Figure 25: Dependencies in case \( l_{ij} = x_k \)

Similarly, it can be shown that \((V, E, D \cup \Delta, L)\) cannot contain any strong-cycles consistent with \(RT_1\) and \(RT_2\) if \(x_k\) is assigned \textit{false}. Thus, \((V, E, D \cup \Delta, L)\) cannot contain any strong-cycles consistent with either \(RT_1\) or \(RT_2\), and is strongly-acyclic with respect to \(R\). \(\square\)
only if: Suppose there exists an assignment of truth values to literals such that $C$ is satisfied. We show that there exists a set of dependencies $\Delta$ such that $D \cup \Delta$ is consistent and $(V, E, D \cup \Delta)$ is strongly-acyclic with respect to $R$. We specify the dependencies in the set $\Delta$. For every literal dependency $(U_k, V_k) \rightarrow (V_k, W_k)$ is added to $\Delta$ if $x_k$ is assigned true, else if $\overline{x_k}$ is assigned true, then dependency $(W_k, V_k) \rightarrow (V_k, U_k)$ is added to $\Delta$. From the construction of $\Delta$, it trivially follows that $D \cup \Delta$ is consistent. We show that it is impossible for $(V, E, D \cup \Delta, L)$ to contain any strong-cycles that are consistent with either $RT_1$ or $RT_2$.

We first show that $(V, E, D \cup \Delta, L)$ cannot contain any strong-cycles consistent with $RT_1$. A strong-cycle consistent with $RT_1$ cannot contain nodes $R_{ijk}, S_{ijk}, T_{ijk}, F_{ijk}, G_{ijk}$ or $H_{ijk}$ due to dependencies $(R_{ijk}, S_{ijk}) \rightarrow (S_{ijk}, T_{ijk})$ and $(H_{ijk}, G_{ijk}) \rightarrow (G_{ijk}, F_{ijk})$. Furthermore, since for every clause there exists a literal $l_{ij}$ that is assigned true, dependency $(N_{ij}, O_{ij}) \rightarrow (O_{ij}, P_{ij})$ is added to $\Delta$. Thus, there are no strong-cycles consistent with $RT_1$ in $(V, E, D \cup \Delta, L)$ involving nodes $N_{ij}, O_{ij}, P_{ij}, j = 1, 2, 3$. Thus, there are no strong-cycles consistent with $RT_1$ in $(V, E, D \cup \Delta, L)$.

We now show that $(V, E, D \cup \Delta, L)$ does not contain any strong-cycles consistent with $RT_2$. A strong-cycle consistent with $RT_2$ cannot contain any of the nodes $N'_{ij}, O'_{ij}$ or $P'_{ij}$ due to the dependency $(P'_{ij}, O'_{ij}) \rightarrow (O'_{ij}, N'_{ij})$, and must involve nodes $M_{ij}, P_{ij}, O_{ij}, N_{ij}, L_{ij}, F_{ijk}, G_{ijk}, H_{ijk}, X_k, U_k, V_k, W_k, Y_k, T_{ijk}, S_{ijk}, R_{ijk}$, for some literal $l_{ij} = x_k$ or $\overline{x_k}$. Let us assume that $x_k$ is true in the assignment.

We consider the following two cases:

$l_{ij} = \overline{x_k}$: In this case (as shown in Figure 24), since $l_{ij}$ is false in the assignment, dependency $(P_{ij}, O_{ij}) \rightarrow (O_{ij}, N_{ij})$ is added to $\Delta$ and thus, it is impossible for there to be any strong-cycle consistent with $RT_2$ involving nodes $P_{ij}, O_{ij}, N_{ij}$.

![Figure 24: Dependencies in case $l_{ij} = \overline{x_k}$](image-url)
\[ l_{ij} = \bar{x}_k: \] In this case (as shown in Figure 23), dependency \((W_k, V_k) \rightarrow (V_k, U_k)\) must belong to \(\Delta\), there would be a strong-cycle in the TSGD \((V, E, D \cup \Delta, L)\) consistent with \(RT_2\). Since \(\Delta\) is consistent, only one of \((W_k, V_k) \rightarrow (V_k, U_k)\) or \((U_k, V_k) \rightarrow (V_k, W_k)\) can belong to \(\Delta\). Thus, \((U_k, V_k) \rightarrow (V_k, W_k)\) can not belong to \(\Delta\), and \(x_k\) is assigned \(false\) (\(\bar{x}_k\) is assigned \(true\)).

Figure 22: Dependencies in case \(l_{ij} = x_k\)

Figure 23: Dependencies in case \(l_{ij} = \bar{x}_k\)
$R_{ijk}, T_{ijk}, F_{ijk}$ and $H_{ijk}$ are transaction nodes, while $G_{ij}$ and $S_{ijk}$ are site nodes. Subtransactions of $R_{ijk}, T_{ijk}, F_{ijk}$ and $H_{ijk}$ at sites $S_{ijk}, Y_k, L_{ij}$ and $G_{ijk}$ respectively are of type $b$, while subtransactions of $R_{ijk}, T_{ijk}, F_{ijk}$ and $H_{ijk}$ at sites $M_{ij}, S_{ijk}, G_{ijk}$ and $X_k$ are of type $a$. Note that there are at most three edges incident on $L_{ij}$ and $M_{ij}$. Also, there are two edges incident on each of $P_{ij}, O_{ij}, N_{ij}, R_{ijk}, S_{ijk}, T_{ijk}, F_{ijk}, G_{ijk}, H_{ijk}, U_k, V_k, W_k, P_{ij}', O_{ij}', N_{ij}'$. Note that the TSGD can be constructed in $O(p + q)$ steps.

The regular specification $R$ contains two regular terms, $RT_1$ and $RT_2$, $RT_1 = (A : a, b) : (A : a, b)$ and $RT_2 = (A : c, c) : ((A : b, a) + (A : c, c))$. We show that $C$ is satisfiable iff there exist a set of dependencies $\Delta$ such that $D \cup \Delta$ is consistent and $(V, E, D \cup \Delta, L)$ is strongly-acyclic with respect to $R$.

Let us assume there exists a set of dependencies $\Delta$ such that $(V, E, D \cup \Delta, L)$ is strongly-acyclic with respect to $R$ and $D \cup \Delta$ is consistent. We need to show that there exists an assignment of truth values to literals such that $C$ is satisfiable. We assign truth values to literals as follows. If dependency $(U_k, V_k) \rightarrow (V_k, W_k) \in \Delta$, then literal $x_k$ is assigned $true$, else $x_k$ is assigned $false$ ($\bar{x}_k$ is assigned $true$). Thus, only one of $x_k$ or $\bar{x}_k$ is assigned $true$.

We further need to show that in every clause $C_i$, there is at least one literal that is $true$. Since $(V, E, D \cup \Delta, L)$ is strongly-acyclic with respect to $R$, for every clause $C_i$, for some $l_{ij}, j = 1, \ldots, n$, there must be a dependency $(N_{ij}, O_{ij}) \rightarrow (O_{ij}, P_{ij})$ (else there would be a strong-cycle in the TS). We show that $l_{ij}$ must be assigned $true$, for which we need only consider the following two cases:

$l_{ij} = x_k$: In this case (as shown in Figure 22), dependency $(U_k, V_k) \rightarrow (V_k, W_k)$ must belong to $L_{ij}$ else there would be a strong-cycle in the TSGD $(V, E, D \cup \Delta, L)$ consistent with $RT_2$. Thus, $x_k$
more understandable). For all $j = 1, 2, 3$, nodes $P_{ij}, N_{ij}, P'_{ij}$ and $N'_{ij}$ are transaction nodes while nodes $M_{ij}, O_{ij}, L_{ij}$ and $O'_{ij}$ are site nodes. Subtransactions of $P_{ij}, N_{ij}, P'_{ij}$ and $N'_{ij}$ at sites $O_{ij}, L_{ij},$ and $M_{ij}$ respectively are of type $a$; while subtransactions of $P_{ij}, N_{ij}, P'_{ij}$ and $N'_{ij}$ at sites $M_{ij}, O_{ij}, L_{i(j\mod 3)+1}$ and $O'_{ij}$ respectively are of type $b$. Furthermore, for every literal $x_k$, we include the nodes and edges shown in Figure 19 in the TSGD.

![Figure 19: Nodes and edges for literal $x_k$](image)

$U_k$ and $W_k$ are transaction nodes, while $Y_k, V_k$ and $X_k$ are site nodes. Subtransactions of $U_k, W_k$ at sites $Y_k, V_k$ and $X_k$ are of type $c$. Also, we introduce additional edges and dependencies in TSGD depending on whether $l_{ij} = x_k$ or $l_{ij} = \bar{x}_k$. If $l_{ij} = x_k$, then the nodes, edges and dependencies illustrated in Figure 20 are added to the TSGD.

![Figure 20: Nodes and edges if $l_{ij} = x_k$](image)

On the other hand, if $l_{ij} = \bar{x}_k$, then we include nodes, edges and dependencies in the TSGD shown.

- $M_{ij}, O_{ij}, L_{ij}$ and $O'_{ij}$ are site nodes.
- $P_{ij}, N_{ij}, P'_{ij}$ and $N'_{ij}$ are transaction nodes.
- Subtransactions of $P_{ij}, N_{ij}, P'_{ij}$ and $N'_{ij}$ at sites $O_{ij}, L_{ij},$ and $M_{ij}$ respectively are of type $a$;
- Subtransactions of $P_{ij}, N_{ij}, P'_{ij}$ and $N'_{ij}$ at sites $M_{ij}, O_{ij}, L_{i(j\mod 3)+1}$ and $O'_{ij}$ respectively are of type $b$.
- For every literal $x_k$, we include the nodes and edges shown in Figure 19 in the TSGD.

$U_k$ and $W_k$ are transaction nodes, while $Y_k, V_k$ and $X_k$ are site nodes. Subtransactions of $U_k, W_k$ at sites $Y_k, V_k$ and $X_k$ are of type $c$. Also, we introduce additional edges and dependencies in TSGD depending on whether $l_{ij} = x_k$ or $l_{ij} = \bar{x}_k$. If $l_{ij} = x_k$, then the nodes, edges and dependencies illustrated in Figure 20 are added to the TSGD.

On the other hand, if $l_{ij} = \bar{x}_k$, then we include nodes, edges and dependencies in the TSGD shown.
Proof of Theorem 8: The above problem is in NP since a non-deterministic algorithm only needs to guess a set $\Delta$ such that there are dependencies between any two edges in the TSGD. $\Delta$ can contain at most $|E|^2$ dependencies since there can be at most $|E|^2$ dependencies in the TSGD $(V, E, D)$. The algorithm then needs to check if (1) $D \cup \Delta$ is consistent, and (2) for every regular term $RT$ in $g\tau$ and every node $v$ in the TSGD, if there is a strong-cycle consistent with $RT$ involving $v$ in the TSGD. Step 1 can be performed in polynomial time and involves detecting cycles in a directed graph. Step 2, too, can be performed in polynomial time using an algorithm similar to Detect.Ins.Opt that given a TSGD such that between any two edges there is a dependency, a node $v$ in the TSGD and a regular term $RT$, precisely detects if the TSGD contains a strong-cycle involving $v$ that is consistent with $RT$.

We show a polynomial transformation from 3-SAT to the above problem. Consider a 3-SAT formula $C = C_1 \land C_2 \land \cdots \land C_p$ that is defined over literals $x_1, x_2, \ldots, x_q$. Let $l_{ij}$ denote the literal in clause $C_i$, $i = 1, 2, \ldots, p$, in position $j$, $j = 1, 2, 3$ ($l_{ij}$ could be either $x_k$ or $\overline{x_k}$, for some $k = 1, 2, \ldots, q$). We construct a TSGD $(V, E, D, L)$ and a regular expression $R$ such that $C$ is satisfiable if and only if there exists a set of dependencies $\Delta$ such that $D \cup \Delta$ is consistent, and the TSGD $(V, E, D \cup \Delta)$ is strongly-acyclic with respect to $R$. Every global transaction in the MDBS has type $A$, that is, $g\tau = \{A\}$. Local DBMSs export procedures whose types are one of $a$, $b$ or $c$, that is, $l\tau = \{a, b, c\}$.

We construct the TSGD as follows. For every clause $C_i$, the TSGD contains the structure shown in Figure 18: Structure for clause $C_i$.
\begin{itemize}
  \item \((x_i, b'_i, N_{1i}, (N_{1i}, Z_{1i}), (Z_{1i}, Y_{1i}), (Y_{1i}, neg_1(1)), (neg_1(1), N_{2i}), (N_{2i}, Z_{2i}), \ldots, (Y_{i_{n+1}}, neg_{i_{n+1}}), (neg_{i_{n+1}}), (N_{i_{n+1}}, e'_i), (e'_i, x_{i+1})\), if \(|neg_1| > 0\),
  \item \((x_i, b'_i, N_{1i}), (N_{1i}, e'_i), (e'_i, x_{i+1})\), if \(|neg_1| = 0\),
\end{itemize}

This is mainly due to

- the dependency \((x_1, s_0) \rightarrow (s_0, C_{p+1})\), and for all \(i = 1, 2, \ldots, p\), dependencies \((x_{i+1}, e_i, (e_i, P_{i_{(i+1)}})), (x_{i+1}, e_i) \rightarrow (e'_i, N_{i_{(i+1)}})\) in \(D\), and

- for all \(l_{ij} = pos_1(k)\), only two edges are incident on each of \(P_{rk}, X_{rk}\) and \(W_{rk}\), and dependecies \((W_{rk}, l_{ij}) \rightarrow (l_{ij}, R_{ij}) \in D\) and \((B_{ij}, A_{ij}) \rightarrow (A_{ij}, C_i) \in D\) (a similar argument can be used \(l_{ij} = neg_1(k)\)).

Finally the strong-cycle contains the edges \((x_{q+1}, s_2)\) and \((s_2, G_i)\). Note that no node in the strong-cycles is visited more than once. Trivially, all the nodes other than \(l_{ij}\) appear only once in the strong-cycle.

Furthermore, if \(l_{ij} = pos_1(k)\) (the argument if \(l_{ij} = neg_1(k)\) is similar), then \(l_{ij}\) cannot be in the sequence of edges between both \(C_i\) and \(C_{i+1}\) as well as \(x_r\) and \(x_{r+1}\) since \(D \cup \{(R_{ij}, l_{ij}) \rightarrow (l_{ij}, B_{ij}), (P_{rk+1}, l_{ij}),(l_{ij}, W_{rk})\}\) is inconsistent, and the sequence of edges are in a strong-cycle.

We now show that there exists an assignment of truth values to \(x_k\) for all \(k = 1, 2, \ldots, q\), so that for all \(i = 1, 2, \ldots, p\), for some \(j = 1, 2, 3\), \(l_{ij}\) is assigned \textit{true}, and thus \(C\) is satisfiable. For \(i = 1, 2, \ldots, p\), for all \(j = 1, 2, 3\), \(l_{ij}\) is assigned \textit{true} if the edges \((B_{ij}, l_{ij}),(l_{ij}, R_{ij})\) are in the strong-cycle. This assignment causes \(C\) to be \textit{true} since as shown earlier, for all \(i = 1, 2, \ldots, p\), for some \(j = 1, 2, 3\), edges \((B_{ij}, l_{ij}),(l_{ij}, R_{ij})\) are in the strong-cycle.

Further, it is not possible that for some \(k = 1, 2, \ldots, q\), \(x_k\) and \(\bar{x}_k\) are both assigned \textit{true}. If \(x_k\) and \(\bar{x}_k\) are both assigned \textit{true}, then there must exist symbols \(l_{ij}\) and \(l_{rs}\) such that edges \((B_{ij}, l_{ij}),(l_{ij}, R_{ij}),(B_{rs}, l_{rs}),(l_{rs}, R_{rs})\) are in the strong-cycle, and \(l_{ij} = x_k\), \(l_{rs} = \bar{x}_k\). Thus, \(|neg_k| > 0\), \(|pos_k| > |neg_k|\), \(l_{ij} = pos_1(u)\), for some \(u, u = 1, 2, \ldots, |pos|\), and \(l_{rs} = neg_1(v)\), for some \(v, v = 1, 2, \ldots, |neg|\).

However, this is not possible, since as we showed earlier, one of \(l_{ij}\) and \(l_{rs}\) is also in the sequence of edges between \(x_k\) and \(x_{k+1}\) in the strong-cycle, and the strong-cycle does not visit a node more than once.

We now show that the problem of computing a set of dependencies, \(\Delta\), that is strongly-minimal with respect to \((V, E, D, L)\) and \(G_i\), is NP-hard.

\textbf{Proof of Theorem 7:} We show that the NP-complete problem of determining if \(\Delta' = \emptyset\) is not strongly-minimal with respect to \(G_i\) and \((V, E, D, L)\) can be Turing-reduced to the problem of computing a \(\Delta\) such that \(D \cup \Delta\) is consistent and \(\Delta\) is strongly-minimal with respect to \(G_i\) and \((V, E, D, L)\).

Consider a subroutine \(S((V, E, D, L), G_i)\) that returns a set of dependencies \(\Delta\) such that \(D \cup \Delta\) is consistent and \(\Delta\) is strongly-minimal with respect to \(G_i\) and \((V, E, D, L)\) (note that such a \(\Delta\) always exists if \((V, E, D, L)\) satisfies the conditions mentioned in the theorem). An algorithm for solving the problem of determining if \(\Delta' = \emptyset\) is not strongly-minimal with respect to \(G_i\) and \((V, E, D, L)\) calls \(S((V, E, D, L), G_i)\). If the set of dependencies \(\Delta\) returned by \(S\) is non-empty, then the algorithm responds \"yes\" (since if \(\Delta' = \emptyset\) is strongly-minimal with respect to \(G_i\) and \((V, E, D, L)\), then an empty \(\Delta\) cannot be strongly-minimal with respect to \(G_i\) and \((V, E, D, L)\), and \(S\) would return \(\emptyset\)). On the other hand, the set of dependencies \(\Delta\) returned by \(S\) is \(\emptyset\), then the algorithm responds \"no\" (since \(\Delta = \emptyset\) is strongly-minimal with respect to \((V, E, D, L)\) and \(G_i\)). \(\Box\)
cycle. Since $C$ is satisfiable, there exists an assignment of truth values to $x_k$, for all $k = 1, 2, \ldots \#$ such that for all $i = 1, 2, \ldots, p$, for some $j = 1, 2, 3$, $l_{ij}$ is assigned true. We now specify the edges in the strong-cycle. Edge sequence $(G_i, s_1)(s_1, C_1)$ is in the strong-cycle. For all $i = 1, 2, \ldots, p$, edge sequence $(C_i, A_{ij})(A_{ij}, B_{ij})(B_{ij}, l_{ij})(l_{ij}, R_{ij})(R_{ij}, Q_{ij})(Q_{ij}, C_{i+1})$ is in the strong-cycle, for some $j = 1, 2, 3$ such that $l_{ij}$ is true in the assignment. Edges $(C_{p+1}, s_0), (s_0, x_1)$ are also in the strong-cycle. For all $i = 1, 2, \ldots, q$, if $x_i$ is false in the assignment, then the following edges are in the strong-cycle:

- $(x_i, b_i), (b_i, P_{i1}), (P_{i1}, x_{i1}), (x_{i1}, W_{i1}), (W_{i1}, \text{pos}_i(1)), (\text{pos}_i(1), P_{i2}), (P_{i2}, x_{i2}), \ldots, (W_{i\lfloor \text{pos}_i \rfloor}, \text{pos}_i(\text{pos}_i)), (\text{pos}_i(\text{pos}_i), P_{i1\lfloor \text{pos}_i \rfloor}), (P_{i1\lfloor \text{pos}_i \rfloor}, e_i), (e_i, x_{i+1})$, if $|\text{pos}_i| > 0$,
- $(x_i, b_i), (b_i, P_{i1}), (P_{i1}, e_i), (e_i, x_{i+1})$, if $|\text{pos}_i| = 0$.

else if $x_i$ is true in the assignment, the strong-cycle contains the edges:

- $(x_i, b'_i), (b'_i, N_{i1}), (N_{i1}, Z_{i1}), (Z_{i1}, Y_{i1}), (Y_{i1}, \text{neg}_i(1)), (\text{neg}_i(1), N_{i1}), (N_{i2}, Z_{i2}), \ldots, (Y_{i\lfloor \text{neg}_i \rfloor}, \text{neg}_i(\text{neg}_i)), (\text{neg}_i(\text{neg}_i), N_{i\lfloor \text{neg}_i \rfloor}), (N_{i\lfloor \text{neg}_i \rfloor}, e'_i), (e'_i, x_{i+1})$, if $|\text{neg}_i| > 0$,
- $(x_i, b'_i), (b'_i, N_{i1}), (N_{i1}, e'_i), (e'_i, x_{i+1})$, if $|\text{neg}_i| = 0$.

Finally, the sequence of edges $(x_{p+1}, s_2)(s_2, G_i)$ are in the strong-cycle.

