}\ \mathbf{v_N}, \mathbf{0}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^{\mathbf{l}}, \mathbf{t}, \mathbf{w}\rangle \to \langle \mathbf{m}, \mathbf{c}[\bar{\mathbf{x_1}}/\bar{\mathbf{x}}]; [\texttt{loop}\ (\mathbf{v_N} - \mathbf{1}), \mathbf{1}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^{\mathbf{l}}, \mathbf{t}, (\mathbf{w} + \mathbf{l})\rangle \to \langle \mathbf{m_1}, [\texttt{loop}\ v_N - 1, 1, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^{l}, t_1, w_1\rangle \dots \langle \mathbf{m_N}, [\texttt{skip}]^{l}, t_N, (w_N \setminus l)\rangle$$

We need to show:

1. $G_s([\texttt{loop}\ v_N, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^{l}, D, \mathbf{m}, w) \subseteq G_{ssa}(M', V', i, i + N * (B + A) + B)$.
2. $K_{[\texttt{loop}\ v_N, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^{l}} \leq |V'|$.

From our assumption $G; w \vDash (\texttt{loop}\ \mathbf{a}, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}), i, i + N * (B + A) + B$, we denote $G' = G[0, \dots, (i - 1)]$, so we have:

$$\frac{\begin{array}{c} G_0 = G' \quad w_0 = w \qquad \forall 0 \leq z < N. G_z + + [\bar{\mathbf{x}}^{(\mathbf{l}, \mathbf{w_z} + \mathbf{l})}]; (w_z + l); \mathbf{c} \to G_{z+1}; w_{z+1} \\ G_f = G_N + + [\bar{\mathbf{x}}^{(\mathbf{l}, \mathbf{w_N} \setminus \mathbf{l})}] \qquad \mathbf{a} = \lceil \mathbf{N} \rceil \end{array}}{G'; w; [\texttt{loop}\ \mathbf{a}, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^{l} \to G_f; w_N \setminus l}\ \textbf{loop}$$

We can show that $G[0, \dots, (i + B - 1)] = G_0 + + [\bar{\mathbf{x}}^{(\mathbf{l}, \mathbf{w_z} + \mathbf{l})}]$ because $B = |\bar{x}|$, and $G[0, \dots, (i + B + A)] = G_1$ because A is the number of variables assigned in $\mathbf{c}$. So we have $G; w \vDash (\mathbf{c}, i + B, i + B + A)$, $G; w_1 \vDash (\mathbf{c}, i + B + 1 * (B + A), i + B + A + 1 * (B + A))$, until $G, w_{N-1} \vDash (\mathbf{c}, i + B + (N - 1) * (B + A), i + B + A + (N - 1) * (B + A))$.

By induction hypothesis on premise $\Gamma \vdash^{(i+B, i+B+A)}_{M_{20}, V_{20}} \mathbf{c}$, we conclude about the first iteration.

$$G_s(\mathbf{c}[\bar{\mathbf{x_1}}/\bar{\mathbf{x}}], D, \mathbf{m}, w) \subseteq G_{ssa}(M_{20}, V_{20}, i + B, i + B + A) \wedge K_c \leq |V_{20}|$$

By induction hypothesis on premise $\Gamma \vdash^{(i+B+j*(B+A), i+B+A+j*(B+A))}_{M_{2j}, V_{2j}} \mathbf{c}$ for $0 < j < N$, we have:

$$G_s(\mathbf{c}[\bar{\mathbf{x_2}}/\bar{\mathbf{x}}], D, \mathbf{m_j}, w_j) \subseteq G_{ssa}(M_{2j}, V_{2j}, i + B + j * (B + A), i + B + A + j * (B + A)) \wedge K_c \leq |V_{2j}|$$

We can show the first goal: $K_{[\texttt{loop}\ v_N, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^{l}} = N * K_c \leq \sum_{0 \leq j < N}(|V_{2j}|)$. Because the non-zeros of any two V_{2j}, V_{2i} $(i \neq j)$ are disjoint.

Then we need to show:

$$G_s([\texttt{loop}\ v_N, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^{l}, D, \mathbf{m}, w) \subseteq G_{ssa}(M', V', i, i + N * (B + A) + B)$$

Unfold the definition of subgraph $\subseteq$, we need to show two cases.

1. The vertices in $G_s([\texttt{loop}\ v_N, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^l, D, \mathbf{m}, w)$ can be mapped to the vertices in $G_{ssa}(M', V', i, i + N * (B + A) + B)$ by $f_{D,m}$. We know that there exists function $f_{D,m_j,c}$ for $G_s(\mathbf{c}, D, \mathbf{m_j}, w_j)$ when $0 \leq j < N$. We can have

$$f_{D,m}(vx) = \quad f_{D,m_j,c}(vx) \quad vx = q(v)^{(l,w_j)}$$

2. Every edge in $G_s([\texttt{loop}\ v_N, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^l, D, \mathbf{m}, w)$ can be mapped to a corresponding path in $G_{ssa}(M', V', i, i + N * (B + A) + B)$. The edges (vx_1, vx_2) can be divided into two categories.

   (a) vx_1, vx_2 appear in the same iteration $(t_{j+1} - t_j)$ when $0 \leq j < N$ and $t_0 = t$. We know that $(vx_1, vx_2) \in G_s(\mathbf{c}, D, \mathbf{m_j}, w_j)$ so we can find a path $p(f_{D,M}(vx_1), f_{D,m}(vx_2))$ in $G_{ssa}(M_{2j}, V_{2j}, i + B + j * (B + A), i + B + A + j * (B + A))$. We can easily show that $G_{ssa}(M_{2j}, V_{2j}, i+B+j*(B+A), i+B+A+j*(B+A))$ is a subgraph of $G_{ssa}(M', V', i, i + N * (B + A) + B)$.

