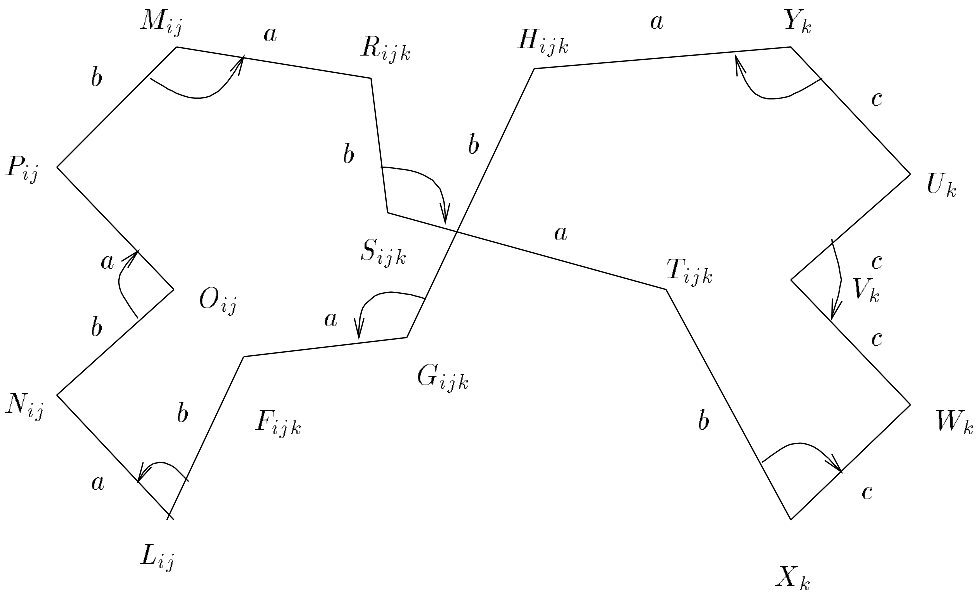


Figure 25: Dependencies in case $l_{ij} = x_k$

Similarly, it can be shown that $(V, E, D \cup \Delta, L)$ cannot contain any strong-cycles consistent w $RT_1$ and $RT_2$ if $x_k$ is assigned $false$. Thus, $(V, E, D \cup \Delta, L)$ cannot contain any strong-cycles consis with either $RT_1$ or $RT_2$, and is strongly-acyclic with respect to $R$. $\Box$

**only if:** Suppose there exists an assignment of truth values to literals such that $C$ is satisfia
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 $\bar{x_k}$ is assigned *true*, t
$(W_k, V_k) \rightarrow (V_k, U_k)$ is added to $\Delta$ (since only one of $x_k$ or $\bar{x_k}$ is *true* in the assignment, addition of
dependencies to $\Delta$ does not make $D \cup \Delta$ inconsistent). Also, for all $l_{ij}$, if $l_{ij}$ is *true* in the assignm
then dependency $(N_{ij}, O_{ij}) \rightarrow (O_{ij}, P_{ij})$ is added to $\Delta$, else dependency $(P_{ij}, O_{ij}) \rightarrow (O_{ij}, N_{ij})$ is ad
to $\Delta$. From the construction of $\Delta$, it trivially follows that $D \cup \Delta$ is consistent. We show that i
impossible for $(V, E, D \cup \Delta, L)$ to contain any strong-cycles that are consistent with either $RT_1$ or $I$

We first show that $(V, E, D \cup \Delta, L)$ cannot contain any strong-cycles consistent with $RT_1$.
strong-cycle consistent with $RT_1$ cannot contain nodes $R_{ijk}, S_{ijk}, T_{ijk}, F_{ijk}, G_{ijk}$ or $H_{ijk}$ due to dep
dencies $(R_{ijk}, S_{ijk}) \rightarrow (S_{ijk}, T_{ijk})$ 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 $\Delta$. T
there cannot be any strong-cycle 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 $RT_2$.
strong-cycle consistent with $RT_2$ cannot involve any of the nodes $N'_{ij}$, $O'_{ij}$ or $P'_{ij}$ due to the depende
$(P'_{ij}, O'_{ij}) \rightarrow (O'_{ij}, N'_{ij})$, 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 $\bar{x_k}$. Let us assume that $x_k$ is *true* in the assignm
We consider the following two cases:

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

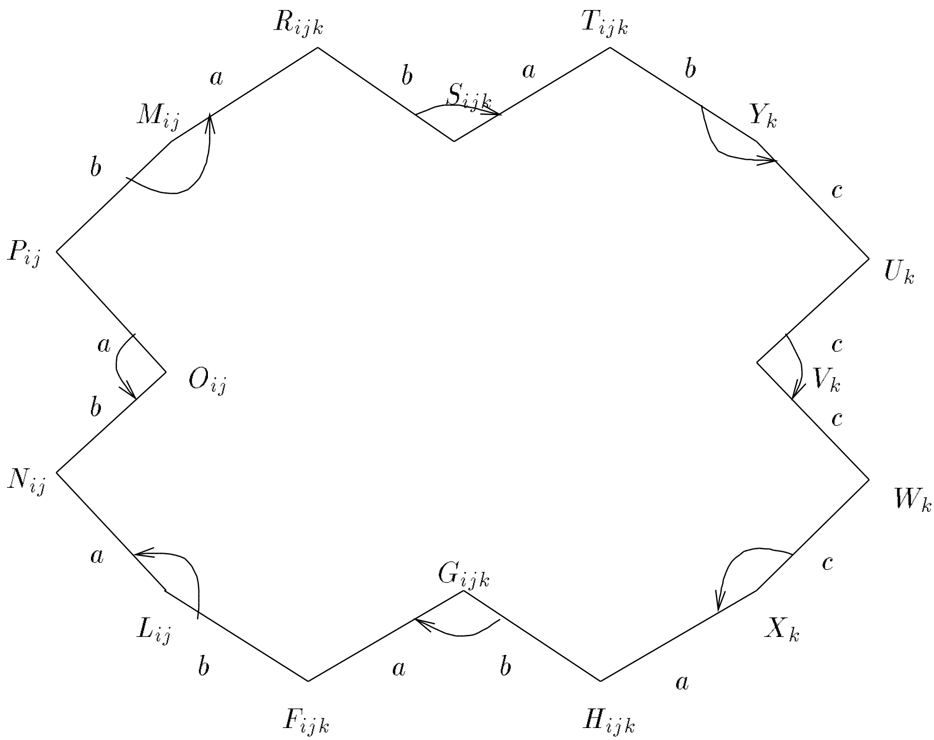


Figure 24: Dependencies in case $l_{ij} = \bar{x_k}$

$l_{ij} = x_k$: In this case (as shown in Figure 25), since dependency $(U_k, V_k) \rightarrow (V_k, W_k)$ is added to

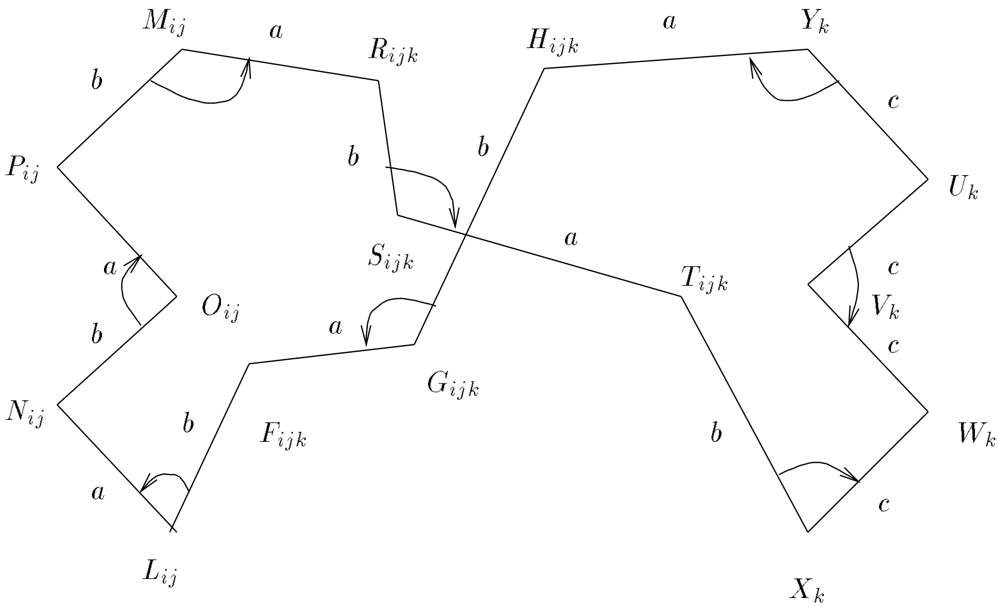


Figure 22: Dependencies in case $l_{ij} = x_k$

$l_{ij} = \bar{x_k}$: 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 consist 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, $(U_k, V_k) \rightarrow (V_k, W_k)$ c not belong to $\Delta$, and $x_k$ is assigned *false* ($\bar{x_k}$ is assigned *true*).

