Program Adaptivity Analysis

Contents

1 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 control flow.

1.1 Syntax and Semantics

Syntax.

$$
\begin{array}{llll}
\text{Arithmatic Operators} & *_a & ::= & + \mid - \mid \times \mid \div \\
\text{Boolean Operators} & *_b & & \\
\text{Relational Operators} & *_r & ::= & < \mid \leq \mid = \\
\text{Label} & l & & \\
\text{While Map} & w & \in & \text{Label} \times \mathbb{N} \\
\text{AExpr} & a & ::= & n \mid x \mid a *_a a \mid [] \mid [a_0, \dots, a_i] \\
\text{BExpr} & b & ::= & \texttt{true} \mid \texttt{false} \mid \neg b \mid b *_b b \mid a *_r a \\
\text{Expr} & e & ::= & a \mid b \\
\text{Command} & c & ::= & [x \leftarrow e]^l \mid [x \leftarrow q]^l \mid [\texttt{switch}\,(e, x, (v_i \rightarrow q_i))]^l \mid \texttt{loop}\ [e_N]^l\ (f)\ \texttt{do}\ c \\
& & & \mid c;c \mid \texttt{if}([b]^l, c_1, c_2) \mid \texttt{while}([b]^l, c) \mid [\texttt{skip}]^l \\
\text{Query} & q & ::= & q_0, q_1, \dots, q_n \\
\text{Memory} & m & ::= & [] \mid m[x^l \to v] \\
\text{Trace} & t & ::= & [] \mid [(q^{(l,w)}, v)] \mid t ++ t
\end{array}
$$

Operational Semantics.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\begin{array}{lll} w \setminus l & = w & l \notin Keys(w) \\ & = w_l & Otherwise \\ w + l & = w[l \to 0] & l \notin Keys(w) \\ & w_l[l \to w(l) + 1] & Otherwise \end{array}$$

1.2 Adaptivity of Programs in Low level language

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

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

$mk(w_i) = MinKey(w_i)$

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

Definition 2 (Query Direction)**.** *Direction between two queries.*
$\forall q_1, q_2, l_1, l_2, w_1, w_2$. $(q_1^{l_1,w_1})\,\mathsf{TO}\,(q_2^{l_2,w_2})$, 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, $\mathsf{IND}(q_i^{(l_1,w_1)}, q_j^{(l_2,w_2)}, c) =.$*

$$
\begin{array}{l}
\forall m, D. \langle m, c, [] \rangle \to^* \langle m_1, c_1, t_1 \rangle \to \langle m_2, c_2, t_1 ++ [(q_i^{(l_1,w_1)}, v_i)] \rangle \to^* \langle m_3, \texttt{skip}, t_3 \rangle \land x = FindVar(q_i^{(l_1,w_1)}, c) \land \\
w_1 = \emptyset \implies \Big(\\
\Big((q_i^{(l_1,w_1)}, v_i) \in t_3 \land (q_j^{(l_2,w_2)}, v_j) \in t_3 \implies \forall v \in codomain(q_i^{l_1}). \Big(\langle m, c[v/q_i], [] \rangle \to^* \langle m', \texttt{skip}, t' \rangle \land (q_j^{l_2}, v_j) \in t' \Big) \Big) \\
\land \Big((q_i^{l_1}, v_i) \in t_3 \land (q_j^{l_2}, v_j) \notin t_3 \implies \forall v \in codomain(q_i^{l_1}). \Big(\langle m, c[v/q_i], [] \rangle \to^* \langle m', \texttt{skip}, t' \rangle \land (q_j^{l_2}, v_j) \notin t' \Big) \Big) \Big) \\
\land w_1 \neq \emptyset \implies \Big(\\
((q_i^{(l_1,w_1)}, v_i) \in t_3 \land (q_j^{(l_2,w_2)}, v_j) \in t_3 \implies \\
\forall v \in codomain(q_i^{l_1}). \Big(\langle m_2[v/x], c_2, t_1 ++ [(q_i^{(l_1,w_1)}, v_i)] \rangle \to^* \langle m', \texttt{skip}, t' \rangle \land (q_j^{l_2}, v_j) \in t' \Big)) \\
\land ((q_i^{l_1}, v_i) \in t \land (q_j^{l_2}, v_j) \notin t \implies \\
\forall v \in codomain(q_i^{l_1}). \Big(\langle m_2[v/x], c_2, t_1 ++ [(q_i^{(l_1,w_1)}, v_i)] \rangle \to^* \langle m', \texttt{skip}, t' \rangle \land (q_j^{l_2}, v_j) \notin t' \Big) \Big) \Big).
\end{array}
$$