   (b) vx_1, vx_2 comes from different iterations $j_1(t_{j_1+1} - t_{j_1}), j_2(t_{j_2+1} - t_{j_2})$ where $0 \leq j_1 < j_2 < N$. We assume $vx_1 = q_1(v_1)^{(l'_1, w'_{j_1})}$ and $vx_2 = q_2(v_2)^{(l'_2, w'_{j_2})}$. We unfold the definition of one edge so that we have:

$$\mathsf{DEP}(vx_1, vx_2, [\texttt{loop}\ v_N, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^l, w) \wedge \mathsf{To}(vx_1, vx_2)$$

   We know that $q_1(v_1)^{(l'_1, w'_{j_1})}$ in the trace $t_{j_1+1} - t_{j_1}$ comes from a command $\mathbf{c_1} = [\mathbf{x} \leftarrow \mathbf{q_1}(\mathbf{e_1})]^{l'_1}$ We assume the loop body $\mathbf{c} = \mathbf{c_0}; \mathbf{c_1}; \mathbf{c_2}$ then we have:

$$\frac{\begin{array}{c}\langle \mathbf{m}, [\texttt{loop}\ v_N, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^l, t, w\rangle \to^* \\ \langle \mathbf{m'_{j_1}}, [\mathbf{x} \leftarrow q_1(v_1)]^{l'_1}; \mathbf{c_2}; [\texttt{loop}\ (v_N - j_1), j_1, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^l, t'_{j_1}, w'_{j_1}\rangle \to^* \\ \langle \mathbf{m_{j_1}}, [\texttt{loop}\ (v_N - j_1), j_1, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^l, t_{j_1}, w_{j_1}\rangle \to^* \\ \langle \mathbf{m_N}, [\texttt{skip}]^l, (t_N, w_N/l)\rangle\end{array}}{\langle \mathbf{m}, [\texttt{loop}\ v_N, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^l, t, w\rangle \to^* \langle \mathbf{m_N}, [\texttt{skip}]^l, (t_N, w_N/l)\rangle}$$

   From the definition of DEP, we conclude that there exists a value v of $q_1(D)$, assigned to variable $\mathbf{x}^{(l'_1, w'_{j_1})}$, such that $q_2(v_2)^{(l'_2, w'_{j_2})} \notin t_{j_2+1} - t_{j_2}$.

   Similarly, we know that $q_2^{(l'_2, w'_{j_2})}$ in the trace comes from a command $\mathbf{c'_2} = [\mathbf{y} \leftarrow q_2(e_2)]^{l'_2}$. We know that the execution of $\mathbf{c'_2}(\mathbf{y}^{(l'_2, w'_{j_2})})$ depends on $\mathbf{x}^{(l'_1, w'_{j_1})}$.

   Similar as we show in sequence case, we now need to show there exists a path in G_{ssa} from $\mathbf{y}^{(l'_2, w'_{j_2})}$ to $\mathbf{x}^{(l'_1, w'_{j_1})}$ if the former affect the appearance of the later.

   We first show that $\mathbf{y}^{(l'_2, w'_{j_2})}$ will not directly depend on $\mathbf{x}^{(l'_1, w'_{j_1})}$ based on the position of $\mathbf{c'_2}$ in $\mathbf{c}$, there are three situations:

   i. $l'_2 < l'_1$ so that $\mathbf{c'_2} \in \mathbf{c_0}$. $\mathbf{y}^{(l'_2, w'_{j_2})}$ will not directly depend on $\mathbf{x}^{(l'_1, w'_{j_1})}$ because c_2 appears before the assignment of $\mathbf{x}$ at line l'_1.

   ii. $l'_2 = l'_1$ so that $q_1 = q_2$ and $\mathbf{x} = \mathbf{y}$. It is the same reason as previous case that $\mathbf{x}^{(l'_1, w'_{j_2})}$ does not directly depend on $\mathbf{x}^{(l'_1, w'_{j_1})}$.

   iii. $l'_2 > l'_1$ so that $\mathbf{c'_2} \in \mathbf{c_2}$. In this case, $\mathbf{y}^{(l'_2, w'_{j_2})}$ still does not directly depend on $\mathbf{x}^{(l'_1, w'_{j_1})}$ because there is only a direct edge between variables in the same iteration, while according to our assumption, it is not true.

Then, we show $\mathbf{y}^{(l'_2, w'_{j2})}$ indirectly depends on $\mathbf{x}^{(l'_1, w'_{j1})}$. There exists a sequence of labelled variables $lv_1, lv_2, \ldots, lv_n$ for such that :1. lv_1 directly depends on $\mathbf{x}^{(l'_1, w'_{j1})}$. 2. lv_i directly depends on lv_j when $i = j+1 \wedge 1 < i < n$. 3. $\mathbf{y}^{(l'_2, w'_{j2})}$ directly depends on lv_n.

We now show that direct dependence between any two labelled variables $\mathbf{y}^{(l,w)}$ and $\mathbf{x}^{(l',w')}$ in ssa language is tracked by our algorithm.

There are three possible situations:

i. The assignment statement $[\mathbf{y} \leftarrow \mathbf{e}]$, and $\mathbf{x}$ appears in $\mathbf{e}$. This case also covers the variables in $\bar{\mathbf{x}}$ of our loop statement, because we treat $[\bar{x}, \bar{x_1}, \bar{x_2}]$ as assignment $\mathbf{x} = \mathbf{x_1} + \mathbf{x_2}$ to allow the values of variables passed to next iteration.

ii. The assignment of $\mathbf{y}$ (either $[\mathbf{y} \leftarrow \mathbf{e}]^l$ or $[\mathbf{y} \leftarrow q]^l$) appears in either branch of an if statement. $\mathbf{x}$ is used in b.

iii. A switch statement assigns variable $\mathbf{y}$ such as $[\mathtt{switch}\,(\mathbf{e}, \mathbf{y}, (v_j \to q_j))]^l$ and the variable $\mathbf{x}$ is used in $\mathbf{e}$.