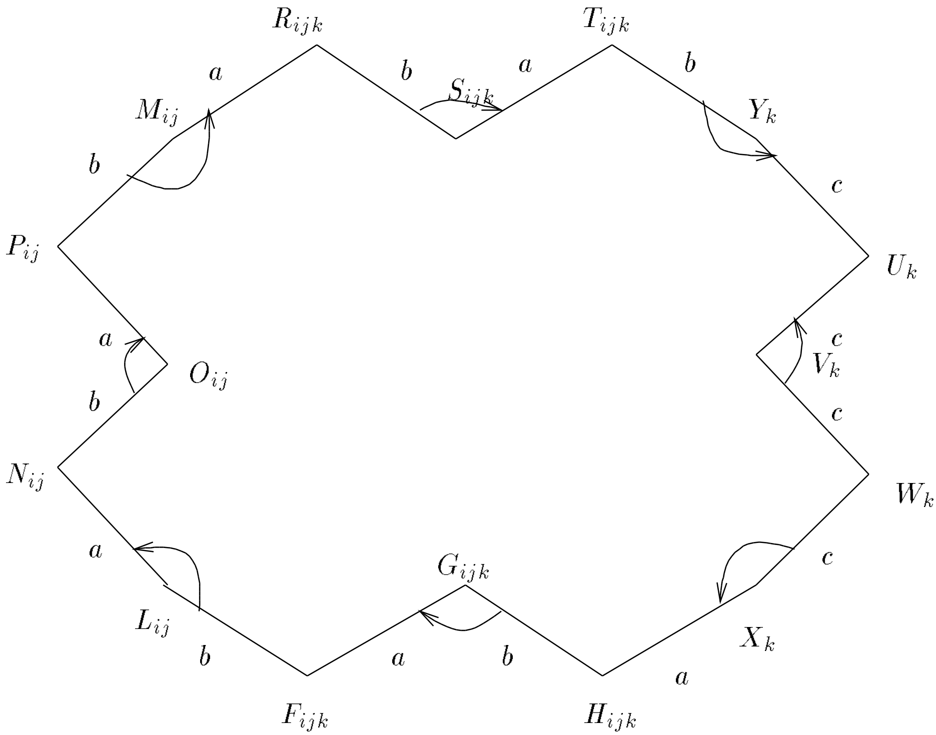


Figure 23: Dependencies in case $l_{ij} = \bar{x_k}$

Thus, since there exists an assignment of truth values to literals such that in every conjunct

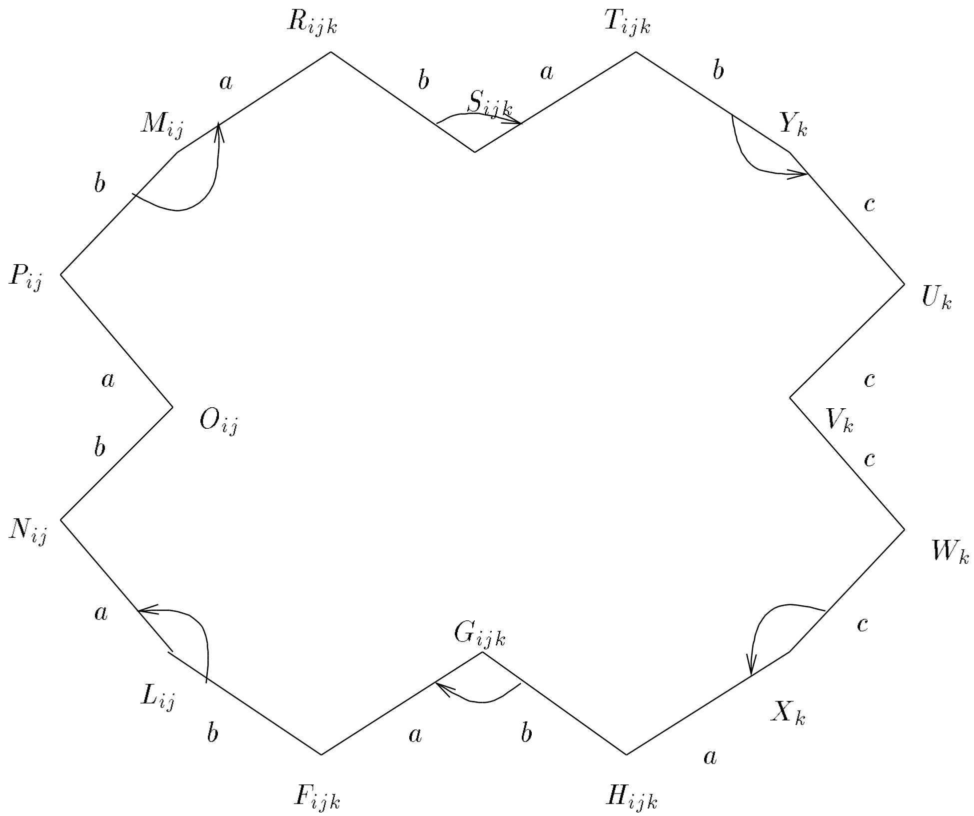


Figure 21: Nodes and edges if $l_{ij} = \bar{x_k}$

$R_{ijk}$, $T_{ijk}$, $F_{ijk}$ and $H_{ijk}$ are transaction nodes, while $G_{ijk}$ and $S_{ijk}$ are site nodes. Subtra actions 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 $b$, w 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 $a$. N that there are at most three edges incident on $L_{ij}$ and $M_{ij}$. Also, there are two edges incident 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 be constructed in $O(p+q)$ steps.

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

**if:** Let us assume there exists a set of dependencies $\Delta$ such that $(V, E, D \cup \Delta, L)$ is strongly-acy with respect to $R$ and $D \cup \Delta$ is consistent. We need to show that there exists an assignment of tr values to literals such that $C$ is satisfiable. We assign truth values to literals as follows. If depende $(U_k, V_k) \rightarrow (V_k, W_k) \in \Delta$, then literal $x_k$ is assigned $true$, else $x_k$ is assigned $false$ ($\bar{x_k}$ is assigned $tr$ 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 $true$. Si $(V, E, D \cup \Delta, L)$ is strongly-acyclic with respect to $R$, for every clause $C_i$, for some $l_{ij}$, $j = 1$, there must be a dependency $(N_{ij}, Oij) \rightarrow (O_{ij}, P_{ij})$ (else there would be a strong-cycle in the TS $(V, E, D \cup \Delta, L)$ consistent with $RT_1$). We show that $l_{ij}$ must be assigned $true$, for which we nee 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 else there would be a strong-cycle in the TSGD $(V, E, D \cup \Delta, L)$ consistent with $RT_2$. Thus, $x$

more understandable). For all $j = 1, 2, 3$, nodes $P_{ij}, N_{ij}, P'_{ij}$ and $N_{ij}$ are transaction nodes while no $M_{ij}, O_{ij}, L_{ij}$ 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 $M_{ij}$, $L_{i(j \bmod 3)+1}$ and $O'_{ij}$ respectively are of type $b$. Furthermore, for every literal $x_k$, we include the no and edges shown in Figure 19 in the TSGD.

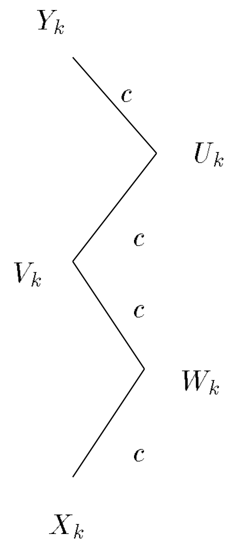


Figure 19: Nodes and edges for literal $x_k$

$U_k$ 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 dependen illustrated in Figure 20 are added to the TSGD.

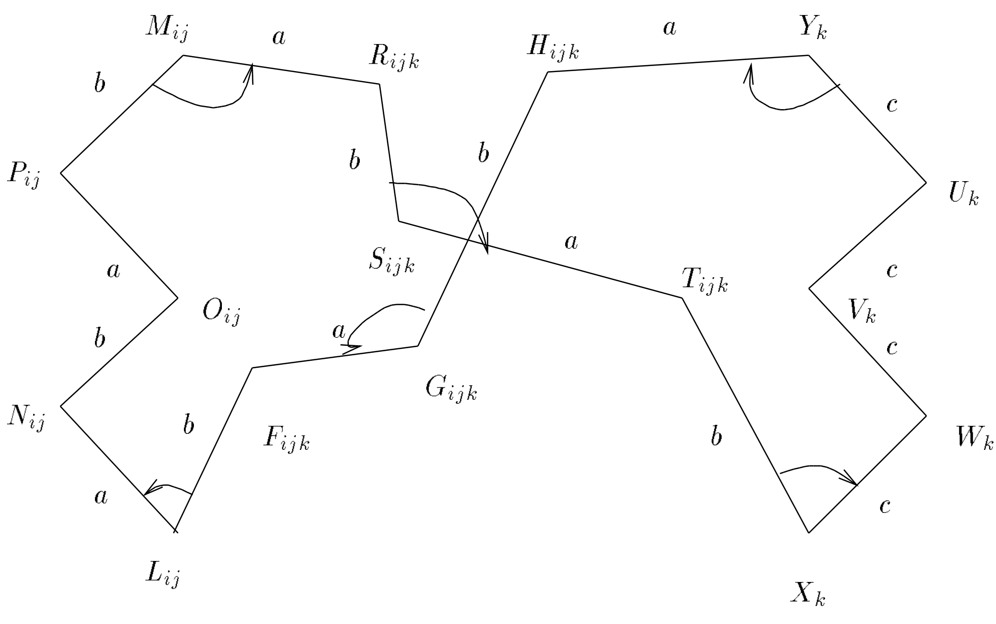


Figure 20: Nodes and edges if $l_{ij} = x_k$

On the other hand, if $l_{ij} = \bar{x_k}$, then we include nodes, edges and dependencies in the TSGD sh