In the above choice of edges, we show that no node appears more than once in the strong-cycle. Nodes other than $l_{ij}$, trivially, appear only once. For any node $l_{ij}$, it is in the sequence of edges between nodes $C_i$ and $C_{i+1}$ only if $l_{ij}$ is true in the assignment. If $l_{ij} = \text{pos}_r(k)$, then $l_{ij} = x_r$, and since $x_r$ is true in the assignment, $l_{ij}$ is not among the nodes in the sequence of edges between $x_r$ and $x_{r+1}$. Similarly, if $l_{ij} = \text{neg}_r(k)$, then $l_{ij} = \bar{x}_r$, and since $x_r$ is false in the assignment, $l_{ij}$ is not among the nodes in the sequence of edges between $x_r$ and $x_{r+1}$. Thus, since

- for any consecutive edges $(v_1, v_2), (v_2, v_3)$ in the sequence, $v_1 \neq v_3$ and dependency $(v_1, v_2, v_3) \notin D$, and
- for all $l_{ij} = \text{pos}_r(k)$, $D \cup \{ (R_{ij}, l_{ij}) \rightarrow (l_{ij}, B_{ij}) \}$ is consistent and $D \cup \{ (P_{r(k+1)}, l_{ij}) \rightarrow (l_{ij}, W_{rk}) \}$ consistent, and
- for all $l_{ij} = \text{neg}_r(k)$, $D \cup \{ (R_{ij}, l_{ij}) \rightarrow (l_{ij}, B_{ij}) \}$ is consistent and $D \cup \{ (N_{r(k+1)}, l_{ij}) \rightarrow (l_{ij}, Y_{rk}) \}$ consistent,

the above sequence of edges constitute a strong-cycle involving $G_i$ in the TSGD.

We now show that if there is a strong-cycle involving $G_i$ in the TSGD, then there exists an assignment of truth values to literals such that $C$ is satisfiable. Any strong-cycle involving $G_i$ in the TSGD must contain the sequence of edges $(G_i, s_1)(s_1, C_1)$. Further, we claim that for all $i = 1, 2, \ldots, p$, sequence of edges $(C_i, A_{ij})(A_{ij}, B_{ij})(B_{ij}, l_{ij})(l_{ij}, R_{ij})(R_{ij}, Q_{ij})(Q_{ij}, C_{i+1})$ are in the strong-cycle, some $j = 1, 2, 3$. This follows from the fact that there are dependencies $(C_{r+1}, Q_{rs}) \rightarrow (Q_{rs}, R_{rs})$, all $r = 1, 2, \ldots, p$, for all $s = 1, 2, 3$ and also if $l_{ij} = \text{pos}_r(k)$, then the dependencies $(W_{rk}, X_{rk}, (W_{rk}, P_{rk}) \in D$ and $(B_{ij}, l_{ij}) \rightarrow (l_{ij}, P_{r(k+1)}) \in D$ (a similar set of dependencies can be identified in case $l_{ij} = \text{neg}_r(k)$). Thus, the strong-cycle also contains edges $(C_{p+1}, s_0), (s_0, x_1)$.

Also, for all $i = 1, 2, \ldots, q$, the strong-cycle contains either edges

- $(x_i, b_i), (b_i, P_{i1}), (P_{i1}, x_{i1}), (x_{i1}, W_{i1}), (W_{i1}, \text{pos}_i(1)), (\text{pos}_i(1), P_{i2}), (P_{i2}, x_{i2}), \ldots, (W_{i\lfloor \text{pos}_i \rfloor}, \text{pos}_i(\text{pos}_i)), (\text{pos}_i(\text{pos}_i), P_{i1\lfloor \text{pos}_i \rfloor}), (P_{i1\lfloor \text{pos}_i \rfloor}, e_i), (e_i, x_{i+1})$, if $|\text{pos}_i| > 0$,
- $(x_i, b_i), (b_i, P_{i1}), (P_{i1}, e_i), (e_i, x_{i+1})$, if $|\text{pos}_i| = 0$. 
only if \( r < s \)). In addition, there is no strong-cycle in \((V', E', D', L')\) consisting of transaction nodes from both \(S_1\) and \(S_2\) since such a strong-cycle must contain the sequence of edges \((v_1, l_{ij})(l_{ij}, v_2)\), for some node \(l_{ij}, v_1 \in S_2\) and \(v_2 \in S_1\) (so and \(l_{ij}\) are the only site nodes that have edges to transaction nodes both \(S_1\) and \(S_2\), and due to the dependency \((x_1, s_0)\rightarrow(s_0, C_{p+1})\), the sequence of edges \((x_1, s_0)(s_0, C_{p})\) cannot be in a strong-cycle). Let \(l_{ij} = pos_r(k)\) (the argument if \(l_{ij} = neg_r(k)\) is similar). Node \(v_1\) can be \(P_{r(k+1)}\) since if \(k < |pos_r|\), then only two edges are incident on each of \(P_{r(k+1)}\) and \(X_{r(k+1)}\), and edges preceding \((v_1, l_{ij})\) in the strong-cycle must be the sequence \((W_{r(k+1)}, X_{r(k+1)}(X_{r(k+1)}, P_{r(k+1)})\). However, due to the dependency \((W_{r(k+1)}, X_{r(k+1)}(X_{r(k+1)}, P_{r(k+1)})\), this is not possible. On the other hand, if \(k = |pos_r|\), then since only two edges are incident on each of \(P_{r(|pos_r|+1)}\) and \(e_r\), edges preceding \((v_1, l_{ij})\) in the strong-cycle must be the sequence \((x_{r+1}, e_r)(e_r, P_{r(|pos_r|+1)})\). However, due to the dependency \((x_{r+1}, e_r)(e_r, P_{r(|pos_r|+1)})\), this is not possible. Thus, \(v_1 = W_{rk}\). However, to the dependency \((W_{rk}, l_{ij})\rightarrow(l_{ij}, R_{ij}), v_2 \neq R_{ij}\). Thus, it must be the case that \(v_2 = B_{ij}\). However, since only two edges are incident on \(A_{ij}\) and \(B_{ij}\), the sequence of edges immediately following \(B_{ij}\) the cycle must be \((B_{ij}, A_{ij})(A_{ij}, C_i)\) which is not possible due to the dependency \((B_{ij}, A_{ij})\rightarrow(A_{ij}, C_i)\). Thus, there can be no strong-cycle in \((V', E', D', L')\) consisting of transaction nodes from both \(S_1\) and \(S_2\), and \((V', E', D', L')\) is strongly-acyclic.

We now show that \(C\) is satisfiable iff \((V, E, D, L)\) contains a strong-cycle involving \(G_i\). If \(C\)
\[- (x_i, b'_i, (b'_i, N_{i_1}), (N_{i_1}, Z_{i_1}), (Z_{i_1}, Y_{i_1}), (Y_{i_1}, neg_i(1)), (neg_i(1), N_{i_2}), (N_{i_2}, Z_{i_2}), \ldots, (Y_{i_1\negneg_i}, neg_i(|neg_i|)), (neg_i(|neg_i|), N_i(|neg_i|+1)), (N_i(|neg_i|+1), e_i'), (e_i', x_{i+1}), \text{if } |neg_i| > 0, \]

\[- (x_i, b_i), (b_i, N_{i_1}), (N_{i_1}, e_i'), (e_i', x_{i+1}), \text{if } |neg_i| = 0, \]

\[(x_{q+1}, s_2), (s_2, G_i), (G_i, s_1), (s_1, C_1). \]

Note that there are two edges incident on each of the symbols \(e_i, e_i', b_i, b_i', A_{ij}, B_{ij}, Q_i, R_{ij}, P_{ij}, W_{ij}, X_{ij}, N_{ij}, Y_{ij}\) and \(Z_{ij}\). In addition, there are four edges incident on every symbol \(l_{ij}\).

- If \(l_{ij} = pos_r(k)\), there are edges \((B_{ij}, l_{ij}), (l_{ij}, R_{ij}), (W_{rk}, l_{ij})\) and \((l_{ij}, P_{r(k+1)})\) in the TSGD.
- If \(l_{ij} = neg_r(k)\), there are edges \((B_{ij}, l_{ij}), (l_{ij}, R_{ij}), (Y_{rk}, l_{ij})\) and \((l_{ij}, N_{r(k+1)})\) in the TSGD.

The set of dependencies \(D\) consist of

\[(B_{ij}, A_{ij}) \rightarrow (A_{ij}, C_i), (C_{i+1}, Q_{ij}) \rightarrow (Q_{ij}, R_{ij}), \text{ for all } i = 1, 2, \ldots, p, \text{ for all } j = 1, 2, 3, \]

\[(x_1, s_0) \rightarrow (s_0, C_{p+1}), \]

for \(i = 1, 2, \ldots, q,

\[(P_{i_1}, b_i) \rightarrow (b_i, x_i), (W_{i_1}, X_{i_1}) \rightarrow (X_{i_1}, P_{i_1}), (W_{i_2}, X_{i_2}) \rightarrow (X_{i_2}, P_{i_2}), \ldots, (W_{i|pos_i|}, X_{i|pos_i|}) \rightarrow (X_{i|pos_i|}, P_{i|pos_i|}), (x_{i+1}, e_i) \rightarrow (e_i, P_{i|pos_i|+1}), \text{if } |pos_i| > 0, \]

\[(P_{i_1}, b_i) \rightarrow (b_i, x_i), (x_{i+1}, e_i) \rightarrow (e_i, P_{i_1}), \text{if } |pos_i| = 0, \]

\[(N_{i_1}, b'_i) \rightarrow (b'_i, x_i), (Y_{i_1}, Z_{i_1}) \rightarrow (Z_{i_1}, N_{i_1}), (Y_{i_2}, Z_{i_2}) \rightarrow (Z_{i_2}, N_{i_2}), \ldots, (Y_{i|neg_i|}, Z_{i|neg_i|}) \rightarrow (Z_{i|neg_i|}, N_{i|neg_i|}), (x_{i+1}, e_i') \rightarrow (e_i', N_{i|neg_i|+1}), \text{if } |neg_i| > 0, \]

\[(N_{i_1}, b'_i) \rightarrow (b'_i, x_i), (x_{i+1}, e_i') \rightarrow (e_i', N_{i_1}), \text{if } |neg_i| = 0, \]

- for each symbol \(l_{ij},\)

\[\begin{align*}
\text{if } l_{ij} = \text{pos}_r(k), & \text{ then the following dependencies are in } D: (W_{rk}, l_{ij}) \rightarrow (l_{ij}, R_{ij}) \quad (B_{ij}, l_{ij}) \rightarrow (l_{ij}, P_{r(k+1)}). \\
\text{if } l_{ij} = \text{neg}_r(k), & \text{ then the following dependencies are in } D: (Y_{rk}, l_{ij}) \rightarrow (l_{ij}, R_{ij}) \text{ and } (B_{ij}, l_{ij}) \rightarrow (l_{ij}, N_{r(k+1)}). \\
(C_1, s_1) \rightarrow (s_1, G_i),
\end{align*}\]

It is easy to see that the number of steps required to construct the TSGD \((V, E, D, L)\) is \(O(p + q)\). If \(C = \bar{x}_2 \vee x_1 \vee x_3\), then the constructed TSGD is as shown in Figure 17.

Our goal is to show that \(C\) is satisfiable iff \((V, E, D, L)\) contains a strong-cycle involving \(G_i\). We begin by showing that the TSGD \((V, E, D, L)\) satisfies the conditions. In \(D\), the only dependencies involving any of \(G_i\)'s edges is \((C_1, s_1) \rightarrow (s_1, G_i)\). Thus, in \(D\), there are only dependencies into \(G_i\)'s edges. Also, the set of dependencies, \(D\), is consistent. Further, we show that the TSGD \((V', E', D')\) is strongly-acyclic, where \(V' = V - G_i, E' = E - \{(G_i, s_1), (G_i, s_2)\}\), and \(D' = D - \{(C_1, s_1) \rightarrow (s_1, G_i)\}\). Let \(S_1 = \{C_1, C_2, \ldots, C_{p+1}\} \cup \{B_{ij}, R_{ij} : i = 1, 2, \ldots, p, j = 1, 2, 3\}\), and \(S_2 = \{x_1, x_2, \ldots, x_{q+1}, N_{r(k), Y_{rk}} : r = 1, 2, \ldots, q, k = 1, 2, \ldots, |neg_r|\} \cup \{P_{rk}, W_{rk} : r = 1, 2, \ldots, q, k = 1, 2, \ldots, |pos_r|\} \cup \{P_{r|pos_r|+1}, N_{r|neg_r|+1} : r = 1, 2, \ldots, q\}.\)

Note that there cannot exist a strong-cycle in \((V', E', D')\), such that all the transaction nodes in the cycle are in \(S_1\) (since there are dependencies \((B_{ij}, A_{ij}), (C_i, C_{i+1}), (Q_{ij}, R_{ij})\), for all \(i = 1, 2, \ldots, p, \) for all \(j = 1, 2, 3\), a sequence of edges from \(C_i\) can be part of a strong-cycle only if \(r < s\)). Similarly, there can be no strong-cycle in \((V', E', D')\), such that all the transaction nodes in the cycle are in \(S_2\) (that is, dependencies \((B_{ij}, A_{ij}), (C_i, C_{i+1}), (Q_{ij}, R_{ij})\), for all \(i = 1, 2, \ldots, p, \) for all \(j = 1, 2, 3\), a sequence of edges from \(C_i\) can be part of a strong-cycle only if \(r < s\)).
Figure 15: Edges and Dependencies if \( l_{ij} = \text{pos}_r(k) \)

On the other hand, if \( l_{ij} = \text{neg}_r(k) \), then edges and dependencies shown in Figure 16 are introduced in the TSGD.

Figure 16: Edges and Dependencies if \( l_{ij} = \text{neg}_r(k) \)

We now describe the nodes, edges and dependencies in the TSGD. The set of nodes \( V \) consists of transaction and site nodes. The set of transaction nodes in the TSGD consists of \( C_1, C_2, \ldots, C_p, C, x_1, x_2, \ldots, x_q, x_{q+1}, B_{ij}, R_{ij}, i = 1, 2, \ldots, p, j = 1, 2, 3, G_i \) (\( C_{p+1}, x_{q+1} \) and \( G_i \) are new symbols added to \( P_{r(|\text{pos}_r|+1)}, P_{rk}, W_{rk}, \) for all \( r = 1, 2, \ldots, q, k = 1, 2, \ldots, |\text{pos}_r| \), and for all \( r = 1, 2, \ldots, N_{r(|\text{neg}_r|+1)}, N_{rk}, Y_{rk} \) \( k = 1, 2, \ldots, |\text{neg}_r| \)). Site nodes consist of \( l_{ij}, A_{ij}, Q_{ij}, i = 1, 2, \ldots, p, j = 1, \) for all \( i, = 1, 2, \ldots, q, e_i, e_i', b_i, b_i', X_{rk} \) for all \( r = 1, 2, \ldots, q, k = 1, 2, \ldots, |\text{pos}_r|, \) and \( Z_{rk} \) for \( r = 1, 2, \ldots, q, k = 1, 2, \ldots, |\text{neg}_r| \) in addition to new symbols \( s_0, s_1, s_2 \).

The set of edges \( E \) consists of

- \( (C_i, A_{ij}), (A_{ij}, B_{ij}), (B_{ij}, l_{ij}), (l_{ij}, R_{ij}), (R_{ij}, Q_{ij}) \) and \( (Q_{ij}, C_{i+1}) \), for all \( i = 1, 2, \ldots, p, \) for \( j = 1, 2, 3, \)

- \( (C_{p+1}, s_0), (s_0, x_1), \)

- for \( i = 1, 2, \ldots, q, \)

Thus, \( (x_i, b_i), (b_i, P_{i1}), (P_{i1}, X_{i1}), (X_{i1}, W_{i1}), (W_{i1}, \text{pos}_i(1)), (\text{pos}_i(1), P_{i2}), (P_{i2}, X_{i2}), \ldots, \)

Thus, \( (P_{d(|\text{pos}|+1)}, e_i), (e_i, x_{i+1}), \) if \( |\text{pos}| > 0. \)
Appendix -E: Intractability results

Theorem 7 is a consequence of the following NP-completeness result.

**Theorem 9:** The following problem is NP-complete: Given a TSGD \((V, E, D, L)\) and a transaction node \(G_i \in V\), such that \(D\) is consistent, and for all transactions \(G_j \in V\), for all sites \(s_k\), dependencies \((G_i, s_k) \rightarrow (s_k, G_j) \notin D\). Also, TSGD \((V', E', D', L')\) resulting due to the deletion of \(G_i\), its edges dependencies from \((V, E, D, L)\), is strongly-acyclic. Is \(\Delta = \emptyset\) not strongly-minimal with respect to TSGD and transaction \(G_i\)?

**Proof:** We begin by showing that \(\Delta = \emptyset\) is not strongly-minimal with respect to \(G_i\) and \((V, E, D)\) iff \((V, E, D, L)\) contains a strong-cycle involving transaction \(G_i\). Since \(\Delta = \emptyset\), and universal quantification over \(\emptyset\) is always true, by the definition of strong-minimality, \(\Delta\) is strongly-minimal with respect to \(G_i\) and \((V, E, D, L)\) iff \((V, E, D, L)\) does not contain any strong-cycles involving \(G_i\). As a result, suffices to show that the following problem is NP-complete: Does \((V, E, D, L)\) contain a strong-cycle involving \(G_i\)?

The above problem is in NP since a non-deterministic algorithm only needs to guess a sequence containing at most \(2|E|^2 + 1\) edges and then check in polynomial time if the sequence of edges results in a strong-cycle involving \(G_i\) in the TSGD \((V, E, D, L)\). The algorithm only needs to guess a sequence of \(2|E|^2 + 1\) edges since any strong-cycle with more than \(2|E|^2 + 1\) edges, a consecutive pair of edges must be repeated (the total number of distinct pairs of edges is \(|E|^2\)). Thus, the strong-cycle must be of the form \((v_1', v_1)(v_1, v_2)(v_2, v_3)\cdots(v_1, v_2)(v_2, v_3)(v_3, v_4)\cdots\) for some nodes \(v_1, v_2, v_3, v_4\) in the TSGD. However, there exists a strong-cycle with fewer edges: \((v_1', v_1)(v_1, v_2)(v_2, v_3)(v_3, v_4)\cdots\). Thus, if \((V, E, D, L)\) contains a strong-cycle involving \(G_i\), then it contains a strong-cycle involving \(G_i\) with no more than \(2|E|^2 + 1\) edges.

We show a polynomial transformation from 3-SAT. Consider a formula in Conjunctive Normal Form (CNF) \(C = C_1 \land C_2 \land \cdots \land C_p\) that is defined over literals \(x_1, x_2, \ldots, x_q\). Let \(l_{ij}\), \(i = 1, 2, \ldots, p, j = 1, 2, \ldots, q\) be a new symbol for the \(j^{th}\) literal in clause \(C_i\). Each symbol \(l_{ij}\) is either \(x_k\) or \(\bar{x}_k\), \(k = 1, 2, \ldots, q\). In addition, for every literal \(x_i\), we introduce new symbols \(e_i, e'_i, b_i\) and \(b'_i\), and for literal \(l_{ij}\), we introduce new symbols \(A_{ij}, B_{ij}, Q_{ij}\) and \(R_{ij}\). For \(r = 1, 2, \ldots, q\), \(pos_r\) denotes the sequence of symbols \(l_{ij}\) in the order of increasing \(i\), such that \(l_{ij} = x_r\). For \(r = 1, 2, \ldots, q\), \(neg_r\) denotes the sequence of symbols \(l_{ij}\) in the order of increasing \(i\), such that \(l_{ij} = \bar{x}_r\). Also \(|pos|\) denotes the number of elements in \(pos\), and \(|neg|\) is similarly defined. For all \(r = 1, 2, \ldots, q\), we introduce new symbols \(P_{rk}, W_{rk}, Z_{rk}\), \(r = 1, 2, \ldots, |pos|\), \(P_{|[pos]|+1}\), \(r = 1, 2, \ldots, q\), new symbols \(N_{rk}, Y_{rk}, Z_{rk}\), \(r = 1, 2, \ldots, |neg|\), \(N_{|[neg]|+1}\). We illustrate the notation by means of the following example (“\(\cdot\)” is the concatenation operator for sequences and “\(\emptyset\)” is the empty sequence).

**Example:** Let \(C = (x_1 \lor \bar{x}_3 \lor x_4) \land (\bar{x}_2 \lor \bar{x}_1 \lor x_3) \land (\bar{x}_2 \lor \bar{x}_4 \lor x_1)\).

\[
l_{1,1} = x_1, \ l_{2,2} = \bar{x}_1, \ l_{3,3} = \bar{x}_4.
\]

\[
pos_1 = l_{1,1}, \ l_{3,3}, \ neg_1 = l_{2,2}, \ pos_2 = e.
\]

Also, \(|pos_1| = 2, |pos_2| = 0, |neg_2| = 2, pos_1(1) = l_{1,1}, pos_1(2) = l_{3,3}, neg_2(1) = l_{2,2}, neg_2(2) = l_{3,3}\). \(\square\)

We now construct the TSGD as follows. The main components in the TSGD are the edges of dependencies that we introduce for literals \(l_{ij}\). If \(l_{ij} = pos_r(k)\), then edges and dependencies show in Figure 15 are included in the TSGD.
We further use Lemma 3 to show that, for $F = FA(RT_2), \text{state}_F(\text{init\_st}_F, \text{edge}(t_1) \cdots \text{edge}(t_{n-1}) \text{(sfirst}(t_0), G_0))$ is an accept state. Let $\text{edge}(t_1) \cdots \text{edge}(t_{n-1}) \text{(sfirst}(t_0), G_0) = (v_1, v_2) \cdots (v_{2m-1}, v_2)$. In order to use Lemma 3, we need to show that there exists a sequence $g_1 \cdots g_{m-1}$ such that

- if $v_{2i} = v_{2i+1}$, then $g_i = L(v_{2i-1}, v_{2i})$, and
- if $v_{2i-1} = v_{2i+1}$, then $g_i = L(v_{2i-1}, v_{2i})$.