In the above three situations, our algorithm shows that $M[\mathbf{y}^{(l,w)}][\mathbf{x}^{(l',w')}] = 1$ so that there is an edge from $\mathbf{z}^{(l,w)}$ to $\mathbf{x}^{(l',w')}$.

So we can show that there is path from the labelled variable in $\mathbf{y}^{(l'_2, w'_{j2})}$ to $\mathbf{x}^{(l'_1, w'_{j1})}$ when $\mathbf{y}^{(l'_2, w'_{j2})}$ indirectly depends on $\mathbf{x}^{(l'_1, w'_{j1})}$.

$\square$

6 [[Examples]]

Example 6.1 (TwoRound Algorithm)**.**

$$
TR^H(k) \triangleq \begin{array}{l}
[a \leftarrow []]^1; \\
\mathtt{loop}\ [k]^2 \\
\mathtt{do} \\
\Big([x \leftarrow q()]^3; \\
[a \leftarrow x :: a]^4\Big); \\
[l \leftarrow q_{k+1}(a)]^5
\end{array}
\quad \Rightarrow \quad
TR^L \triangleq \begin{array}{l}
[a \leftarrow []]^1; \\
\mathtt{loop}\ [k]^2 \\
\mathtt{do} \\
\Big([x \leftarrow q]^3; \\
[a \leftarrow x :: a]^4\Big); \\
\left[\mathtt{switch}\left(a, l, \left(\begin{array}{l} [-n, -n, -n, \cdots, -n] \to q_{k+1,1}, \\ \cdots \\ [n, n, n, \cdots, n] \to q_{k+1, n^k} \end{array}\right)\right)\right]^5
\end{array}
$$

$$
TR^{ssa} \triangleq \begin{array}{l}
[a_1 \leftarrow []]^1; \\
\mathtt{loop}\ [k]^2, 0, [a_3, a_1, a_2] \\
\mathtt{do} \\
\Big([x_1 \leftarrow q]^3; \\
[a_2 \leftarrow x_1 :: a_3]^4\Big); \\
\left[\mathtt{switch}\left(a_3, l_1, \left(\begin{array}{l} [-n, -n, -n, \cdots, -n] \to q_{k+1,1}, \\ \cdots \\ [n, n, n, \cdots, n] \to q_{k+1, n^k} \end{array}\right)\right)\right]^5
\end{array}
$$

Adapt(TR^L) = 1

Using AdaptFun, we first generate a global list G from an empty list [] *and empty whlemap* $\emptyset$.

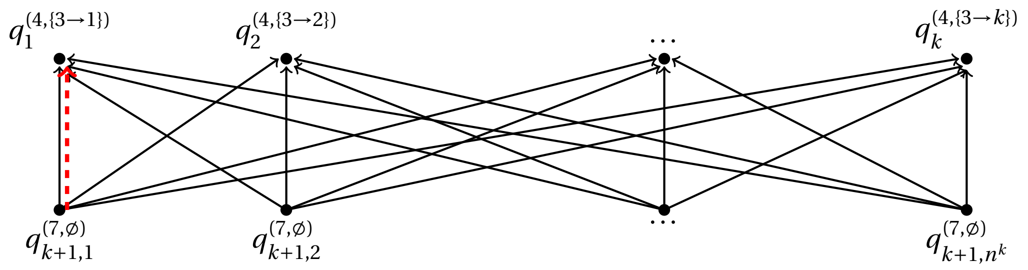
$$[];\emptyset; TR^{ssa} \to G; w \wedge w = \emptyset$$

.

$$G_{k=2} = \left[{a_1}^{(1,\emptyset)}, {a_3}^{(2,[2:1])}, {x_1}^{(3,[2:1])}, {a_2}^{(4,[2:1])}, {a_3}^{(2,[2:2])}, {x_1}^{(3,[2:2])}, {a_2}^{(4,[2:2])}, {a_3}^{(2,\emptyset)}, {l_1}^{(5,\emptyset)}\right]$$

We denote a_1^1 *short for* ${a_1}^{(1,\emptyset)}$ *and* ${a_3}^{(2,1)}$ *short for* ${a_3}^{(2,[2:1])}$*, where the label* $(2,1)$ *represents at line number* 2 *and in the* 1 *st iteration.*

$$M = \left[\begin{array}{cccccccccc} & a_1^1 & a_3^{(2,1)} & x_1^{(3,1)} & a_2^{(4,1)} & a_3^{(2,2)} & x_1^{(3,2)} & a_2^{(4,2)} & a_3^2 & l_1^5 \\ a_1^1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a_3^{(2,1)} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_1^{(3,1)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a_2^{(4,1)} & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ a_3^{(2,2)} & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ x_1^{(3,2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a_2^{(4,2)} & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ a_3^2 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ l_1^5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right], V = \left[\begin{array}{cc} a_1^1 & 0 \\ a_3^{(2,1)} & 0 \\ x_1^{(3,1)} & 1 \\ a_2^{(4,1)} & 0 \\ a_3^{(2,2)} & 0 \\ x_1^{(3,2)} & 1 \\ a_2^{(4,2)} & 0 \\ a_3^2 & 0 \\ l_1^5 & 1 \end{array}\right]$$