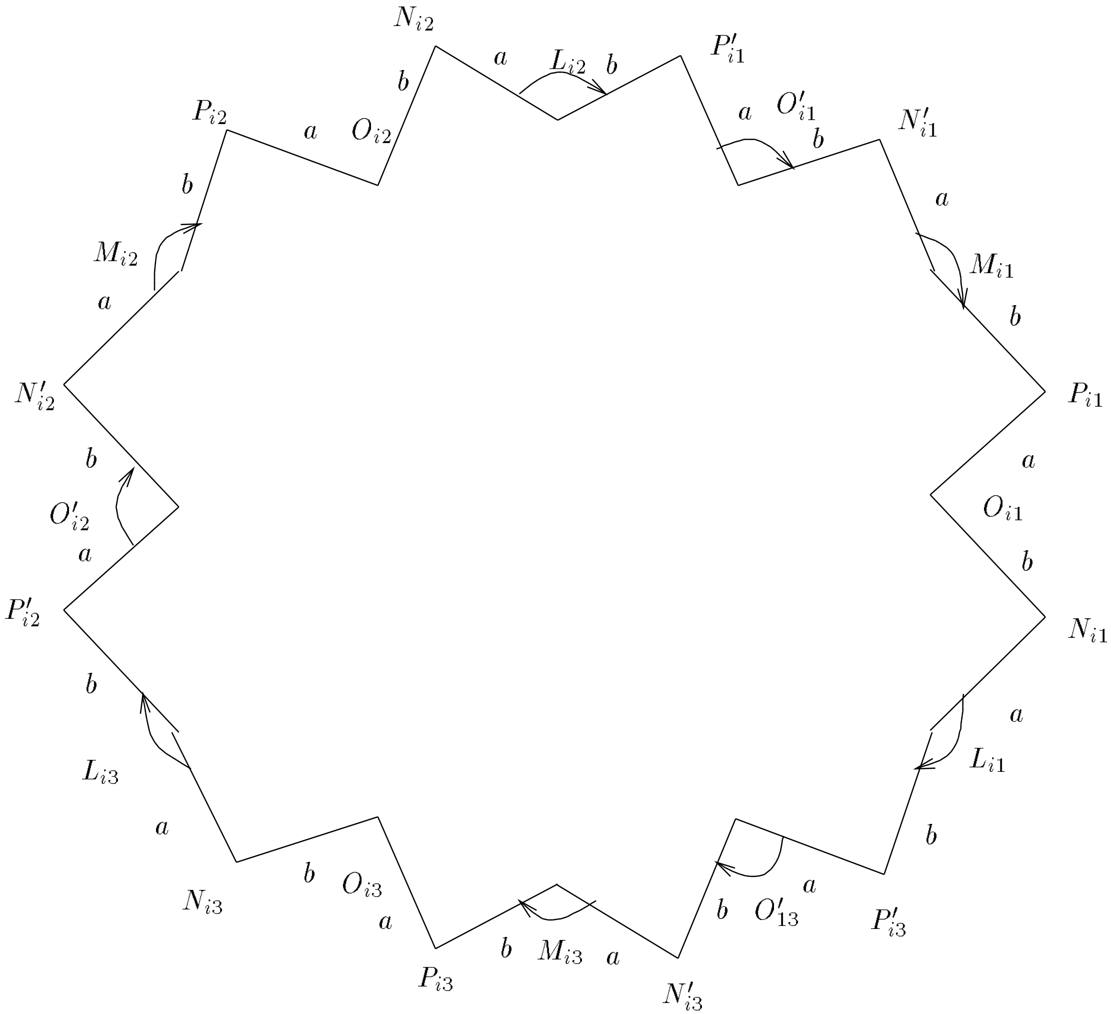

Figure 18: Structure for clause $C_i$

**Proof of Theorem 8:** The above problem is in NP since a non-deterministic algorithm only ne to guess a set $\Delta$ such that there are dependencies between any two edges in the TSGD. $\Delta$ can con 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 $RT$ i and every node $v$ in the TSGD, if there is a strong-cycle consistent with $RT$ involving $v$ in the TS Step 1 can be performed in polynomial time and involves detecting cycles in a directed graph. Ste , too, can be performed in polynomial time using an algorithm similar to Detect_Ins_Opt that gi arguments a TSGD such that between any two edges there is a dependency, a node $v$ in the TSGD a regular term $RT$, precisely detects if the TSGD contains a strong-cycle involving $v$ that is consis with $RT$.

We show a polynomial transformation from 3-SAT to the above problem. Consider a 3-SAT form $C = C_1 \wedge C_2 \wedge \cdots \wedge C_p$ that is defined over literals $x_1, x_2, \ldots, x_q$. Let $l_{ij}$ denote the literal in cla $C_i$, $i = 1, 2, \ldots, p$, in position $j$, $j = 1, 2, 3$ ($l_{ij}$ could be either $x_k$ or $\bar{x_k}$, for some $k = 1, 2, \ldots$ We construct a TSGD $(V, E, D, L)$ and a regular expression $R$ such that $C$ is satisfiable if and o 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 $g\tau = \{A\}$. 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 show

- $(x_i, b'_i), (b'_i, N_{i1}), (N_{i1}, Z_{i1}), (Z_{i1}, Y_{i1}), (Y_{i1}, neg_i(1)), (neg_i(1), N_{i2}), (N_{i2}, Z_{i2}), \ldots,$ $(Y_{i|neg_i|}, neg_i(|neg_i|)), (neg_i(|neg_i|), N_{i(|neg_i+1)}), (N_{i(|neg_i+1)}, e'_i), (e'_i, x_{i+1})$, if $|neg_i| > 0$,

- $(x_i, b'_i), (b'_i, N_{i1}), (N_{i1}, e'_i), (e'_i, x_{i+1})$, if $|neg_i| = 0$,

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 $(x_{i+1}, e_i$ $(e_i, P_{i(|pos_i|+1)})$, $(x_{i+1}, e'_i) \rightarrow (e'_i, N_{i(|neg_i|+1)})$ in $D$, and

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

Finally the strong-cycle contains the edges $(x_{q+1}, s_2)$ and $(s_2, G_i)$. Note that no node in the strong-c is visited more than once. Trivially, all the nodes other than $l_{ij}$ appear only once in the strong-cy Furthermore, if $l_{ij} = pos_r(k)$ (the argument if $l_{ij} = neg_r(k)$ is similar), then $l_{ij}$ cannot be in the seque of edges between both $C_i$ and $C_{i+1}$ as well as $x_r$ and $x_{r+1}$ since $D \cup \{(R_{ij}, l_{ij}) \rightarrow (l_{ij}, B_{ij}), (P_{r(k+1)}, l_{i}$ $(l_{ij}, W_{rk})\}$ 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 $k = 1, 2, \ldots, q$, s that for all $i = 1, 2, \ldots, p$, for some $j = 1, 2, 3$, $l_{ij}$ is assigned $true$, and thus $C$ is satisfiable. For $i = 1, 2, \ldots, p$, for all $j = 1, 2, 3$, $l_{ij}$ is assigned $true$ iff the edges $(B_{ij}, l_{ij}), (l_{ij}, R_{ij})$ are in the stro cycle. This assignment causes $C$ to be $true$ since as shown earlier, for all $i = 1, 2, \ldots, p$, for s $j = 1, 2, 3$, 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 $true$. If $x_k$ $\bar{x_k}$ are both assigned $true$, then there must exist symbols $l_{ij}$ and $l_{rs}$ such that edges $(B_{ij}, l_{ij}), (l_{ij}, I$ $(B_{rs}, l_{rs}), (l_{rs}, R_{rs})$ are in the strong-cycle, and $l_{ij} = x_k$, $l_{rs} = \bar{x_k}$. Thus, $|neg_k| > 0$, $|pos_k| >$ $l_{ij} = pos_k(u)$, for some $u$, $u = 1, 2, \ldots, |pos_k|$, and $l_{rs} = neg_k(v)$, for some $v$, $v = 1, 2, \ldots, |ne$ However, this is not possible, since as we showed earlier, one of $l_{ij}$ and $l_{rs}$ is also in the sequenc edges between $x_k$ and $x_{k+1}$ in the strong-cycle, and the strong-cycle does not visit a node more t once. □

