Adaptivity Analysis

Abstract

A data analysis is said to *generalize* when it allows one to draw conclusions from the data that are true of the population from which the data are sampled. Statistician and data scientist have devised several methods aimed to guarantee generalization in data analyses, and avoid in this way overfitting to the data. Guaranteeing generalization is more difficult when data analyses are *adaptive*: when the result of an analysis depends on the result of previous analyses.

A recent line of work focuses on methods aimed at guaranteeing generalization in adaptive data analysis through the addition of carefully calibrated statistical noise to the empirical results of the analysis on the sampled data. In these works, the confidence intervals on the generalization error that one can achieve for a given analysis usually depend on the *level of adaptivity* of the analysis: the number of adaptive steps that depend on each other.

In this work we introduce a programming framework, named AdaptFun, aimed at supporting the study of the generalization error for adaptive data analysis. Through its analysis system, an upper bound on the adaptivity – depth (the length of the longest chain of queries) of a program implementing an adaptive data analysis can be overestimated. We show how this language can help to analyze the generalization error between two data analyses with different adaptivity structures.

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 support the model by providing a "high" level language, which is expressive to represent the adaptive mechnisms. In this language, the queries are allow to carry arguments, for example, the expression $q(e)$ simulates the process of submitting a query to the interface in the model while the argument e is consumed to construct the query. For instance, one submitted query may use the average of answers of previous queries, can be expressed $q(x)$ where x stores the expected average results in our language. In this sense,the result of the query from a specific D can vary under different contexts.

Nevertheless, in *statistical query model*, the dependency between two submitted queries is vague, especially when we study in adaptive scenarios. Our definition of the dependency lies in the independence between two queries: 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. Following such definition, the dependency between two queries comes from either the control flow or the argument of the function q_2, which brings us difficulty to distinguish them at the semantic level.
An example to show how do control flow and query argument may bring challenges.

To simplify the model, we start from a low level language where the queries are atomic – q – given data base D, the result of the query from D is deterministic. Hence, we define the adaptivity of the program under this model based on only the control flow. The arguments used in queries in the high level language are rearranged and the program expressed in the high level one can be transformed into its low level version.

We give the definition of adaptivity of a low level program by graphs, called dependency graph. These graph is produced using a trace of queries which is generated along with the semantics of the program. The queries in the trace consists of the nodes in the graph while the edge represent dependency. If there is no dependency between two node(queries), there will be no edge.

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 Low Level Control Flow Based Language

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

2.1 Syntax and Semantics

Syntax.

Arithmatic Operators	$*_a$	::=	$+ \mid - \mid \times \mid \div$
Boolean Operators	$*_b$		
Relational Operators	$*_r$	::=	$< \mid \leq \mid =$
Label	l		
While Map	w	$\in$	$\text{Label} \times \mathbb{N}$
AExpr	a	::=	$n \mid x \mid a *_a a \mid [] \mid [a_0, \ldots, a_i] \mid \texttt{uniform} \mid \texttt{bernoulli} \mid a \times a$
BExpr	b	::=	$\texttt{true} \mid \texttt{false} \mid \neg b \mid b *_b b \mid a *_r a$
Expr	e	::=	$a \mid b$
Command	c	::=	$[x \leftarrow e]^l \mid [x \leftarrow q]^l \mid [\texttt{switch}\,(e,x,(v_i \to q_i))]^l \mid \texttt{loop}\ [e_N]^l\ (f)\ \texttt{do}\ c$
			$\mid c;c \mid \texttt{if}([b]^l, c_1, c_2) \mid \texttt{while}([b]^l, c) \mid [\texttt{skip}]^l \mid \texttt{unfold}([b]^l, c)$
Memory	m	::=	$[] \mid m[x^l \to v]$
Trace	t	::=	$[] \mid [(q,v)^{(l,w)}] \mid t ++ t$

Operational Semantics.

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

$$\frac{}{\langle m, x \rangle \to_a m(x)} \qquad \frac{\langle m, a_1 \rangle \to_a a_1'}{\langle m, a_1 *_a a_2 \rangle \to_a e_1' *_a a_2} \qquad \frac{\langle m, a_2 \rangle \to_a a_2'}{\langle m, n_1 *_a a_2 \rangle \to_a n_1 *_a a_2'}$$

$$\frac{n_3 = n_1 *_a n_2}{\langle m, n_1 *_a n_2 \rangle \to_a n_3}$$

$$\boxed{\langle m, b \rangle \to_b b' : Memory \times BExpr \Rightarrow BExpr}$$

$$\frac{}{\langle m, \texttt{true} \rangle \to_b \texttt{true}} \qquad \frac{}{\langle m, \texttt{false} \rangle \to_b \texttt{false}} \qquad \frac{\langle m, a_1 \rangle \to_a a_1'}{\langle m, a_1 *_r a_2 \rangle \to_b e_1' *_r a_2}$$

$$\frac{\langle m, a_2 \rangle \to_a a_2'}{\langle m, n_1 *_r a_2 \rangle \to_b n_1 *_r a_2'} \qquad \frac{b_3 = n_1 *_r n_2}{\langle m, n_1 *_r n_2 \rangle \to_b b_3} \qquad \frac{\langle m, b_1 \rangle \to_b b_1'}{\langle m, b_1 *_b b_2 \rangle \to_b b_1' *_b b_2}$$

$$\frac{\langle m, b_2 \rangle \to_b b_2'}{\langle m, \texttt{true} *_b b_2 \rangle \to_b \texttt{true} *_b b_2'} \qquad \frac{\langle m, b_2 \rangle \to_b b_2'}{\langle m, \texttt{false} *_b b_2 \rangle \to_b \texttt{false} *_b b_2'} \qquad \frac{\langle m, b \rangle \to_b b'}{\langle m, \neg b \rangle \to_b \neg b'}$$

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

$$Memory \times Com \times Db \times Trace \times WhileMap \Rightarrow Memory \times Com \times Db \times Trace \times WhileMap$$