$\text{state}_F(\text{init\_st}_F, g_1 \cdots g_{m-1})$ is an accept state. We construct the sequence $g_1 \cdots g_{m-1}$ with the above properties as follows. For all $i = 1, \ldots, n-1$, let $f_i = (\text{type}(\text{hdr}(t_i)), \text{type}(\text{first}(t_i)))$, if $\text{arity}(t_i) = \text{arity}(t_i) = 0, \text{else, } f_i = (\text{type}(\text{hdr}(t_i)), \text{type}(\text{first}(t_i)))((\text{type}(\text{hdr}(t_i)), \text{type}(\text{last}(t_i)))).$ Since $\text{type}(t_1) \cdots \text{type}(t_{n-1})$ is a string in $L(\text{reg\_exp})$, by the construction of $FA(RT_2)$, it follows that $\text{state}_F(\text{init\_st}_F, f_1 \cdots f_{n-1})$ is an accept state. Let $g_1 \cdots g_{m-1} = f_1 \cdots f_{n-1}$, such that every $g_i \in \Sigma_F$. Furthermore, from the definition of $\text{edge}$ and $f_i$, it follows that, if for some $i = 1, \ldots, m-1$, if $(v_{2i-1}, v_{2i}) \in \text{edge}(t_k)$ and $\text{arity}(t_k) = \text{arity}(t_k) = 0, \text{then } g_i = L(v_{2i-1}, v_{2i}), \text{else } g_i = L(v_{2i-1}, v_{2i})$.

In order to show that $\text{state}_F(\text{init\_st}_F, (v_1, v_2), \ldots, (v_{m-1}, v_m))$ is an accept state, we need to show that for all $i = 1, 2, \ldots, m-1$, if $v_{2i} = v_{2i+1}$, then $g_i = L(v_{2i-1}, v_{2i})$ and if $v_{2i-1} = v_{2i+1}$, then $g_i = L(v_{2i-1}, v_{2i})$. We first show that if $v_{2i} = v_{2i+1}$, and $(v_{2i-1}, v_{2i}) \in \text{edge}(t_k)$ for some $k$, then $\text{arity}(t_k) = \text{arity}(t_k) = 0, \text{which leads to a contradiction. Thus, } \text{arity}(t_k) = 2, \text{and } g_i = L(v_{2i-1}, v_{2i}).$ Also, it can be shown that if $v_{2i-1} = v_{2i+1}$, and $(v_{2i-1}, v_{2i}) \in \text{edge}(t_k)$, then $\text{arity}(t_k) = 1$. Suppose $\text{arity}(t_k) = 2, \text{and } v_{2i} = G_k, \text{then } v_{2i} = v_{2i+1} = G_k$, which leads to a contradiction. If $v_{2i-1} = G_k$, then since $\text{last}(t_k)$, if $\text{first}(t_{k+1})\mod n$ execute at the same site, $\text{last}(t_k) = v_{2i-1}, \text{first}(t_{k+1})\mod n = v_{2i+1}$, it follows that $v_{2i-1} = v_{2i+1}$, which leads to a contradiction. Thus, $\text{arity}(t_k) = 1, \text{and, } g_i = L(v_{2i-1}, v_{2i})$.

Thus, by Lemma 3, $\text{state}_F(\text{init\_st}_F, \text{edge}(t_1) \cdots \text{edge}(t_{n-1})\text{(sfirst}(t_0), G_0))$ is an accept state. Then, by corollaries 8 and 10, during the execution of Detect.Ins.TSGD$^\Sigma(V, E, D, L, G_0, \text{slast}(t_0), set_1, RT_2)$, dependency $\text{prev\_anc}(\text{first}(t_0)), \text{first}(t_0)\rightarrow (\text{first}(G_0), G_0)\text{ is added to } \Delta, \text{and the} \text{prev\_anc}(\text{first}(t_0)), \text{first}(t_0)\rightarrow (\text{first}(t_0), G_0)\in \Delta_F$. However, this leads to a contradiction since we showed earlier that $\text{prev\_anc}(\text{first}(t_0)), \text{first}(t_0)\rightarrow (\text{first}(t_0), G_0)\not\in \Delta_F$. Thus, every schedule $S$ is correct. \□
When \( \text{init}_0 \) is processed, the procedure Detect\_Ins\_TSGD? is invoked with arguments that include the TSGD \((V, E, D, L), G_0, \text{slast}(t_0), \text{set}_1, \) and \( RT_2 \) since \( \text{type}(G_0) = \text{hdr}(c_0) \) and \( \text{type}(\text{last}(t_0)) = \text{last}(c_0) \). Also, \( s\text{first}(t_0) \in \text{set}_1 \) (if \( \text{arity}(t_0) = 1 \), then since \( s\text{first}(t_0) = \text{slast}(t_0) \), \( s\text{first}(t_0) \in \text{set}_1 \) if binary\( (t_0) \), then since \( s\text{first}(t_0) \neq \text{slast}(t_0) \), and \( \text{type}(\text{first}(t_0)) = \text{first}(c_0) \), \( s\text{first}(t_0) \in \text{set}_1 \). Furthermore, all the edges belonging to \( G_0, \ldots, G_{n-1} \) are in the TSGD when Detect\_Ins\_TSGD? is invoked. In order to show this, we first show that \( G_j \)'s edges cannot be deleted from the TSGD before \( G_{j+1} \)’s edges are deleted from the TSGD, for all \( j, j = 1, 2, \ldots, n-1 \). Suppose, some \( j, j = 1, 2, \ldots, n-1 \), \( G_j \)'s edges are deleted from the TSGD before \( G_{j+1} \)'s edges are deleted from the TSGD. Let \( \text{slast}(t_j) = s_k \). Since \( G_{jk} \) is serialized after \( G_{j+1} \) before \( G_j \), if \( s\text{first}(G_{j+1}) \) executes before \( s\text{first}(G_j) \). Thus, \( G_{j+1} \)’s edges must be in the TSGD when \( G_j \)’s edges are deleted (since we have assumed that \( G_j \)’s edges are deleted before \( G_{j+1} \)’s edges are deleted). Furthermore, since \( s\text{first}(G_j) \) and \( s\text{first}(G_{j+1}) \) must have both executed when \( G_j \)’s edges are deleted, \( G_{j+1} \) is serialized before \( G_j \) when \( G_j \)’s edges are deleted. However, this leads to a contradiction, since edges belonging to \( G_j \) and \( G_{j+1} \) are deleted together when \( \text{fin}_t \) for some transaction \( G_i \) is processed (since \( G_{j+1} \) is serialized before \( G_j \), if \( \text{for every transaction } G_k \in V \text{ serialized before } G_j \), \( \text{val}_k \) has been processed, then for every transaction \( G_k \in V \text{ serialized before } G_{j+1} \) also, \( \text{val}_k \) must have been processed). Thus, \( G_j \)’s edges are not deleted from the TSGD before \( G_2 \)’s edges are deleted, \( \ldots, G_{n-1} \)’s edges are not deleted from the TSGD before \( G_0 \)’s edges are deleted. By transitivity and since \( G_0 \)’s edges are deleted after \( \text{init}_0 \) has been processed, when Detect\_Ins\_TSGD? is invoked during the processing of \( \text{init}_0 \), TSGD contains all the edges belonging to transactions \( G_0, G_1, \ldots, G_{n-1} \).

Let \( \Delta_F \) be the set of dependencies returned by Detect\_Ins\_TSGD?. We now show that \((G_0, \text{slast}(t_1), \ldots, \text{slast}(t_{n-1})(s\text{first}(t_0), G_0)\) is a path in the TSGD \((V, E, D \cup \Delta_F)\). We begin by showing that any two consecutive edges in the path have a common node. Consecutive edges in the path could be one of the following:

- \((s\text{first}(G_j), G_j, \text{slast}(G_j)), j = 1, 2, \ldots, n-1, \) where \( \text{arity}(t_j) = 2 \) (\( G_j \) is the common node).
- \((G_j, \text{slast}(t_j))(s\text{first}(t_{j+1})) \text{mod}(n), G_{j+1} \text{mod}(n)), j = 0, 1, \ldots, n-1, \) where \( \text{arity}(t_{j+1}) = 1 \) or \( j = 0 \) \( \text{arity}(t_{j+1}) = 1 \) or \( j = 0, 1, \ldots, n-1, \) \( \text{last}(t_j) \) and \( \text{first}(t_{j+1}) \) execute at the same site, \( \text{slast}(t_j) = s\text{first}(t_{j+1}) \text{mod}(n) \) is the common node).
- \((s\text{first}(t_j), G_j, s\text{first}(t_{j+1})) \text{mod}(n), G_{j+1} \text{mod}(n)), j = 1, 2, \ldots, n-1, \) where \( \text{arity}(t_j) = 1 \), \( \text{arity}(t_{j+1}) = 1 \) (since \( \text{arity}(t_j) = 1 \) implies that \( s\text{first}(t_j) = \text{slast}(t_j) \), and \( \text{slast}(t_j) = s\text{first}(t_{j+1}) \text{mod}(n) \) is the common node).

Also, for the sequence of edges \((s\text{first}(t_j), G_j, \text{slast}(t_j))\) in the path, \( j = 1, 2, \ldots, n-1, \) it may be the case that \( \text{arity}(t_j) = 2 \), and thus \( s\text{first}(t_j) \neq \text{slast}(t_j) \). Also, if for some \( j, k, j = 0, 1, \ldots, n-1, \) \( j < k \leq n \), the sequence of edges \((G_j, s\text{first}(t_j)) \text{mod}(n), G_{j+1} \text{mod}(n)\), \((s\text{first}(t_{k+1})) \text{mod}(n), G_{k+1} \text{mod}(n)\) is in the path, then it must be the case that for all \( j < l < k, \text{arity}(t_l) = 1 \).

Thus, by Property 1, it follows that \( \text{slast}(t_j) = s\text{first}(t_{j+1}) \text{mod}(n) = \cdots = s\text{first}(t_{k+1}) \text{mod}(n) \), and for \( r, s, j \leq r < s \leq k \),

- \( G_r \neq G_{s \text{mod}(n)} \), and
- \( G_r \) is serialized after \( G_{s \text{mod}(n)} \) at site \( s\text{first}(G_{s \text{mod}(n)}) \). Thus, by Lemma 14, dependency \((G_r, s\text{first}(G_{s \text{mod}(n)}) \rightarrow (s\text{first}(G_{s \text{mod}(n)}), G_{s \text{mod}(n)}) \) does not belong to \( D \cup \Delta_F \) (since \( \Delta_F \) is added to the list of dependencies \( D \) in the TSGD immediately after \( \text{init}_0 \) is processed).
However, since in state $S_t^d$, no forward transition can be made due to edge $(S_t^d, v, v_{m+2})$, it must be the case that

- if $v_{m+2} = v_{m+3}$, then $S_{t'}^d = V_{set}(v_{m+2})$ already contains $(st_{m+1}, (S_{t'}^d, v_{m+1}), prev(v_{m+3}) = v_{m+1}, prev.anc(v_{m+3}) = v_{m+1}, (st_{m+1}, (prev.anc(v_{m+3}), prev(v_{m+3}))))$ is added to $V_{set}(v_{m+3})$ during the execution of $Detext.Ins.TSGD2$.

- if $v_{m+1} = v_{m+3}$, then $S_{t'}^d = V_{set}(S_{t'}^d, v)$ already contains $(st_{m+1}, (prev.anc(v_{m+1}), v_{m+1}, st_{m+1}, (prev.anc(v_{m+3}), prev(v_{m+3}))))$ is added to $V_{set}(v_{m+3})$ during the execution of $Detext.Ins.TSGD2$. □

**Corollary 10:** Let $Detext.Ins.TSGD2((V, E, D, L), v_1, v_2, set_1, RT)$ return the set of dependencies $\Delta_F$. If the TSGD $(V, E, D \cup \Delta_F)$ contains a path $(v_1, v_2, \cdots, v_{m-1}, v_m)\cdots (v_{m-1}, v_m, v_1)$, $v_1$, $v_2$, $\cdots$, $v_{m-1}$, $v_m$, is an accept state and $v_{m+1} \in set_1$, then during the execution of $Detext.Ins.TSGD2$, dependency $(prev.anc(v_{m+1}), v_{m+1}) \rightarrow (v_{m+1}, v_1)$ is added to $\Delta$.

**Proof:** By Lemma 13, $(st, (prev.anc(v_{m+1}), prev(v_{m+1})))$ is added to $V_{set}(v_{m+1})$. Since $prev(v_{m+1}) \neq v_1$ and $prev.anc(v_{m+1}) \neq v_1$ (by definition of path), $Detext.Ins.TSGD2$ makes a forward state transition when $(st, (prev.anc(v_{m+1}), prev(v_{m+1})))$ is added to $V_{set}(v_{m+1})$. However, just before $(st, (prev.anc(v_{m+1}), prev(v_{m+1})))$ is added to $V_{set}(v_{m+1})$, since $st$ is an accept state, $(prev.anc(v_{m+1}) \neq v_1, prev(v_{m+1}) \neq v_1$ and $v_{m+1} \in set_1$, dependency $(prev.anc(v_{m+1}), v_{m+1}) \rightarrow (v_{m+1}, v_1)$ is added to $\Delta$. □

We are now in a position to prove that the TSGD scheme ensures the correctness of $S$. Before we present the proof, we prove the following lemma.

**Lemma 14:** If, in the TSGD scheme, for some site $s_k$, transactions $G_i, G_j, G_{ik}$ is serialized before $G_{jk}$ at site $s_k$, then there does not exist a dependency $(G_j, s_k) \rightarrow (s_k, G_i)$ in the TSGD.

**Proof:** Suppose there exists a dependency $(G_j, s_k) \rightarrow (s_k, G_i)$ in the TSGD. The dependency cannot be added to the TSGD once $act(ser_k(G_i))$ has executed. Thus, dependency $(G_j, s_k) \rightarrow (s_k, G_i)$ must be added to the TSGD before $act(ser_k(G_i))$ executes. However, if this were the case, $act(ser_k(G_i))$ would not execute until $act(ack(ser_k(G_j)))$ completes execution (the dependency $(G_j, s_k) \rightarrow (s_k, G_i)$ is deleted from the TSGD only after $act(ack(ser_k(G_j)))$ is processed). Thus, $ser_k(G_j)$ would execute before $ser_k(G_i)$ and $G_{jk}$ would be serialized before $G_{ik}$ at site $s_k$, which leads to a contradiction. □

**Proof of Theorem 5:** Suppose $S$ is not correct. Thus, there exists a regular term $RT$ in $R$, an instantiation $I$ of $RT$ in $S$. Let $G_0$ be the transaction in $I$ such that $init_0$ is processed after $init_1$ every other transaction $G_i$ in $I$ is processed. By Lemma 1, since $R$ is complete, there exists a regular term $RT_2 = c_0 : reg.exp$ and an instantiation $t_0 : t_1 t_2 \cdots t_{n-1}$ of $RT_2$ in $S$ such that $hdr(t_0) = T_0$.

Thus, for all $j, j = 0, 1, \ldots, n - 1$,

1. $t_j \in \Sigma_S$ (without loss of generality, let $hdr(t_j) = G_j$), and
2. $last(t_j)$ and $first(t_{(j+1)modn})$ execute at the same site, and $last(t_j)$ is serialized at $first(t_{(j+1)modn})$ at the site, and
the conditions in Step 2 need to be checked, on an average, for \(v_S\) edges (the average number of sites a global transaction executes at is \(v_S\)), while every time a site node is visited, the conditions in Step 2 need to be checked for at most \(n_G\) edges (since the number of transaction nodes in the TSGD is at most \(n_G\)). Furthermore, every transaction node can be visited at most \(v_S^2n_S\) times, while every site node can be visited at most \(n_G^2n_S\) times (every node \(v\) in the TSGD can be visited in a state \(st\) of \(F\) at most once for every pair of nodes \(u, w\) such that \((v, w)\) and \((v, u)\) are edges in the TSGD, and \(F\) has at most \(n_S\) states). Since there are \(m\) site nodes and at most \(n_G\) transaction nodes in the TSGD, the number of times \(\text{Detect}_\text{Ins}_\text{TSGD2}\) checks if an edge satisfies the conditions in Step 2 is \(n_G^3mn_S + n_G^3n_S^3\). Since each of the conditions in Step 2 can be checked in constant time and \(v_S \ll n_G, v_S < m\), \(\text{Detect}_\text{Ins}_\text{TSGD2}\) terminates in \(O(n_G^3mn_S)\) steps. \(\square\)

We now show that \(\text{Detect}_\text{Ins}_\text{TSGD2}\) traverses edges in the TSGD in a manner that ensures it detects instantiations of regular terms.

**Lemma 13:** Let \(\text{Detect}_\text{Ins}_\text{TSGD2}(\langle V, E, D, L \rangle, v_1, v_2, st_1, RT)\) return the set of dependencies \(\Delta_F\). If the TSGD \(\langle V, E, D \cup \Delta_F \rangle\) contains a path \((v_1, v_2), (v_3, v_4), \ldots, (v_{2n-3}, v_{2n-2}), (v_{2n-1}, v_2)\), \(v_2 = v_3\), such that for the regular term \(RT, F = FA(\text{RT})\), \(\text{state}_F(\text{initial}_st_F, (v_3, v_4), \ldots, (v_{2n-1}, v_2))\) is defined, then during the execution of \(\text{Detect}_\text{Ins}_\text{TSGD2}(\langle st, (\text{prev}\_\text{anc}(v_2i+1), \text{prev}(v_2i+1)) \rangle)\) is added to \(V\_\text{set}(v_{2i+1})\), where \(st = \text{state}_F(\text{initial}_st_F, (v_3, v_4) \ldots (v_{2i-1}, v_2)(v_{2i+1}, v_{2i+2}))\), for all \(i, i = 1, 2, 3, \ldots\).

**Proof:** We prove the above lemma by induction on \(i\). We prove that for all \(i, i = 1, 2, \ldots, n\), \(V\_\text{set}(v_{2i+1})\) is added to \(V\_\text{set}(v_{2i+1})\), where \(st = \text{state}_F(\text{initial}_st_F, (v_3, v_4) \ldots (v_{2i-1}, v_2)(v_{2i+1}, v_{2i+2}))\).

**Basis** \((i = 1)\): In Step 1 of \(\text{Detect}_\text{Ins}_\text{TSGD2}\), \((\text{initial}_st_F, (v_1, v_2))\) is added to \(V\_\text{set}(v_2)\). Since \(v_2 = \text{prev}_\text{anc}(v_3) = \text{prev}(v_3) = v_1\), and \(\text{state}_F(\text{initial}_st_F, (v_3, v_4)) = \text{initial}_st_F\), the lemma is true for \(i = 1\).

**Induction:** Let us assume that the lemma is true for \(i = m, 1 \leq m < n - 1\). Thus, \(\text{(state}_st_F(st_{m}, \text{initial}_st_F, (v_3, v_4))\) is added to \(V\_\text{set}(v_{2m+1})\), where \(st_m = \text{state}_F(\text{initial}_st_F, (v_3, v_4) \ldots (v_{2m-1}, v_{2m})(v_{2m+1}, v_{2m+2}))\). We show the lemma to be true for \(i = m + 1\). Thus, we need to show that \((\text{state}_st_F(st_m, \text{initial}_st_F, (v_3, v_4))\) is added to \(V\_\text{set}(v_{2m+3})\), where \(st_{m+1} = \text{state}_F(\text{initial}_st_F, (v_3, v_4) \ldots (v_{2m+1}, v_{2m+2})(v_{2m+3}, v_{2m+4}))\). By the definition of \(\text{state}_F\), \(st_{m+1} = st_F(st_m, L(v_{2m+1}, v_2)\) if \(v_{2m+2} = v_{2m+3}\) and \(st_{m+1} = st_F(st_m, L(v_{2m+1}, v_{2m+2}))\), if \(v_{2m+3} = v_{2m+2}\).

Let \(St_k\) be the resulting state of \(\text{Detect}_\text{Ins}_\text{TSGD2}\) after \((st_{m}, \text{prev}\_\text{anc}(v_{2m+1}), \text{prev}(v_{2m+1}))\) is added to \(V\_\text{set}(v_{2m+1})\) (the state \(St_k\) results either due to the forward transition \(St_j - St_k\), either \(St_j \cdot v = v_{2m+1}\) or \(St_j \cdot v = \text{prev}(v_{2m+1})\), or due to Step 1). Thus, \(St_k \cdot v = v_{2m+1}, St_k - \text{cur}\_\text{st} = st_m\) in state \(St_k\), \(\text{head}(\text{state}_st_F(St_k \cdot v)) = \text{prev}\_\text{anc}(v_{2m+1}, \text{prev}(v_{2m+1}))\). Furthermore, it follows from Lemma 12 that after a finite number of steps, \(\text{Detect}_\text{Ins}_\text{TSGD2}\) is in a state \(St_k'\) in which no further transitions can be made from \(St_k'\). Thus, in state \(St_k'\),

- Since \(\text{prev}(v_{2m+1}) \neq v_{2m+2}\) and \(\text{prev}\_\text{anc}(v_{2m+1}) \neq v_{2m+2}\), \(\text{head}(\text{state}_st_F(St_k' \cdot v))[1] \neq v_{2m+2}\), \(\text{head}(\text{state}_st_F(St_k' \cdot v))[2] \neq v_{2m+2}\).