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

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

Consider a subroutine $S((V, E, D, L), G_i)$ that returns a set of dependencies $\Delta$ such that $D \cup$ consistent and $\Delta$ is strongly-minimal with respect to $G_i$ and $(V, E, D, L)$ (note that such a $\Delta$ alw exists if $(V, E, D, L)$ satisfies the conditions mentioned in the theorem). An algorithm for solving 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 algori responds "yes" (since if $\Delta' = \emptyset$ is strongly-minimal with respect to $G_i$ and $(V, E, D, L)$, then a r 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 " (since $\Delta' = \emptyset$ is strongly-minimal with respect to $(V, E, D, L)$ and $G_i$). □

We now prove Theorem 8.

cycle. Since $C$ is satisfiable, there exists an assignment of truth values to $x_k$, for all $k = 1, 2, ..$ such that for all $i = 1, 2, \ldots, p$, for some $j = 1, 2, 3$, $l_{ij}$ is assigned *true*. We now specify the e 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 s $j = 1, 2, 3$ such that $l_{ij}$ is *true* in the assignment. Edges $(C_{p+1}, s_0), (s_0, x_1)$ are also in the strong-cy For all $i = 1, 2, \ldots, q$, if $x_i$ is *false* in the assignment, then the following edges are in the strong-cy

- $(x_i, b_i), (b_i, P_{i1}), (P_{i1}, X_{i1}), (X_{i1}, W_{i1}), (W_{i1}, pos_i(1)), (pos_i(1), P_{i2}), (P_{i2}, X_{i2}), \ldots,$ $(W_{i|pos_i|}, pos_i(|pos_i|)), (pos_i(|pos_i|), P_{i(|pos_i+1)}), (P_{i(|pos_i+1)}, e_i), (e_i, x_{i+1})$, if $|pos_i| > 0$,
- $(x_i, b_i), (b_i, P_{i1}), (P_{i1}, e_i), (e_i, x_{i+1})$, if $|pos_i| = 0$,

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

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

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-cy Nodes other than $l_{ij}$, trivially, appear only once. For any node $l_{ij}$, it is in the sequence of edges betw nodes $C_i$ and $C_{i+1}$ only if $l_{ij}$ is *true* in the assignment. If $l_{ij} = pos_r(k)$, then $l_{ij} = x_r$, and since is *true* in the assignment, $l_{ij}$ is not among the nodes in the sequence of edges between $x_r$ and $x$ Similarly, if $l_{ij} = neg_r(k)$, then $l_{ij} = \bar{x_r}$, and since $x_r$ is *false* in the assignment, $l_{ij}$ is not among 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 $(v_1, v_2$ $(v_2, v_3) \notin D$, and
- for all $l_{ij} = pos_r(k)$, $D \cup \{(R_{ij}, l_{ij}) \rightarrow (l_{ij}, B_{ij})$ is consistent and $D \cup \{(P_{r(k+1)}, l_{ij}) \rightarrow (l_{ij}, W_{rk})$ consistent, and
- for all $l_{ij} = neg_r(k)$, $D \cup \{(R_{ij}, l_{ij}) \rightarrow (l_{ij}, B_{ij})$ 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 ass ment of truth values to literals such that $C$ is satisfiable. Any strong-cycle involving $G_i$ in the TS 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 $l_{ij} = pos_r(k)$, then the dependencies $(W_{rk}, X_{rk}$ $(X_{rk}, P_{rk}) \in D$ and $(B_{ij}, l_{ij}) \rightarrow (l_{ij}, P_{r(k+1)}) \in D$ (a similar set of dependencies can be identified in $l_{ij} = neg_r(k)$). 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

- $(x_i, b_i), (b_i, P_{i1}), (P_{i1}, X_{i1}), (X_{i1}, W_{i1}), (W_{i1}, pos_i(1)), (pos_i(1), P_{i2}), (P_{i2}, X_{i2}), \ldots,$ $(W_{i|pos_i|}, pos_i(|pos_i|)), (pos_i(|pos_i|), P_{i(|pos_i+1)}), (P_{i(|pos_i+1)}, e_i), (e_i, x_{i+1})$, if $|pos_i| > 0$,
- $(x_i, b_i), (b_i, P_{i1}), (P_{i1}, e_i), (e_i, x_{i+1})$, if $|pos_i| = 0$,

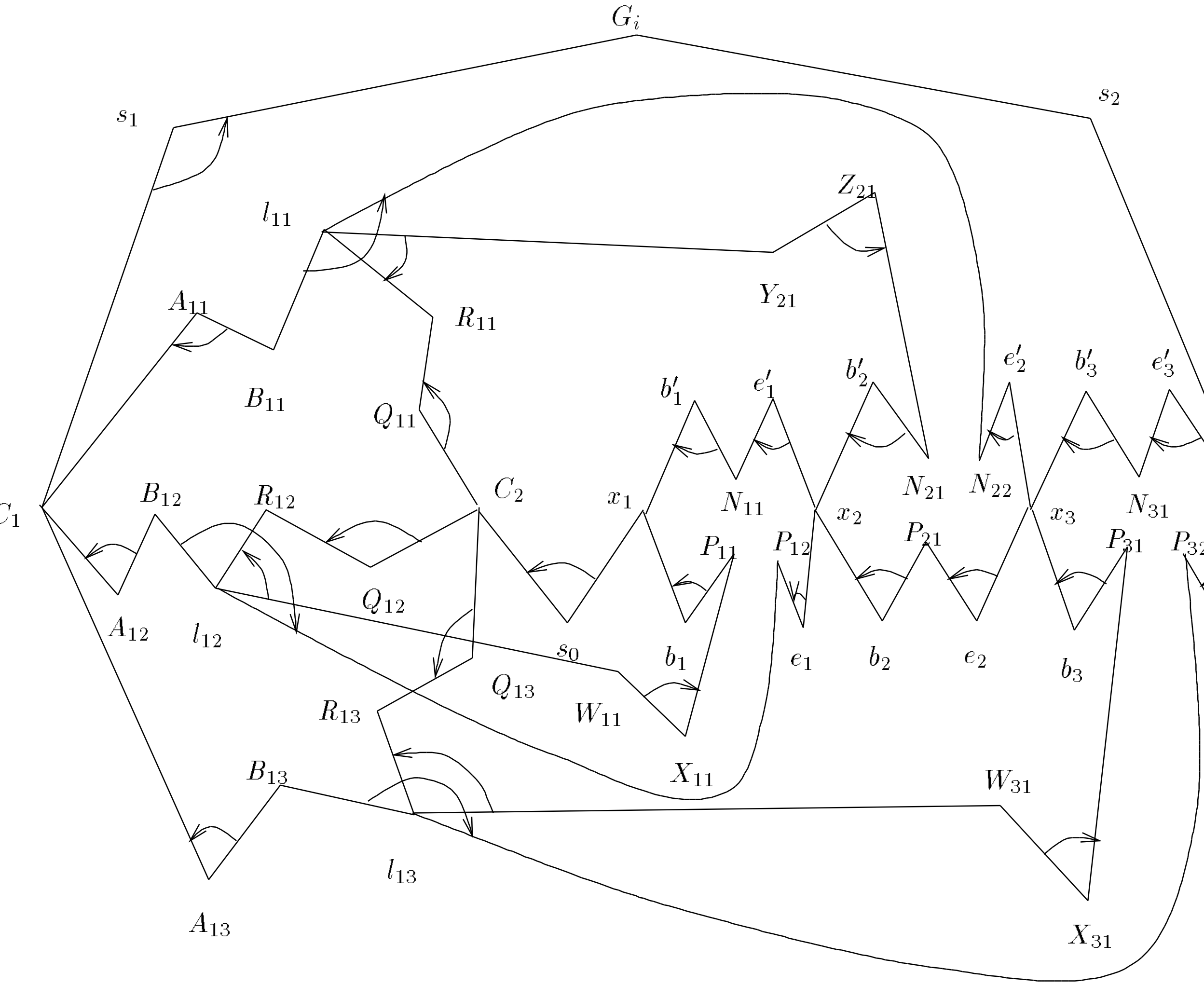


Figure 17: TSGD

only if $r < s$). In addition, there is no strong-cycle in $(V', E', D', L')$ consisting of transaction nodes f 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 $v_2 \in S_1$ ($s_0$ and $l_{ij}$ are the only site nodes that have edges to transaction node 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 $(W_{r(k+1)}, X_{r(k+1)})(X_{r(k+1)}, P_{r(k+}$ However, due to the dependency $(W_{r(k+1)}, X_{r(k+1)}) \rightarrow (X_{r(k+1)}, P_{r(k+1)})$, this is not possible. On 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 $(x_{r+1}, e_r)(e_r, P_{r(|pos_r|+1)})$. Howe due to the dependency $(x_{r+1}, e_r) \rightarrow (e_r, P_{r(k+1)})$, 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 $v_2 = B_{ij}$. Howe since only two edges are incident on $A_{ij}$ and $B_{ij}$, the sequence of edges immediately following $B_{i}$ the cycle must be $(B_{ij}, A_{ij})(A_{ij}, C_i)$ which is not possible due to the dependency $(B_{ij}, A_{ij}) \rightarrow (A_{ij},$ Thus, there can be no strong-cycle in $(V', E', D', L')$ consisting of transaction nodes from both $S_1$ $S_2$, 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