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

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

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

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

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

$$\frac{}{\langle m, \mathtt{while}([b]^l, c), D, t, w\rangle \to \langle m, \mathtt{unfold}([b^l], \mathtt{while}([b]^l, c)), D, t, w\rangle}\ \textbf{while}$$

$$\frac{\langle m, b\rangle \to b'}{\langle m, \mathtt{unfold}([b]^l, c), D, t, w\rangle \to \langle m, \mathtt{unfold}([b']^l, c), D, t, w\rangle}\ \textbf{unfold}$$

$$\frac{}{\langle m, \mathtt{unfold}([\mathtt{false}]^l, c), D, t, w\rangle \to \langle m, [\mathtt{skip}]^l, D, t, (w \setminus l)\rangle}\ \textbf{unfold-f}$$

$$\frac{}{\langle m, \mathtt{unfold}([\mathtt{true}]^l, c), D, t, w\rangle \to \langle m, c, D, t, (w + l)\rangle}\ \textbf{unfold-t}$$

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

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

$$\frac{\langle m, e_N \to e_N'\rangle}{\langle m, \mathtt{loop}\ [e_N]^l\ (f)\ \mathtt{do}\ c, D, t, w\rangle \to \langle m, [\mathtt{loop}\ [e_N]^l\ (f)\ \mathtt{do}\ c]^l, D, t, w\rangle}\ \textbf{loop}$$

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

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

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

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

2.2 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.*
$\forall Q_1, Q_2, l_1, l_2, w_1, w_2.$ $(Q_1^{l_1,w_1})$ TO $(Q_2^{l_2,w_2})$, *denoted as* $\mathsf{To}(Q_1^{l_1,w_1}, Q_2^{l_2,w_2})$
iff $(Q_1^{l_1,w_1}) <_q (Q_2^{l_2,w_2})$
where
$(Q_1^{l_1,w_1}) <_q (Q_2^{l_2,w_2})$ *is defined:*

$$
\begin{array}{ll}
l_1 < l_2 & w_1 = \emptyset \lor w_2 = \emptyset \lor w_1 =_w w_2 \\
w_1 <_w w_2 & \text{Otherwise}
\end{array}
$$

Independence between two queries in Low level language When two queries q_1, q_2 are independent in a program c, suppose q_1 appears before q_2 in the program c, we think the choice of queries starting from q_1, ending with query q_2 should be fixed no matter the change of the result of q_1.

Definition 3 (Query Independence)**.** *Two queries q_i and q_j in a program c are independent,* $\mathsf{IND}(q_i^{l_1}, q_j^{l_2}, c)$.

$$
\begin{array}{l}
\quad \forall m, D. \Big(\langle m, c, D, [] \rangle \to \langle m', \texttt{skip}, D, t \rangle \\
\land \Big((q_i^{l_1}, v_i) \in t \land (q_j^{l_2}, v_j) \in t \implies \forall v \in codomain(q_i^{l_1}). \big(\langle m, c[v/q_i], D, [] \rangle \to \langle m', \texttt{skip}, D, t' \rangle \land (q_j^{l_2}, v_j) \in t' \big) \Big) \\
\land \Big((q_i^{l_1}, v_i) \in t \land (q_j^{l_2}, v_j) \notin t \implies \forall v \in codomain(q_i^{l_1}). \big(\langle m, c[v/q_i], D, [] \rangle \to \langle m', \texttt{skip}, D, t' \rangle \land (q_j^{l_2}, v_j) \notin t' \big) \Big) \Big).
\end{array}
$$

In following examples:

$$
c_1 \triangleq \begin{array}{l}
\left[x \leftarrow q_1\right]^1; \\
\texttt{if } [x]^2 \\
\text{then } \left[y \leftarrow q_2\right]^3 \\
\text{else } \left[y \leftarrow 0\right]^4; \\
\texttt{if } [x]^5 \\
\text{then } \left[y \leftarrow 0\right]^6 \\
\text{else } \left[y \leftarrow q_2\right]^7;
\end{array}
\qquad
c_2 \triangleq \begin{array}{l}
\left[x \leftarrow q_1\right]^1; \\
\left[y \leftarrow q_2\right]^2; \\
\texttt{if } \left[x + y = 5\right]^3 \\
\text{then } \left[z \leftarrow q_3\right]^4 \\
\text{else } \left[\texttt{skip}\right]^5; \\
\left[w \leftarrow q_4\right]^6;
\end{array}
\qquad
c_3 \triangleq \begin{array}{l}
\left[x \leftarrow q_1\right]^1; \\
\texttt{while}\Big([x \leq 100]^2, \\
\left[y \leftarrow q_2\right]^3; \\
\left[z \leftarrow q_3\right]^4; \\
\left[x \leftarrow y + z + x\right]^5 \Big);
\end{array}
\qquad
c_4 \triangleq \begin{array}{l}
\left[x \leftarrow q_1\right]^1; \\
\texttt{while}\Big([x \leq 100]^2, \\
\left[y \leftarrow q_2 + x\right]^3; \\
\texttt{while}\Big(\left[y \leq 50\right]^4, \\
\left[z \leftarrow q_3\right]^5; \\
\left[y \leftarrow y + z\right]^6; \Big) \\
\left[x \leftarrow y + x\right]^7 \Big);
\end{array}
$$

We have the dependency as:
In program c_1: $\mathsf{IND}(q_2^3, q_2^7, c_1)$,
In program c_2: $\mathsf{IND}(q_1^1, q_2^2, c_4)$, $\mathsf{IND}(q_3^3, q_4^4, c_4)$, $\mathsf{IND}(q_1^1, q_4^4, c_4)$, $\mathsf{IND}(q_2^2, q_4^4, c_4)$,
In program c_3: $\mathsf{IND}(q_2^{(3,l)}, q_3^{(4,l)}, c_2)$ for all l.
In program c_4: No independent queries.

Dependency between multiple queries

Definition 4 (Dependency Graph)**.** *A dependency graph over a program P is defined as $G(P) = (V, E, \mathsf{end})$, where V is set of verticals and E is the set of directed edges:*
$V = \{q_1, q_2, \cdots, q_n\}$, where $q_1, q_2, \cdots, q_n$ are reachable queries in the program P.
$E = \{(q_i, q_j) | \exists m.\mathsf{Dep}(q_i, q_j, P, m)\}$, in the program P.
$\mathsf{end} = \{q_{e_1}, \cdots, q_{e_l}\}$ be the set of queries related to the return value.