$q_1^{(4,\{3\to1\})}$
$q_2^{(4,\{3\to2\})}$
$\cdots$
$q_k^{(4,\{3\to k\})}$
$q_{k+1,1}^{(7,\emptyset)}$
$q_{k+1,2}^{(7,\emptyset)}$
$\cdots$
$q_{k+1,n^k}^{(7,\emptyset)}$

Example 6.2 (Multi-Round Algorithm)**.**

$$MR^H \triangleq \begin{array}{l} [I \leftarrow []]^1; \\ \texttt{loop}\ [k]^2 \\ \ \ \texttt{do} \\ \Big([p \leftarrow c]^3; \\ \left[a \leftarrow q(p,I)\right]^4; \\ \left[I \leftarrow \texttt{update}\ (\ I\ ,(a,p))\right]^5\Big) \end{array} \quad \Rightarrow \quad MR^L \triangleq \begin{array}{l} [I \leftarrow []]^1; \\ \texttt{loop}\ [k]^2 \\ \ \ \texttt{do} \\ \Big([p \leftarrow c]^3; \\ \left[\texttt{switch}\left(I, a\begin{pmatrix} [] \to q_{j,1}, \\ \cdots \\ [1,2,3,\cdots,n] \to q_{j,n!} \end{pmatrix}\right)\right]^4; \\ \left[I \leftarrow \texttt{update}\ (\ I\ ,(a,p))\right]^5\Big) \end{array}$$

Adapt(MR^L*) = 2 when* $k=2$

$$MR^{ssa} \triangleq \begin{array}{l} [I_1 \leftarrow []]^1; \\ \texttt{loop}\ [k]^2, 0, [I_3, I_1, I_2] \\ \ \texttt{do} \\ \Big([p_1 \leftarrow c]^3; \\ \left[\texttt{switch}\left(I_3, a_1\left(\begin{array}{l} [] \rightarrow q_{j,1}, \\ \cdots \\ [1,2,3,\cdots,n] \rightarrow q_{j,n!} \end{array}\right)\right)\right]^4; \\ [I_2 \leftarrow \texttt{update}\ (\ I_3\ , (a_1, p_1))]^5\Big) \end{array}$$

Using AdaptFun, we first generate a global list G from an empty list $[]$ *and empty whlemap* $\emptyset$.

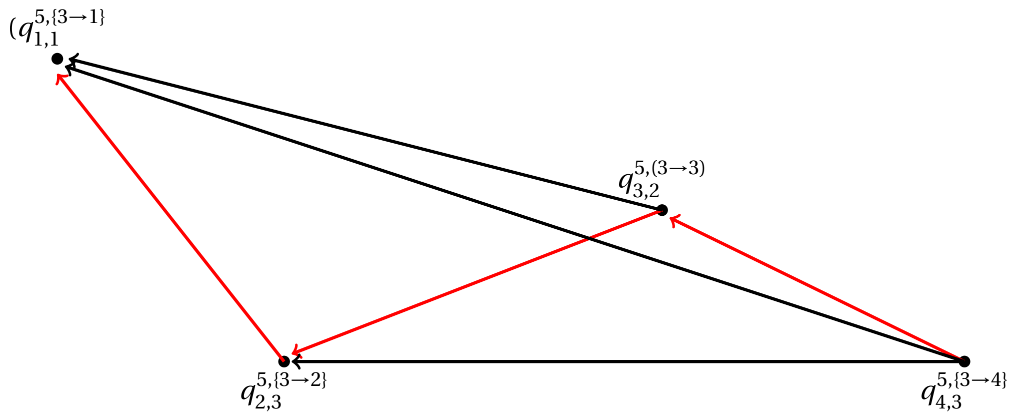
$$[]; \emptyset; MR^{ssa} \rightarrow G; w \wedge w = \emptyset$$

.

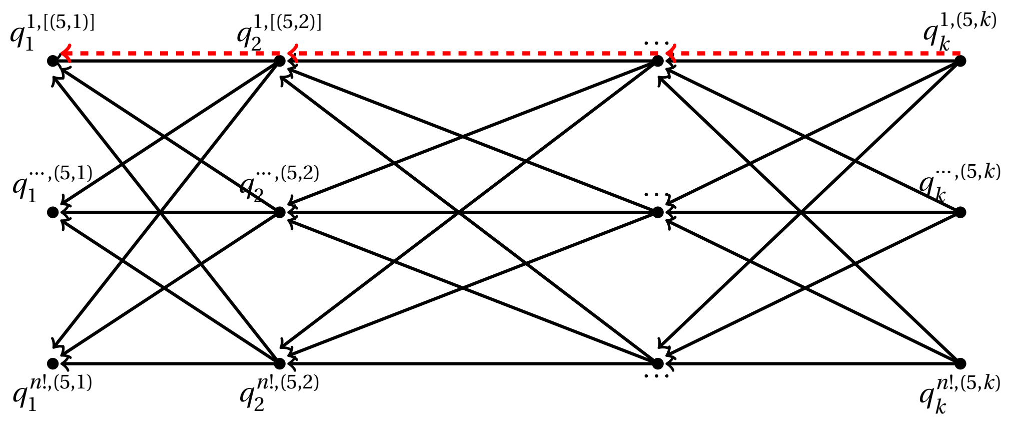
$$G_{k=2} = \left[{I_1}^{(1,\emptyset)}, {I_3}^{(2,[2:1])}, {p_1}^{(3,[2:1])}, {a_1}^{(4,[2:1])}, {I_2}^{(5,[2:1])}, {I_3}^{(2,[2:2])}, {p_1}^{(3,[2:2])}, {a_1}^{(4,[2:2])}, {I_2}^{(5,[2:2])}, {I_3}^{(2,\emptyset)}\right]$$