  - $(x_i, b'_i), (b'_i, N_{i1}), (N_{i1}, Z_{i1}), (Z_{i1}, Y_{i1}), (Y_{i1}, neg_i(1)), (neg_i(1), N_{i2}), (N_{i2}, Z_{i2}), \ldots,$ $(Y_{i|neg_i|}, neg_i(|neg_i|)), (neg_i(|neg_i|), N_{i(|neg_i|+1)}), (N_{i(|neg_i|+1)}, e'_i), (e'_i, x_{i+1})$, if $|neg_i| > 0$,
  - $(x_i, b'_i), (b'_i, N_{i1}), (N_{i1}, e'_i), (e'_i, x_{i+1})$, 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 $e_i$, $e'_i$, $b_i$, $b'_i$, $A_{ij}$, $B_{ij}$, $Q_{ij}$, $R_{ij}$, $P_{ij}$, $W$ $X_{ij}$, $N_{ij}$ $Y_{ij}$ 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

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

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

- for $i = 1, 2, \ldots, q$,

  - $(P_{i1}, b_i) \rightarrow (b_i, x_i), (W_{i1}, X_{i1}) \rightarrow (X_{i1}, P_{i1}), (W_{i2}, X_{i2}) \rightarrow (X_{i2}, P_{i2}), \ldots,$ $(W_{i|pos_i|}, X_{i|pos_i|}) \rightarrow (X_{i|pos_i|}, P_{i|pos_i|}), (x_{i+1}, e_i) \rightarrow (e_i, P_{i(|pos_i|+1)})$, if $|pos_i| > 0$,
  - $(P_{i1}, b_i) \rightarrow (b_i, x_i), (x_{i+1}, e_i) \rightarrow (e_i, P_{i1})$, if $|pos_i| = 0$,
  - $(N_{i1}, b'_i) \rightarrow (b'_i, x_i), (Y_{i1}, Z_{i1}) \rightarrow (Z_{i1}, N_{i1}), (Y_{i2}, Z_{i2}) \rightarrow (Z_{i2}, N_{i2}), \ldots,$ $(Y_{i|neg_i|}, Z_{i|neg_i|}) \rightarrow (Z_{i|neg_i|}, N_{i|neg_i|}), (x_{i+1}, e'_i) \rightarrow (e'_i, N_{i(|neg_i|+1)})$, if $|neg_i| > 0$,
  - $(N_{i1}, b'_i) \rightarrow (b'_i, x_i), (x_{i+1}, e'_i) \rightarrow (e'_i, N_{i1})$, if $|neg_i| = 0$,

- for each symbol $l_{ij}$,

  - if $l_{ij} = pos_r(k)$, then the following dependencies are in $D$: $(W_{rk}, l_{ij}) \rightarrow (l_{ij}, R_{ij})$ $(B_{ij}, l_{ij}) \rightarrow (l_{ij}, P_{r(k+1)})$.
  - if $l_{ij} = neg_r(k)$, then the following dependencies are in $D$: $(Y_{rk}, l_{ij}) \rightarrow (l_{ij}, R_{ij})$ and $(B_{ij}, l_{i}$ $(l_{ij}, N_{r(k+1)})$.

- $(C_1, s_1) \rightarrow (s_1, G_i)$,

It is easy to see that the number of steps required to construct the TSGD $(V, E, D, L)$ is $O(p + q)$ $C = \bar{x_2} \vee x_1 \vee x_3$, 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 $G_i$. begin by showing that the TSGD $(V, E, D, L)$ satisfies the conditions. In $D$, the only depende 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 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 $D' = D - \{(C_1, s_1) \rightarrow (s_1, G$ 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 $S_2 = \{x_1, x_2, \ldots, x_{q+1}$ $\{N_{rk}, Y_{rk} : r = 1, 2, \ldots, q, k = 1, 2, \ldots, |neg_r|\} \cup \{P_{rk}, W_{rk} : r = 1, 2, \ldots, q, k = 1, 2, \ldots, |pos_r|$ $\{P_{r(|pos_r|+1)}, N_{r(|neg_r|+1)} : r = 1, 2, \ldots, q\}$. 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 $(B_{ij}, A_{ij}$ $(A_{ij}, C_i)$, $(C_{i+1}, Q_{ij}) \rightarrow (Q_{ij}, R_{ij})$, for all $i = 1, 2, \ldots, p$, for all $j = 1, 2, 3$, a sequence of edges from $C$ $C_s$ can be part of a strong-cycle only if $r < s$). Similarly, there can be no strong-cycle in $(V', E', D'$

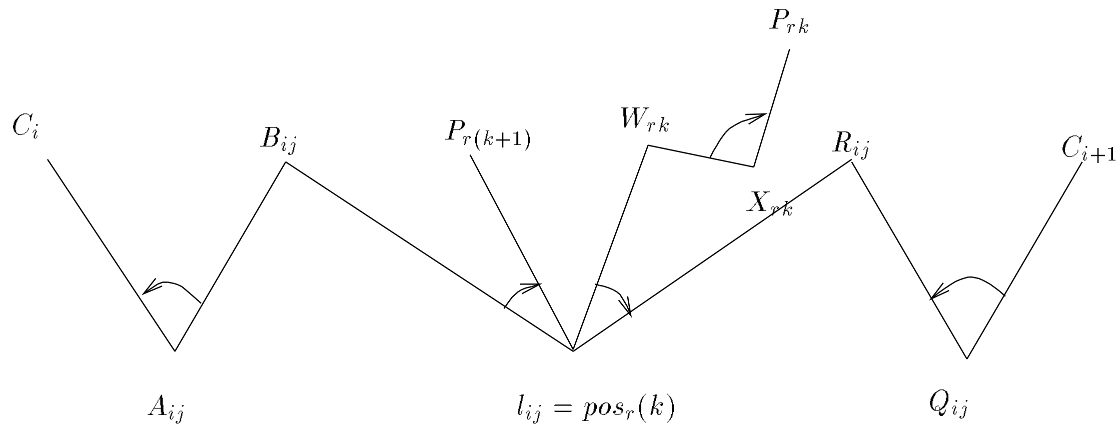


Figure 15: Edges and Dependencies if $l_{ij} = pos_r(k)$

On the other hand, if $l_{ij} = neg_r(k)$, then edges and dependencies shown in Figure 16 are introdu
in the TSGD.

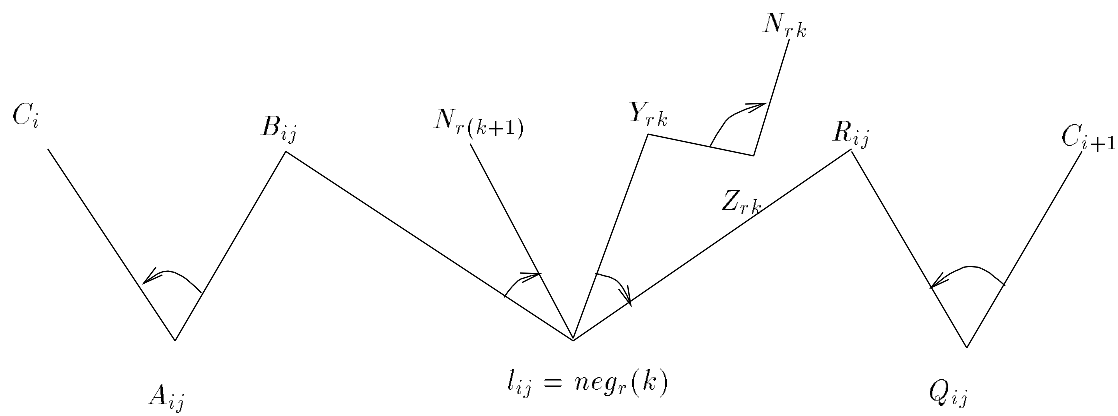


Figure 16: Edges and Dependencies if $l_{ij} = neg_r(k)$

We now describe the nodes, edges and dependencies in the TSGD. The set of nodes $V$ consis 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 addition to $P_{r(|pos_r|+1)}$, $P_{rk}$, $W_{rk}$, for all $r = 1, 2, \ldots, q$, $k = 1, 2, \ldots, |pos_r|$, and for all $r = 1, 2, \ldots$ $N_{r(|neg_r|+1)}$, $N_{rk}$, $Y_{rk}$ $k = 1, 2, \ldots, |neg_r|$. Site nodes consist of $l_{ij}$, $A_{ij}, Q_{ij}$, $i = 1, 2, \ldots, p$, $j = 1,$ for all $i$, $i = 1, 2, \ldots, q$, $e_i, e_i', b_i, b_i'$, $X_{rk}$ for all $r = 1, 2, \ldots, q$, $k = 1, 2, \ldots, |pos_r|$, and $Z_{rk}$ fo $r = 1, 2, \ldots, q$, $k = 1, 2, \ldots, |neg_r|$ in addition to new symbols $s_0, s_1, s_2$.