In following examples:

$$
c_1 \triangleq \begin{array}{l}
[x \leftarrow q_1]^1; \\
\texttt{if } [x]^2 \\
\text{then } [y \leftarrow q_2]^3 \\
\text{else } [y \leftarrow 0]^4; \\
\texttt{if } [x]^5 \\
\text{then } [y \leftarrow 0]^6 \\
\text{else } [y \leftarrow q_2]^7;
\end{array}
\quad
c_2 \triangleq \begin{array}{l}
[x \leftarrow q_1]^1; \\
\texttt{if } [x]^2 \\
\text{then } [y \leftarrow q_2]^3 \\
\text{else } [y \leftarrow q_3]^4; \\
[z \leftarrow q_2]^5;
\end{array}
\quad
c_3 \triangleq \begin{array}{l}
[x \leftarrow q_1]^1; \\
\texttt{if } [x]^2 \\
\text{then } [y \leftarrow q_3]^3 \\
\text{else } [y \leftarrow q_4]^4; \\
[z \leftarrow q_2]^5;
\end{array}
\quad
c_4 \triangleq \begin{array}{l}
[x \leftarrow q_1]^1; \\
[y \leftarrow q_2]^2; \\
\texttt{if } [x + y = 5]^3 \\
\text{then } [z \leftarrow q_3]^4 \\
\text{else } [\texttt{skip}]^5; \\
[w \leftarrow q_4]^6;
\end{array}
$$

We have the dependency as:
In program c_1: $\mathsf{IND}(q_2^3, q_2^7, c_1)$,
In program c_2: $\mathsf{IND}(q_1^1, q_2^5, c_2)$, $\mathsf{IND}(q_2^3, q_3^4, c_2)$,
In program c_3: $\mathsf{IND}(q_1^1, q_2^5, c_3)$, $\mathsf{IND}(q_3^3, q_4^4, c_3)$,
In program c_4: $\mathsf{IND}(q_1^1, q_2^2, c_4)$, $\mathsf{IND}(q_3^3, q_4^4, c_4)$, $\mathsf{IND}(q_1^1, q_4^4, c_4)$, $\mathsf{IND}(q_2^2, q_4^4, c_4)$,
In following examples containing `while` loop:

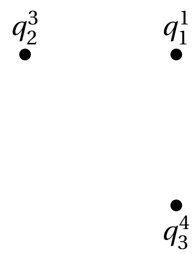
$$
c_1 \triangleq \begin{array}{l} [x \leftarrow q_1]^1; \\ \texttt{while}\Big([x \leq 100]^2, \\ [x \leftarrow q_2 + x]^3\Big); \end{array}
\qquad
c_2 \triangleq \begin{array}{l} [x \leftarrow q_1]^1; \\ \texttt{while}\Big([x \leq 100]^2, \\ [y \leftarrow q_2]^3; \\ [z \leftarrow q_3]^4; \\ [x \leftarrow y + z + x]^5\Big); \end{array}
\qquad
c_3 \triangleq \begin{array}{l} [x \leftarrow q_1]^1; \\ \texttt{while}\Big([x \leq 100]^2, \\ [y \leftarrow q_2 + x]^3; \\ \texttt{while}\Big([y \leq 50]^4, \\ [z \leftarrow q_3]^5; \\ [y \leftarrow y + z]^6;\Big) \\ [x \leftarrow y + x]^7\Big); \end{array}
$$

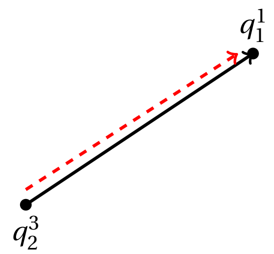
we have the dependency as:
In program c_1: No independent queries.
In program c_2: $\mathsf{IND}(q_2^{(3,l)}, q_3^{(4,l)}, c_2)$ for all l.
In program c_3: No independent queries.