- Since \(St_k' \cdot \Delta \subseteq \Delta_F\) and \((v_1, v_2) \ldots (v_{2m+2}, v_{2m+2})\) is a path in \((V, E, D \cup \Delta_F)\), there are no dependencies \((\text{prev}(v_{2m+1}), v_{2m+1}) \rightarrow (v_{2m+1}, v_{2m+2})\) and \((\text{prev}\_\text{anc}(v_{2m+1}), v_{2m+1}) \rightarrow (v_{2m+1}, v_{2m+2})\) in \(D \cup \Delta_F\); thus, dependencies \(\text{(head}(\text{state}_st_F(St_k' \cdot v))[2], St_k' \cdot v) \rightarrow (St_k' \cdot v, v_{2m+2})\) and \((\text{head}(\text{state}_st_F(St_k' \cdot v))[1], St_k' \cdot v) \rightarrow (St_k' \cdot v, v_{2m+2})\) are not in \(D \cup St_k' \cdot \Delta\).

- Since \(\text{state}_F(\text{initial}_st_F, (v_3, v_4) \ldots (v_{2m+1}, v_{2m+2})(v_{2m+3}, v_{2m+4}))\) is defined, if \(v_{2m+2} = v_{2m+3}\), then \(\text{state}_F(\text{state}_st_F(St_k' \cdot v), v_{2m+2}) = \text{state}_F(\text{state}_st_F(St_k' \cdot v), v_{2m+2})\).
For $St_j.v = \text{prev.anc}(v_{m+1})$, or due to Step 1. Thus, $St_k.v = v_{m+1}$, $St_k.\text{cur. st} = st_m$ and in state $St_k$, head$(St_k.\text{anc}(St_k.v)) = \text{prev.anc}(v_{m+1})$. Furthermore, it follows from Lemma 10 that after a finite number of steps, Detect.Ins.TSGD1 is in a state $St_k'$ such that $St_k' \equiv St_k$ and no further forward transitions can be made from $St_k'$. Thus, in state $St_k'$,

- Since $\text{prev.anc}(v_{m+1}) \neq v_{m+2}$ (by the definition of path), head$(St_k'.\text{anc}(St_k'.v)) \neq v_{m+2}$,
- Since $St_k'.\Delta \subseteq \Delta_F$, and $(v_1, v_2) \cdots (v_{m+1}, v_{m+2})$ is a path in $(V, E, D \cup \Delta_F)$, there is no dependency $(\text{prev.anc}(v_{m+1}), v_{m+1}) \rightarrow (v_{m+1}, v_{m+2})$ in $D \cup \Delta_F$; thus, there is no dependency $(head(St_k'.\text{anc}(St_k'.v)), St_k'.v) \rightarrow (St_k'.v, v_{m+2})$ in $D \cup St_k'.\Delta$,
- Since state_F$(init.st_F, (v_3, v_4) \cdots (v_{m+3}, v_{m+4}))$ is defined, if $v_{m+2} = v_{m+3}$, then $st_{m+1} = st_F(St_k'.\text{cur. st}, L(St_k'.v, v_{m+2}))$ is defined, else if $v_{m+1} = v_{m+3}$, then $st_{m+1} = st_F(St_k'.\text{cur. st}, L(St_k'.v, v_{m+2}))$ is defined.

However, since in state $St_k'$, no forward transition can be made due to edge $(St_k'.v, v_{m+2})$, it must be the case that

- if $v_{m+2} = v_{m+3}$, then $St_k'.V.set(v_{m+2})$ already contains $(st_{m+1}, St_k'.v)$. Thus, since $St_k'.v = v_{m+1}$, $\text{prev.anc}(v_{m+3}) = v_{m+1}$, $(st_{m+1}, \text{prev.anc}(v_{m+3}))$ is added to $V.set(v_{m+3})$ during the execution of Detect.Ins.TSGD1.
- if $v_{m+1} = v_{m+3}$, then $St_k'.V.set(St_k'.v)$ already contains $(st_{m+1}, \text{prev.anc}(v_{m+1}))$. Thus, since $St_k'.v = v_{m+1}$, $\text{prev.anc}(v_{m+3}) = \text{prev.anc}(v_{m+1})$, $(st_{m+1}, \text{prev.anc}(v_{m+3}))$ is added to $V.set(v_{m+3})$ during the execution of Detect.Ins.TSGD1. □

**Corollary 8:** Let Detect.Ins.TSGD1($(V, E, D, L, v_1, v_2, set_1, RT)$) return the set of dependencies $\Delta_F$. If the TSGD $(V, E, D \cup \Delta_F)$ contains a path $(v_1, v_2) \cdots (v_{n-1}, v_n)(v_{n+1}, v_1)$, $v_2 = v_3$, so that for the regular term $RT$, $F = FA(RT)$, $st = state_F(init.st_F, (v_3, v_4) \cdots (v_{n-1}, v_n)(v_{n+1}, v_1)$, is an accept state and $v_{n+1} \in set_1$, then during the execution of Detect.Ins.TSGD1, dependency $(\text{prev.anc}(v_{n+1}), v_{n+1}) \rightarrow (v_{n+1}, v_1)$ is added to $\Delta$.

**Proof:** By Lemma 11, $(st, \text{prev.anc}(v_{m+1}))$ is added to $V.set(v_{m+1})$. Since $\text{prev.anc}(v_{m+1}) \neq Detect.Ins.TSGD1$ makes a forward state transition when $(st, \text{prev.anc}(v_{m+1}))$ is added to $V.set(v_2)$. However, just before $(st, \text{prev.anc}(v_{m+1}))$ is added to $V.set(v_{m+1})$, since $st$ is an accept state, $\text{prev.anc}(v_{m+1}) \neq v_1$ and $v_{m+1} \in set_1$, dependency $(\text{prev.anc}(v_{m+1}), v_{m+1}) \rightarrow (v_{m+1}, v_1)$ is added to $\Delta$. □

We now show that Detect.Ins.TSGD2 terminates in $O(n^2_\Delta mn_S)$ steps, for which we need to prove the following lemma.

**Lemma 12:** If during its execution, Detect.Ins.TSGD2 is in state $St_k$, then after a finite number of steps, it enters a state $St_k' \equiv St_k$ such that no forward transitions from $St_k'$ are possible.

**Proof:** Similar to proof of Lemma 8. □

**Corollary 9:** Procedure Detect.Ins.TSGD2 terminates in $O(n^2_\Delta mn_S)$ steps.

**Proof:** Detect.Ins.TSGD2 can be shown to terminate as a result of Lemma 12 using a similar argument as in Corollary 3.

The number of steps Detect.Ins.TSGD2 terminates in is equal to the product of the number of times Detect.Ins.TSGD2 checks if an edge satisfies the conditions in Step 2 and the number of steps in Step 3 to check if a state is an accept state.
Appendix -D- : TSGD Schemes

In this appendix, we prove Theorem 5. We begin by showing that Detect_Ins.TSGD1 and Detect_Ins.TSGD2 detect instantiations of regular terms in $S$. States $St_k$ between the execution of two steps of Detect_Ins.TSGD1 and Detect_Ins.TSGD2 are as defined earlier for Detect_Ins.Opt.

**Lemma 10:** If during its execution, Detect_Ins.TSGD1 is in state $St_k$, then after a finite number of steps, it enters a state $St'_k \equiv St_k$ such that no forward transitions from $St'_k$ are possible.

**Proof:** Similar to proof of Lemma 2. □

**Corollary 7:** Procedure Detect_Ins.TSGD1 terminates in $O(n_G^2 mn_S)$ steps.

**Proof:** Detect_Ins.TSGD1 can be shown to terminate as a result of Lemma 10 using a similar argument as in Corollary 3.

The number of steps Detect_Ins.TSGD1 terminates in is equal to the product of the number of times Detect_Ins.TSGD1 checks if an edge satisfies the conditions in Step 2 and the number of steps required to check if an edge satisfies the conditions in Step 2. Every time a transaction node is visited, the conditions in Step 2 need to be checked, on an average, for $v_S$ edges (the average number of site nodes a global transaction executes at is $v_S$), while every time a site node is visited, the conditions in Step 2 need to be checked for at most $n_G$ edges (since the number of transaction nodes in the TSGD is most $n_G$). Furthermore, every transaction node can be visited at most $v_S n_S$ times, while every site node can be visited at most $n_G n_S$ times (every node $v$ in the TSGD can be visited in a state $st$ at most once for every node $w$ such that edge $(v, w)$ is in the TSGD, and $F$ has at most $n_S$ states). Since there are $m$ site nodes and at most $n_G$ transaction nodes in the TSGD, the number of times Detect_Ins.TSGD1 checks if an edge satisfies the conditions in Step 2 is $n_G^2 mn_S + n_G v_S^2 n_S$. Since each of the conditions in Step 2 can be checked in constant time and $v_S \ll n_G$, $v_S < m$, Detect_Ins.TSGD1 terminates in $O(n_G^2 mn_S)$ steps. □

We now show that Detect_Ins.TSGD1 traverses edges in the TSGD in a manner that ensures it detects instantiations of regular terms.

**Lemma 11:** Let Detect_Ins.TSGD1($(V, E, D, L), v_1, v_2, set_1, RT$) return the set of dependent edges $\Delta_F$. If the TSGD $(V, E, D \cup \Delta_F)$ contains a path $(v_1, v_2), (v_3, v_4), \ldots, (v_{2n-3}, v_{2n-2}), (v_{2n-1}, v_{2n}), v_3, v_4$ such that for the regular term $RT$, $F = FA(RT)$, $state_F(init_st_F, (v_3, v_4), \ldots, (v_{2n-1}, v_{2n}))$ is defined, then during the execution of Detect_Ins.TSGD1, for all $i = 1, 2, 3, \ldots, n-1$, $(st, prev_\text{anc}(v_{2i+1}))$ is added to $V\cdot set(v_{2i+1})$, where $st = state_F(init_st_F, (v_3, v_4)) \ldots (v_{2i-1}, v_{2i})(v_{2i+1}, v_{2i+2})$.

**Proof:** We prove the above lemma by induction on $i$. We prove that for all $i = 1, 2, \ldots, n$, $(st, prev_\text{anc}(v_{2i+1}))$ is added to $V\cdot set(v_{2i+1})$, where $st = state_F(init_st_F, (v_3, v_4)) \ldots (v_{2i-1}, v_{2i})(v_{2i+1}, v_{2i+2})$.

**Basis (i = 1):** In Step 1 of Detect_Ins.TSGD1, $(init_st_F, v_1)$ is added to $V\cdot set(v_2)$. Since $v_2 = prev_\text{anc}(v_3) = v_1$, and $state_F(init_st_F, (v_3, v_4)) = init_st_F$, the lemma is true for $i = 1$ ($(init_st_F, prev_\text{anc}(v_3))$ is added to $V\cdot set(v_3)$).

**Induction:** Let us assume that the lemma is true for $i = m, 1 \leq m \leq n-1$. Thus, $(st_m, prev_\text{anc}(v_{2m+1}))$ is added to $V\cdot set(v_{2m+1})$, where $st_m = state_F(init_st_F, (v_3, v_4)) \ldots (v_{2m-1}, v_{2m})(v_{2m+1}, v_{2m+2})$. To show the lemma to be true for $i = m + 1$. Thus, we need to show that $(st_{m+1}, prev_\text{anc}(v_{2m+3}))$ is added to $V\cdot set(v_{2m+3})$, where $st_{m+1} = state_F(init_st_F, (v_3, v_4)) \ldots (v_{2m+1}, v_{2m+2})(v_{2m+3}, v_{2m+4})$. By the definition of $state_F$, $st_{m+1} = st_F(st_m, L(v_{2m+1}, v_{2m+2}))$, if $v_{2m+2} = v_{2m+3}$ and $st_{m+1} = st_F(st_m, L(v_{2m+1}, v_{2m+2}))$, if $v_{2m+1} = v_{2m+3}$.
$j \leq k \leq n$, the sequence of edges $(G_j, slast(t_j))(sfirst(t_{j+1})modn), G,(j+1)modn), \ldots$.

(sfirst(t_{k+1})modn), G_{k+1}modn) is in the path, then it must be the case that for all $j < l < k$, \text{arity}(t_l) = 1. Thus, by Property 1, it follows that slast(t_j) = sfirst(t_{j+1})modn) = \cdots = sfirst(t_{k+1})modn), and all $r, s, j \leq r < s \leq k$, $G_r \neq G_{smodn}$. Thus, $(G_0, slast(t_0))edge(t_1) \cdots edge(t_{n-1})(sfirst(t_0), G_0)$ is a path in the TSG $(V, E, L)$. Furthermore, if $\Delta_F$ is the set of site nodes returned by \text{Detect.Ins.TSG}, then for some $j = 0, 1, \ldots, n - 1$, if $sfirst(t_{j+1})modn) \in set_2 \cup \Delta_F$, then $G_{(j+1)modn} \neq G_0$ and $s_k = sfirst(t_{j+1})modn) = slast(t_j)$. If $s_k \in set_2 \cup \Delta_F$ and $G_{(j+1)modn} = G_0$, then $sfirst(t_{j+1})modn)$ in the queue is marked when $init_0$ is processed. Since $init_0$ is processed after $init_j$, $sfirst(t_{j+1})modn)$ is inserted into the queue for site $s_k$ before $sfirst(G_{j+1}modn)$ is inserted into the queue for $s_k$. The $sfirst(G_{j+1}modn)$ executes after $sfirst(G_j)$, and $sfirst(t_{j+1})modn) = G_{(j+1)modn}$ must be serial after last(t_j) = s_{j+1} at site $s_k$, which leads to a contradiction. Thus, $sfirst(t_0) \notin set_2 \cup \Delta_F$. Then the path $(G_0, slast(t_0))edge(t_1) \cdots edge(t_{n-1})(sfirst(t_0), G_0)$ is consistent with respect to $set_2 \cup \Delta_F$.

We further use Lemma 3 to show that, for $F = FA(RT_2)$, $state_F(init_st_F, edge(t_1) \cdots edge(t_{n-1})(sfirst(t_0), G_0))$ is an accept state. Let $edge(t_1) \cdots edge(t_{n-1})(sfirst(t_0), G_0) = (v_1, v_2) \cdots (v_{m-1}, v_m)$.

In order to use Lemma 3, we need to show that there exists a sequence $g_1 \cdots g_{m-1}$ such that

- if $v_{2i} = v_{2i+1}$, then $g_i = L(v_{2i-1}, v_{2i})$, and
- if $v_{2i-1} = v_{2i+1}$, then $g_i = L(v_{2i-1}, v_{2i})$, and

Therefore, $state_F(init_st_F, g_1 \cdots g_{m-1})$ is an accept state. We construct the sequence $g_1 \cdots g_{m-1}$ with the above properties as follows. For all $i = 1, \ldots, m - 1$, let $f_i = (type(hdr(t_i)), type(first(t_i)))$, if \text{arity}(t_i) = 1, else $f_i = (type(hdr(t_i)), type(first(t_i)))((type(hdr(t_i)), type(last(t_i))))$. Since $t_1 \cdots t_{n-1}$ is a string in $L(reg, exp)$, by the construction of $FA(RT_2)$, it follows that $state_F(init_st_F, f_1 \cdots f_{n-1})$ is an accept state. Let $g_1 \cdots g_{m-1} = f_1 \cdots f_{n-1}$, such that every $g_i \in \Sigma_F$. Furthermore, from the definition of $edge$ and $f_j$, it follows that, if for some $i = 1, \ldots, m - 1$, if $(v_{2i-1}, v_{2i}) \in edge(t_k)$ and \text{arity}(t_k) = 1, then $g_i = L(v_{2i-1}, v_{2i})$, else $g_i = L(v_{2i-1}, v_{2i})$.

In order to show that $state_F(init_st_F, (v_1, v_2), \ldots, (v_{m-1}, v_m))$ is an accept state, we need to show that for all $i = i, i = 1, 2, \ldots, m - 1$, if $v_{2i} = v_{2i+1}$, then $g_i = L(v_{2i-1}, v_{2i})$ and if $v_{2i-1} = v_{2i+1}$, then $g_i = L(v_{2i-1}, v_{2i})$. We first show that if $v_{2i} = v_{2i+1}$, and $(v_{2i-1}, v_{2i}) \in edge(t_k)$ for some $k, k = 1, 2, \ldots, n - 1$, then \text{arity}(t_k) = 2. Suppose \text{arity}(t_k) = 1. Since $sfirst(t_{k+1})modn) \neq G_k$, which leads to a contradiction. Thus, \text{arity}(t_k) = 2, and $g_i = L(v_{2i-1}, v_{2i})$. Also, it can be shown that if $v_{2i-1} = v_{2i+1}$, and $(v_{2i-1}, v_{2i}) \in edge(t_k)$, then \text{arity}(t_k) = 1. Suppose \text{arity}(t_k) = 2, then $v_{2i} = G_k$, then $v_{2i} = v_{2i+1} = G_k$, which leads to a contradiction. If $v_{2i-1} = G_k$, then since $sfirst(t_{k+1})modn) \neq G_k$, which leads to a contradiction. If $v_{2i-1} = G_k$, then since $sfirst(t_{k+1})modn) \neq G_k$, which leads to a contradiction. Thus, \text{arity}(t_k) = 1, and $g_i = L(v_{2i-1}, v_{2i})$.

Thus, by Lemma 3, $state_F(init_st_F, edge(t_1) \cdots edge(t_{n-1})(sfirst(t_0), G_0))$ is an accept state. Therefore, during the execution of $\text{Detect.Ins.TSG}\square(V, E, L), G_0, (sfirst(t_0), set_1, set_2, R)$, if $sfirst(t_0)$ is added to $\Delta$, and thus $sfirst(t_0) \in \Delta_F$. However, this leads to a contradiction since we showed earlier that $sfirst(t_0) \notin set_2 \cup \Delta_F$. Thus, every schedule $S$ is correct. □
Proof of Theorem 3: Suppose $S$ is not correct. Thus, there exists a regular term $RT$ in $R$ an instantiation $I$ of $RT$ in $S$. Let $G_0$ be the transaction in $I$ such that $init_0$ is processed after $init_0$ every other transaction $G_i$ in $I$ is processed. By Lemma 1, since $R$ is complete, there exists a regular term $RT_2 = \epsilon_0 : \text{reg exp}$ and an instantiation $t_0 : t_1 t_2 \cdots t_{n-1}$ of $RT_2$ in $S$ such that $hdr(t_0) = \epsilon_0$.

Thus,

- for all $j$, $j = 0, 1, \ldots, n-1$,
  1. $t_j \in \Sigma_S$ (without loss of generality, let $hdr(t_j) = G_j$), and
  2. $last(t_j)$ and $first(t_{(j+1)\mod n})$ execute at the same site, and $last(t_j)$ is serialized at $first(t_{(j+1)\mod n})$ at the site, and
- $type(t_0) = \epsilon_0$ and $type(t_1) \cdots type(t_{n-1})$ is a string in $L(\text{reg exp})$.

When $init_0$ is processed, the procedure Detect Ins TSG is invoked with arguments that include $TSG (V, E, L), G_0, lsl(t_0), set_1, set_2$ and $RT_2$ since $type(G_0) = hdr(\epsilon_0)$ and $type(last(t_0)) = last(E)$. Also, $sfirst(t_0) \in set_1$ (if $arity(t_0) = 1$, then since $sfirst(t_0) = lsl(t_0)$, $sfirst(t_0) \in set_1$ binary(t_0), then since $sfirst(t_0) \neq lsl(t_0)$, and $type(first(t_0)) = first(\epsilon_0)$, $first(t_0) \in set_1$. Furthermore, all the edges belonging to $G_0, \ldots, G_{n-1}$ are in the TSG when Detect Ins TSG is invoked.

In order to show this, we first show that $G_j$'s edges cannot be deleted from the TSG before $G_{(j+1)\mod n}$'s edges are deleted from the TSG, for all $j, j = 1, 2, \ldots, n-1$. Suppose, for some $j, j = 1, 2, \ldots, n-1$, $G_j$'s edges are deleted from the TSG before $G_{(j+1)\mod n}$'s edges are deleted from the TSG. Let $slast(t_j) =$ \text{SomeValue}.

Since $G_{jk}$ is serial after $G_{(j+1)\mod n}$, at site $s_k$, $ser_k(G_{(j+1)\mod n})$ executes before $ser_k(G_j)$. Then, since $G_{(j+1)\mod n}$'s edges are inserted into the TSG before $ser_k(G_{(j+1)\mod n})$ executes, while $G_{j}$'s edge are deleted after $ser_k(G_j)$ executes, $G_{(j+1)\mod n}$'s edges must be in the TSG when $G_j$'s edges are deleted (since we have assumed that $G_j$'s edges are deleted before $G_{(j+1)\mod n}$'s edges are deleted). However, this leads to a contradiction, since edges belonging to $G_j$ and $G_{(j+1)\mod n}$ are deleted together with $fin_i$ for some transaction $G_i$ is processed (due to the sequence of edges between $G_j$ and $G_{(j+1)\mod n}$, $(G_j, s_k)(s_k, G_{(j+1)\mod n})$, if for every transaction $G_k \in V$ such that there is a sequence of edges from $G_j$ to $G_k$ in the TSG, $val_k$ has been processed, then for every transaction $G_k \in V$ such that there is a sequence of edges from $G_{(j+1)\mod n}$ to $G_k$, $val_k$ must also have been processed). Thus, $G_j$'s edges are not deleted from the TSG before $G_j$'s edges are deleted, ..., $G_{n-1}$'s edges are not deleted from the TSG before $G_0$'s edges are deleted. By transitivity and since $G_0$'s edges are deleted only after $init_0$ has been processed, when Detect Ins TSG is invoked during the processing of $init_0$, the TSG $(V, E)$ contains all the edges belonging to transactions $G_0, G_1, \ldots, G_{n-1}$.