The set of edges $E$ consist 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 $i = 1, 2, \ldots, p$, fo $j = 1, 2, 3$,

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

- for $i = 1, 2, \ldots, q$,

  - $(x_i, b_i), (b_i, P_{i1}), (P_{i1}, X_{i1}), (X_{i1}, W_{i1}), (W_{i1}, pos_i(1)), (pos_i(1), P_{i2}), (P_{i2}, X_{i2}), \ldots,$
  $(W_{i|pos_i|}, pos_i(|pos_i|)), (pos_i(|pos_i|), P_{i(|pos_i|+1)}), (P_{i(|pos_i|+1)}, e_i), (e_i, x_{i+1})$, if $|pos_i| > 0$,

## 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 transac node $G_i \in V$, such that $D$ is consistent, and for all transactions $G_j \in V$, for all sites $s_k$, depende $(G_i, s_k) \rightarrow (s_k, G_j) \notin D$. 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 quantif tion over $\emptyset$ is always *true*, by the definition of strong-minimality, $\Delta$ is strongly-minimal with resp to $G_i$ and $(V, E, D, L)$ iff $(V, E, D, L)$ does not contain any strong-cycles involving $G_i$. As a resul suffices to show that the following problem is NP-complete: Does $(V, E, D, L)$ contain a strong-c involving $G_i$ ?

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

We show a polynomial transformation from 3-SAT. Consider a formula in *Conjunctive Normal F* (CNF) $C = C_1 \wedge C_2 \wedge \cdots \wedge C_p$ that is defined over literals $x_1, x_2, \ldots, x_q$. Let $l_{ij}$, $i = 1, 2, \ldots, p$, $j = 1,$ be a new symbol for the $j^{th}$ literal in clause $C_i$. Each symbol $l_{ij}$ is either $x_k$ or $\bar{x_k}$, $k = 1, 2, \ldots, q$ 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 introd new symbols $A_{ij}$, $B_{ij}$, $Q_{ij}$ and $R_{ij}$. For $r = 1, 2, \ldots, q$, $pos_r$ denotes the sequence of symbols $l_i$ the order of increasing $i$, such that $l_{ij} = x_r$. For $r = 1, 2, \ldots, q$, $neg_r$ denotes the sequence of syml $l_{ij}$ in the order of increasing $i$, such that $l_{ij} = \bar{x_r}$. Also $|pos_r|$ denotes the number of elements in sequence $pos_r$ and for $k = 1, 2, \ldots, |pos_r|$, $pos_r(k)$ denotes the $k^{th}$ element in the sequence $pos_r$ ($|n$ and $neg_r(k)$ are similarly defined). For all $r = 1, 2, \ldots, q$, we introduce new symbols $P_{rk}$, $W_{rk}$, for each $pos_r(k)$, $k = 1, 2, \ldots, |pos_r|$, and $P_{r(|pos_r|+1)}$; for $r = 1, 2, \ldots, q$, new symbols $N_{rk}$, $Y_{rk}$, $Z_{rk}$ each $neg_r(k)$, $k = 1, 2, \ldots, |neg_r|$, and $N_{r(|neg_r|+1)}$. We illustrate the notation by means of the follow example ("$\cdot$" is the concatenation operator for sequences and "$\epsilon$" is the empty sequence).

**Example:** Let $C = (x_1 \vee \bar{x_3} \vee x_4) \wedge (\bar{x_2} \vee \bar{x_1} \vee x_3) \wedge (\bar{x_2} \vee \bar{x_4} \vee x_1)$.
$l_{1,1} = x_1$, $l_{2,2} = \bar{x_1}$, $l_{3,2} = \bar{x_4}$.
$pos_1 = l_{1,1} \cdot l_{3,3}$, $neg_1 = l_{2,2}$, $pos_2 = \epsilon$.
Also, $|pos_1| = 2$, $|pos_2| = 0$, $|neg_2| = 2$.
$pos_1(1) = l_{1,1}$, $pos_1(2) = l_{3,3}$, $neg_1(1) = l_{2,2}$, $neg_2(2) = l_{3,1}$. □

We now construct the TSGD as follows. The main components in the TSGD are the edges 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 $F = FA(RT_2)$, $state_F(init\_st_F, edge(t_1) \cdots edge(t_{n-}$
$(sfirst(t_0), G_0))$ is an accept state. Let $edge(t_1) \cdots edge(t_{n-1})(sfirst(t_0), G_0) = (v_1, v_2) \cdots (v_{2m-1},$
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 $g_i = \overline{L(v_{2i-1}, v_{2i})}$, and

$st_F(init\_st_F, g_1 \cdots g_{m-1})$ is an accept state. We construct the sequence $g_1 \cdots g_{m-1}$ with the ab
properties as follows. For all $i = 1, \ldots, n-1$, let $f_i = \overline{(type(hdr(t_i)), type(first(t_i)))}$, if $arity(t_i)$
else, $f_i = (type(hdr(t_i)), type(first(t_i)))(type(hdr(t_i)), type(last(t_i)))$. Since $type(t_1) \cdots type(t_{n-1}$
a string in $L(reg\_exp)$, by the construction of $FA(RT_2)$, it follows that $st_F(init\_st_F, f_1 \cdots f_{n-1})$ is
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 defini
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 $arity(t_k)$
then $g_i = L(v_{2i-1}, v_{2i})$, else $g_i = \overline{L(v_{2i-1}, v_{2i})}$.

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 s
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 $v_{2i-1} = v_{2i+1}$, t
$g_i = \overline{L(v_{2i-1}, v_{2i})}$. We first show that if $v_{2i} = v_{2i+1}$, and $(v_{2i-1}, v_{2i}) \in edge(t_k)$ for some $k$,
$1, 2, \ldots, n-1$, then $arity(t_k) = 2$. Suppose $arity(t_k) = 1$. Since $last(t_k)$ and $first(t_{(k+1)\mathrm{mo}}$
execute at the same site, $slast(t_k) = v_{2i-1}$, $sfirst(t_{(k+1)\mathrm{mod}n}) = v_{2i+1}$, it follows that $v_{2i-1} = v_2$
which leads to a contradiction. Thus, $arity(t_k) = 2$, and $g_i = L(v_{2i-1}, v_{2i})$. Also, it can be sh
that if $v_{2i-1} = v_{2i+1}$, and $(v_{2i-1}, v_{2i}) \in edge(t_k)$, then $arity(t_k) = 1$. Suppose $arity(t_k) = 2$.
$v_{2i} = G_k$, then $v_{2i} = v_{2i+1} = G_k$, which leads to a contradiction. If $v_{2i-1} = G_k$, then since $last(t_k)$
$first(t_{(k+1)\mathrm{mod}n})$ execute at the same site, $slast(t_k) = v_{2i}$, $sfirst(t_{(k+1)\mathrm{mod}n}) = v_{2i+1}$, it follows
$v_{2i} = v_{2i+1}$, which leads to a contradiction. Thus, $arity(t_k) = 1$, and, $g_i = \overline{L(v_{2i-1}, v_{2i})}$.

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. T
by corollaries 8 and 10, during the execution of Detect_Ins_TSGD?$((V, E, D, L), G_0, slast(t_0),$
$set_1, RT_2)$, dependency $(prev\_anc(sfirst(t_0)), sfirst(t_0)) \rightarrow (sfirst(G_0), G_0)$ is added to $\Delta$, and
$(prev\_anc(sfirst(t_0)), sfirst(t_0)) \rightarrow (sfirst(t_0), G_0) \in \Delta_F$. However, this leads to a contradiction s
we showed earlier that $(prev\_anc(sfirst(t_0)), sfirst(t_0)) \rightarrow (sfirst(t_0), G_0) \notin \Delta_F$. Thus, every sched
$S$ is correct. $\Box$