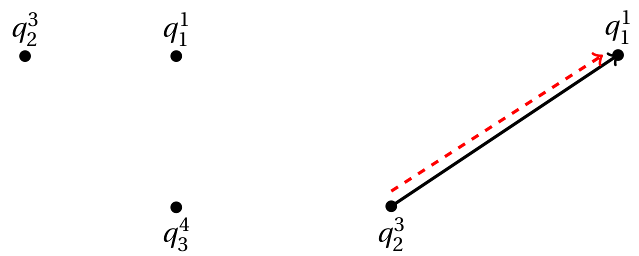
Definition 5 (Reachable Query)**.** *. A query q in program P is reachable iff q may be executed.*

Definition 6 (Adaptivity)**.** *Given a program P and its dependency graph $G(P) = (V, E, \mathsf{end})$, the adaptivity of the program is defined as $A(P)$, s.t.: for every $q_i \in \mathsf{end}$, let p_i be the longest path starting from q_i with length l_i,*

$$A(P) = \max_{q_i \in \mathsf{end}} \{l_i \mid l_i = |p_i|\}$$

Example 2.1. ***Dependency graphs for programs containing 3 atomic queries***
Let $q_1 \triangleq \lambda D.D_1 * D_j$, $q_2 \triangleq \lambda D.D_3 * D_4$ and $q_3 \triangleq \lambda D.D_3 * D_2$. in program P_1 and P_2 as follows:

$$P_1 \triangleq \begin{array}{l} [w \leftarrow 100]^0; \\ \left[x \leftarrow q_1\right]^1; \\ \mathtt{if}([w > 1]^2) \\ \quad \mathit{then}\ \left[y \leftarrow q_2\right]^3; \\ \quad \mathit{else}\ \left[z \leftarrow q_3\right]^4 \end{array} \qquad P_2 \triangleq \begin{array}{l} \left[x \leftarrow q_1\right]^1; \\ \mathtt{if}([x > 1]^2) \\ \quad \mathit{then}\ \left[y \leftarrow q_2\right]^3; \\ \quad \mathit{else}\ [z \leftarrow 10]^4 \end{array}$$



3 High level Language

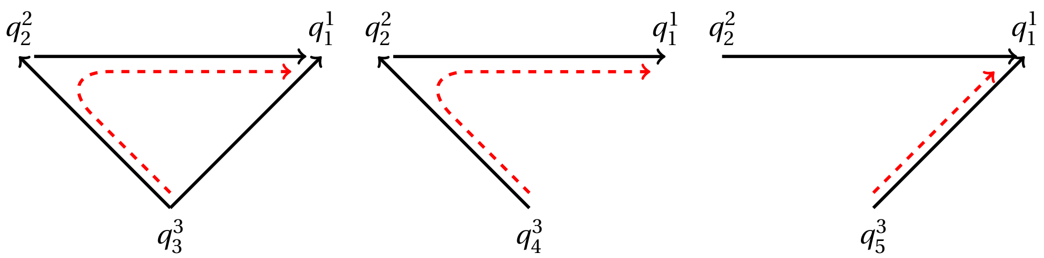
3.1 Syntax and Semantics

Syntax.

Arithmatic Operators	$*_a$	$::=$	$+ \mid - \mid \times \mid \div$
Boolean Operators	$*_b$		
Relational Operators	$*_r$	$::=$	$< \mid \leq \mid =$
Label	l		
While Map	w	$\in$	$\text{Label} \times \mathbb{N}$
AExpr	a	$::=$	$n \mid x \mid a *_a a \mid [] \mid [a_0, \ldots, a_i] \mid \texttt{uniform} \mid \texttt{bernoulli} \mid a \times a$
BExpr	b	$::=$	$\texttt{true} \mid \texttt{false} \mid \neg b \mid b *_b b \mid a *_r a$
Command	c	$::=$	$[x \leftarrow e]^l \mid [x \leftarrow q\,(e)]^l$
			$\mid c; c \mid \texttt{if}([b]^l, c_1, c_2) \mid \texttt{while}([b]^l, c) \mid [\texttt{skip}]^l \mid \texttt{loop}\ [v_N]^l\ (f)\ \texttt{do}\ c$
Memory	m	$::=$	$[] \mid m[x^l \to v]$
Trace	t	$::=$	$[] \mid [(q, v)^{(l,w)}] \mid t ++ t$

Example 3.1. ***Dependency graphs for high level programs containing non-atomic queries***
Let $q_1 = \lambda D.D_i * D_j$,
Let $q_2(x_1) = \lambda D.D_i * D_j + x_1$.
Let $q_3(x_1 - x_2) = \lambda D.D_i * D_j + x_1 - x_2$, $q_4(x_2) = \lambda D.D_i * D_j + x_2$, *and* $q_5(x_1) = \lambda D.D_i * D_j + x_1$.
in program P_1, P_2 *and* P_3 *as following:*

$$P_1 \triangleq \begin{array}{c} [x_1 \leftarrow q_1]^1; \\ [x_2 \leftarrow q_2]^2; \\ [x_3 \leftarrow q_3]^3 \end{array} \qquad P_2 \triangleq \begin{array}{c} [x_1 \leftarrow q_1]^1; \\ [x_2 \leftarrow q_2]^2; \\ [x_3 \leftarrow q_4]^3 \end{array} \qquad P_3 \triangleq \begin{array}{c} [x_1 \leftarrow q_1]^1; \\ [x_2 \leftarrow q_2]^2; \\ [x_3 \leftarrow q_5]^3 \end{array}$$



3.2 Rewriting from High Level Program into Low Level Program

The transformation $(\!| e^h |\!) = e^l$ 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.

$$
\begin{array}{lll}
(\!|[x \leftarrow q\ (e)]^l|\!) & \Rightarrow & \left[\ \texttt{switch}\ \left(e, x, \left(\begin{array}{l} v_1 \rightarrow q_1, \\ \cdots, \\ v_m \rightarrow q_m \end{array}\right)\right)\ \right]^l \\
(\!|q()|\!) & \Rightarrow & q \\
(\!|[x \leftarrow \texttt{uniform}]^l|\!) & \Rightarrow & [x \leftarrow c_u]^l \\
(\!|[x \leftarrow e]^l|\!) & \Rightarrow & [x \leftarrow e]^l \\
(\!|c_1; c_2|\!) & \Rightarrow & (\!|c_1|\!); (\!|c_2|\!) \\
(\!|\texttt{if}([b]^l, c_1, c_2)|\!) & \Rightarrow & \texttt{if}([b]^l, (\!|c_1|\!), (\!|c_2|\!)) \\
(\!|\texttt{while}([b]^l, c)|\!) & \Rightarrow & \texttt{while}([b]^l, (\!|c|\!))
\end{array}
$$