We now show that $(G_0, lsl(t_0))$ \text{edge}(t_1) \cdots \text{edge}(t_{n-1})(sfirst(t_0), G_0)$ is a path in the TSG $(V, E)$. We begin by showing that any two consecutive edges in the path have a common node. Consecutive edges in the path could be one of the following:

- $(sfirst(G_j), G_j)(G_j, lsl(G_j))$, $j = 1, 2, \ldots, n-1$, where $arity(t_j) = 2$ ($G_j$ is the common node).
- $(G_j, lsl(t_j))(sfirst(t_{(j+1)\mod n}), G_{(j+1)\mod n})$, $j = 0, 1, \ldots, n-1$, where $arity(t_j) = 2$ or $arity(t_{(j+1)\mod n}) = 1$ or 2 (since for all $j$, $j = 0, 1, \ldots, n-1$, $last(t_j)$ and $first(t_{(j+1)\mod n})$ execute at the same site, $last(t_j) = sfirst(t_{(j+1)\mod n})$ is the common node).
- $(sfirst(t_j), G_j)(sfirst(t_{(j+1)\mod n}), G_{(j+1)\mod n})$, $j = 1, 2, \ldots, n-1$, where $arity(t_j) = 1$, or $arity(t_{(j+1)\mod n}) = 1$ or 2 (since $arity(t_j) = 1$ implies that $sfirst(t_j) = lsl(t_j)$, and $sfirst(t_j)$ and $sfirst(t_{(j+1)\mod n})$, it follows that $sfirst(t_j) = sfirst(t_{(j+1)\mod n})$ is the common node).

Also, for the sequence of edges $(sfirst(t_j), G_j)(G_j, lsl(t_j))$ in the path, $j = 1, 2, \ldots, n-1$, it is
or $s t_{j,v} = \text{prev.anc}(v_{m+1})$, or due to Step 1). Thus, $s t_{k,v} = v_{m+1}$, $s t_{k,cur-st} = st_{m}$ and in state $s t_{k}$, $\text{head}(s t_{k,\text{anc}(s t_{k,v})}) = (\text{prev.anc}(v_{m+1}), v_{j})$. Furthermore, it follows from Lemma 8 that a finite number of steps, $\text{Detect.Ins.TSG2}$ is in a state $s t_{k}'$ such that $s t_{k}' \equiv s t_{k}$ and no further forward transitions can be made from $s t_{k}'$. Thus, in state $s t_{k}'$:

- Since $s t_{k}' \cdot \Delta \subseteq \Delta_F$, and $(v_3, v_4) \cdots (v_{m+1}, v_{m+2})$ is consistent with $s e t_2 \cup \Delta_F$, $(v_3, v_4) \cdots (v_{m+1}, v_{m+2})$ is consistent with $s e t_2 \cup s t_{k}' \cdot \Delta$; thus, if $s t_{k}' \cdot v \in (s e t_2 \cup s t_{k}' \cdot \Delta)$, then $v_{m+2} \neq v_{m+3}$.

However, since in state $s t_{k}'$, no forward transition can be made due to edge $(s t_{k}' \cdot v, v_{m+2})$, it must be the case that:

- if $v_{m+2} = v_{m+3}$, then either

  1. $s t_{k}' \cdot V \cdot \text{set}(v_{m+2})$ already contains $(s t_{m+1}, (s t_{k}' \cdot v, s t_{k}' \cdot v))$. Thus, since $s t_{k}' \cdot v = v_{m+1}$, $s t_{m+1} \neq v_{m+4}$, $\text{prev.anc}(v_{m+3}) = v_{m+1}$, $(s t_{m+1}, (\text{prev.anc}(v_{m+3}), v_{j}'))$ is added to $V \cdot \text{set}(v_{m+3})$ during the execution of $\text{Detect.Ins.TSG2}$, $v_{j}' \neq v_{m+4}$.

  2. $s t_{k}' \cdot V \cdot \text{set}(v_{m+2})$ already contains $(s t_{m+1}, (s t_{k}' \cdot v, u_2))$ and $(s t_{m+1}, (s t_{k}' \cdot v, u_3))$, $u_2 \neq u_3$. Thus, since $s t_{k}' \cdot v = v_{m+1}$, either $u_2 \neq v_{m+4}$ or $u_3 \neq v_{m+4}$ (since $u_2 \neq u_3$), $\text{prev.anc}(v_{m+3}) = v_{m+1}$, $(s t_{m+1}, (\text{prev.anc}(v_{m+3}), v_{j}'))$ is added to $V \cdot \text{set}(v_{m+3})$ during the execution of $\text{Detect.Ins.TSG2}$, $v_{j}' \neq v_{m+4}$.

- if $v_{m+1} = v_{m+3}$, then either

  1. $s t_{k}' \cdot V \cdot \text{set}(s t_{k}' \cdot v)$ already contains $(s t_{m+1}, (\text{prev.anc}(v_{m+1}, v_{m+2})))$. Thus, since $s t_{k}' \cdot v = v_{m+1}$, $v_{m+2} \neq v_{m+4}$, $\text{prev.anc}(v_{m+3}) = \text{prev.anc}(v_{m+1})$, $(s t_{m+1}, (\text{prev.anc}(v_{m+3}), v_{j}'))$ is added to $V \cdot \text{set}(v_{m+3})$ during the execution of $\text{Detect.Ins.TSG2}$, $v_{j}' \neq v_{m+4}$.

  2. $s t_{k}' \cdot V \cdot \text{set}(s t_{k}' \cdot v)$ already contains $(s t_{m+1}, (\text{prev.anc}(v_{m+1}, u_2))$ and $(s t_{m+1}, (\text{prev.anc}(v_{m+1}, u_3))$, $u_2 \neq u_3$. Thus, since $s t_{k}' \cdot v = v_{m+1}$, either $u_2 \neq v_{m+4}$ or $u_3 \neq v_{m+4}$ (since $u_2 \neq u_3$), $\text{prev.anc}(v_{m+3}) = \text{prev.anc}(v_{m+1})$, $(s t_{m+1}, (\text{prev.anc}(v_{m+3}), v_{j}'))$ is added to $V \cdot \text{set}(v_{m+3})$ during the execution of $\text{Detect.Ins.TSG2}$, $v_{j}' \neq v_{m+4}$. $\square$

**Corollary 6:** Let $\text{Detect.Ins.TSG2}((V, E, L, v_1, v_2, s e t_1, s e t_2, R T)$ return the set of site nodes $\Delta_F$. If the TSG $(V, E, L)$ contains a path $(v_1, v_2)(v_3, v_4) \cdots (v_{n-1}, v_{n})(v_{n+1}, v_1)$, $v_2 = v_3$, consistent with $s e t_2 \cup \Delta_F$, such that for the regular term $R T$, $F = F A(R T)$, $s t = s t a t e_F(\text{init.st.F}, (v_3, v_4) \cdots (v_{n-1}, v_{n})(v_{n+1}, v_1))$ is an accept state and $v_{n+1} \in s e t_1$, then during the execution of $\text{Detect.Ins.TSG2}$, $v_{n+1}$ is added to $\Delta$.

**Proof:** By Lemma 9, $(s t, (\text{prev.anc}(v_{n+1}), v_j))$ is added to $V \cdot \text{set}(v_{n+1})$, where $v_j \neq v_1$. Since $\text{prev.anc}(v_{n+1}) \neq v_1$ and $v_j \neq v_1$, $\text{Detect.Ins.TSG2}$ makes a forward state transition when $(s t, (\text{prev.anc}(v_{n+1}), v_j))$ is added to $V \cdot \text{set}(v_{n+1})$. However, just before $(s t, (\text{prev.anc}(v_{n+1}), v_j))$ is added to $V \cdot \text{set}(v_{n+1})$, since $s t$ is an accept state, $\text{prev.anc}(v_{n+1}) \neq v_1$, $v_j \neq v_1$ and $v_{n+1} \in s e t_1$, $v_{n+1}$ is added to $\Delta$. $\square$
Corollary 5: Procedure Detect$_{Ins}$TSG2 terminates in $O(n_G^2 m n_S)$ steps.

Proof: Detect$_{Ins}$TSG2 can be shown to terminate as a result of Lemma 8 using a similar argument as in Corollary 3.

The number of steps Detect$_{Ins}$TSG2 terminates in is equal to the product of the number of times Detect$_{Ins}$TSG2 checks if an edge satisfies the conditions in Step 2 and the number of steps required to check if an edge satisfies the conditions in Step 2. Every time a transaction node is visited, the conditions in Step 2 need to be checked, on average, for $v_S$ edges (the average number of sites a global transaction executes at is $v_S$), while every time a site node is visited, the conditions in Step 2 need be checked for at most $n_G$ edges (since the number of transaction nodes in the TSG is at most $n_G$). Furthermore, every transaction node can be visited at most $2v_S n_S$ times, while every site node can be visited at most $2n_G n_S$ times (every node $v$ in the TSG can be visited in a state $st$ of $F$ at most twice every node $w$ such that edge $(v, w)$ is in the TSG, and $F$ has at most $n_S$ states). Since there are $m$ nodes and at most $n_G$ transaction nodes in the TSG, the number of times Detect$_{Ins}$TSG2 checks if an edge satisfies the conditions in Step 2 is $2n_G^2 m n_S + 2n_G v_S n_S$. Since each of the conditions in Step 2 can be checked in constant time and $v_S \ll n_G, v_S < m$, Detect$_{Ins}$TSG2 terminates in $O(n_G^2 m n_S)$ steps.

In order to show that Detect$_{Ins}$TSG2 traverses edges in the TSG in a manner that ensures it detects instantiations of regular terms, we define the following.

Definition 13: Consider a TSG/TSGD containing a path $(v_1, v_2)(v_3, v_4)\cdots(v_{2n-1}, v_{2n})$, $v_2 \neq v_1$. For all $i = 1, 2, \ldots, n-1$, $prev_{anc}(v_{2i+1})$ is defined as follows.

$$prev_{anc}(v_{2i+1}) = \begin{cases} prev_{anc}(v_{2i-1}) & \text{if } v_{2i-1} = v_{2i+1} \\ v_{2i-1} & \text{if } v_{2i} = v_{2i+1} \end{cases} \square$$

Note that, by the definition of path, it follows that for all $i, i = 1, 2, \ldots, n-1, v_{2i+2} \neq prev_{anc}(v_{2i+1})$ and dependency $(prev_{anc}(v_{2i+1}), v_{2i+1}) \rightarrow (v_{2i+1}, v_{2i+2})$ does not belong to the TSGD.

Lemma 9: Let Detect$_{Ins}$TSG2($TSG, v_1, v_2, set_1, set_2, RT$) return the set of nodes $\Delta_F$. If TSG ($V, E, L$) contains a path $(v_1, v_2), (v_3, v_4), \ldots, (v_{2n-3}, v_{2n-2}), (v_{2n-1}, v_{2n})$, $v_2 = v_3$, consistent with respect to $set_2 \cup \Delta_F$, such that for the regular term $RT$, $F = FA(RT)$, $state_F(init_{st_F}, (v_3, v_4), \ldots, (v_{2n-1}, v_{2n}))$ is defined, then during the execution of Detect$_{Ins}$TSG2, for all $i, i = 1, 2, 3, \ldots, n$, there exists a node $v_j$, $v_j \neq v_{2i+2}$, such that $state_F(init_{st_F}, (v_3, v_4)\cdots(v_{2i-1}, v_{2i})(v_{2i+1}, v_{2i+2}))$.

Proof: We prove the above lemma by induction on $i$. We prove that for all $i, i = 1, 2, \ldots, n$, there exists a $v_j \neq v_{2i+2}$, such that $state_F(init_{st_F}, (v_3, v_4)\cdots(v_{2i-1}, v_{2i})(v_{2i+1}, v_{2i+2}))$.

Basis ($i = 1$): In Step 1 of Detect$_{Ins}$TSG2, $(init_{st_F}, (v_1, v_1))$ is added to $V_set(v_2)$. Since $v_2 = prev_{anc}(v_3) = v_1$, $v_1 \neq v_4$, and $state_F(init_{st_F}, (v_3, v_4)) = init_{st_F}$, the lemma is true for $i = 1$.

Induction: Let us assume that the lemma is true for $i = m, 1 \leq m < n - 1$. Thus, $(st_{m+1}, prev_{anc}(v_{2m+1}), v_j))$ is added to $V_set(v_{2m+1})$, where $v_j \neq v_{2m+2}$, $st_{m+1} = state_F(init_{st_F}, (v_3, v_4)\cdots(v_{2m-1}, v_{2m})(v_{2m+1}, v_{2m+2}))$. We show the lemma to be true for $i = m + 1$. Thus, we need to show that $(st_{m+1}, prev_{anc}(v_{2m+3}), v'_j)$ is added to $V_set(v_{2m+3})$, where $v'_j \neq v_{2m+4}$, $st_{m+1} = state_F(init_{st_F}, (v_3, v_4)\cdots(v_{2m-1}, v_{2m})(v_{2m+1}, v_{2m+2}))$. By the definition of $state_F, st_{m+1} = state_F(L(v_{2m+1}, v_{2m+2}))$, if $v_{2m+2} = v_{2m+3}$ and $st_{m+1} = state_F(L(v_{2m+1}, v_{2m+2}))$, if $v_{2m+3} = v_{2m+1}$. Let St$_1$ be the state of Detect$_{Ins}$TSG2 after $(st_{m+1}, prev_{anc}(v_{2m+3}), v_j)$ is added.
Corollary 4: Let Detect\_Ins\_TSG1((V, E, L), v_1, v_2, set_1, set_2, RT). return the set of site not in \(\Delta F\). If the TSG \((V, E, L)\) contains a path \((v_1, v_2, v_3, v_4) \cdots (v_{n-1}, v_n, v_1)\), then \(v_2 \in \Delta F\) with \(\Delta F \cup \Delta F\), such that for the regular term \(RT\), \(F = FA(RT)\), \(st = state_F\) \((v_3, v_4) \cdots (v_{n-1}, v_n, v_1)\) is an accept state and \(v_{n+1} \in set_1\), then during the execution of Detect\_Ins\_TSG1, \(v_{n+1}\) is added to \(\Delta\).

Proof: By Lemma 7, \((st, v_j)\) is added to \(V\_set(v_{n+1})\), where \(v_j \neq \text{foll}(v_{n+1})\). Since \(\text{foll}(v_{n+1}) \neq v_1\) and Detect\_Ins\_TSG1 makes a forward state transition when \((st, v_j)\) is added to \(V\_set(v_{n+1})\). However, just before \((st, v_j)\) is added to \(V\_set(v_{n+1})\), since \(st\) is an accept state, \(v_j \neq v_1\) \(v_{n+1} \in set_1\), \(v_{n+1}\) is added to \(\Delta\). \(\square\)

We now show that Detect\_Ins\_TSG2 terminates in \(O(n^2 G m v_S)\) steps, for which we need to prove the following lemma.

Lemma 8: If during its execution, Detect\_Ins\_TSG2 is in state \(St_k\), then after a finite number of steps, it enters a state \(St'_k \equiv St_k\) such that no forward transitions from \(St'_k\) are possible.

Proof: We prove the lemma by induction on \(num\), the number of elements in \((st, v_1, v_2, v_3, v_4) \in F \wedge (v_1, v_2, v_3) \in V \wedge ((st, (v_1, v_2)) \notin V\_set(v_3))\) in state \(St_k\).

Basis \((num = 0)\): If \(num = 0\) in state \(St_k\), then, in state \(St_k\), for every edge \((St_k, v, u)\), if \(st_F((St_k, cur\_st, L(St_k, v, u)))\) is defined, then \((st, (St_k, v, St_k, v)) \in St_k, V\_set(u)\) (alternatively, if \(st_F((St_k, cur\_st, L(St_k, v, u)))\) is defined, then \((st', \text{head}(St_k, anc\_st(St_k, v))[1], u) \in St_k, V\_set(St_k, v))\). No forward transition can be made from state \(St_k\) (since every edge \((St_k, v, u)\) satisfies the third condition in Step 2).

Induction: Let us assume the lemma is true for \(num \leq m\), \(m \geq 0\). We show that the lemma is true if \(num \leq m + 1\) in state \(St_k\). We show that after a finite number of moves, Detect\_Ins\_TSG2 is in a state \(St'_k\) such that \(St'_k \equiv St_k\) and no forward transitions can be made from state \(St'_k\).

Let \(St''_k\) be any state equivalent to \(St_k\) such that in \(St''_k\), \(num \leq m + 1\). If Detect\_Ins\_TSG2 makes the forward transition \(St''_k \rightarrow St_l\) due to some edge \((St'_l, v, u)\) and \(L(St''_k, v, u)\), then it must be the case that \(St_l, v = u\), \(St_l, cur\_st = st_F((St''_k, cur\_st, L(St''_k, v, u)))\). Furthermore, in state \(St_l\), \((St_l, cur\_st, (St''_k, v, St''_k, v)) \notin St''_k, V\_set(u)\) and in state \(St_l\), \((St_l, cur\_st, (St''_k, v, St''_k, v)) \in St_l, V\_set(u)\) (since the transition \(St''_k \rightarrow St_l\) causes \((St_l, cur\_st, (St''_k, v, St''_k, v))\) to be added to \(V\_set(u)\)). Note that since before the transition is made, \((St_l, cur\_st, (St''_k, v, St''_k, v))\) does not belong to \(V\_set(u)\) and \(num \leq m + 1\) in \(St''_k\), after the transition \(St''_k \rightarrow St_l\), \(num \leq m\) in \(St_l\). By IH, after a finite number of steps, Detect\_Ins\_TSG2 enters a state \(St'_l \equiv St_l\), such that no forward transitions are possible from \(St'_l\).

Thus, Detect\_Ins\_TSG2 makes the reverse transition \(St'_l \rightarrow St''_k\) after a finite number of steps, where \(St''_k \equiv St''_k \equiv St_k\). Furthermore, in state \(St''_k\), \((St_l, cur\_st, (St''_k, v, St''_k, v)) \in St''_k, V\_set(u)\) and \(L(St''_k, v, u)\), and thus, no forward transition can be made from state \(St''_k\) due to edge \((St''_k, v, u)\) \((L(St''_k, v, u)\) (edge \((St''_k, v, u)\) does not satisfy the condition in Step 3(b)). Using a similar argument, it can be shown that if Detect\_Ins\_TSG2 makes a forward transition \(St''_k \rightarrow St_l\) due to edge \((St''_k, v, u)\) \((L(St''_k, v, u)\), then in a finite number of steps, Detect\_Ins\_TSG2 enters a state \(St''_k \equiv St_k\) such that no forward transitions are possible from \(St''_k\) due to edge \((St''_k, v, u)\) and \((L(St''_k, v, u)\).

Thus, once a forward transition is made by Detect\_Ins\_TSG2 due to an edge \(e\) and \((L(e)/L(e))\) from a state equivalent to \(St_k\), then no further forward transitions can be made by Detect\_Ins\_TSG2 due to \(e\) and \((L(e)/L(e))\) from any state equivalent to \(St_k\). Furthermore, every time a forward transition is made from a state \(St''_k\) that is equivalent to \(St_k\) such that \(num \leq m + 1\) in \(St''_k\), a reverse transition is made by Detect\_Ins\_TSG2 to a state \(St''_k\) equivalent to \(St_k\) such that \(num \leq m + 1\) in \(St''_k\). Since there are a finite number of edges incident on each node in state \(St_k\), \(num \leq m + 1\), eventually, Detect\_Ins\_TSG2 would be in a state \(St'_k \equiv St_k\), such that no further forward transitions can be made.
1, there exists a node \( v_j, v_j \neq \text{foll}(v_{2i+1}) \), such that \((st, v_j)\) is added to \( V_{set}(v_{2i+1}) \), where \( st = \text{state}_F(init_{st}, (v_3, v_4) \cdots (v_{2i-1}, v_{2i}))(v_{2i+1}, v_{2i+2}) \).

**Proof:** We prove the above lemma by induction on \( i \). We prove that for all \( i, i = 1, 2, \ldots, n \), there exists a \( v_j \neq \text{foll}(v_{2i+1}) \), such that \((st, v_j)\) is added to \( V_{set}(v_{2i+1}) \), where \( st = \text{state}_F(init_{st}, (v_3, v_4) \cdots (v_{2i-1}, v_{2i}))(v_{2i+1}, v_{2i+2}) \).

**Basis \((i = 1)\):** In Step 1 of Detect_Ins-TSG1, \((init_{st}, v_1)\) is added to \( V_{set}(v_2) \). Since \( v_2 = (st, v_j) \), \( v_1 \neq \text{foll}(v_3) \), and \( \text{state}_F(init_{st}, (v_3, v_4)) = \text{init}_{st} \), the lemma is true for \( i = 1 \) ((\( init_{st}, v_j \) added to \( V_{set}(v_3) \), \( v_j \neq \text{foll}(v_3) \)).

**Induction:** Let us assume that the lemma is true for \( i = m, 1 \leq m < n - 1 \). Thus, \((st_m, v_j)\) is added to \( V_{set}(v_{2m+1}) \), where \( v_j \neq \text{foll}(v_{2m+1}) \), \( st_m = \text{state}_F(init_{st}, (v_3, v_4) \cdots (v_{2m+1}, v_{2m+2})) \). We show the lemma to be true for \( i = m + 1 \). Thus, we need to show that \((st_{m+1}, v'_j)\) is added to \( V_{set}(v_{2m+2}) \), where \( v'_j \neq \text{foll}(v_{2m+3}) \), \( st_{m+1} = \text{state}_F(init_{st}, (v_3, v_4) \cdots (v_{2m+3}, v_{2m+4})) \).