When $init_0$ is processed, the procedure Detect_Ins_TSGD? is invoked with arguments that incl the TSGD $(V, E, D, L)$, $G_0$, $slast(t_0)$, $set_1$, and $RT_2$ since $type(G_0) = hdr(e_0)$ and $type(last(t_0)$ $last(e_0)$. Also, $sfirst(t_0) \in set_1$ (if $arity(t_0) = 1$, then since $sfirst(t_0) = slast(t_0)$, $sfirst(t_0) \in s$ if $binary(t_0)$, then since $sfirst(t_0) \neq slast(t_0)$, and $type(first(t_0)) = first(e_0)$, $sfirst(t_0) \in se$ Furthermore, all the edges belonging to $G_0, \ldots, G_{n-1}$ are in the TSGD when Detect_Ins_TSGD invoked. In order to show this, we first show that $G_j$'s edges cannot be deleted from the TS before $G_{(j+1)\mathrm{mod}n}$'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 $G_{(j+1)\mathrm{mod}n}$'s edges deleted from the TSGD. Let $slast(t_j) = s_k$. Since $G_{jk}$ is serialized after $G_{((j+1)\mathrm{mod}n)k}$, at site $ser_k(G_{(j+1)\mathrm{mod}n})$ executes before $ser_k(G_j)$. Thus, since $G_{(j+1)\mathrm{mod}n}$'s edges are inserted into the TS before $ser_k(G_{(j+1)\mathrm{mod}n})$ executes, while $G_j$'s edges are deleted after $ser_k(G_j)$ executes, $G_{(j+1)\mathrm{mo}}$ edges must be in the TSGD when $G_j$'s edges are deleted (since we have assumed that $G_j$'s e are deleted before $G_{(j+1)\mathrm{mod}n}$'s edges are deleted). Furthermore, since $ser_k(G_j)$ and $ser_k(G_{(j+1)\mathrm{mo}}$ must have both executed when $G_j$'s edges are deleted, $G_{(j+1)\mathrm{mod}n}$ is serialized before $G_j$ when edges are deleted. However, this leads to a contradiction, since edges belonging to $G_j$ and $G_{(j+1)\mathrm{n}}$ are deleted together when $fin_l$ for some transaction $G_l$ is processed (since $G_{(j+1)\mathrm{mod}n}$ is seria before $G_j$, if for every transaction $G_k \in V$ serialized before $G_j$, $val_k$ has been processed, then for ev transaction $G_k \in V$ serialized before $G_{(j+1)\mathrm{mod}n}$ also, $val_k$ must have been processed). Thus, edges are not deleted from the TSGD before $G_2$'s edges are deleted, …, $G_{n-1}$'s edges are not dele from the TSGD before $G_0$'s edges are deleted. By transitivity and since $G_0$'s edges are deleted after $init_0$ has been processed, when Detect_Ins_TSGD? is invoked during the processing of $init_0$, 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 $(G_0, slast($ $edge(t_1) \cdots edge(t_{n-1})(sfirst(t_0), G_0)$ is a path in the TSGD $(V, E, D \cup \Delta_F)$. We begin by show that any two consecutive edges in the path have a common node. Consecutive edges in the path co be one of the following:

- $(sfirst(G_j), G_j)(G_j, slast(G_j))$, $j = 1, 2, \ldots, n-1$, where $arity(t_j) = 2$ ($G_j$ is the common no
- $(G_j, slast(t_j))(sfirst(t_{(j+1)\mathrm{mod}n}), G_{(j+1)\mathrm{mod}n})$, $j = 0, 1, \ldots, n-1$, where $arity(t_j) = 2$ or $j =$ $arity(t_{(j+1)\mathrm{mod}n}) = 1$ or $2$ (since for all $j$, $j = 0, 1, \ldots, n-1$, $last(t_j)$ and $first(t_{(j+1)\mathrm{mo}}$ execute at the same site, $slast(t_j) = sfirst(t_{(j+1)\mathrm{mod}n})$ is the common node).
- $(sfirst(t_j), G_j)(sfirst(t_{(j+1)\mathrm{mod}n}), G_{(j+1)\mathrm{mod}n})$, $j = 1, 2, \ldots, n-1$, where $arity(t_j) = 1$, ar $t_{(j+1)\mathrm{mod}n}) = 1$ or $2$ (since $arity(t_j) = 1$ implies that $sfirst(t_j) = slast(t_j)$, and $slast(t_j$ $sfirst(\ t_{(j+1)\mathrm{mod}n})$, it follows that $sfirst(t_j) = sfirst(t_{(j+1)\mathrm{mod}n})$ is the common node).

Also, for the sequence of edges $(sfirst(t_j), G_j)(G_j, slast(t_j))$ in the path, $j = 1, 2, \ldots, n-1$, it m be the case that $arity(t_j) = 2$, and thus $sfirst(t_j) \neq slast(t_j)$. Also, if for some $j, k$, $j = 0, 1, \ldots, n$ $j < k \leq n$, the sequence of edges $(G_j, slast(t_j))(sfirst(t_{(j+1)\mathrm{mod}n}), G_{(j+1)\mathrm{mod}n}), \ldots,$ $(sfirst(t_{k\mathrm{mod}n}), G_{k\mathrm{mod}n})$ is in the path, then it must be the case that for all $j < l < k$, $arity(t_l) =$ Thus, by Property 1, it follows that $slast(t_j) = sfirst(t_{(j+1)\mathrm{mod}n}) = \cdots = sfirst(t_{k\mathrm{mod}n})$, and fo $r, s$, $j \leq r < s \leq k$,

- $G_r \neq G_{s\mathrm{mod}n}$, and
- $G_r$ is serialized after $G_{s\mathrm{mod}n}$ at site $sfirst(G_{s\mathrm{mod}n})$. Thus, by Lemma 14, dependency $(G_r, sfi$ $G_{s\mathrm{mod}n}))\rightarrow(sfirst(G_{s\mathrm{mod}n}), G_{s\mathrm{mod}n})$ does not belong to $D \cup \Delta_F$ (since $\Delta_F$ is added to the of dependencies $D$ in the TSGD immediately after $init_0$ is processed).

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 $(V, E, D \cup \Delta_F)$. T

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

- if $v_{2m+2} = v_{2m+3}$, then $St'_k.V\_set(v_{2m+2})$ already contains $(st_{m+1}, (St'_k.v, St'_k.v))$. Thus, si $St'_k.v = v_{2m+1}$, $prev(v_{2m+3}) = v_{2m+1}$, $prev\_anc(v_{2m+3}) = v_{2m+1}$, $(st_{m+1}, (prev\_anc(v_{2m+3}),$ $prev(v_{2m+3})))$ is added to $V\_set(v_{2m+3})$ during the execution of Detect_Ins_TSGD2.

- if $v_{2m+1} = v_{2m+3}$, then $St'_k.V\_set(St'_k.v)$ already contains $(st_{m+1}, (prev\_anc(v_{2m+1}), v_{2m+}$ Thus, since $St'_k.v = v_{2m+1}$, $prev\,(v_{2m+3}) = v_{2m+2}$, $prev\_anc\,(v_{2m+3}) = prev\_anc\,(v_{2m}$ $(st_{m+1},\ (prev\_anc\,(v_{2m+3}),\ prev\,(v_{2m+3})\,)\,)$ is added to $V\_set\,(v_{2m+3})$ during the executio Detect_Ins_TSGD2. □

**Corollary 10:** Let Detect_Ins_TSGD2($(V, E, D, L), v_1, v_2, set_1, RT$) return the set of dependen $\Delta_F$. If the TSGD $(V, E, D \cup \Delta_F)$ contains a path $(v_1, v_2)\cdots(v_{2n-1}, v_{2n})(v_{2n+1}, v_1)$, $v_2 = v_3$, s that for the regular term $RT$, $F = FA(RT)$, $st = state_F(init\_st_F, (v_3, v_4)\cdots(v_{2n-1}, v_{2n})(v_{2n+1},$ is an accept state and $v_{2n+1} \in set_1$, then during the execution of Detect_Ins_TSGD2, depende $(prev\_anc(v_{2n+1}), v_{2n+1}) \rightarrow (v_{2n+1}, v_1)$ is added to $\Delta$.

**Proof:** By Lemma 13, $(st,\ (prev\_anc\,(v_{2n+1}),\ prev\,(v_{2n+1})\,)\,)$ is added to $V\_set\,(v_{2n+1})$. Si $prev(v_{2n+1}) \neq v_1$ and $prev\_anc(v_{2n+1}) \neq v_1$ (by definition of path), Detect_Ins_TSGD2 makes a ward state transition when $(st, (prev\_anc(v_{2n+1}), prev(v_{2n+1})))$ is added to $V\_set(v_{2n+1})$. Howe just before $(st,\ (prev\_anc(v_{2n+1}), prev(v_{2n+1})))$ is added to $V\_set(v_{2n+1})$, since $st$ is an accept st $prev\_anc(v_{2n+1}) \neq v_1$, $prev(v_{2n+1}) \neq v_1$ and $v_{2n+1} \in set_1$, dependency $(prev\_anc(v_{2n+1}), v_{2n+1}$ $(v_{2n+1}, v_1)$ is added to $\Delta$. □

We are now in a position to prove that the TSGD scheme ensures the correctness of $S$. Before 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 be $G_{jk}$ 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 can 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 added to the TSGD before $act(ser_k(G_i))$ executes. However, if this were the case, $act(ser_k(G_i))$ wo not execute until $act(ack(ser_k(G_j)))$ completes execution (the dependency $(G_j, s_k) \rightarrow (s_k, G_i)$ is dele from the TSGD only after $ack(ser_k(G_j))$ is processed). Thus, $ser_k(G_j)$ would execute before $ser_k($ 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 $init_i$ every other transaction $G_i$ in $I$ is processed. By Lemma 1, since $R$ is complete, there exists a reg term $RT_2 = e_0 : reg\_exp$ and an instantiation $t_0 : t_1 t_2 \cdots t_{n-1}$ of $RT_2$ in $S$ such that $hdr(t_0) =$ 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 $first(t_{(j+1) \bmod n})$ execute at the same site, and $last(t_j)$ is serialized a $first(t_{(j+1) \bmod n})$ at the site, and

the conditions in Step 2 need to be checked, on an average, for $v_S$ edges (the average number of s a global transaction executes at is $v_S$), while every time a site node is visited, the conditions in Ste need to be checked for at most $n_G$ edges (since the number of transaction nodes in the TSGD is at m $n_G$). Furthermore, every transaction node can be visited at most $v_S^2 n_S$ times, while every site n can be visited at most $n_G^2 n_S$ times (every node $v$ in the TSGD can be visited in a state $st$ of $F$ at m once for every pair of nodes $u, w$ such that $(v, w)$ and $(v, u)$ are edges in the TSGD, and $F$ has at m $n_S$ states). Since there are $m$ site nodes and at most $n_G$ transaction nodes in the TSGD, the num of times Detect_Ins_TSGD2 checks if an edge satisfies the conditions in Step 2 is $n_G^3 m n_S + n_G v_S^3$ Since each of the conditions in Step 2 can be checked in constant time and $v_S \ll n_G, v_S < m$, tect_Ins_TSGD2 terminates in $O(n_G^3 m n_S)$ steps. □

We now show that Detect_Ins_TSGD2 traverses edges in the TSGD in a manner that ensure detects instantiations of regular terms.