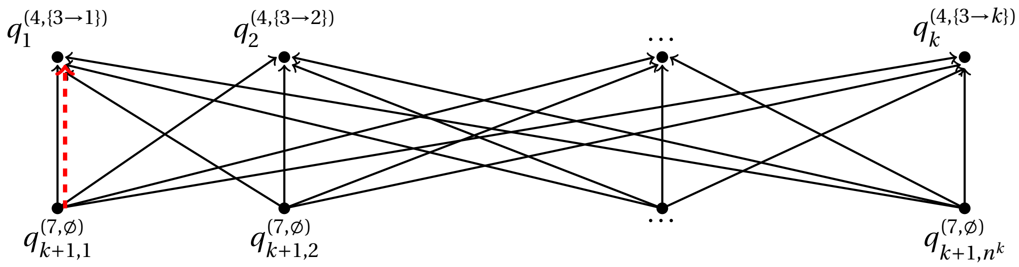
Example 3.2 (Two Round Algorithm)**.**

$$
TR^H(k) \triangleq (\!|\ \begin{array}{l}
\left[j \leftarrow 1\right]^1; \\
\left[a \leftarrow []\right]^2; \\
\texttt{while}\Big(\left[j \leq k\right]^3, \\
\left[x \leftarrow q_j()\right]^4; \\
\left[a \leftarrow x :: a\right]^5; \\
\left[j \leftarrow j+1\right]^6\Big); \\
\left[l \leftarrow q_{k+1}(a)\right]^7;
\end{array}\ |\!) \quad \Rightarrow \quad TR^L \triangleq \begin{array}{l}
\left[j \leftarrow 1\right]^1; \\
\left[a \leftarrow []\right]^2; \\
\texttt{while}\Big(\left[j \leq k\right]^3, \\
\left[x \leftarrow q_j\right]^4; \\
\left[a \leftarrow x :: a\right]^5; \\
\left[j \leftarrow j+1\right]^6\Big); \\
\left[\texttt{switch}\left(a, x, \left(\begin{array}{l} [-n, -n, -n, \cdots, -n] \rightarrow q_{k+1,1}, \\ \cdots \\ [n, n, n, \cdots, n] \rightarrow q_{k+1,n^k} \end{array}\right)\right)\right]^7
\end{array}
$$

$\langle \emptyset, TR^L, D, [], \emptyset \rangle \rightarrow \langle m, \texttt{skip}, D, t, w \rangle$.

$$
t = [(q_1^{(4,\{3 \rightarrow 1\})}, v_1), (q_2^{(4,\{3 \rightarrow 2\})}, v_2), \ldots, (q_k^{(4,\{3 \rightarrow k\})}, v_k), (q_{k+1,i}^{(7,\emptyset)}, v_1)]
$$

A(TR^L) = 1



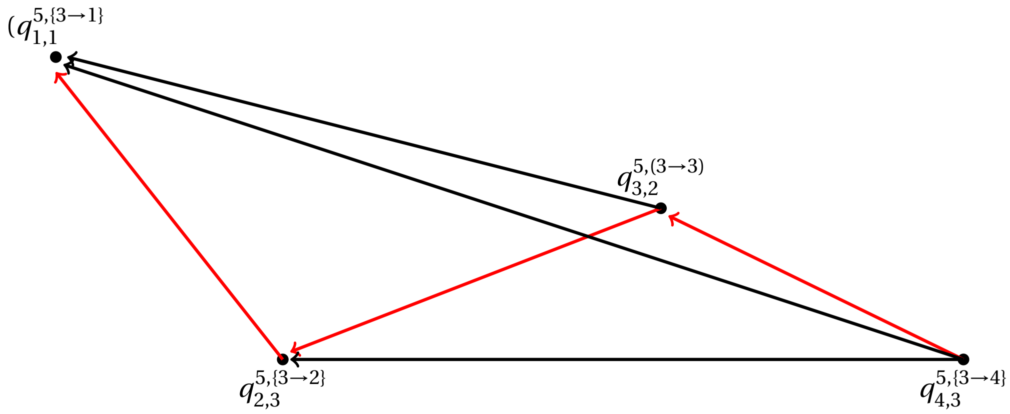
Example 3.3 (Multi-Round Algorithm)**.**

$$
MR^H \triangleq \begin{array}{l}
[j \leftarrow 0]^1; \\
[I \leftarrow []]^2; \\
\mathtt{while}\Big([j \leq k]^3, \\
[p \leftarrow \mathtt{uniform(0,1)}]^4; \\
[a \leftarrow q_j(p, I)]^5; \\
[I \leftarrow \mathsf{update}\ (\ I\ , (a,p))]^6; \\
[j \leftarrow j+1]^7\Big);
\end{array}
\quad \Rightarrow \quad
MR^L \triangleq \begin{array}{l}
[j \leftarrow 0]^1; \\
[I \leftarrow []]^2; \\
\mathtt{while}\Big([j \leq k]^3, \\
[p \leftarrow 0]^4; \\
\left[\mathtt{switch}\left(I, x \begin{pmatrix} [] \rightarrow q_{j,1}, \\ \cdots \\ [1,2,3,\cdots,n] \rightarrow q_{j,n!} \end{pmatrix}\right)\right]^5 \\
[I \leftarrow \mathsf{update}\ (\ I\ , (a,p))]^6; \\
[j \leftarrow j+1]^7\Big);
\end{array}
$$