By the definition \( st_{m+1} = st_F(st_m, L(v_{2m+1}, v_{2m+2})) \), if \( v_{2m+2} = v_{2m+2} \) and \( st_{m+1} = st_F(st_m, L(v_{2m+1}, v_{2m+2})) \) if \( v_{2m+1} = v_{2m+3} \).

Let \( St_k \) be the resulting state of Detect_Ins-TSG1 after \((st_m, v_j)\) is added to \( V_{set}(v_{2m+1}) \). The state \( St_k \) results either due to the forward transition \( St_j \rightarrow St_k \), either \( St_j.v = v_{2m+1} \) or \( St_j.v = v_j \) (due to Step 1). Thus, \( St_k.v = v_{2m+1} \), \( St_k.cur.st = st_m \) and in state \( St_k \), \( head(St_k.anc(St_k.v)) \).

Furthermore, it follows from Lemma 6 that after a finite number of steps, Detect_Ins-TSG1 is in a state \( St'_k \) such that \( St'_k \equiv St_k \) and no further forward transitions can be made from \( St'_k \). Thus, in state \( St'_k \).

- Since \( \text{state}_F(init_{st}, (v_3, v_4) \cdots (v_{2m+1}, v_{2m+2}))(v_{2m+3}, v_{2m+4}) \) is defined, if \( v_{2m+2} = v_{2m+3} \), then \( st_{m+1} = st_F(st'_k.cur.st, L(st'_k.v, v_{2m+2})) \) is defined, else if \( v_{2m+1} = v_{2m+3} \), then \( st_{m+1} = st_F(st'_k.cur.st, L(st'_k.v, v_{2m+2})) \) is defined.

- Since \( st'_k.\Delta \subseteq \Delta_F \), and \((v_3, v_4) \cdots (v_{2m+1}, v_{2m+2}) \) is consistent with \( set_2 \cup \Delta_F \), \((v_3, v_4) \cdots (v_{2m+1}, v_{2m+2}) \) is consistent with \( set_2 \cup st'_k.\Delta \); thus, if \( St'_k.v \in (set_2 \cup st'_k.\Delta) \), then \( v_{2m+2} \neq \text{foll}(v_{2m+2}) \).

However, since in state \( St'_k \), no forward transition can be made due to edge \((St'_k.v, v_{2m+2}) \), it must be the case that

- if \( v_{2m+2} = v_{2m+3} \), then \( \text{foll}(v_{2m+1}) = v_{2m+2} \) and since \( v_j \neq \text{foll}(v_{2m+1}) \) (by the definition of \( path \), \( head(St'_k.anc(St'_k.v)) \) \neq v_{2m+2} \), and thus, either
  1. \( St'_k.V.set(v_{2m+2}) \) already contains \((st_{m+1}, St'_k.v)\). Thus, since \( St'_k.v = v_{2m+1} \), \( v_{2m+2} \neq \text{foll}(v_{2m+3}) \), \((st_{m+1}, v'_j) \) is added to \( V.set(v_{2m+3}) \) during the execution of Detect_Ins_TSG1, \( v'_j \neq \text{foll}(v_{2m+3}) \).
  2. \( St'_k.V.set(v_{2m+2}) \) already contains \((st_{m+1}, u_2) \) and \((st_{m+1}, u_3) \), \( u_2 \neq u_3 \). Thus, since either \( u_2 \neq \text{foll}(v_{2m+3}) \) or \( u_3 \neq \text{foll}(v_{2m+3}) \) (since \( u_2 \neq u_3 \), \((st_{m+1}, v'_j) \) is added to \( V.set(v_{2m+3}) \) during the execution of Detect_Ins_TSG1, \( v'_j \neq \text{foll}(v_{2m+3}) \).

- if \( v_{2m+1} = v_{2m+3} \), then either
  1. \( St'_k.V.set(St'_k.v) \) already contains \((st_{m+1}, v_j)\). Thus, since \( St'_k.v = v_{2m+1} \), \( v_{2m+1} \neq \text{foll}(v_{2m+3}) \), \((st_{m+1}, v'_j) \) is added to \( V.set(v_{2m+3}) \) during the execution of Detect_Ins_TSG1, \( v'_j \neq \text{foll}(v_{2m+3}) \).
  2. \( St'_k.V.set(St'_k.v) \) already contains \((st_{m+1}, u_2) \) and \((st_{m+1}, u_3) \), \( u_2 \neq u_3 \). Thus, since either \( u_2 \neq \text{foll}(v_{2m+3}) \) or \( u_3 \neq \text{foll}(v_{2m+3}) \) (since \( u_2 \neq u_3 \), \((st_{m+1}, v'_j) \) is added to \( V.set(v_{2m+3}) \) during the execution of Detect_Ins_TSG1, \( v'_j \neq \text{foll}(v_{2m+3}) \). □
Appendix -C- : TSG Schemes

In this appendix, we prove Theorem 3. We begin by showing that Detect.Ins.TSG1 and Detect.Ins.TSG2 detect instantiations of regular terms in $S$. States $S_{tk}$ between the execution of any two steps of Detect.Ins.TSG1 and Detect.Ins.TSG2 are as defined earlier for Detect.Ins.Opt.

Lemma 6: If during its execution, Detect.Ins.TSG1 is in state $S_{tk}$, then after a finite number of steps, it enters a state $S_{tk} \equiv S_{tk}$ such that no forward transitions from $S_{tk}$ are possible.

Proof: Similar to proof of Lemma 2. □

Corollary 3: Procedure Detect.Ins.TSG1 terminates in $O(n_G mn_S)$ steps.

Proof: We first show that Detect.Ins.TSG1 terminates in a finite number of steps. Let $S_{t1}$ denote the state immediately after the execution of Step 1 of algorithm Detect.Ins.TSG1. By Lemma 5; after a finite number of steps, Detect.Ins.TSG1 is in a state $S_{t1} \equiv S_{t1}$ such that no further forward transitions can be made from $S_{t1}$. Detect.Ins.TSG1, thus executes Step 4 and since, in state $\text{head}(S_{t1}, F, List(S_{t1}, v)) = (s^*, G_i)$, Detect.Ins.TSG1 terminates in a finite number of steps.

The number of steps Detect.Ins.TSG1 terminates in is equal to the product of the number of times Detect.Ins.TSG1 checks if an edge satisfies the conditions in Step 2 and the number of steps required to check if an edge satisfies the conditions in Step 2. Every time a transaction node is visited, conditions in Step 2 need to be checked on a average, for $v_S$ edges (the average number of site-local transaction executes at $v_S$), while every time a site node is visited, the conditions in Step 2 need to be checked for at most $n_G$ edges (since the number of transaction nodes in the TSG is at most $n_G$). Furthermore, every transaction and site node can be visited at most $2n_S$ times (every node in the TSG can be visited in a state $st$ of $F$ at most twice, and $F$ has at most $n_S$ states). Since there are $n$ site nodes and at most $n_G$ transaction nodes in the TSG, the number of times Detect.Ins.TSG1 checks if an edge satisfies the conditions in Step 2 is $2n_G mn_S + 2n_G v_S n_S$. Since each of the conditions in Step 2 can be checked in constant time and $v_S < m$, Detect.Ins.TSG1 terminates in $O(n_G mn_S)$ steps.

In order to show that Detect.Ins.TSG1 traverses edges in the TSG in a manner that ensures it detects instantiations of regular terms, we define the following.

Definition 11: Consider a TSG containing a path $(v_1, v_2)(v_3, v_4) \cdots (v_{2n-1}, v_{2n})$. For all $i = 1, 2, \ldots, n-1$, we define $\text{foll}(v_{2i-1})$ as follows.

$$\text{foll}(v_{2i-1}) = \begin{cases} \text{foll}(v_{2i+1}) & \text{if } i < n \text{ and } v_{2i-1} = v_{2i+1} \\ v_{2i} & \text{if } i = n \text{ or } v_{2i} = v_{2i+1} \end{cases}$$

Note that, by the definition of path, for all $i = 1, 2, \ldots, n-1$, if $v_{2i} = v_{2i+1}$, then $v_{2i-1} \neq \text{foll}(v_{2i-1})$.

Definition 12: Consider a TSG containing a path $(v_1, v_2) \cdots (v_{2n-1}, v_{2n})$. The path is said to be consistent with a set of nodes $\text{set}$ if for all $i, i = 1, \ldots, n$, if $v_{2i-1} \in \text{set}$, then $v_{2i} \neq v_1$. □

Lemma 7: Let Detect.Ins.TSG1($(V, E, L), v_1, v_2, \text{set}_1, \text{set}_2, RT$) return the set of site nodes $S$. If the TSG $(V, E, L)$ contains a path $(v_1, v_2), (v_3, v_4), \ldots, (v_{2n-3}, v_{2n-2}), (v_{2n-1}, v_{2n})$, $v_2 = v_3$, consistent with $\text{set}_2 \cup \Delta_F$ such that for the regular term $RT, F = FA(RT)$, $\text{state}_F(\text{init}_st_F, (v_3, v_4), \ldots, (v_{2n-1}, v_{2n}))$ is defined, then during the execution of Detect.Ins.TSG1, for all $i, i = 1, 2, 3, \ldots, n$, if $v_{2i-1} \in \text{set}_1$, then $v_{2i} \neq v_1$. □
\( (sfirst(G_j), G_j, slast(G_j)), j = 1, 2, \ldots, n - 1, \) where \( \text{arity}(t_j) = 2 \) (\( G_j \) is the common node).

\( (G_j, slast(t_j))(sfirst(t_{(j+1) mod n}), G_{(j+1) mod n}, j = 0, 1, \ldots, n - 1, \) where \( \text{arity}(t_{(j+1) mod n}) = 1 \) or 2 (since for all \( j, j = 0, 1, \ldots, n - 1, \) \( sfirst(t_{(j+1) mod n}) \) execute at the same site, \( slast(t_j) = sfirst(t_{(j+1) mod n}) \) is the common node).

\( (sfirst(t_j), G_j)(sfirst(t_{(j+1) mod n}), G_{(j+1) mod n}, j = 1, 2, \ldots, n - 1, \) where \( \text{arity}(t_j) = 1, \) and
\( t_{(j+1) mod n}) = 1 \) or 2 (since \( \text{arity}(t_j) = 1 \) implies that \( sfirst(t_j) = slast(t_j) \), and \( slast(t_j) \) \( sfirst(t_{(j+1) mod n}) \), it follows that \( sfirst(t_j) = sfirst(t_{(j+1) mod n}) \) is the common node).

Also, for the sequence of edges \( (sfirst(t_j), G_j)(sfirst(t_{(j+1) mod n}), Г_{(j+1) mod n}) \) in the path, \( j = 1, 2, \ldots, n - 1, \) it may be the case that \( \text{arity}(t_j) = 2, \) and thus \( sfirst(t_j) \neq slast(t_j) \). Also, if for some \( j, k, j = 0, 1, \ldots, n - 1, \) \( j < k < n \), the sequence of edges \( (sfirst(t_j), G_j)(sfirst(t_{(j+1) mod n}), G_{(j+1) mod n}) \), \( \ldots \), \( (sfirst(t_{(j-1) mod n}), G_{(j-1) mod n}) \) is in the path, then it must be the case that for all \( j < l < k, \) \( \text{arity}(t_l) = 1 \).

Thus, Property 1, it follows that \( slast(t_j) = sfirst(t_{(j+1) mod n}) = \ldots = sfirst(t_{(j-1) mod n}), \) and for \( r, s, j \leq r < s \leq k, \)
\( G_r \neq G_{s mod n}, \) and
\( G_r \) is serialized after \( G_{s mod n} \) at site \( sfirst(G_{s mod n}). \) Thus, by Lemma 5, dependency
\( (G_r, sfirst(G_{s mod n})) \rightarrow (sfirst(G_{s mod n}), G_{s mod n}) \) does not belong to \( D'. \)

Thus, \( (G_0, slast(t_0))(edge(t_1) \cdots edge(t_{n-1}))(sfirst(t_0), G_0) \) is a path in the TSGD \( (V', E', D', L'). \)

We further use Lemma 3 to show that, for \( F = FA(RT_2), state_F(init_{stF}, edge(t_1) \cdots edge(t_{n-1}))(sfirst(t_0), G_0) \) is an accept state. Let \( edge(t_1) \cdots edge(t_{n-1}))(sfirst(t_0), G_0) = (v_1, v_2) \cdots (v_{m-2}, v_m), \)

In order to use Lemma 3, we need to show that there exists a sequence \( g_1 \cdots g_{m-1} \) such that

- if \( v_{2i} = v_{2i+1}, \) then \( g_i = L(v_{2i-1}, v_{2i}), \) and
- if \( v_{2i-1} = v_{2i+1}, \) then \( g_i = L(v_{2i-1}, v_{2i}), \) and

\( stF(init_{stF}, g_1 \cdots g_{m-1}) \) is an accept state. We construct the sequence \( g_1 \cdots g_{m-1} \) with the above properties as follows. For all \( i = 1, \ldots, n - 1, \) let
\( f_i = \{ type(hdr(t_i)), type(first(t_i)) \}, \) if \( \text{arity}(t_i) = 1 \)
\( f_i = \{ type(hdr(t_i)), type(first(t_i)), type(last(t_i)) \}. \) Since \( type(t_1) \cdots type(t_{n-1}) \) is a string in \( L(reg.exp), \) by the construction of \( FA(RT_2), \) it follows that \( stF(init_{stF}, f_1 \cdots f_{n-1}) \) is an accept state. Let \( g_1 \cdots g_{m-1} = f_1 \cdots f_{n-1}, \) such that every \( g_i \in \Sigma_F. \) Furthermore, from the definition of \( edge \) and \( f_i, \) it follows that, for some \( i = 1, \ldots, m - 1, \) if \( (v_{2i-1}, v_{2i}) \in edge(t_k) \) and \( \text{arity}(t_k) = 1 \) then \( g_i = L(v_{2i-1}, v_{2i}), \) else \( g_i = L(v_{2i-1}, v_{2i}). \)

In order to show that \( state_F(init_{stF}, (v_1, v_2), \ldots, (v_{m-2}, v_{m}))) \) is an accept state, we need to show that for all \( i, i = 1, 2, \ldots, m - 1, \) if \( v_{2i} = v_{2i+1}, \) then \( g_i = L(v_{2i-1}, v_{2i}) \) and if \( v_{2i-1} = v_{2i+1}, \) then \( g_i = L(v_{2i-1}, v_{2i}). \) We first show that if \( v_{2i} = v_{2i+1}, \) and \( (v_{2i-1}, v_{2i}) \in edge(t_k) \) for some \( k, k = 1, 2, \ldots, n - 1, \) then \( \text{arity}(t_k) = 1. \) Since \( last(t_k) \) and \( first(t_{(k+1) mod n}) \) execute at the same site, \( slast(t_k) = v_{2i-1}, sfirst(t_{(k+1) mod n}) = v_{2i+1}, \) it follows that \( v_{2i-1} = v_{2i+1} \), which leads to a contradiction. Thus, \( \text{arity}(t_k) = 2, \) and \( g_i = L(v_{2i-1}, v_{2i}). \) Also, it can be shown that if \( v_{2i-1} = v_{2i+1}, \) and \( (v_{2i-1}, v_{2i}) \in edge(t_k), \) then \( \text{arity}(t_k) = 1. \) Suppose \( \text{arity}(t_k) = 2, \) then \( v_{2i-1} = v_{2i+1}, \) then \( v_{2i+1} = G_k, \) then \( v_{2i} = v_{2i+1} = G_k, \) which leads to a contradiction. If \( v_{2i-1} = G_k, \) then \( last(t_k) \) and \( first(t_{(k+1) mod n}) \) execute at the same site, \( slast(t_k) = v_{2i}, sfirst(t_{(k+1) mod n}) = v_{2i+1}, \) it follows that \( v_{2i} = v_{2i+1}, \) which leads to a contradiction. Thus, \( \text{arity}(t_k) = 1, \) and \( g_i = L(v_{2i-1}, v_{2i}). \)

Thus, by Lemma 3, \( state_F(init_{stF}, edge(t_1) \cdots edge(t_{n-1}))(sfirst(t_0), G_0)) \) is an accept state. Then, by Corollary 2, \( Detect_{Ins}Opt((V', E', D', L'), G_0, slast(t_0), set_1, RT_2) \) returns abort and \( G_0 \) is aborted by the optimistic scheme. However, this leads to a contradiction since \( G_0 \) is a transaction in \( I \) that cannot be committed. Therefore, the claim is proven.
dependency is added during the execution of $act(ser_k(G_i))$, then $act(ser_k(G_j))$ must have already executed. On the other hand, if the dependency were added to the TSGD before $act(ser_k(G_i))$ executed, then $act(ser_k(G_j))$ would not execute until $act(ack(ser_k(G_j)))$ completes execution (the dependency $(G_j, s_k) \rightarrow (s_k, G_i)$ is deleted from the TSGD only after $ack(ser_k(G_j))$ is processed). Thus, in both cases $ser_k(G_j)$ executes before $ser_k(G_i)$, and thus, $G_{jk}$ is serialized before $G_{ik}$ at site $s_k$, which leads to a contradiction. □

For an element $t_i \in \Sigma_S$, we denote by $\text{slast}(t_i)$ and $\text{sfist}(t_i)$, the sites at which $\text{last}(t_i)$ and $\text{first}(t_i)$ execute, respectively. Also, if $\text{arity}(t_i) = 1$, then $\text{edge}(t_i) = (\text{sfist}(t_i), \text{hdr}(t_i)), \text{edge}(t_i) = (\text{sfist}(t_i), \text{hdr}(t_i))(\text{hdr}(t_i), \text{slast}(t_i))$.

**Proof of Theorem 1:** Suppose $S$ is not correct. Thus, there exists a regular term $RT$ in $R$ an instantiation $I$ of $RT$ in $S$. Let $G_0$ be the transaction in $I$ such that $val_0$ is processed after $val_i$ every other transaction $G_i$ in $I$ is processed. By Lemma 1, since $R$ is complete, there exists a regular term $RT_2 = e_0 : \text{reg.exp}$ and an instantiation $t_0 : t_1t_2 \cdots t_{n-1}$ of $RT_2$ in $S$ such that $\text{hdr}(t_0) = e_0$. Thus,

- for all $j, j = 0, 1, \ldots, n-1$,
  1. $t_j \in \Sigma_S$ (without loss of generality, let $\text{hdr}(t_j) = G_j$), and
  2. $\text{last}(t_j)$ and $\text{first}(t_{(j+1) \mod n})$ execute at the same site, and $\text{last}(t_j)$ is serialized at
    $\text{first}(t_{(j+1) \mod n})$ at the site, and

- type($t_0$) = $e_0$ and type($t_1$) \cdots type($t_{n-1}$) is a string in $L(\text{reg.exp})$.

When $val_0$ is processed, Detect_Ins_Opt is invoked with arguments that include the TSGD $(V', E', L')$, $G_0$, $\text{slast}(t_0)$, $set_1$, and $RT_2$ since $type(G_0) = \text{hdr}(e_0)$ and $type(\text{last}(t_0)) = \text{last}(e_0)$. At $\text{sfist}(t_0) \in set_1$ (if $\text{arity}(t_0) = 1$, then since $\text{sfist}(t_0) = \text{slast}(t_0)$, $\text{sfist}(t_0) \in set_1$; if $\text{arity}(t_0) \geq 2$, then since $\text{sfist}(t_0) \neq \text{slast}(t_0)$, and $\text{type}(\text{first}(t_0)) = \text{first}(e_0)$, $\text{sfist}(t_0) \in set_1$). Furthermore, the edges belonging to $G_0, \ldots, G_{n-1}$ are in the TSGD when Detect_Ins_Opt is invoked. In order to show this, we first show that $G_j$’s edges cannot be deleted from the TSGD before $G_{(j+1) \mod n}$’s edges are deleted from the TSGD, for all $j, j = 1, 2, \ldots, n-1$. Suppose, for some $j, j = 2, \ldots, n-1$, $G_j$’s edges are deleted from the TSGD before $G_{(j+1) \mod n}$’s edges are deleted from the TSGD. Let $\text{slast}(t_j) = s_k$. Since $G_{jk}$ is serialized after $G_{(j+1) \mod k}$, at site $s_k$, $\text{ser}_k(G_{(j+1) \mod n})$ executes before $\text{ser}_k(G_j)$. Thus, since $G_{(j+1) \mod n}$’s edges are inserted into the TSGD before $\text{ser}_k(G_{(j+1) \mod n})$ executes, while $G_j$’s edges are deleted after $\text{ser}_k(G_j)$ executes, $G_{(j+1) \mod n}$’s edges must be in the TSGD when $G_j$’s edges are deleted (since we have assumed that $G_j$’s edges are deleted before $G_{(j+1) \mod n}$’s edges are deleted). Furthermore, since $\text{ser}_k(G_j)$ and $\text{ser}_k(G_{(j+1) \mod n})$ must have both executed when $\text{sfist}(t_j)$ is a contradiction, since edges belonging to $G_j$ and $G_{(j+1) \mod n}$ are deleted together when $\text{act}(f)$ for some transaction $G_i$ executes (since $G_{(j+1) \mod n}$ is serialized before $G_j$, if for every transaction $G_k \in V$ serialized before $G_j$, $val_k$ has been processed, then for every transaction $G_k \in V$ serialized before $G_{(j+1) \mod n}$ also, $val_k$ must have been processed). Thus, $G_1$’s edges are not deleted from the TSGD before $G_2$’s edges are deleted, $\ldots$, $G_{n-1}$’s edges are not deleted from the TSGD before $G_n$’s edges are deleted. By transitivity and since $G_0$’s edges are deleted only after $val_0$ has been processed when Detect_Ins_Opt is invoked during the processing of $val_0$, the TSGD $(V', E', D', L')$ contains the edges belonging to transactions $G_0, G_1, \ldots, G_{n-1}$ (since for all $i = 1, \ldots, n-1$, $val_i$ is processed before $val_0$ is processed).