Dependency between multiple queries

Example 1.1. ***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 c_1 and c_2 as follows:

$$
c_1 \triangleq \begin{array}{l} [w \leftarrow 100]^0; \\ [x \leftarrow q_1]^1; \\ \texttt{if}([w > 1]^2) \\ \quad then\ [y \leftarrow q_2]^3; \\ \quad else\ [z \leftarrow q_3]^4 \end{array}
\qquad
c_2 \triangleq \begin{array}{l} [x \leftarrow q_1]^1; \\ \texttt{if}([x > 1]^2) \\ \quad then\ [y \leftarrow q_2]^3; \\ \quad else\ [z \leftarrow 10]^4 \end{array}
$$





2 High level Language

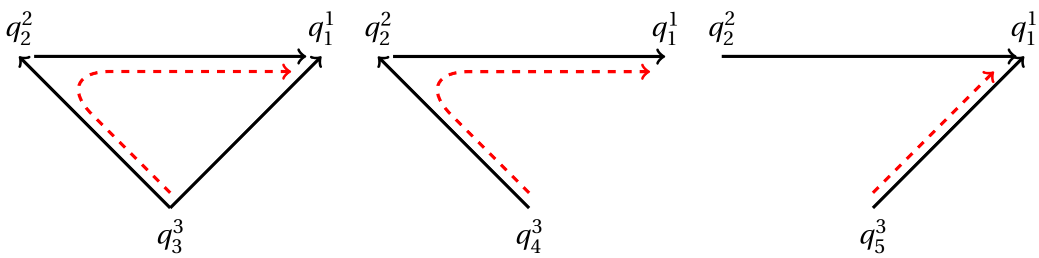
2.1 Syntax and Semantics

Syntax.

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

Example 2.1. ***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 c_1, c_2 *and* c_3 *as following:*

$$
c_1 \triangleq \begin{array}{l} [x_1 \leftarrow q_1]^1; \\ [x_2 \leftarrow q_2]^2; \\ [x_3 \leftarrow q_3]^3 \end{array}
\qquad
c_2 \triangleq \begin{array}{l} [x_1 \leftarrow q_1]^1; \\ [x_2 \leftarrow q_2]^2; \\ [x_3 \leftarrow q_4]^3 \end{array}
\qquad
c_3 \triangleq \begin{array}{l} [x_1 \leftarrow q_1]^1; \\ [x_2 \leftarrow q_2]^2; \\ [x_3 \leftarrow q_5]^3 \end{array}
$$



2.2 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.