We denote I_1^1 *short for* ${I_1}^{(1,\emptyset)}$ *and* ${I_3}^{(2,1)}$ *short for* ${I_3}^{(2,[2:1])}$, *where the label* $(2,1)$ *represents at line number* 2 *and in the* 1 *st iteration.*

$$M = \begin{bmatrix} I_1^1 & I_3^{(2,1)} & p_1^{(3,1)} & a_1^{(4,1)} & I_2^{(5,1)} & I_3^{(2,2)} & p_1^{(3,2)} & a_1^{(4,2)} & I_2^{(5,2)} & I_3^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, V = \begin{bmatrix} I_1^1 & 0 \\ I_3^{(2,1)} & 0 \\ p_1^{(3,1)} & 0 \\ a_1^{(4,1)} & 1 \\ I_2^{(5,1)} & 0 \\ I_3^{(2,2)} & 1 \\ p_1^{(3,2)} & 0 \\ a_1^{(4,2)} & 1 \\ I_2^{(5,2)} & 0 \\ I_3^2 & 0 \end{bmatrix}$$


$(q_{1,1}^{5,\{3\rightarrow 1\}}$
$q_{3,2}^{5,(3\rightarrow 3)}$
$q_{2,3}^{5,\{3\rightarrow 2\}}$
$q_{4,3}^{5,\{3\rightarrow 4\}}$

$\forall k.\forall D$, *we have* $A(TR^L) = (k-1)$ *given all possible execution traces.*



7 Analysis of Generalization Error

Example 7.1 (Two Round Algorithm)**.**

$$TR^H(k) \triangleq \begin{array}{l} [a_1 \leftarrow []]^1; \\ \texttt{loop } [k]^2 \; (a_2 \leftarrow f(1, a_1, a_3)) \\ \;\texttt{do} \\ \Big([x_1 \leftarrow q()]^3; \\ [a_3 \leftarrow x_1 :: a_2]^4 \Big); \\ [l \leftarrow q_{k+1}(a_3)]^5 \end{array}$$

Example 7.2 (Multi-Round Algorithm)**.**

$$MR^H \triangleq \begin{array}{l} [I_2 \leftarrow []]^1; \\ \texttt{loop } [k]^2 \; (I_2 \leftarrow f(2, I_1, I_3)) \\ \;\texttt{do} \\ \Big([p_1 \leftarrow c]^3; \\ [a_1 \leftarrow \delta(q(p, I_2))]^4; \\ [I_3 \leftarrow \mathsf{update}\ (\ I_2\ , (a_1, p))]^5 \Big) \end{array}$$

By applying different mechanisms $\delta()$ over the queries $q(\cdot)$, we have different error bounds.
Gaussian Mechanism: $N(0, \sigma)$ [1]:
Adaptivity $r = 2$: $\sigma = O\left(\frac{\sqrt{r \log(k)}}{\sqrt{n}}\right)$ (also known as expected error);
Adaptivity unknown: $\sigma = O\left(\frac{\sqrt[4]{k}}{\sqrt{n}}\right)$;
Mean Squared Error Bound: $\frac{1}{2n} \min_{\lambda \in [0,1]} \left(\frac{2\rho k n - \ln(1-\lambda)}{\lambda}\right) + 2\mathbb{E}_{Z_i \sim N(0, \frac{1}{2n^2\rho})} \left[\max_{i \in [k]} (Z_i^2)\right]$
Confidence Bounds: minimize τ where $\tau \geq \sqrt{\frac{2}{n\beta} \min_{\lambda \in [0,1]} \left(\frac{2\rho k n - \ln(1-\lambda)}{\lambda}\right)}$ and $\tau \geq \frac{2}{n}\sqrt{\frac{\ln(4n/\beta}{\rho'}}$ with confidence level $1 - \beta$.
$(\epsilon, \delta) - DP$ **mechanism**:
Confidence Bounds: $\tau \geq \sqrt{\frac{48}{n} \ln(4/\beta)}$ with $\epsilon \leq \frac{\tau}{4}$ and $\delta = \exp\left(\frac{-4\ln(8/\beta)}{\tau}\right)$
Sample Splitting:

Expected Error: $O\left(\frac{\sqrt{k\log(k)}}{\sqrt{n}}\right)$
Thresholdout: B, σ, T, h
Confidence bounds: $\tau = \max\left\{\sqrt{\frac{2\zeta}{h\beta}}, 2\sigma\ln(\frac{\beta}{2}), \sqrt{\frac{1}{\beta}}\cdot\left(\sqrt{T^2+56\sigma^2}+\sqrt{\frac{\zeta}{4h}}\right)\right\}$, for $\zeta = \min_{\lambda\in[0,1)}\left(\frac{2B\ (\sigma^2 h)-\ln(1-\lambda)}{\lambda}\right)$

8 new **AdaptFun**with loop

8.1 Analysis Rules/Algorithms of **AdaptFun**

There are two steps for our algorithm to get the estimation of the adaptivity of a program **c** in the ssa form.

1. Estimate the variables that are new generated (via assignment) and store these variables in a global list G. We have the algorithm of the form : $\mathsf{VetxEst}(G; w; \mathbf{c}) \to (G'; w')$.
2. We start to track the dependence between variables in a matrix M, whose size is $|G| \times |G|$, and track whether arbitrary variable is assigned with a query result in a vector V with size $|G|$. The algorithm to fill in the matrix is of the form: $\mathsf{GraphGen}(\Gamma; \mathbf{c}; i_1) \to (M; V; i_2)$. Γ is a vector records the variables the current program **c** depends on, the index i_1 is a pointer which refers to the position of the first new-generated variable in **c** in the global list G, and i_2 points to the first new variable that is not in **c** (if exists).