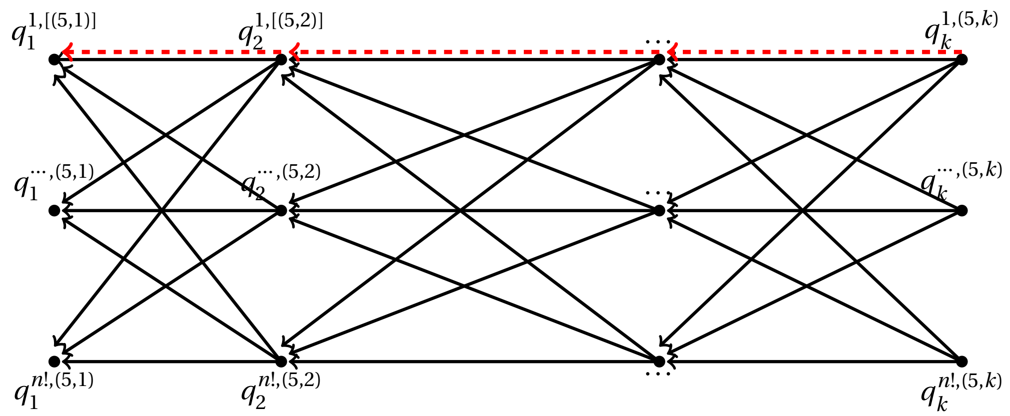
Let $k = 4$, given a specific database D, we have $\langle \emptyset, MR^L, D, [], \emptyset \rangle \rightarrow \langle m, \mathtt{skip}, D, t, w \rangle$ and the trace as:

$$
t = \Big[(q_{1,1}^{5,\{3\rightarrow 1\}}, v_1), (q_{2,3}^{5,\{3\rightarrow 2\}}, v_2), (q_{3,2}^{5,\{3\rightarrow 3\}}, v_3)(q_{4,3}^{5,\{3\rightarrow 4\}}, v_4)\Big]
$$

$A(TR^L)$ = 3



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



4 Towards Probability

4.1 Syntax and Semantics

Syntax.

$$
\begin{array}{llll}
\text{Arithmatic Operators} & *_a & ::= & + \mid - \mid \times \mid \div \\
\text{Boolean Operators} & *_b & ::= & \vee \mid \wedge \\
\text{Relational Operators} & *_r & ::= & < \mid \leq \mid = \\
\text{Label} & l & & \\
\text{While Map} & w & \in & \text{Label} \times \mathbb{N} \triangleq w + l \mid w \setminus l \mid w \oplus_\rho w \\
\text{AExpr} & a & ::= & n \mid x \mid a *_a a \mid [] \mid [a_0, \ldots, a_i] \mid a \times a \\
\text{BExpr} & b & ::= & \texttt{true} \mid \texttt{false} \mid \neg b \mid b *_b b \mid a *_r a \\
\text{Deterministic Expr} & e_d & ::= & a \mid b \\
\text{Randomized AExpr} & a_r & ::= & x_r \mid n \mid a_r *_a a_r \\
\text{Randomized BExpr} & b_r & ::= & \neg b_r \mid b_r *_b b_r \mid a_r *_r a_r \\
\text{Randomized Expr} & e_r & ::= & e_d \mid a_r \mid b_r \\
\text{Command} & C & ::= & [x \leftarrow e_d]^l \mid [x_r \leftarrow e_r]^l \mid [x \leftarrow q]^l \mid [x_r \leftarrow \texttt{uniform}]^l \mid [x_r \leftarrow \texttt{bernoulli}]^l \\
& & & \mid P; P \mid \texttt{if}_D([b]^l, P_1, P_2) \mid \texttt{if}_R([b_r]^l, P_1, P_2) \mid \texttt{while}([b]^l, P) \mid [\texttt{skip}]^l \\
& & & \mid \texttt{unfold}([b]^l, P) \ \mid [\texttt{switch}\,(e, x, (v_i \to q_i))]^l \mid \texttt{loop}\ [v_N]^l\ (f)\ \texttt{do}\ P \\
\text{Memory} & m & ::= & [] \mid m[x^l \to v] \\
\text{Trace} & t & ::= & [] \mid [(q, v)^{(l,w)}] \mid t ++ t \mid t \oplus_\rho t
\end{array}
$$

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

Semantics We have a countable set RV of random variables($x_r \in \mathsf{RV}$), a countable set Val of values. For any subset $\mathsf{S} \subseteq \mathsf{RV}$, we denote $\mathsf{RanM[S]} \triangleq \mathsf{S} \to \mathsf{Val}$. We let DV as a countable set of deterministic variables (x) and the deterministic memory $\mathsf{DetM} \triangleq \mathsf{DV} \to \mathsf{Val}$. Distribution over A as $D(A)$. The *distribution unit* unit : $A \to D(A)$. The *distribution bind* bind : $D(A) \to (A \to D(B)) \to D(B)$. $[\![C]\!]$: $(\mathsf{DetM} \times D(\mathsf{RandM}) \times \mathsf{Trace} \times \mathsf{WhileMap} \times \mathsf{DB}) \to (\mathsf{DetM} \times D(\mathsf{RandM}) \times \mathsf{Trace} \times \mathsf{WhileMap} \times \mathsf{DB})$

$$\langle(\sigma, \mu_1, t_1, w_1, D)\rangle \oplus_\rho \langle(\sigma, \mu_2, t_2, w_2, D)\rangle \triangleq \langle\sigma, \mu_1 \oplus_\rho \mu_2, t_1 \oplus_\rho t_2, w_1 \oplus_\rho w_2, D\rangle$$

$$(t_1 \oplus_\rho t_2) ++ t_3 \triangleq (t_1 ++ t_3) \oplus_\rho (t_2 ++ t_3)$$

$$t_3 ++ (t_1 \oplus_\rho t_2) \triangleq (t_3 ++ t_1) \oplus_\rho (t_3 ++ t_2)$$

$$(w_1 \oplus_\rho w_2) + l \triangleq (w_1 + l) \oplus_\rho (w_2 + l)$$

$$(w_1 \oplus_\rho w_2) \setminus l \triangleq (w_1 \setminus l) \oplus_\rho (w_2 \setminus l)$$