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

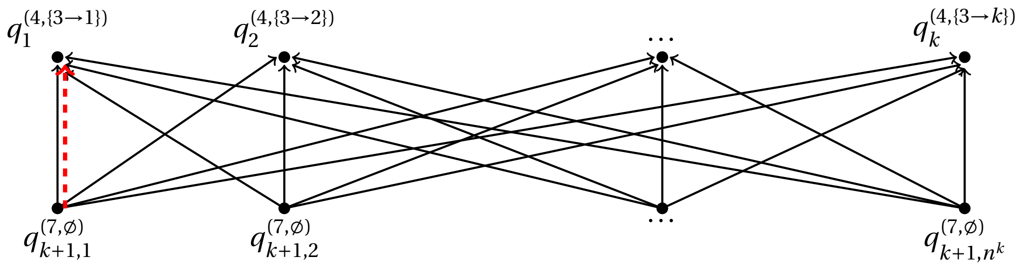
Example 2.2 (Two Round Algorithm)**.**

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

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

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

A(TR^{L}) = 1



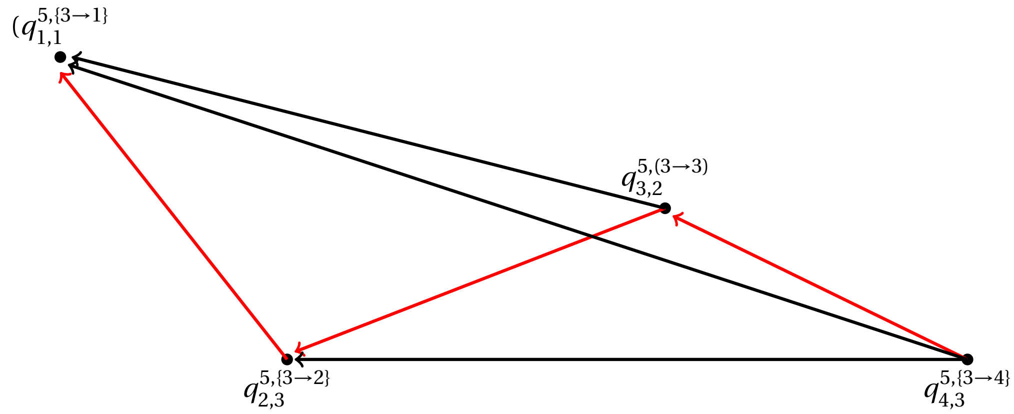
Example 2.3 (Multi-Round Algorithm)**.**

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

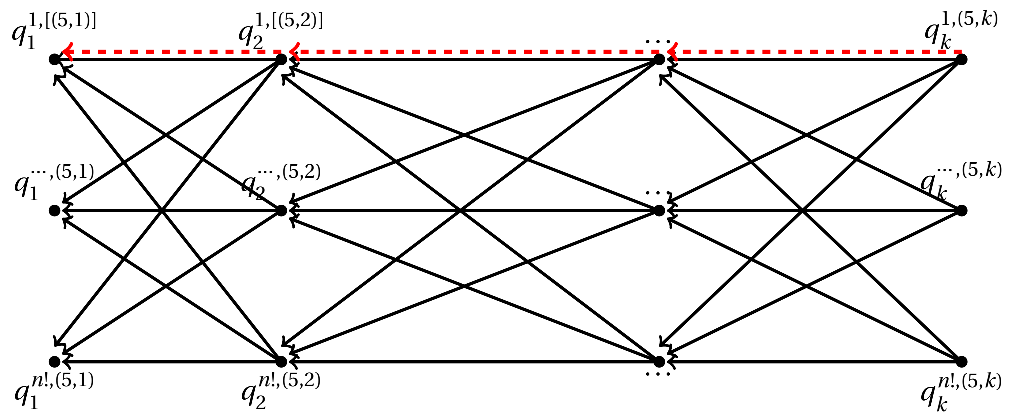
Let $k = 4$, given a specific database D, we have $\langle \emptyset, MR^L, D, [], \emptyset \rangle \rightarrow \langle m, \texttt{skip}, D, t, w \rangle$ and the trace as:

$$t = \left[(q_{1,1}^{5,\{3\rightarrow1\}}, v_1), (q_{2,3}^{5,\{3\rightarrow2\}}, v_2), (q_{3,2}^{5,\{3\rightarrow3\}}, v_3)(q_{4,3}^{5,\{3\rightarrow4\}}, v_4)\right]$$

$A(TR^L) = 3$



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



3 Towards Probability

3.1 Syntax and Semantics

Syntax.

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

$$w \setminus l = \begin{cases} w & l \notin Keys(w) \\ w_l & Otherwise \end{cases}$$

$$w + l = \begin{cases} w[l \to 0] & l \notin Keys(w) \\ w_l[l \to w(l)+1] & Otherwise \end{cases}$$

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 : $D(A) \to (A \to D(B)) \to D(B)$. $[\![C]\!] : (\mathsf{DetM} \times D(\mathsf{RandM}) \times [\mathsf{Trace}] \times [\mathsf{WhileMap}] \times \mathsf{DB}) \to (\mathsf{DetM} \times D(\mathsf{RandM}) \times [\mathsf{Trace}] \times [\mathsf{WhileMap}] \times \mathsf{DB})$. The upper case character T represents the list of traces and W represents the list of while map. The @ operator represents the list concatenation, and the $T.i$ represents the i^{th} element in the list T.