We now show that $(G_0, \text{slast}(t_0)) \text{edge}(t_1) \cdots \text{edge}(t_{n-1})(\text{sfist}(t_0), G_0)$ is a path in the TSGD, which implies that the transactions do not execute in the order they are generated by the system.
By the definition of $state_F$, $st_{m+1} = st_F(st_m, L(v_{2m+1}, v_{2m+2}))$, if $v_{2m+2} = v_{2m+3}$ and $st_F(st_m, L(v_{2m+1}, v_{2m+2}))$, if $v_{2m+1} = v_{2m+3}$.

Let $St_k$ be the resulting state of Detect_INS_Opt after $(st_m, prev(v_{2m+1}))$ is added to $V_{-set}(v_{2m+1})$ (the state $St_k$ results either due to the forward transition $St_j \rightarrow St_k$, either $St_j.v = v_{2m+1}$ or $St_j.prev(v_{2m+1})$, or to Step 1). Thus, $St_k.v = v_{2m+1}$, $St_k.cur.st = st_m$ and in state $St_k$, $head(St_k, (St_k.v))[2] = prev(v_{2m+1})$. Furthermore, since Detect_INS_Opt does not return abort, it follows from Lemma 2 that after a finite number of steps, Detect_INS_Opt is in a state $St'_k$ such that $St'_k \equiv St_k$ and no further forward transitions can be made from $St'_k$. Thus, in state $St'_k$,

- Since $prev(v_{2m+1}) \neq v_{2m+2}$ (by the definition of path), $head(St'_k.anc(St'_k.v)) \neq v_{2m+2}$.
- Since $(v_1, v_2) \cdots (v_{2m+1}, v_{2m+2})$ is a path in $(V, E, D)$, there is no dependency $(prev(v_{2m+1}), v_{2m+2})$ in $D$; thus, there is no dependency $(head(St'_k.anc(St'_k.v)), St'_k.v) \rightarrow (St'_k.v, v_{2m+2})$ in $D$.
- Since $state_F(init.st_F, (v_3, v_4) \cdots (v_{2m+1}, v_{2m+2})(v_{2m+3}, v_{2m+4}))$ is defined, if $v_{2m+2} = v_{2m+3}$, then $st_{m+1} = st_F(St'_k.cur.st, L(St'_k.v, v_{2m+2}))$ is defined, else if $v_{2m+1} = v_{2m+3}$, then $st_{m+1} = st_F(St'_k.cur.st, L(St'_k.v, v_{2m+2}))$ is defined.

However, since in state $St'_k$, no forward transition can be made due to edge $(St'_k.v, v_{2m+2})$ Detect_INS_Opt does not return abort, it must be the case that

- if $v_{2m+2} = v_{2m+3}$, then $St'_k. V_{-set}(v_{2m+3})$ already contains $(st_{m+1}, St'_k.v)$. Thus, since $St'_k. v_{2m+1}, prev(v_{2m+3}) = v_{2m+1}, (st_{m+1}, prev(v_{2m+3}))$ is added to $V_{-set}(v_{2m+3})$ during the execution of Detect_INS_Opt.
- if $v_{2m+1} = v_{2m+3}$, then $St'_k. V_{-set}(St'_k.v)$ already contains $(st_{m+1}, v_{2m+2})$. Thus, since $St'_k. v_{2m+1}, prev(v_{2m+3}) = v_{2m+2}, (st_{m+1}, prev(v_{2m+3}))$ is added to $V_{-set}(v_{2m+3})$ during the execution of Detect_INS_Opt. \(\square\)

**Corollary 2:** Consider a TSGD $(V, E, D, L)$ containing a path $(v_1, v_2) \cdots (v_{2n-1}, v_{2n})(v_{2n+1}, v_{2}) = v_3$. If, for a regular term $RT, F = FA(RT), st = state_F(init.st_F, (v_3, v_4) \cdots (v_{2n-1}, v_{2n})(v_{2n+1}, v_3)$ is an accept state, and $v_{2n+1} \in set_1$, then Detect_INS_Opt($(V, E, D, L), v_1, v_2, set_1, RT)$ returns abort.

**Proof:** Suppose Detect_INS_Opt does not return abort. By Lemma 4, $(st, prev(v_{2n+1}))$ is added to $V_{-set}(v_{2n+1})$. Since $prev(v_{2n+1}) \neq v_1$, Detect_INS_Opt makes a forward state transition with $(st, prev(v_{2n+1}))$ is added to $V_{-set}(v_{2n+1})$. However, just before $(st, prev(v_{2n+1}))$ is added to $V_{-set}(v_{2n+1})$ since $st$ is an accept state, $prev(v_{2n+1}) \neq v_1$, $v_{2n+1} \in set_1$, and dependency $(prev(v_{2n+1}), v_{2n+1})$ $(v_{2n+1}, v_1)$ does not belong to $D$, Detect_INS_Opt returns abort. This leads to a contradiction, thus, it must be the case that Detect_INS_Opt returns abort. \(\square\)

We are now in a position to prove Theorem 1. Before we present the proof, we introduce some additional notation and the following lemma.

**Lemma 5:** If, in the optimistic scheme, for some site $s_k$, transactions $G_i, G_j, G_{ik}$ is serializable, before $G_{jk}$ at site $s_k$, then there does not exist a dependency $(G_j, s_k)-(s_k, G_i)$ in the TSGD.

**Proof:** Suppose there exists a dependency $(G_j, s_k)-(s_k, G_i)$ in the TSGD. The dependency could have not been added to the TSGD after $act(set_k(G_i))$ has executed. Thus, dependency $(G_j, s_k)$ have not been added to the TSGD after $act(set_k(G_i))$ has executed. Thus, dependency $(G_j, s_k)$ have not been added to the TSGD after $act(set_k(G_i))$ has executed.
The above definition of $state_F$ is recursive. In the following lemma, we show that an alternate non-recursive definition of $state_F$ is possible.

**Lemma 3:** Consider a TSGD containing a path $(v_1, v_2)(v_3, v_4)\cdots(v_{2n-1}, v_{2n})$. If $e_1e_2\cdots, e_{n-1}$ is a sequence such that

- if $v_{2i} = v_{2i+1}$, then $e_i = L(v_{2i-1}, v_{2i})$, and
- if $v_{2i-1} = v_{2i+1}$, then $e_i = \overline{L(v_{2i-1}, v_{2i})}$,

then for a regular term $RT$ and a state $st$ of $F = FA(RT)$, $state_F(st, (v_1, v_2)(v_3, v_4)\cdots(v_{2n-1}, v_{2n})$, $st_F(st, e_1\cdots e_{n-1})$.

**Proof:** We use induction on $i$ to prove that for all $i = 1, \ldots, n$, $state_F(st, (v_1, v_2)(v_3, v_4)\cdots(v_{2i-1}, v_{2i})$, $st_F(st, e_1\cdots e_i)$.

**Basis** ($i = 1$): $state_F(st, (v_1, v_2)) = st_F(st, e) = st$.

**Induction:** Assume true for $i = m, 1 \leq m < n$, that is, $state_F(st, (v_1, v_2)\cdots(v_{2m-1}, v_{2m}), st_F(st, e_1\cdots e_{m-1})$. We prove the claim for $i = m + 1$, that is, we need to show that $state_F(st, (v_1, v_2)\cdots(v_{2m+1}, v_{2m+2})) = st_F(st, e_1\cdots e_m)$. By the definition of $state_F$,

$$state_F(st, (v_1, v_2)\cdots(v_{2m+1}, v_{2m+2})) = \begin{cases} st_F(st', L(v_{2m-1}, v_{2m})) & \text{if } v_{2m} = v_{2m+1} \\ st_F(st', \overline{L(v_{2m-1}, v_{2m})}) & \text{if } v_{2m-1} = v_{2m+1} \end{cases}$$

where $st' = state_F(st, (v_1, v_2)\cdots(v_{2m-1}, v_{2m})$. Thus,

$$state_F(st, (v_1, v_2)\cdots(v_{2m+1}, v_{2m+2})) = \begin{cases} st_F(st, e_1\cdots e_{m-1}L(v_{2m-1}, v_{2m})) & \text{if } v_{2m} = v_{2m+1} \\ st_F(st, e_1\cdots e_{m-1}\overline{L(v_{2m-1}, v_{2m})}) & \text{if } v_{2m-1} = v_{2m+1} \end{cases}$$

Thus, $state_F(st, (v_1, v_2)\cdots(v_{2m+1}, v_{2m+2})) = st_F(st, e_1\cdots e_m)$. □

For every instantiation of a regular term $RT$, there is a corresponding path in the TSGD which $state_F (F = FA(RT))$ with respect to the initial state $init-st_F$ is an accept state. The following lemma lays the groundwork for showing that Detect-Ins-Opt detects instantiation by detecting appropriate paths in the TSGD.

**Lemma 4:** Consider a TSGD $(V, E, D, L)$ containing a path $(v_1, v_2)\cdots(v_{2n-3}, v_{2n-2}), (v_{2n-1}, v_{2n})$, such that for a regular term $RT, F = FA(RT)$, $state_F(init-st_F, (v_3, v_4)\cdots(v_{2n-1}, v_{2n}))$ is defined. If Detect-Ins-Opt($(V, E, D, L), v_1, v_2, set_1, RT$) does not return abort, then during the execution of Detect-Ins-Opt (before it returns commit), for all $i, i = 1, 2, 3, \ldots, n - 1$, $(st, prev(v_{2i+1}))$ is added to $V-set(v_{2i+1})$, where $st = state_F(init-st_F, (v_3, v_4)\cdots(v_{2i-1}, v_{2i})(v_{2i+1}, v_{2i+2}))$.

**Proof:** We prove the above lemma by induction on $i$. We prove that if Detect-Ins-Opt does not return abort, then for all $i, i = 1, 2, \ldots, n - 1$, $(st, prev(v_{2i+1}))$ is added to $V-set(v_{2i+1})$, where $st = state_F(init-st_F, (v_3, v_4)\cdots(v_{2i-1}, v_{2i})(v_{2i+1}, v_{2i+2}))$.

**Basis** ($i = 1$): In Step 1 of Detect-Ins-Opt, $(init-st_F, v_1)$ is added to $V-set(v_2)$. Since $v_2 = prev(v_3) = v_1$, and $state_F(init-st_F, (v_3, v_4)) = init-st_F$, the lemma is true for $i = 1$ ($(init-st_F, prev(v_2)$ is added to $V-set(v_2)$).

**Induction:** Let us assume that the lemma is true for $i = m, 1 \leq m < n - 1$. Thus, if Detect-Ins-Opt does not return abort, then $(st_m, prev(v_{2m+1}))$ is added to $V-set(v_{2m+1})$, where $st_m = state_F(init-st_F, (v_3, v_4)\cdots(v_{2m-1}, v_{2m})(v_{2m+1}, v_{2m+2}))$. We show the lemma to be true for $i = n - 1$.

Thus, we need to show that if Detect-Ins-Opt does not return abort, then $(st_{n-1}, prev(v_{2n}))$ is added to $V-set(v_{2n})$. □
node can be visited at most $n_G n_S$ times (every node $v$ in the TSGD can be visited in a state $s_t$ of $F$ at most once for every node $w$ such that edge $(v, w)$ is in the TSGD, and $F$ has at most $n_S$ states).

Since there are $m$ site nodes and at most $n_G$ transaction nodes in the TSGD, the number of times $\text{Detect}_\text{Ins}_\text{Opt}$ checks if an edge satisfies the conditions in Step 2 is $n_G^2 m n_S + n_G v_S^2 n_S$. Since $v_S$ is of the conditions in Step 2 can be checked in constant time and $v_S \ll n_G$, $v_S < m$, $\text{Detect}_\text{Ins}_\text{Opt}$ terminates in $O(n_G^2 m n_S)$ steps. □

Before we show that $\text{Detect}_\text{Ins}_\text{Opt}$ detects instantiations, we define the notion of a path in order to capture the notion of instantiations in the TSGD. Corresponding to every instantiation, there is a path, defined below, in the TSGD (paths are similarly defined for a TSG; the requirement that there are no dependencies between certain edges is trivially satisfied in a TSG).

**Definition 9:** Consider a TSG/TSGD containing the sequence of edges $(v_1, v_2)(v_3, v_4)\cdots(v_{2n-1}, v_{2n})$, $n > 1$. The sequence of edges is a path if

- for every pair of consecutive edges $(v_{2i-1}, v_{2i}),(v_{2i+1}, v_{2i+2})$, $i = 1, \ldots, n - 1$, either $v_{2i} = v_{2i+1}$ or $v_{2i-1} = v_{2i+1}$, and
- if for some $j$, $k = 1, 2, \ldots, n$, $j \leq k$, $v_{2j-1} = v_{2j+1} = v_{2j+3} = \cdots = v_{2k-1}$, then
  1. if $j < k$, then $v_{2j} \neq v_{2j+2} \neq v_{2j+4} \neq \cdots \neq v_{2k}$, and for all $l, m, j \leq l < m \leq k$, there is no dependency $(v_{2l}, v_{2l-1}) \rightarrow (v_{2m-1}, v_{2m})$ in the TSG/TSGD, and
  2. if $j > 1$ and $v_{2j-2} = v_{2j-1}$, then for all $l = j, j + 1, \ldots, k$, $v_{2j-3} \neq v_{2l}$, and there is no dependency $(v_{2j-3}, v_{2j-2}) \rightarrow (v_{2l-1}, v_{2l})$ in the TSG/TSGD. □

Thus, it follows from the definition of path that for every pair of consecutive edges $(v_{2i-1}, v_{2i}),(v_{2i+1}, v_{2i+2})$, $i = 1, \ldots, n - 1$, either

- $v_{2i} = v_{2i+1}$, $v_{2i-1} \neq v_{2i+2}$, and dependency $(v_{2i-1}, v_{i}) \rightarrow (v_{2i+1}, v_{2i+2})$ is not in the TSGD, or
- $v_{2i-1} = v_{2i+1}$, $v_{2i} \neq v_{2i+2}$ and dependency $(v_{2i}, v_{2i-1}) \rightarrow (v_{2i+1}, v_{2i+2})$ is not in the TSGD.

Furthermore, for the path $(v_1, v_2)(v_3, v_4)\cdots(v_{2n-1}, v_{2n})$, for $i = 1, 2, \ldots, n - 1$, we define $\text{prev}(v_{2i+1})$ as follows.

$$
\text{prev}(v_{2i+1}) = \begin{cases} 
  v_{2i} & \text{if } v_{2i} = v_{2i+1} \\
  v_{2i-1} & \text{if } v_{2i-1} = v_{2i+1}
\end{cases}
$$

Note that, by the definition of path, $\text{prev}(v_{2i+1}) \neq v_{2i+2}$ and there is no dependency $(\text{prev}(v_{2i+1}), v_{2i+1}) \rightarrow (v_{2i+1}, v_{2i+2})$ in the TSGD. Only certain paths in the TSG/TSGD in which the sequence of transaction types is a string in $\mathit{reg}\_\mathit{exp}$ correspond to instantiations of $\mathit{RT} = c_0 : \mathit{reg}\_\mathit{exp}$ in $S$. In order to ensure that transaction type information can be taken into account when detecting paths in the TSG/TSGD, we define $\mathit{state}_F$ below.

**Definition 10:** Consider a TSG/TSGD containing a path $(v_1, v_2)\cdots(v_{2n-1}, v_{2n})$. Let $\mathit{RT}$ be a regular term and $F = FA(\mathit{RT})$. We define $\mathit{state}_F$ for the sequence of edges in the path and a state of $F$, using $\mathit{st}_F$, as follows.

$$
\mathit{state}_F(st, (v_1, v_2)\cdots(v_{2i-1}, v_{2i})) = \begin{cases} 
  st & \text{if } i = 1 \\
  \mathit{st}_F(st', \mathit{L}(v_{2i-3}, v_{2i-2})) & \text{if } i > 1 \text{ and } v_{2i-2} = v_{2i-1} \\
  \mathit{st}_F(st', \mathit{L}(v_{2i-3}, v_{2i-2})) & \text{if } i > 1 \text{ and } v_{2i-3} = v_{2i-1}
\end{cases}
$$

and $\mathit{state}_F$ is in $(v_1, v_2)\cdots(v_{2n-1}, v_{2n})$. □
Basis ($num = 0$): If $num = 0$ in state $St_k$, then in state $St_k$, for every edge $(St_k, v, u)$, if $st_F(St_k, cur, St, L(St_k, v, u))$ is defined, then $(st, St_k, v) \in St_k \cdot V \cdot set(u)$ (alternatively, if $st' = (St_k, cur, St, L(St_k, v, u))$ is defined, then $(st', u) \in St_k \cdot V \cdot set(St_k, v)$). Thus, no forward transition can be made from state $St_k$ (since every edge $(St_k, v, u)$ satisfies the last condition in Step 2).

**Induction:** Let us assume the lemma is true if $num \leq m$ in state $St_k$, $m \geq 0$. We show that the lemma is true if $num \leq m + 1$ in state $St_k$. We show that if Detect Ins Opt does not return abort, then after a finite number of moves, Detect Ins Opt is in a state $St'_k$ such that $St'_k \equiv St_k$ and no forward transitions can be made from state $St'_k$.

Let $St''_k$ be any state equivalent to $St_k$ such that in $St''_k$, $num \leq m + 1$. If Detect Ins Opt makes the forward transition $St''_k \rightarrow St_l$ due to some edge $(St''_k, v, u)$ and $L(St''_k, v, u)$, then it must be the case that $St_l, v = u, St_l, cur \cdot st \equiv st_F(St''_k, cur, St, L(St''_k, v, u))$. Furthermore, in state $St_l$, $(St_l, cur, St, L(St''_k, v, u)) \in St_l \cdot V \cdot set(u)$ and in state $St_l$, $(St_l, cur, St, L(St''_k, v, u)) \in St_l \cdot V \cdot set(u)$ (since the transition $St''_k \rightarrow St_l$ causes $(St_l, cur, St, L(St''_k, v, u))$ to be added to $V \cdot set(u)$). Note that, since before the transition is made, $(St_l, cur, St, L(St''_k, v, u))$ does not belong to $V \cdot set(u)$ and $num \leq m + 1$ in $St''_k$, after the transition $St''_k \rightarrow St_l$ is made, $num \leq m$ in $St_l$. By IH, since Detect Ins Opt does not return abort, and in a finite number of steps, Detect Ins Opt enters a state $St'_l \equiv St_l$, such that no forward transitions are possible from $St'_l$. Thus, since it does not return abort, Detect Ins Opt makes the reverse transition $St'_l \rightarrow St''_k$ after a finite number of steps, where $St''_k \equiv St_k$. Furthermore, in state $St_l$, $(St_l, cur, St, L(St''_k, v, u)) \in St_l \cdot V \cdot set(u)$ and $St''_k \cdot v = St''_k \cdot v$, and thus, no forward transition can be made from state $St''_k$ due to edge $(St''_k, v, u)$ and $L(St''_k, v, u)$ (edge $(St''_k, v, u)$ does not satisfy the condition in Step 3(c)). Using a similar argument, it can be shown that if Detect Ins Opt makes a forward transition $St''_k \rightarrow St_l$ due to edge $(St''_k, v, u)$ and $L(St''_k, v, u)$, then in a finite number of steps, Detect Ins Opt enters a state $St''_k \equiv St_k$ such that no forward transitions are possible from $St''_k$ due to edge $(St''_k, v, u)$ and $L(St''_k, v, u)$.

Thus, once a forward transition is made by Detect Ins Opt due to an edge $e$ and $L(e) / L(e)$ from any state equivalent to $St_k$, then no further forward transitions can be made by Detect Ins Opt due to $L(e) / L(e)$ from any state equivalent to $St_k$. Furthermore, every time a forward transition is made, from a state $St''_k$ that is equivalent to $St_k$ such that $num \leq m + 1$ in $St''_k$, a reverse transition is made by Detect Ins Opt to a state $St''_k$ equivalent to $St_k$ such that $num \leq m + 1$ in $St''_k$. Since there are a finite number of edges incident on each node, Detect Ins Opt does not return abort, and in state $St''_k$, $num \leq m + 1$, eventually, Detect Ins Opt would be in a state $St'_k \equiv St_k$ such that no further forward transitions can be made.

**Corollary 1:** Procedure Detect Ins Opt terminates in $O(n^2 mns)$ steps.

**Proof:** We first show that Detect Ins Opt terminates in a finite number of steps. Let $St_0$ denote the state immediately after the execution of Step 1 of algorithm Detect Ins Opt. If Detect Ins Opt does not return abort, then by Lemma 1, after a finite number of steps, Detect Ins Opt is in a state $St'_1 \equiv St_1$ such that no further forward transitions can be made from $St'_1$. Detect Ins Opt then executes Step 4 and, since, in state $St'_1$, head($St'_1, F, list(St'_1, v)$) = $(s, G_i)$, Detect Ins Opt terminates in a finite number of steps. If, on the other hand, Detect Ins Opt returns abort, then it trivially terminates in a finite number of steps.