We give an example of M and V of the program c.

$$
c = \begin{array}{c} \mathbf{x_1} \leftarrow \mathbf{q}; \\ \mathbf{x_2} \leftarrow \mathbf{x_1} + \mathbf{1}; \\ \mathbf{x_3} \leftarrow \mathbf{x_2} + \mathbf{2} \end{array}
\qquad
M = \left[\begin{array}{cccc} & (x_1) & (x_2) & (x_3) \\ (x_1) & 0 & 0 & 0 \\ (x_2) & 1 & 0 & 0 \\ (x_3) & 1 & 1 & 0 \end{array}\right], V = \left[\begin{array}{cc} (x_1) & 1 \\ (x_2) & 0 \\ (x_3) & 0 \end{array}\right]
$$

Still use the program c as the example, the global list G is now : $[x_1, x_2, x_3]$. The function Left and Right is used to generate the corresponding vector of the left side and right side of an assignment. Take $x_2 \leftarrow x_1 + 1$ as an example, the result is shown as follows.

$$
\mathsf{L}(1) = \left[\begin{array}{cc} 0 & (x_1) \\ 1 & (x_2) \\ 0 & (x_3) \end{array}\right] \qquad \mathsf{R}(\mathsf{x_1} + 1, 1) = \left[\begin{array}{ccc} 1 & 0 & 0 \\ (x_1) & (x_2) & (x_3) \end{array}\right]
$$

Now let us think about the loop.

$$
\mathbf{c_3} \triangleq \begin{array}{l} \left[\mathbf{x_1} \leftarrow \mathbf{q_1}\right]^1; \\ \mathtt{loop}\ [2]^2, 0, \\ [\mathbf{x_3}, \mathbf{x_1}, \mathbf{x_2}]\ \mathtt{do} \\ \Big(\left[\mathbf{y_1} \leftarrow \mathbf{q_2}\right]^3; \\ \left[\mathbf{x_2} \leftarrow \mathbf{y_1} + \mathbf{x_3}\right]^5\Big); \\ [z_1 \leftarrow x_3 + 2]^6 \end{array}
\qquad
M = \left[\begin{array}{cccccccccc} & (x_1) & (x_3^1) & (y_1^1) & (x_3^1) & (x_3^2) & (y_1^2) & (x_2^2) & (x_3^f) & (z_1) \\ (x_1) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (x_3^1) & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (y_1^1) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (x_2^1) & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ (x_3^2) & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ (y_1^2) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (x_2^2) & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ (x_3^f) & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ (z_1) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right], V = \left[\begin{array}{cc} (x_1) & 1 \\ (x_3^1) & 0 \\ (y_1^1) & 1 \\ (x_2^1) & 0 \\ (x_3^2) & 0 \\ (y_1^2) & 1 \\ (x_2^2) & 0 \\ (x_3^f) & 0 \\ (z_1) & 0 \end{array}\right]
$$

$$
\mathbf{c_3'} \triangleq \begin{array}{l} \left[\mathbf{x_1} \leftarrow \mathbf{q_1}\right]^1; \\ \mathtt{loop}\ [0]^2, 0, \\ [\mathbf{x_3}, \mathbf{x_1}, \mathbf{x_2}]\ \mathtt{do} \\ \Big(\left[\mathbf{y_1} \leftarrow \mathbf{q_2}\right]^3; \\ \left[\mathbf{x_2} \leftarrow \mathbf{y_1} + \mathbf{x_3}\right]^5\Big); \\ [z_1 \leftarrow x_3 + 2]^6 \end{array}
\qquad
M = \left[\begin{array}{cccc} & (x_1) & (x_3^f) & (z_1) \\ (x_1) & 0 & 0 & 0 \\ (x_3^f) & 1 & 0 & 0 \\ (z_1^2) & 0 & 1 & 0 \end{array}\right], V = \left[\begin{array}{cc} (x_1) & 1 \\ (x_3^f) & 0 \\ (z_1) & 0 \end{array}\right]
$$

We can now look at the if statement.

$$
c_4 \triangleq \begin{array}{l}
[x \leftarrow q_1]^1; \\
[y \leftarrow q_2]^2; \\
\texttt{if } (x + y == 5)^3 \\
\mathsf{then}\ [x \leftarrow q_3]^4 \\
\mathsf{else(}\ [x \leftarrow q_4]^5; \\
y \leftarrow 2); \\
[z \leftarrow x + y]^6;
\end{array}
\quad \hookrightarrow \quad
\mathbf{c_4} \triangleq \begin{array}{l}
[\mathbf{x_1} \leftarrow \mathbf{q_1}]^1; \\
[\mathbf{y_1} \leftarrow \mathbf{q_2}]^2; \\
\texttt{if } (\mathbf{x_1} + \mathbf{y_1} == \mathbf{5})^3, [x_4, x_2, x_3], [], [y_3, y_1, y_2] \\
\mathsf{then}\ [\mathbf{x_2} \leftarrow \mathbf{q_3}]^4 \\
\mathsf{else(}\ [\mathbf{x_3} \leftarrow \mathbf{q_4}]^5; \\
\mathbf{y_2} \leftarrow \mathbf{2}) \\
[\mathbf{z_1} \leftarrow \mathbf{x_4} + \mathbf{y_3}]^6;
\end{array}
$$

$$
M_{c4} = \left[\begin{array}{ccccccccc}
 & (x_1) & (y_1) & (x_2) & (x_3) & (y_2) & (x_4) & (y_3) & (z_1) \\
(x_1) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(y_1) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(x_2) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(x_3) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(y_2) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(x_4) & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
(y_3) & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
(z_1) & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0
\end{array}\right], V_{c4} = \left[\begin{array}{cc}
(x_1) & 1 \\
(y_1) & 1 \\
(x_2) & 1 \\
(x_3) & 1 \\
(y_2) & 0 \\
(x_4) & 0 \\
(y_3) & 0 \\
(z_1) & 0
\end{array}\right]
$$