$$\langle(\sigma, \mu_1, T_1, W_1, D)\rangle \oplus_\rho \langle(\sigma, \mu_2, T_2, W_2, D)\rangle \triangleq \langle\sigma, \mu_1 \oplus_\rho \mu_2, T_1 @ T_2, W_1 @ W_2, D\rangle$$

$$T{+}{+}t \triangleq [T.i{+}{+}t]$$

$$W + l \triangleq [W.i + l]$$

$$W \setminus l \triangleq [W.i \setminus l]$$

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

Figure 1: Semantics of programs

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

3.2 Extending Adaptivity onto Probabilistic Program

Definition 4. *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:
$V_k = \{q^{l,w} | \forall m, w. \langle \emptyset, P, D, [], \emptyset \rangle \to \langle m, \mathtt{skip}, D, T, \emptyset \rangle \wedge q^{l,w} \in T.k\}$;
$E_k = \left\{(q_i^{l,w}, q_j^{l',w'}) \;\middle|\; \neg\mathsf{IND}(q_i^{l,w}, q_j^{l',w'}, P) \wedge \mathsf{To}(q_j^{l',w'}, q_i^{l,w})\right\}$,

Definition 5. *Dependency Graph.*
Given a program P, a database D, dependency graph $G(P,D) \triangleq (V,E)$ is defined as:
$V = \bigcup\{V_k | G_k \in F(P,D)\}$;
$E_k = \left\{(q_i, q_j) \;\middle|\; (q_i, q_j) \in E_k, G_k \in F(P,D)\right\}$,

Definition 6. *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 $q_j^{l',w'}$,

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

4 Analysis of Program Adaptivity

We have the anaylsis rules of the form : $\vdash_{M,V}^{i_1,i_2} C$. 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 i_1, i_2 is a counter specifying the range of "living" or "active" variables in the matrix and vector. i_1 is the starting line (and column) in the matrix where the new generated variables in program P starts to show up. Likewise, i_2 states the ending position of active range by C. The new generated variables will be treated carefully when we handle recursive statement, in our case, the loop statement.

We assume the program C 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, i_1, i_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 $i_2 = i_1 + 1$, telling us the variable x is at the i_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 C.

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 C.

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

Analysis Rules.

$$\begin{array}{ll}
\mathsf{Predict}(x \leftarrow e)(\Gamma, i) & = (L(x) * (R(e) + \Gamma), V, i+1) \\
\mathsf{Predict}(x \leftarrow q)(\Gamma, i) & = (L(x) * (R(\emptyset) + \Gamma), L(x), i+1) \\
\mathsf{Predict}(\texttt{if}\ e_b \text{ then } C_1 \text{ else } C_2)(\Gamma, i) & = \texttt{let}\ (M_1, v_1, i_1) = \mathsf{Predict}(C_1)(\Gamma + R(e_b), i)\ \texttt{in} \\
& \texttt{let}\ (M_2, v_2, i_2) = \mathsf{Predict}(C_2)(\Gamma + R(e_b), i_1)\ \texttt{in}\ (M_1 \uplus M_2, V_1 \uplus V_2, i_2) \\
\mathsf{Predict}(C_1; C_2)(\Gamma, i) & = \texttt{let}\ (M_1, v_1, i_1) = \mathsf{Predict}(C_1)(\Gamma + R(e_b), i)\ \texttt{in} \\
& \texttt{let}\ (M_2, v_2, i_2) = \mathsf{Predict}(C_2)(\Gamma + R(e_b), i_1)\ \texttt{in}\ (M_1 \cdot M_2, V_1 \uplus V_2, i_2)
\end{array}$$

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

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

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

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

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

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

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

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

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

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

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

Definition 8. *Two queries q_i and q_j in a program c are independent,* $\mathsf{IND}(q_i^{l_1}, q_j^{l_2}, P)$.