$$
\begin{array}{rcl}
\llbracket [\texttt{skip}]^{l} \rrbracket (\sigma, \mu, t, w, D) & \triangleq & (\sigma, \mu, t, w, D) \\
\llbracket [x \leftarrow e_d]^{l} \rrbracket (\sigma, \mu, t, w, D) & \triangleq & (\sigma[x \to \llbracket e_d \rrbracket (\sigma)], \mu, t, w, D) \\
\llbracket [x_r \leftarrow e_r]^{l} \rrbracket (\sigma, \mu, t, w, D) & \triangleq & (\sigma, bind(\mu, m \to unit(m[x_r \to \llbracket e_r \rrbracket (\sigma, m)])), t, w, D) \\
\llbracket [x \leftarrow q]^{l} \rrbracket (\sigma, \mu, t, w, D) & \triangleq & (\sigma[x \to v], \mu, t ++ [(q, v)]^{(l,w)}, w, D) \qquad : v = q(D) \\
\llbracket [x_r \leftarrow \texttt{uniform}]^{l} \rrbracket (\sigma, \mu, t, w, D) & \triangleq & (\sigma, bind(\mu, m \to bind(\texttt{uniform}, u \to m[x_r \to u])), t, w, D) \\
\llbracket [x_r \leftarrow \texttt{bernoulli}]^{l} \rrbracket (\sigma, \mu, t, w, D) & \triangleq & (\sigma, bind(\mu, m \to bind(\texttt{bernoulli}, u \to m[x_r \to u])), t, w, D) \\
\llbracket P; P' \rrbracket (\sigma, \mu, t, w, D) & \triangleq & \llbracket P' \rrbracket (\llbracket P \rrbracket \sigma, \mu, t, w) \\
\llbracket \texttt{if}_D([b]^{l}, P_1, P_2) \rrbracket (\sigma, \mu, t, w, D) & \triangleq & \begin{cases} \llbracket P_1 \rrbracket (\sigma, \mu, t, w, D) & : \llbracket b \rrbracket (\sigma) = \texttt{true} \\ \llbracket P_1 \rrbracket (\sigma, \mu, t, w, D) & : \llbracket b \rrbracket (\sigma) = \texttt{false} \end{cases} \\
\llbracket \texttt{if}_R([b_r]^{l}, P_1, P_2) \rrbracket (\sigma, \mu, t, w, D) & \triangleq & \begin{cases} \llbracket P_1 \rrbracket (\sigma, \mu | \llbracket b_r \rrbracket \sigma = \texttt{true}, t, w, D) \oplus_{\rho} \llbracket P_2 \rrbracket (\rho, \mu | \llbracket b_r \rrbracket \sigma = \texttt{false}, t, w, \\ \llbracket P_1 \rrbracket (\sigma, \mu | \llbracket b_r \rrbracket \sigma = \texttt{true}, t, w, D) \qquad \rho = 1 \\ \llbracket P_2 \rrbracket (\sigma, \mu | \llbracket b_r \rrbracket \sigma = \texttt{false}, t, w, D) \qquad \rho = 0 \end{cases} \\
& & \text{where} \quad \rho = \mu(\llbracket b_r \rrbracket \sigma = \texttt{true}) \\
\llbracket \texttt{while}([b]^{l}, P) \rrbracket (\sigma, \mu, t, w, D) & \triangleq & \llbracket \texttt{unfold}([b]^{l}, \texttt{while}([b]^{l}, P)) \rrbracket (\sigma, \mu, t, w, D) \\
\llbracket \texttt{unfold}([b]^{l}, P) \rrbracket (\sigma, \mu, t, w, D) & \triangleq & \begin{cases} \llbracket P \rrbracket (\sigma, \mu, t, w + l, D) & : \llbracket b \rrbracket (\sigma) = \texttt{true} \\ \llbracket \texttt{skip} \rrbracket (\sigma, \mu, t, w - l, D) & : \llbracket b \rrbracket (\sigma) = \texttt{false} \end{cases} \\
\llbracket [\texttt{switch}\,(e, x, (v_i \to q_i))]^{l} \rrbracket (\sigma, \mu, t, w, D) & \triangleq & \llbracket [x \leftarrow q_1]^{l} \rrbracket (\sigma, \mu, t, w, D) \qquad : v_1 = \llbracket e \rrbracket (\sigma) \\
\llbracket \texttt{loop}\ [e_N]^{l}\ (f)\ \texttt{do}\ P \rrbracket (\sigma, \mu, t, w, D) & \triangleq & \begin{cases} \llbracket f; P; \texttt{loop}\ [e_N - 1]^{l}\ (f)\ \texttt{do}\ P \rrbracket (\sigma, \mu, t, w + l, D) & : \llbracket e_N \rrbracket (\sigma) > 0 \\ \llbracket \texttt{skip} \rrbracket (\sigma, \mu, t, w - l, D) & : \llbracket e_N \rrbracket (\sigma) = 0 \end{cases}
\end{array}
$$

Figure 1: Semantics of programs

$$
\begin{array}{lcl}
\begin{array}{l}
[j \leftarrow 0]^{1}; \\
[I \leftarrow []]^{2}; \\
\texttt{while}\Big([j \leq k]^{3}, \\
[p \leftarrow 0]^{4}; \\
\left[\texttt{switch}\left(I, x \begin{pmatrix} [] \to q_{j,1}, \\ \cdots \\ [1,2,3,\cdots,n] \to q_{j,n!} \end{pmatrix}\right)\right]^{5} \\
[I \leftarrow \text{update}\ (\ I\ , (x, p))]^{6}; \\
[j \leftarrow j + 1]^{7}\Big);
\end{array}
& \Rightarrow \ SSA \triangleq &
\begin{array}{l}
[j \leftarrow 0]^{1}; \\
[I_1 \leftarrow []]^{2}; \\
\texttt{while}\Big([j \leq k]^{3}, \phi : I_2 = f(I_1, I_3) \\
[p \leftarrow 0]^{4}; \\
\left[\texttt{switch}\left(I_2, x \begin{pmatrix} [] \to q_{j,1}, \\ \cdots \\ [1,2,3,\cdots,n] \to q_{j,n!} \end{pmatrix}\right)\right]^{5} \\
[I_3 \leftarrow \text{update}\ (\ I_2\ , (x, p))]^{6}; \\
[j \leftarrow j + 1]^{7}\Big);
\end{array}
\end{array}
$$

4.2 Extending Adaptivity onto Probabilistic Program

Definition 7. *Dependency Forest.*
Given a program P, a database D, dependency forest $F(P,D) \triangleq \{G_1, G_2, \cdots, G_m\}$, $G_k = (V_k, E_k)$ is defined as:
$V_k = \{q^{l,w} | \forall m, w. \langle \emptyset, P, D, [], \emptyset \rangle \to \langle m, \texttt{skip}, D, T, \emptyset \rangle \wedge q^{l,w} \in \pi_k(T)\}$;
$E_k = \left\{(q_i^{l,w}, q_j^{l',w'}) \;\middle|\; \neg\mathsf{IND}(q_i^{l,w}, q_j^{l',w'}, P) \wedge \mathsf{To}(q_j^{l',w'}, q_i^{l,w})\right\}$,
where $\pi_k(T) = t_k$ s.t. $T = t_1 \oplus_{\rho_1} t_2 \oplus_{\rho_2} \cdots \oplus_{m-1} t_m$ and there is no $\oplus$ in t_i.