The number of steps Detect Ins Opt terminates in is equal to the product of the number of times Detect Ins Opt checks if an edge satisfies the conditions in Step 2 and the number of steps required to check if an edge satisfies the conditions in Step 2. Every time a transaction node is visited, the conditions in Step 2 need to be checked, on average, for $v_S$ edges (the average number of site global transaction executes at is $v_S$), while every time a site node is visited, the conditions in Step 2 need to be checked for at most $n_G$ edges (since the number of transaction nodes in the TSGD is $n_G$). Thus, the number of steps required to check if an edge satisfies the conditions in Step 2 is the number of times we visit both types of nodes.
Appendix B: Optimistic Scheme

Before we prove Theorem 1, we need to prove certain lemmas. In the following lemma, we study the implications of complete regular specifications.

Lemma 1: Let $RT_1$ be a regular term in the regular specification $R$, $I$ be an instantiation of $S$ in the global schedule $S$, and $G_0$ be a transaction in $I$. If $R$ is complete, then there exists a regular term $RT_2$ and an instantiation $t_0 : t_1 \cdots t_{m-1}$ of $RT_2$ in $S$ such that $hdr(t_0) = G_0$.

Proof: Let $RT_1 = e_0' : \text{reg-exp}_1$ and $I = t'_0 : t'_1 \cdots t'_{n-1}$, $n > 1$. Since $I$ is an instantiation of $S$, in $S$,

- for all $j$, $j = 0, 1, \ldots, n - 1$,
  1. $t'_j \in \Sigma_S$, and
  2. $\text{last}(t'_j)$ and $\text{first}(t'_{(j+1) \mod n})$ execute at the same site, and $\text{last}(t'_j)$ is serialized at $\text{first}(t'_{(j+1) \mod n})$ at the site, and

- $\text{type}(t'_0) = e'_0$ and $\text{type}(t'_1) \cdots \text{type}(t'_{n-1})$ is a string in $L(\text{reg-exp}_1)$.

Let $G_0 = \text{hdr}(t'_k)$, for some $k$, $k = 0, 1, \ldots, n - 1$. Since $R$ is complete, there exists a regular term $RT_2 = type(t'_k) : \text{reg-exp}_2$ such that

$$\text{type}(t'_{(k+1) \mod n}) \cdots \text{type}(t'_{(k+n-1) \mod n})$$

is a string in $\in L(\text{reg-exp}_2)$. Thus,

$$t'_k : t'_{(k+1) \mod n} \cdots t'_{(k+n-1) \mod n}$$

is the required instantiation of $RT_2$ in $S$. □

We next show that the manner in which Detect_ins Opt traverses edges in the TSGD ensures that it detects instantiations of regular terms in the TSGD. We first introduce the following additional notation.

Between the execution of any two steps\(^3\) of Detect_ins Opt, the contents of $v$, $\text{cur-st}$, $\Delta$, $\text{anc}(V\_set(v))$ and $F\_List(v_i)$ for all $v_i \in V$ constitute a state $St_k$ of Detect_ins Opt. We denote the contents of $v$, $\text{cur-st}$, $\Delta$, $\text{anc}(v_i)$, $V\_set(v_i)$, and $F\_List(v_i)$ for any $v_i \in V$ in state $St_k$ by $St_k.v$, $St_k.\text{cur-st}$, $St_k.\Delta$, $St_k.\text{anc}(v_i)$, $St_k.V\_set(v_i)$ and $St_k.F\_List(v_i)$ respectively. State changes in Detect_ins Opt are caused by steps 1, 3, and 4. We refer to state transition $St_j \rightarrow St_k$ due to Step 3 as a forward transition while a state transition $St_j \rightarrow St_k$ due to Step 4 is referred to as a reverse transition. Also, two states $St_j$ and $St'_j$ are said to be equivalent (denoted by $St_j \equiv St'_j$) if $St_j.v = St'_j.v$, $St_j.\text{cur-st} = St'_j.\text{cur-st}$, and for all $v_i \in V$, $St_j.\text{anc}(v_i) = St'_j.\text{anc}(v_i)$, $St_j.F\_List(v_i) = St'_j.F\_List(v_i)$. Detect_ins Opt has the following interesting property: if it makes a forward transition $St_j \rightarrow St_k$ and for a state $St'_k \equiv St_k$ makes a reverse transition $St'_k \rightarrow St'_j$, then $St_j \equiv St'_j$.

Lemma 2: If Detect_ins Opt does not return abort and during its execution, Detect_ins Opt is in state $St_k$, then after a finite number of steps, it enters a state $St'_k \equiv St_k$ such that no forward transitions from $St'_k$ are possible.

Proof: We prove the lemma by induction on $num$, the number of elements in $\{(st, v_1, v_2) : (st \text{ is a state of } F) \land (v_1, v_2 \in V) \land ((st, v_1) \notin V\_set(v_2))\}$ in state $St_k$. 


procedure Detect_Ins_TSGD2((V, E, D, L), G_i, s_k, set_1, RT):

1. For all nodes v in the TSGD, set F_list(v) = [], anc(v) = [], V_set(v) = ∅.
   v = s_k, F_list(s_k) = [(st*, G_i)], anc(s_k) = [(G_i, G_i)], F = FA(RT), V_set(s_k) =
   \{init_st_F, (G_i, G_i)\} and cur_st = init_st_F. Set Δ = ∅.

2. If, for every edge (v, u) one of the following is true:
   • head(anc(v))[1] = u.
   • head(anc(v))[2] = u.
   • There is a dependency (head(anc(v))[1], v)→(v, u) in D ∪ Δ.
   • There is a dependency (head(anc(v))[2], v)→(v, u) in D ∪ Δ.
   • If st = st_F(cur_st, L(v, u)) is defined then (st, (v, v)) ∈ V_set(u), and
     if st' = st_F(cur_st, L(v, u)) is defined then (st', (head(anc(v))[1], u)) ∈ V_set(v).

   then go to Step 4.

3. Choose an edge (v, u) such that
   (a) head(anc(v))[1] ≠ u, and
   (b) head(anc(v))[2] ≠ u, and
   (c) there is no dependency (head(anc(v))[1], v)→(v, u) in D ∪ Δ, and
   (d) there is no dependency (head(anc(v))[2], v)→(v, u) in D ∪ Δ, and
   (e) st = st_F(cur_st, L(v, u)) is defined and (st, (v, v)) ∉ V_set(u), or
     st' = st_F(cur_st, L(v, u)) is defined and (st', (head(anc(v))[1], u)) ∉ V_set(v).

   If st is defined and (st, (v, v)) ∉ V_set(u), then do
     • If st is an accept state, u ∈ set_1 and v ≠ G_i, then Δ := Δ ∪ {(v, u)→(u, G_i)}.
     • F_list(u) := (cur_st, v) ∘ F_list(u), anc(u) := (v, v) ∘ anc(u), cur_st := st, V_set(u) :=
       V_set(u) ∪ {(st, (v, v))}, v := u. Go to Step 2.

   If st' is defined and (st', (head(anc(v))[1], u)) ∉ V_set(v) then do
     • If st' is an accept state, v ∈ set_1, u ≠ G_i and head(anc(v))[1] ≠ G_i, then Δ
       Δ ∪ {(head(anc(v))[1], v)→(v, G_i)}.
     • F_list(v) := (cur_st, v) ∘ F_list(v), anc(v) := (head(anc(v))[1], u) ∘ anc(v), cur_st :=
       V_set(v) := V_set(v) ∪ {(st', (head(anc(v))[1], u))}. Go to Step 2.

4. If head(F_list(v)) ≠ (st*, G_i), then temp1 := head(F_list(v))[1], temp2 :=
   head(F_list(v))[2], F_list(v) := tail(F_list(v)), anc(v) = tail(anc(v)), cur_st := temp1,
   v := temp2 and go to Step 2.

5. return(Δ).

Figure 14: Procedure Detect_Ins_TSGD2
procedure Detect_INS_TSGD1((V, E, D, L), G_i, s_k, set_1, RT):

1. For all nodes v in the TSGD, set F_list(v) = [], anc(v) = [], V_set(v) = ∅. Set v = F_list(s_k) = [(st*, G_i)], anc(s_k) = [G_i], F = FA(RT), V_set(s_k) = {(init_st_F, G_i)}
cur_st = init_st_F. Set Δ = ∅.

2. If, for every edge (v, u) one of the following is true:
   - head(anc(v)) = u.
   - There is a dependency (head(anc(v))→(v, u)) in D ∪ Δ.
   - If st = st_F(cur_st, L(v, u)) is defined then (st, v) ∈ V_set(u), and if st' = st_F(cur_st, L(v, u)) is defined then (st', head(anc(v))) ∈ V_set(v).

   then go to Step 4.

3. Choose an edge (v, u) such that
   
   (a) head(anc(v)) ≠ u, and
   
   (b) there is no dependency (head(anc(v))→(v, u)) in D ∪ Δ, and
   
   (c) st = st_F(cur_st, L(v, u)) is defined and (st, v) ∉ V_set(u), or
   
   st' = st_F(cur_st, L(v, u)) is defined and (st', head(anc(v))) ∉ V_set(v).

   If st is defined and (st, v) ∉ V_set(u), then do
   - if st is an accept state, u ∈ set_1 and v ≠ G_i, then Δ := Δ ∪ {(v, u)→(u, G_i)}.
   - F_list(u) := (cur_st, v)→ F_list(u), anc(u) := v→ anc(u), cur_st := st, V_set(u) := V_set(u) ∪ {(st, v)}, v := u. Go to Step 2.

   If st' is defined and (st', head(anc(v))) ∉ V_set(v), then do
   - if st' is an accept state, v ∈ set_1 and head(anc(v)) ≠ G_i, then Δ := Δ ∪ {(head(anc(v)), v)→(v, G_i)}.
   - F_list(v) := (cur_st, v)→ F_list(v), anc(v) := head(anc(v))→ anc(v), cur_st := V_set(v) = V_set(v) ∪ {(st', head(anc(v)))}. Go to Step 2.

4. If head(F_list(v)) ≠ (st*, G_i), then temp_1 := head(F_list(v))[1], temp_2 := head(F_list(v))[2], F_list(v) := tail(F_list(v)), anc(v) = tail(anc(v)), cur_st := temp_2 and goto Step 2.

5. return(Δ).

Figure 13: Procedure Detect_INS_TSGD1
procedure Detect_Ins.TSG2\((V, E, L, G_i, s_k, \text{set}_1, \text{set}_2, RT)\):

1. For all nodes \(v\) in the TSG, set \(F\_list(v) = \emptyset, \\text{anc}(v) = \emptyset, V\_set(v) = \emptyset\). Set \(v = s_k, F\_list([s_k, G_i]), \\text{anc}(s_k) = [G_i, G_i], F = FA(RT), V\_set(s_k) = \{(\text{init}\_st}_F, (G_i, G_i)\}\) and \(\text{cur}_\_st\). Set \(\Delta = \emptyset\).

2. If, for every edge \((v, u)\) one of the following is true:
   - \(\text{head} (\text{anc}(v))[1] = u\) or \(\text{head} (\text{anc}(v))[2] = u\).
   - If \(st = st_F(\text{cur}_\_st, L(v, u))\) is defined then
     (a) there exist nodes \(u_2, u_3, u_2 \neq u_3\), such that \((st, (v, u_2)) \in V\_set(u), (st, (v, u_3)) \in V\_set(u)\)
     or
     (b) \((st, (v, v)) \in V\_set(u)\),
     and if \(st' = st_F(\text{cur}_\_st, L(v, u))\) is defined then
     (a') there exist nodes \(u_2, u_3, u_2 \neq u_3\), such that \((st', (\text{head}(\text{anc}(v))[1], u_2)) \in V\_set(v), (st', (\text{head}(\text{anc}(v))[1], u_3)) \in V\_set(v)\), or
     (b') \((st', (\text{head}(\text{anc}(v))[1], u)) \notin V\_set(v)\).

3. Choose an edge \((v, u)\) such that
   - \(\text{head} (\text{anc}(v))[1] \neq u\) and \(\text{head} (\text{anc}(v))[2] \neq u\), and
   - \(st = st_F(\text{cur}_\_st, L(v, u))\) is defined and
     (a) there do not exist nodes \(u_2, u_3, u_2 \neq u_3\), such that \((st, (v, u_2)) \in V\_set(u), (st, (v, u_3)) \in V\_set(u)\), and
     (b) \((st, (v, v)) \notin V\_set(u)\),
     or
     \(st' = st_F(\text{cur}_\_st, L(v, u))\) is defined and
     (a') there do not exist nodes \(u_2, u_3, u_2 \neq u_3\), such that \((st', (\text{head}(\text{anc}(v))[1], u_2)) \in V\_set(v), (st', (\text{head}(\text{anc}(v))[1], u_3)) \in V\_set(v)\), and
     (b') \((st', (\text{head}(\text{anc}(v))[1], u)) \notin V\_set(v)\), and
   - \(v \notin (\text{set}_2 \cup \Delta)\) or \(u \neq G_i\).

If \(st\) is defined, 3(a) and 3(b) then do
   - If \(st\) is an accept state, \(u \in \text{set}_1\) and \(v \neq G_i\), then \(\Delta := \Delta \cup \{u\}\).
   - \(F\_list(u) := (\text{cur}_\_st, v) \circ F\_list(u), \\text{anc}(u) := (v, v) \circ \text{anc}(u), \text{cur}_\_st := st, V\_set(u) := V\_set(u) \cup \{(st, (v, v))\}, v := u\). Go to Step 2.

If \(st'\) is defined, 3(a') and 3(b') then do
   - If \(st'\) is an accept state, \(v \in \text{set}_1, u \neq G_i\) and \(\text{head}(\text{anc}(v))[1] \neq G_i\), then \(\Delta := \Delta \cup \{v\}\).
   - \(F\_list(v) := (\text{cur}_\_st, v) \circ F\_list(v), \\text{anc}(v) := (\text{head}(\text{anc}(v))[1], u) \circ \text{anc}(v), \text{cur}_\_st := st, V\_set(v) := V\_set(v) \cup \{(st', (\text{head}(\text{anc}(v))[1], u))\}\). Go to Step 2.

4. If \(\text{head}(F\_list(v)) \neq (st, G_i)\), then \(\text{temp}_1 := \text{head}(F\_list(v))[1], \text{temp}_2 := \text{head}(F\_list(v))\), \(F\_list(v) := \text{tail}(F\_list(v)), \\text{anc}(v) := \text{tail}(\text{anc}(v)), \text{cur}_\_st := \text{temp}_1, v := \text{temp}_2\) and go to Step

5. return(\(\Delta\)).

Figure 12: Procedure Detect_Ins.TSG2
procedure Detect_Ins_TSG1((V, E, L), Gi, sk, set1, set2, RT):

1. For all nodes v in the TSG, set \(F_{\text{list}}(v) = [], \text{anc}(v) = [], V_{\text{set}}(v) = \emptyset\). Set \(v = s_k, F_{\text{list}}(s_k) = [(st_s, G_i)], \text{anc}(sk) = [G_i], F = FA(RT), V_{\text{set}}(s_k) = \{(\text{init}_{st_F}, G_i)\}\) and \(\text{cur.st} = \text{init}_{st_F}\).

2. If, for every edge \((v, u)\) one of the following is true:
   - If \(st = st_F(\text{cur.st}, L(v, u))\) is defined then either
     - (a) \(\text{head}(\text{anc}(v)) = u\) or
     - (b) \((st, v) \in V_{\text{set}}(u)\) or
     - (c) there exist two distinct nodes \(v_1, v_2\) such that \((st, v_1) \in V_{\text{set}}(u)\) and \((st, v_2) \in V_{\text{set}}(u)\).
   - If \(st' = st_F(\text{cur.st}, L(v, u))\) is defined then either
     - (a) \((st', \text{head}(\text{anc}(v))) \notin V_{\text{set}}(v),\) or
     - (b) there exist two distinct nodes \(v_1, v_2\) such that \((st', v_1) \in V_{\text{set}}(v)\) and \((st', v_2) \in V_{\text{set}}(v)\).
   - \(v \in (\text{set}_2 \cup \Delta)\) and \(u = G_i\).

   then go to Step 4.

3. Choose an edge \((v, u)\) such that
   - \(st = st_F(\text{cur.st}, L(v, u))\) is defined and
     - (a) \(\text{head}(\text{anc}(v)) \neq u\) and
     - (b) \((st, v) \notin V_{\text{set}}(u)\), and
     - (c) there do not exist two distinct nodes \(v_1, v_2\) such that \((st, v_1) \in V_{\text{set}}(u)\) and \((st, v_2) \in V_{\text{set}}(u)\).
   - \(st' = st_F(\text{cur.st}, L(v, u))\) is defined and
     - (a') \((st', \text{head}(\text{anc}(v))) \notin V_{\text{set}}(v),\) and
     - there do not exist two distinct nodes \(v_1, v_2\) such that \((st', v_1) \in V_{\text{set}}(v)\) and \((st', v_2) \in V_{\text{set}}(v)\).
   - \(v \notin (\text{set}_2 \cup \Delta)\) or \(u \neq G_i\).

   If \(st\) is defined, 3(a), 3(b) and 3(c), then do
   - If \(st\) is an accept state, \(u \in \text{set}_1\) and \(v \neq G_i\), then \(\Delta := \Delta \cup \{u\}\).
   - \(F_{\text{list}}(u) := (\text{cur.st}, v) \circ F_{\text{list}}(u), \text{anc}(u) := v \circ \text{anc}(u), \text{cur.st} := st, V_{\text{set}}(u) = V_{\text{set}}(v) \cup \{(st, v)\}, v := u\). Go to Step 2.

   If \(st'\) is defined, 3(a') and 3(b'), then do
   - If \(st'\) is an accept state, \(v \in \text{set}_1\) and \(\text{head}(\text{anc}(v)) \neq G_i\), then \(\Delta := \Delta \cup \{v\}\).
   - \(F_{\text{list}}(v) := (\text{cur.st}, v) \circ F_{\text{list}}(v), \text{anc}(v) := \text{head}(\text{anc}(v)) \circ \text{anc}(v), \text{cur.st} := st', V_{\text{set}}(v) \cup \{(st', \text{head}(\text{anc}(v)))\}. Go to Step 2.

4. If \(\text{head}(F_{\text{list}}(v)) \neq (st_s, G_i)\), then \(\text{temp1} := \text{head}(F_{\text{list}}(v))[1], \text{temp2} := \text{head}(F_{\text{list}}(v))[2], F_{\text{list}}(v) := \text{tail}(F_{\text{list}}(v)), \text{anc}(v) := \text{tail}(\text{anc}(v)), \text{cur.st} := \text{temp1}, v := \text{temp2},\) and go to Step 2.

5. return(\(\Delta\)).

Figure 11: Procedure Detect_Ins_TSG1
Appendix - A - : Procedures

procedure Detect_Ins_Opt((V, E, D, L), G_i, s_k, set_1, RT):

1. For all nodes v in the TSGD, set $F_{list}(v) = []$ ([] is the empty list), $\text{anc}(v) = [], V_{set}(v)$
   Set $v = s_k$, $F_{list}(s_k) = [(st*, G_i)]$ ($st*$ is a special termination state), $\text{anc}(s_k) = F = FA(RT)$, $V_{set}(s_k) = \{(init-st_F, G_i)\}$ and $\text{cur-st} = \text{init-st}_F$.

2. If, for every edge $(v, u)$ one of the following is true:
   - $\text{head}(\text{anc}(v)) = u$.
   - There is a dependency $(\text{head}(\text{anc}(v)), v)\rightarrow (v, u)$ in $D$.
   - if $st = st_F(\text{cur-st}, L(v, u))$ is defined then $(st, v) \notin V_{set}(u)$, and
     if $st' = st_F(\text{cur-st}, L(v, u))$ is defined then $(st', u) \notin V_{set}(v)$.

   then go to Step 4.

3. Choose an edge $(v, u)$ such that
   - (a) $\text{head}(\text{anc}(v)) \neq u$, and
   - (b) there is no dependency $(\text{head}(\text{anc}(v)), v)\rightarrow (v, u)$ in $D$, and
   - (c) $st = st_F(\text{cur-st}, L(v, u))$ is defined and $(st, v) \notin V_{set}(u)$, or
     $st' = st_F(\text{cur-st}, L(v, u))$ is defined and $(st', u) \notin V_{set}(v)$.

   If $st$ is defined and $(st, v) \notin V_{set}(u)$ then do
   - If $st$ is an accept state, $u \in set_1$, $v \neq G_i$ and there is no dependency $(v, u)\rightarrow (u, G_i)$ in $D$, then return(abort).
   - $F_{list}(u) := (\text{cur-st}, v) \circ F_{list}(u)$, $\text{anc}(u) = v \circ \text{anc}(u)$, $\text{cur-st} := st$, $V_{set}((st, v) \cup V_{set}(u))$, $v := u$. Go to Step 2.

   If $st'$ is defined and $(st', u) \notin V_{set}(v)$ then do
   - If $st'$ is an accept state, $v \in set_1$, $u \neq G_i$ and there is no dependency $(u, v)\rightarrow (v, G_i)$ then return(abort).
   - $F_{list}(v) := (\text{cur-st}, v) \circ F_{list}(v)$, $\text{anc}(v) = u \circ \text{anc}(v)$, $\text{cur-st} := st'$, $V_{set}((st', u) \cup V_{set}(v))$. Go to Step 2.

4. If $\text{head}(F_{list}(v)) \neq (st*, G_i)$, then $temp1 := \text{head}(F_{list}(v))[1]$, $temp2 := \text{head}(F_{list}(v))[2]$,
   $\text{anc}(v) = \text{tail}(\text{anc}(v))$, $F_{list}(v) := \text{tail}(F_{list}(v))$, $\text{cur-st} := temp2$, $v := temp2$ and go to Step 2.

5. return(commit).

Figure 10: Procedure Detect_Ins_Opt