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

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 $q_j^{l',w'}$,

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

Definition 10 (Adapt)**.** *Given a program P s.t. $\cdot \vdash^{c_1,c_2}_{M,V} P : \Phi \implies \Psi$, there exists 1 and only one Graph $G(M,V) = (Nodes, Edges, Weights)$ defined as:*
$Nodes = \{i | i \in V\}$
$Edges = \{(i,j) | M[i][j] \geq 1\}$

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

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

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

Definition 11 (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 4.1 (Soundness)**.** *Given a program P, Γ, μ, c_1, c_2 and σ s.t. $FreeVar(P) \subseteq dom(\sigma) \cup dom(\mu) \land \Gamma \subseteq FreeVar(P) \land \Gamma \vdash^{c_1,c_2}_{M,V} P : \Phi \implies \Psi$, for all database D, $(\sigma, \mu) \vDash t$ s.t. $[\![P]\!](\sigma,\mu,t,w,D) \triangleq (\sigma', \mu', t', w', D)$, then*

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

Proof. By induction on the judgment $\Gamma \vdash^{c_1,c_2}_{M,V} P : \Phi \implies \Psi$.

- **Case:** $\dfrac{M = L(x) * (R(e) + \Gamma)}{\Gamma \vdash^{(c,c+1)}_{M,V_\emptyset} x \leftarrow e : \Phi \implies \Psi}$
  Given σ, μ, t and w, for arbitrary database D, by induction on e, we have the following semantic.

  $$[\![[x \leftarrow e_d]^l]\!](\sigma,\mu,t,w,D) \triangleq (\sigma[x \to [\![e_d]\!]\sigma], \mu, t, w, D)$$

  $$[\![[x_r \leftarrow e_r]^l]\!](\sigma,\mu,t,w,D) \triangleq (\sigma, bind(\mu, m \to unit(m[x_r \to [\![e_r]\!](\sigma,m)])), t, w, D)$$

  In both of the 2 cases, no newly added node in the trace t. Then we can derive that $A(P) = 0$.
  We also know $V_\emptyset = \emptyset$, i.e., $Adapt(M, V_\emptyset) = 0$.
  Since $0 \leq 0$, this case is proved.

- **Case:** $\dfrac{M = L(x) * (R(\emptyset) + \Gamma) \qquad V = L(x)}{\Gamma \vdash^{(c,c+1)}_{M,V} x \leftarrow q : \Phi \implies \Psi}$
  Given σ, μ, t and w, for arbitrary database D, we have the following semantic.

  $$[\![[x \leftarrow q]^l]\!](\sigma,\mu,t,w,D) \triangleq (\sigma[x \to v], \mu, t ++ [(q,v)]^{(l,w)}, w, D) \qquad : v = q(D)$$

  There is only one newly added node in the trace for all the possible database D. Follow the definition of $Adapt(M,V)$, we know that the claim holds.

- **Case:** $\dfrac{\Gamma + R(e_b) \vdash^{(c_1,c_2)}_{M_1,V_1} P_1 : \Phi \implies \Psi \qquad \Gamma + R(e_b) \vdash^{(c_2,c_3)}_{M_2,V_2} P_2 : \Phi \implies \Psi}{\Gamma \vdash^{(c_1,c_3)}_{M_1 \uplus M_2, V_1 \uplus V_2} \texttt{if } e_b \textbf{ then } P_1 \textbf{ else } P_2 : \Phi \implies \Psi}$
  The semantics depends on the evaluated value of the conditional.

  $$[\![\texttt{if}_D([b]^l, P_1, P_2)]\!](\sigma,\mu,t,w,D) \triangleq \begin{cases} [\![P_1]\!](\sigma,\mu,t,w,D) & : [\![b]\!](\sigma) = \texttt{true} \\ [\![P_2]\!](\sigma,\mu,t,w,D) & : [\![b]\!](\sigma) = \texttt{false} \end{cases}$$

  By induction hypothesis, we have:
  $A(P_1) \leq Adapt(M_1, V_1)$
  $A(P_2) \leq Adapt(M_2, V_2)$
  By definition of $A(P)$ and $Adapt(M,V)$, we have:
  $A(P) \leq \max(A(P_1), A(P_2)) \leq \max(Adapt(M_1,V_1), Adapt(M_2,V_2)) \leq Adapt(M_1 \uplus M_2, V_1 \uplus V_2)$.
  This case is proved.