Definition 8. *Dependency Graph.*
Given a program P, a database D, dependency graph $G(P,D) \triangleq (V, E)$ is defined as:

$V = \bigcup\{V_k | G_k \in F(P, D)\}$;
$E_k = \{(q_i, q_j) \mid (q_i, q_j) \in E_k, G_k \in F(P, D)\}$,

Definition 9. *Adaptivity.*
Given a program P and for all the database D in a set of DBS of databases, the total dependency graph G is the combination of all the dependent graphs over every single database $G(P, D) = (V, E)$, the adaptivity of the program is defined as $A(P)$, s.t.: for every pair (i, j) let $p_{(i,j)}$ be the longest path starting from $q_i^{l,w}$ to $q_j^{l',w'}$,

$$A(P) = \max_{q_i^{l,w}, q_j^{l',w'} \in V} \{l_i \mid l_i = |p_(i, j)|\}$$

5 Analysis of Program Adaptivity

We have the judgment of the form : $\vdash^{c_1, c_2}_{M,V} P : \Phi \implies \Psi$. Our grade is a combination of a matrix M, used to track the dependency of variables appeared in the statement S, and a vector V indicating the variables associated with results from queries q. The size of the matrix M is $L \times L$, and vector V of size N, where L is the total size of variables needed in the program, which is fixed per program. The superscript c_1, c_2 is a counter specifying the range of "living" or "active" variables in the matrix and vector. c_1 is the starting line (and column) in the matrix where the new generated variables in program P starts to show up. Likewise, c_2 states the ending position of active range by P. The new generated variables will be treated carefully when we handle recursive statement, in our case, the loop statement.

We assume the program P is in the static single assignment form. That is to say, $x \leftarrow e_1; x \leftarrow e_2$ will be rewritten as $x_1 \leftarrow e_1; x_2 \leftarrow e_2$. And the if condition $\texttt{if}\ e_b$ then $x \leftarrow e_1$ else $x \leftarrow e_2$ will look like $\texttt{if}\ e_b$ then $x_1 \leftarrow e_1$ else $x_2 \leftarrow e_2$. As we have seen, ssa provides unique variables, and these newly generated variables will be recorded in the matrix M. Worth to mention, c_1, c_2 can be used to track the exact location of newly generated variables. For example, the assignment statement $x \leftarrow e$ or $x \leftarrow q$ holds $c_2 = c_1 + 1$, telling us the variable x is at the c_1th line(column) of the matrix. As we can notice, the loop increases the matrix by $N \times a$ where N is the number of rounds of the loop and a is the size of the variables generated in the loop body P.

We will have a global map, which maps the variable name to the position in the vector. We call it $G : VAR \to \mathbb{N}$,

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

$$P = \begin{array}{c} x_1 \leftarrow q; \\ x_2 \leftarrow x_1 + 1; \\ x_3 \leftarrow x_2 + 2 \end{array} \qquad M = \begin{array}{ccc} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{array}, V = \begin{array}{c} 1 \\ 0 \\ 0 \end{array}$$

Analysis Rules. $\boxed{\Gamma \vdash^{c_1,c_2}_{M,V} P : \Phi \implies \Psi}$

$$\frac{M = L(x) * (R(e) + \Gamma)}{\Gamma \vdash^{(c,c+1)}_{M,V_\emptyset} x \leftarrow e \colon \Phi \implies \Psi} \qquad \frac{M = L(x) * (R(\emptyset) + \Gamma) \qquad V = L(x)}{\Gamma \vdash^{(c,c+1)}_{M,V} x \leftarrow q \colon \Phi \implies \Psi}$$

$$\frac{\Gamma + R(e_b) \vdash^{(c_1,c_2)}_{M_1,V_1} P_1 : \Phi \implies \Psi \qquad \Gamma + R(e_b) \vdash^{(c_2,c_3)}_{M_2,V_2} P_2 : \Phi \implies \Psi}{\Gamma \vdash^{(c_1,c_3)}_{M_1 \uplus M_2, V_1 \uplus V_2} \texttt{if } e_b \text{ then } P_1 \text{ else } P_2 : \Phi \implies \Psi}$$

$$\frac{\Gamma \vdash^{(c_1,c_2)}_{M_1,V_1} P_1 : \Phi \implies \Psi' \qquad \Gamma \vdash^{(c_2,c_3)}_{M_2,V_2} P_2 : \Psi' \implies \Psi}{\Gamma \vdash^{(c_1,c_3)}_{M_1 \cdot M_2, V_1 \uplus V_2} P_1; P_2 : \Phi \implies \Psi}$$

$$\frac{\forall 0 \leq z < N . \Gamma \vdash^{(c,c+a)}_{M,V} f; P : \{\Phi \wedge e_N = z+1\} \implies \{\Phi \wedge e_N = z\}}{\Gamma \vdash^{(c,c+N*a)}_{M^N_{c,a}(f), V^N_{c,a}} \texttt{loop } e_N \ (f) \texttt{ do } P : \{\Phi \wedge e_N = N\} \implies \{\Phi \wedge e_N = 0\}}$$

$$\frac{\Gamma + R(e) \vdash^{(c_1,c_1+1)}_{M_i,V_i} x_i \leftarrow q_i \qquad i \in \{1, \ldots, N\}}{\Gamma \vdash^{(c_1,c_1+N)}_{\sum_{i=0}^N M_i, \sum_{i=0}^N V_i} \texttt{switch}\,(e, x, (v_i \rightarrow q_i))}$$

$$V_1 \uplus V_2 \quad := \quad \begin{cases} 1 & (V_1[i] = 1 \vee V_2[i] = 1) \wedge i = 1, \cdots, N \wedge |V_1| = |V_2| \\ 0 & o.w. \end{cases}$$

$$M_1 \uplus M_2 \quad := \quad \begin{cases} 1 & (M_1[i][j] = 1 \vee M_2[i][j] = 1) \wedge i, j = 1, \cdots, N \wedge |M_1| = |M_2| \\ 0 & (M_1[i][j] = 0 \wedge M_2[i,j] = 0) \wedge i, j = 1, \cdots, N \wedge |M_1| = |M_2| \\ \bot & o.w. \end{cases}$$