8.2 The algorithm to estimate the matrix and vector

We first generate a list of variables G that will be assigned with values (via the command $x \leftarrow e$ or $x \leftarrow q$).

$$
\frac{}{\mathsf{VetxEst}(G; w; [\mathbf{x} \leftarrow \mathbf{e}]^{\mathbf{l}}) \to (G + +[\mathbf{x}^{(l,w)}]; w)}\ \textbf{ag-asgn}
$$

$$
\frac{}{\mathsf{VetxEst}(G; w; [\mathbf{x} \leftarrow q(\mathbf{e})]^{l}) \to (G + +[\mathbf{x}^{(l,w)}]; w)}\ \textbf{ag-query}
$$

$$
\frac{\begin{array}{c}\mathsf{VetxEst}(G; w; \mathbf{c_1}) \to (G_1; w_1) \qquad \mathsf{VetxEst}(G_1; w; \mathbf{c_2}) \to (G_2; w_2) \\ G_3 = G_2 + +[\bar{\mathbf{x}}^{(\mathbf{l,w})}] + +[\bar{\mathbf{y}}^{(\mathbf{l,w})}] + +[\bar{\mathbf{z}}^{(\mathbf{l,w})}]\end{array}}{\mathsf{VetxEst}(G; w; [\texttt{if}\,(\mathbf{b}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2})]^{l}) \to (G_3; w)}\ \textbf{ag-if}
$$

$$
\frac{\mathsf{VetxEst}(G; w; \mathbf{c_1}) \to (G_1; w_1) \qquad \mathsf{VetxEst}(G_1; w_1; \mathbf{c_2}) \to (G_2; w_2)}{\mathsf{VetxEst}(G; w; (\mathbf{c_1}; \mathbf{c_2})) \to (G_2; w_2)}\ \textbf{ag-seq}
$$

$$
\frac{\begin{array}{c}G_0 = G \quad w_0 = w \qquad \forall 0 \le z < N. \mathsf{VetxEst}(G_z + +[\bar{\mathbf{x}}^{(\mathbf{l,w_z+l})}]; (w_z + l); \mathbf{c}) \to (G_{z+1}; w_{z+1}) \\ G_f = G_N + +[\bar{\mathbf{x}}^{(\mathbf{l,w_N \setminus l})}] \qquad \mathbf{a} = N\end{array}}{\mathsf{VetxEst}(G; w; [\texttt{loop}\ \mathbf{a}, n, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^{l}) \to (G_f; w_N \setminus l)}\ \textbf{ag-loop}
$$

Analysis Logic Rules. Γ is a matrix of one row and N columns, $N = |G| = |V|$.

$\mathsf{L(i)}$ generates a matrix of one column, N rows, where the $i-th$ row is 1, all the other rows are 0.

$\mathsf{R}(\mathsf{e}, \mathsf{i})$ generates a matrix of one row and N columns, where the locations of variables in e is marked as 1. To handle loop, for instance, the variable y appears many times in G, the argument i helps to find the location of the current living variable y in the expression e, which is the latest y with the largest location $i_y < i$ in our global variable list G.

$\forall 0 \leq z < |\bar{x}|. \bar{x}(z) = x_z, \bar{x_1}(z) = x_{1z}, \bar{x_2}(z) = x_{2z}$

$\Gamma \vdash^{i,i+|\bar{\mathbf{x}}|}_{M,v_\emptyset} [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \triangleq \forall 0 \leq z < |\bar{x}|. \Gamma \vdash^{i+z,i+z+1}_{M_{x_z},V_\emptyset} x_z \leftarrow x_{1z} + x_{2z}$ where $M = \sum_{z \in [|\bar{x}|]} M_{x_z}$

$\boxed{\Gamma \vdash^{i_1,i_2}_{M,V} c}$

$$\frac{M = \mathsf{L}(i) * (\mathsf{R}(\mathbf{e}, i) + \Gamma)}{\mathsf{GraphGen}(\Gamma; [\mathbf{x} \leftarrow \mathbf{e}]^l; i) \to (M; V_\emptyset; i+1)}\ \textbf{ad-asgn}$$

$$\frac{M = \mathsf{L}(i) * (\mathsf{R}(\mathbf{e}, i) + \Gamma) \qquad V = \mathsf{L}(i)}{\mathsf{GraphGen}(\Gamma; [\mathbf{x} \leftarrow q(\mathbf{e})]^l; i) \to (M; V; i+1)}\ \textbf{ad-query}$$