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

  Given σ, μ, t and w, for arbitrary database D, we have the following semantic.

  $$[\![P_1; P_2]\!](\sigma,\mu,t,w,D) \triangleq [\![P_2]\!]([\![P_1]\!](\sigma,\mu,t,w,D))$$

  Let $[\![P_1]\!](\sigma,\mu,t,w,D) = (\sigma_1,\mu_1,t_1,w_2,D)$, $[\![P_2]\!]([\![P_1]\!](\sigma,\mu,t,w,D)) = (\sigma_2,\mu_2,t_2,w_2,D)$.
  The goal is to show: $A(P_1;P_2) \leq Adapt(M_1 \cdot M_2, V_1 \uplus V_2)$
  By induction hypothesis, we have: $A(P_1) \leq Adapt(M_1,V_1)$ and $A(P_2) \leq Adapt(M_2,V_2)$.
  By definition of V and t, we know the newly added queries in t_2 compared to the original trace t must be the same as newly added queries marked in $V_1 \uplus V_2$, i.e., the query nodes in the dependency graph must be contained in the Adapt graph generated by $M_1 \cdot M_2, V_1 \uplus V_2$.
  On the other hand, any dependency between newly added queries in $t_2 - t$ is tracked by $M_1 \cdot M_2$. It is shown in 3 cases: (1) dependency between queries nodes in P_1 is recorded in M_1. (2) dependency between queries nodes in P_2 is recorded in M_2.(3) dependency between query nodes from P_1 and P_2 respectively is tracked by $M_2 \times M_1$. To sum up, the dependency relation must be contained in $M_1 \cdot M_2$.
  Then we can conclude in this case, the longest path in the dependency graph of $P_1;P_2$ must be contained in the Adapt graph generated by $M_1 \cdot M_2, V_1 \uplus V_2$, i.e., $A(P_1;P_2) \leq Adapt(M_1 \cdot M_2, V_1 \uplus V_2)$.
  This case is proved.

- **Case:** $\dfrac{\Gamma + R(e) \vdash^{(c_1,c_1+1)}_{M_i,V_i} x_i \leftarrow q_i \qquad i \in \{1,\ldots,N\}}{\Gamma \vdash^{(c_1,c_1+N)}_{\sum_{i=0}^N M_i, \sum_{i=0}^N V_i} \mathtt{switch}\,(e,x,(v_i \to q_i)}$

  Given σ, μ, t and w, for arbitrary database D, we have the following semantic.

  $$[\![[\mathtt{switch}\,(e,x,(v_i \to q_i))]^l]\!](\sigma,\mu,t,w,D) \triangleq [\![[x \leftarrow q_1]^l]\!](\sigma,\mu,t,w,D) \qquad : v_1 = [\![e]\!](\sigma)$$

  Let $[\![[x \leftarrow q_1]^l]\!](\sigma,\mu,t,w,D) = (\sigma[x \to v_1'],\mu,t++[(v_1,q_1)],w,D)$.
  We then have: $[\![[\mathtt{switch}\,(e,x,(v_i \to q_i))]^l]\!](\sigma,\mu,t,w,D) = (\sigma[x \to v_1'],\mu,t++[(v_1,q_1)],w,D)$
  The goal is to show: $A([\mathtt{switch}\,(e,x,(v_i \to q_i))]^l) \leq Adapt(\sum_{i=0}^N M_i, \sum_{i=0}^N V_i)$
  By induction hypothesis, we have: $A([x \leftarrow q_i]^l) \leq Adapt(M_i,V_i)$ for any $v_i = q_i$.
  Then we have $A([\mathtt{switch}\,(e,x,(v_i \to q_i))]^l) \leq Adapt(M_i,V_i)$ for any $v_i = q_i$.
  Since we also have: $Adapt(M_i,V_i) \leq Adapt(\sum_{i=0}^N M_i, \sum_{i=0}^N V_i)$ for any $v_i = q_i$, this case is proved.