$$V^N_{(c,a)} \quad := \quad \begin{cases} V[c + i * a, c + (i+1) * a - 1] = V[c, c + a - 1] & i = 1, \cdots, N-1 \\ \bot & o.w. \end{cases}$$

$$M^N_{(c,a)}(f) \quad := \quad \begin{cases} M[c + i * a, c + (i+1) * a - 1][c + i * a, c + (i+1) * a - 1] & \\ = M[c, c + a - 1][c, c + a - 1] & i = 1, \cdots, N-1 \\ M[c + i * a, c + (i+1) * a - 1][0, c + (i) * a - 1] = 0 & i = 1, \cdots, N-1 \\ M[0, c + i * a - 1][c + i * a, c + (i+1) * a - 1] & \\ = M[0, c + (i-1) * a - 1][c + (i-1) * a, c + (i) * a - 1] & i = 1, \cdots, N-1 \\ M[l][k] = 1 & \begin{array}{l} l \in [c + (i-1) * a, c + (i) * a - 1], \\ k \in [c + i * a, c + (i+1) * a - 1] \\ \wedge i = 1, \cdots, N \wedge, l = G(x_l) \\ \wedge k = G(x_k) \wedge x_l = f(\cdots, x_k, \cdots) \end{array} \\ \bot & o.w. \end{cases}$$

1	...	c-1	c	...	c+a-1	c+a	...	c+2a-1	...	c+N*a-1	c+N*a	...
...			0	0	0	0	0	0				
c-1			0	0	0	0	0	0				
c						0	0	0				
...						0	0	0				
c+a-1						0	0	0				
c+a												
...				f								
c+2a-1												
...												
c+N*a-1												
c+N*a												
...												

Definition 10. *Dependency Graph.*
Given a program P, a database D, dependency graph $G(P,D) = (V,E)$ is defined as:
$V = \{q_i^{l,w} | \forall \sigma, \sigma', \mu, \mu', w, w', t, t'. [\![P]\!](\sigma,\mu,t,w,D) \triangleq (\sigma',\mu',t',w',D) \land q_i^{l,w} \in (t'-t)\}$.
$E = \left\{(q_i^{l,w}, q_j^{l',w'}) \;\middle|\; \neg\mathsf{IND}(q_i^{l,w}, q_j^{l',w'}, P) \land \mathsf{To}(q_j^{l',w'}, q_i^{l,w})\right\}$.

Definition 11. *Two queries q_i and q_j in a program c are independent, $\mathsf{IND}(q_i^{l_1}, q_j^{l_2}, P)$.*

$$\forall m, D.\Big([\![P]\!](\sigma,\mu,t,w,D) \triangleq (\sigma',\mu',t',w',D)$$
$$\land\Big((q_i^{l_1}, v_i) \in t' \land (q_j^{l_2}, v_j) \in t' \implies \forall v \in \mathit{Codom}(q_i^{l_1}).\Big([\![P[v/q_i]]\!](\sigma,\mu,t,w,D) \triangleq (\sigma',\mu',t'',w',D) \land (q_j^{l_2}, v_j) \in t''\Big)\Big)$$
$$\land\Big((q_i^{l_1}, v_i) \in t \land (q_j^{l_2}, v_j) \notin t \implies \forall v \in \mathit{Codom}(q_i^{l_1}).\Big([\![P[v/q_i]]\!](\sigma,\mu,t,w,D) \triangleq (\sigma',\mu',t'',w',D) \land (q_j^{l_2}, v_j) \notin t''\Big)\Big)\Big).$$

Definition 12. *Adaptivity.*
Given a program P and for all the database D in a set of DBS of databases, the total dependency graph G is the combination of all the dependent graphs over every single database $G(P,D) = (V,E)$, the adaptivity of the program is defined as $A(P)$, s.t.: for every pair (i,j) let $p_{(i,j)}$ be the longest path starting from $q_i^{l,w}$ to $q_j^{l',w'}$,

$$A(P) = \max_{q_i^{l,w}, q_j^{l',w'} \in V} \{l_i \mid l_i = |p(i,j)|\}$$

Definition 13 (Adapt)**.** *Given a program P s.t. $\cdot \vdash_{M,V}^{c_1,c_2} P : \Phi \implies \Psi$, there exists 1 and only one Graph $G(M,V) = (Nodes, Edges, Weights)$ defined as:*
$Nodes = \{i | i \in V\}$
$Edges = \{(i,j) | M[i][j] \geq 1\}$

$Weights = \{1 | i \in V \wedge V[i] = 1\} \cup \{-1 | \in V \wedge V[i] = 0\}$
Adaptivity of the program defined according to the logic is as:

$$Adapt(M, V) := \max_{k,l \in Nodes} \{i | V[i] = 1 \wedge i \in p(k,l)\},$$

where $p(k,l)$ is the longest weighted path in graph $G(M,V)$ starting from k to l.

Definition 14 (Subgraph)**.** *Given two graphs $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$, $G_1 \subseteq G_2$ iff:*
1. $V_1 \subseteq V_2$;
2. $\forall (q_i, q_j) \in E_1$, $\exists$ a path in G_2 from q_i to q_j.

Theorem 5.1 (Soundness)**.** *Given a program P, Γ, μ, c_1, c_2 and σ s.t. $FreeVar(P) \subseteq dom(\sigma) \cup dom(\mu) \wedge \Gamma \subseteq FreeVar(P) \wedge \Gamma \vdash^{c_1,c_2}_{M,V} P : \Phi \implies \Psi$, for all database D, $(\sigma, \mu) \models t$ s.t. $[\![P]\!](\sigma, \mu, t, w, D) \triangleq (\sigma', \mu', t', w', D)$, then*

$$A(P) \leq Adapt(M, V)$$

Lemma 1 (Subgraph)**.** *Given a program P, Γ, μ, c_1, c_2 and σ s.t. $FreeVar(P) \subseteq dom(\sigma) \cup dom(\mu) \wedge \Gamma \subseteq FreeVar(P) \wedge \Gamma \vdash^{c_1,c_2}_{M,V} P : \Phi \implies \Psi$, for all database D, $(\sigma, \mu) \models t$ s.t. $[\![P]\!](\sigma, \mu, t, w, D) \triangleq (\sigma', \mu', t', w', D)$, then*

$$G(P, D) \subseteq G(M, V)$$