$$\frac{\begin{array}{c}\mathsf{GraphGen}(\Gamma + \mathsf{R}(\mathbf{b}, i_1); \mathbf{c_1}; i_1) \to (M_1; V_1; i_2) \qquad \mathsf{GraphGen}(\Gamma + \mathsf{R}(\mathbf{b}, i_1); \mathbf{c_2}; i_2) \to (M_2; V_2; i_3) \\ \mathsf{GraphGen}(\Gamma; [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]; i_3) \to (M_x; V_\emptyset; i_3 + |\bar{\mathbf{x}}|) \qquad \mathsf{GraphGen}(\Gamma; [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}]; i_3 + |\bar{\mathbf{x}}|) \to (M_y; V_\emptyset; i_3 + |\bar{\mathbf{x}}| + |\bar{\mathbf{y}}|) \\ \mathsf{GraphGen}(\Gamma; [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}]; i_3 + |\bar{\mathbf{x}}| + |\bar{\mathbf{y}}|) \to (M_y; V_\emptyset; i_3 + |\bar{\mathbf{x}}| + |\bar{\mathbf{y}}| + |\bar{\mathbf{z}}|) \\ M = (M_1 + M_2) + M_x + M_y + M_z\end{array}}{\mathsf{GraphGen}(\Gamma; \mathtt{if}([\mathbf{b}]^l, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2}); i_1) \to (M; V_1 \uplus V_2; i_3 + |\bar{x}| + |\bar{y}| + |\bar{z}|)}\ \textbf{ad-if}$$

$$\frac{\mathsf{GraphGen}(\Gamma; \mathbf{c_1}; i_1) \to (M_1; V_1; i_2) \qquad \mathsf{GraphGen}(\Gamma; \mathbf{c_2}; i_2) \to (M_2; V_2; i_3)}{\mathsf{GraphGen}(\Gamma; (\mathbf{c_1}; \mathbf{c_2}); i_1) \to ((M_1; M_2); V_1 \uplus V_2; i_3)}\ \textbf{ad-seq}$$

$$\frac{\begin{array}{c} B = |\bar{\mathbf{x}}| \qquad A = |\mathbf{c}| \\ \forall 0 \leq j < N. \mathsf{GraphGen}(\Gamma; [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]; i + j * (B + A)) \to (M_{1j}; V_{1j}; i + B + j * (B + A)) \\ \mathsf{GraphGen}(\Gamma; \mathbf{c}; i + B + j * (B + A)) \to (M_{2j}; V_{2j}; i + B + A + j * (B + A)) \\ \mathsf{GraphGen}(\Gamma; [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]; i + N * (B + A)) \to (M; V; i + N * (B + A) + B) \\ \mathbf{a} = N \qquad M' = M + \sum\limits_{0 \leq j < N} (M_{1j} + M_{2j}) \qquad V' = V \uplus \sum\limits_{0 \leq j < N} (V_{1j} \uplus V_{2j}) \end{array}}{\mathsf{GraphGen}(\Gamma; \mathtt{loop}\ [\mathbf{a}]^l,\ 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \mathtt{do}\ \mathbf{c}, i) \to (M'; V'; i + N * (B + A) + B)}\ \textbf{ad-loop}$$

$$\begin{array}{rcl} M_1; M_2 & := & M_2 \cdot M_1 + M_1 + M_2 \\ V_1 \uplus V_2 & := & \left\{ \begin{array}{ll} 1 & (V_1[i] = 1 \vee V_2[i] = 1) \wedge i = 1, \cdots, N \wedge |V_1| = |V_2| \\ 0 & o.w. \end{array} \right. \end{array}$$

$$
\begin{array}{lcl}
|[\texttt{switch}\,(\mathbf{e},\mathbf{x},(v_j \to q_j))]^{l}|_{low} & = & [\texttt{switch}\,(|\mathbf{e}|_{low},|x|_{low},(v_j \to q_j))]^{l} \\
|[\texttt{loop}\;\mathbf{a},n,[\bar{\mathbf{x}},\bar{\mathbf{x_1}},\bar{\mathbf{x_2}}]\;\texttt{do}\;\mathbf{c}]^{l}|_{low} & = & [\texttt{loop}\;|\mathbf{a}|_{low},\;\texttt{do}\;|\mathbf{c}|_{low}]^{l} \\
|\mathbf{c_1};\mathbf{c_2}|_{low} & = & |\mathbf{c_1}|_{low};|\mathbf{c_2}|_{low} \\
|[\texttt{if}\,(\mathrm{b},[\bar{\mathbf{x}},\bar{\mathbf{x_1}},\bar{\mathbf{x_2}}],[\bar{\mathbf{y}},\bar{\mathbf{y_1}},\bar{\mathbf{y_2}}],[\bar{\mathbf{z}},\bar{\mathbf{z_1}},\bar{\mathbf{z_2}}],\mathbf{c_1},\mathbf{c_2})]^{l}|_{low} & = & [\texttt{if}\,(|\mathrm{b}|_{low},|\mathbf{c_1}|_{low},|\mathbf{c_2}|_{low})]^{l} \\
|[\mathbf{x} \leftarrow \mathbf{e}]^{l}|_{low} & = & [|\mathbf{x}|_{low} \leftarrow |\mathbf{e}|_{low}]^{l} \\
|[\mathbf{x} \leftarrow q]^{l}|_{low} & = & [|\mathbf{x}|_{low} \leftarrow q]^{l} \\
|x_i|_{low} & = & x \\
|n|_{low} & = & n \\
|\mathbf{a_1} \oplus_a \mathbf{a_2}|_{low} & = & |\mathbf{a_1}|_{low} \oplus_a |\mathbf{a_2}|_{low} \\
|\mathbf{b_1} \oplus_b \mathbf{b_2}|_{low} & = & |\mathbf{b_1}|_{low} \oplus_b |\mathbf{b_2}|_{low}
\end{array}
$$

Figure 3: The erasure of SSA

References

[1] Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Leon Roth. Preserving statistical validity in adaptive data analysis. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 117–126, 2015.

[2] Sumit Gulwani and Florian Zuleger. The reachability-bound problem. In *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '10, page 292–304, New York, NY, USA, 2010. Association for Computing Machinery.

[3] Panagiotis Vekris, Benjamin Cosman, and Ranjit Jhala. Refinement types for typescript. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 310–325, 2016.