- **Case:** $\dfrac{\Gamma \vdash^{(c,c+a)}_{M,V} f;P : \{\Phi \land e_N = z+1\} \implies \{\Phi \land e_N = z\}}{\Gamma \vdash^{(c,c+N*a)}_{M^N_{c,a}(f),V^N_{c,a}} \mathtt{loop}\ e_N\ (f)\ \mathtt{do}\ P : \{\Phi \land e_N = N\} \implies \{\Phi \land e_N = 0\}}$

  Given σ, μ, t and w, for arbitrary database D, we have the following semantic.

  $$[\![\mathtt{loop}\ [e_N]^l\ (f)\ \mathtt{do}\ P]\!](\sigma,\mu,t,w,D) \triangleq \begin{cases} [\![f;P;\mathtt{loop}\ [e_N-1]^l\ (f)\ \mathtt{do}\ P]\!](\sigma,\mu,t,w+l,D) & : [\![e_N]\!](\sigma) > 0 \\ [\![\mathtt{skip}]\!](\sigma,\mu,t,w-l,D) & : [\![e_N]\!](\sigma) = 0 \end{cases}$$

  By induction on $[\![e_N]\!](\sigma)$, we have sub-cases of $[\![e_N]\!](\sigma) = 0$ and $[\![e_N]\!](\sigma) > 0$.
  Sub-case of $[\![e_N]\!](\sigma) = 0$ is obviously true.
  Sub-case of $[\![e_N]\!](\sigma) > 0$:
  The goal is to prove $A(f;P;\mathtt{loop}\ [e_N-1]^l\ (f)\ \mathtt{do}\ P) \leq Adapt(M^N_{c,a}(f), V_{c,a})$.

By unfolding the semantics of sequence, we have:
$[\![f;P;\texttt{loop}\ [e_N-1]^l\ (f)\ \texttt{do}\ P]\!](\sigma,\mu,t,w+l,D) = [\![\texttt{loop}\ [e_N-1]^l\ (f)\ \texttt{do}\ P]\!]([\![f;P]\!](\sigma,\mu,t,w+l,D))$.
By induction hypothesis, we have: $A(f;P) \leq Adapt(M,V)$.
Let $[\![f;P]\!](\sigma,\mu,t,w+l,D) = (\sigma_1,\mu_1,t_1,w_1,D)$. By Lemma 1(Subgraph), we know
(a). the newly added queries in t_1 must be contained in the queries nodes in V,
(b). the dependency between newly added queries must be contained in M.
Let $[\![\texttt{loop}\ [e_N-1]^l\ (f)\ \texttt{do}\ P]\!](\sigma_1,\mu_1,t_1,w_1,D) = (\sigma_N,\mu_N,t_N,w_{Nl},D)$, by definition of $V^N_{(c,a)}$ and $M^N_{(c,a)}(f)$ and (a) and (b), we have:
(1). All newly added queries in t_N must be contained in the queries nodes in $V^N_{(c,a)}$, this is proved jointly by (a).
(2). All the dependency between queries internally in P for all the N rounds are contained in $M^N_{(c,a)}(f)$ according to its definition case 1, this is proved jointly by (b).
(3). For all the dependency between queries in different rounds(for example, one query in the second iteration depending on the result of another query in the first iteration), they are recorded in f, which is also contained in $M^N_{(c,a)}(f)$ according to its definition case 3.
(4) For all the dependency between queries in P and outside P, since the newly added queries are all comes from P, we don't consider the adaptivity outside the scope of program.
According to (1) - (4), we can conclude in this case, the longest path in the dependency graph of $P;\texttt{loop}\ [e_N-1]^l\ (f)\ \texttt{do}\ P$ must be contained in the Adapt graph generated by $M^N_{(c,a)}(f), V^N_{(c,a)}$, i.e., $A(P;\texttt{loop}\ [e_N-1]^l\ (f)\ \texttt{do}\ P) \leq Adapt(M^N_{(c,a)}(f), V^N_{(c,a)})$.
This case is proved.

$\square$

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, $(\sigma,\mu) \vDash t$ *s.t.* $[\![P]\!](\sigma,\mu,t,w,D) \triangleq (\sigma',\mu',t',w',D)$, *then*

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