**Lemma 13:** Let Detect_Ins_TSGD2$((V, E, D, L), v_1, v_2, set_1, RT)$ return the set of depend cies $\Delta_F$. If the TSGD $(V, E, D \cup \Delta_F)$ contains a path $(v_1, v_2), (v_3, v_4), \ldots, (v_{2n-3}, v_{2n-2}), (v_{2n-1}, v$ $v_2 = v_3$, such that for the regular term $RT$, $F = FA(RT)$, $state_F(init\_st_F, (v_3, v_4), \ldots, (v_{2n-1}, v_{2n}$ defined, then during the execution of Detect_Ins_TSGD2, $(st, (prev\_anc(v_{2i+1}), prev(v_{2i+1})))$ is adde $V\_set(v_{2i+1})$, where $st = state_F(init\_st_F, (v_3, v_4) \cdots (v_{2i-1}, v_{2i})(v_{2i+1}, v_{2i+2}))$, for all $i$, $i = 1, 2, 3, \ldots$ 1.

**Proof:** We prove the above lemma by induction on $i$. We prove that for all $i$, $i = 1, 2, \ldots, n$ $(st, (prev\_anc(v_{2i+1}), prev(v_{2i+1})))$ 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_TSGD2, $(init\_st_F, (v_1, v_1))$ is added to $V\_set(v_2)$. Since $v_2 =$ $prev\_anc(v_3) = prev(v_3) = v_1$, and $state_F(init\_st_F, (v_3, v_4)) = init\_st_F$, the lemma is true for $i$ $((init\_st_F, (prev\_anc(v_3), prev(v_3)))$ is added to $V\_set(v_3)$).

**Induction:** Let us assume that the lemma is true for $i = m$, $1 \leq m < n - 1$. Thus, $(st_m, (prev\_anc(v_{2m+1}), prev(v_{2m+1})))$ is added to $V\_set(v_{2m+1})$, where $st_m = state_F(init\_st_F, (v_3$ $\cdots (v_{2m-1}, v_{2m})(v_{2m+1}, v_{2m+2}))$. We show the lemma to be true for $i = m + 1$. Thus, we need to sh that $(st_{m+1}, (prev\_anc(v_{2m+3}), prev(v_{2m+3})))$ is added to $V\_set(v_{2m+3})$, where $st_{m+1} = state_F(init\_$ $(v_3, v_4) \cdots (v_{2m+1}, v_{2m+2})(v_{2m+3}, v_{2m+4}))$. By the definition of $state_F$, $st_{m+1} = st_F(st_m, L(v_{2m+1}, v_2$ if $v_{2m+2} = v_{2m+3}$ and $st_{m+1} = st_F(st_m, \overline{L(v_{2m+1}, v_{2m+2})})$, if $v_{2m+1} = v_{2m+3}$.

Let $St_k$ be the resulting state of Detect_Ins_TSGD2 after $(st_m, (prev\_anc(v_{2m+1}), prev(v_{2m+1}))$ added to $V\_set(v_{2m+1})$ (the state $St_k$ results either due to the forward transition $St_j \rightarrow St_k$, eit $St_j.v = v_{2m+1}$ or $St_j.v = prev(v_{2m+1})$, or due to Step 1). Thus, $St_k.v = v_{2m+1}$, $St_k.cur\_st = st_m$ in state $St_k$, $head(St_k.anc(St_k.v)) = (prev\_anc(v_{2m+1}), prev(v_{2m+1}))$. Furthermore, it follows f Lemma 12 that after a finite number of steps, Detect_Ins_TSGD2 is in a state $St'_k$ such that $St'_k \equiv$ and no further forward transitions can be made from $St'_k$. Thus, in state $St'_k$,

- Since $prev(v_{2m+1}) \neq v_{2m+2}$ and $prev\_anc(v_{2m+1}) \neq v_{2m+2}$, $head(St'_k.anc(St'_k.v))[1] \neq v_{2m}$ $head(St'_k.anc(St'_k.v))[2] \neq v_{2m+2}$,

- Since $St'_k.\Delta \subseteq \Delta_F$ and $(v_1, v_2) \cdots (v_{2m+1}, v_{2m+2})$ is a path in $(V, E, D \cup \Delta_F)$, there are no dep dencies $(prev(v_{2m+1}), v_{2m+1}) \rightarrow (v_{2m+1}, v_{2m+2})$ and $(prev\_anc(v_{2m+1}), v_{2m+1}) \rightarrow (v_{2m+1}, v_{2m+2}$ $D \cup \Delta_F$; thus, dependencies $(head(St'_k.anc(St'_k.v))[2], St'_k.v) \rightarrow (St'_k.v, v_{2m+2})$ and $(head(St'_k.anc(St'_k.v))[1], St'_k.v) \rightarrow (St'_k.v, v_{2m+2})$ are not in $D \cup St'_k.\Delta$,

- 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 $v_{2m+2} = v_{2m}$

or $St_j.v = prev\_anc(v_{2m+1})$, or due to Step 1). Thus, $St_k.v = v_{2m+1}$, $St_k.cur\_st = st_m$ and in st $St_k$, $head(St_k.anc(St_k.v)) = prev\_anc(v_{2m+1})$. Furthermore, it follows from Lemma 10 that aft finite number of steps, Detect_Ins_TSGD1 is in a state $St'_k$ such that $St'_k \equiv St_k$ and no further forw transitions can be made from $St'_k$. Thus, in state $St'_k$,

- Since $prev\_anc(v_{2m+1}) \neq v_{2m+2}$ (by the definition of path), $head(St'_k.anc(St'_k.v)) \neq v_{2m+2}$,
- Since $St'_k.\Delta \subseteq \Delta_F$, and $(v_1, v_2) \cdots (v_{2m+1}, v_{2m+2})$ is a path in $(V, E, D \cup \Delta_F)$, there is no pendency $(prev\_anc(v_{2m+1}), v_{2m+1}) \rightarrow (v_{2m+1}, v_{2m+2})$ in $D \cup \Delta_F$; thus, there is no depende $(head(St'_k.anc(St'_k.v)), St'_k.v) \rightarrow (St'_k.v, v_{2m+2})$ in $D \cup St'_k.\Delta$,
- 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 $v_{2m+2} = v_{2n}$ 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 $st_{m+}$ $st_F(St'_k.cur\_st, \overline{L(St'_k.v, v_{2m+2})})$ is defined.

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

- if $v_{2m+2} = v_{2m+3}$, then $St'_k.V\_set(v_{2m+2})$ already contains $(st_{m+1}, St'_k.v)$. Thus, since $St'_k.$ $v_{2m+1}$, $prev\_anc(v_{2m+3}) = v_{2m+1}$, $(st_{m+1}, prev\_anc(v_{2m+3}))$ is added to $V\_set(v_{2m+3})$ during execution of Detect_Ins_TSGD1.
- if $v_{2m+1} = v_{2m+3}$, then $St'_k.V\_set(St'_k.v)$ already contains $(st_{m+1}, prev\_anc(v_{2m+1}))$. Th since $St'_k.v = v_{2m+1}$, $prev\_anc(v_{2m+3}) = prev\_anc(v_{2m+1})$, $(st_{m+1}, prev\_anc(v_{2m+3}))$ is ad to $V\_set(v_{2m+3})$ 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 dependen $\Delta_F$. If the TSGD $(V, E, D \cup \Delta_F)$ contains a path $(v_1, v_2) \cdots (v_{2n-1}, v_{2n})(v_{2n+1}, v_1)$, $v_2 = v_3$, s that for the regular term $RT$, $F = FA(RT)$, $st = state_F(init\_st_F, (v_3, v_4) \cdots (v_{2n-1}, v_{2n})(v_{2n+1},$ is an accept state and $v_{2n+1} \in set_1$, then during the execution of Detect_Ins_TSGD1, depende $(prev\_anc(v_{2n+1}), v_{2n+1}) \rightarrow (v_{2n+1}, v_1)$ is added to $\Delta$.

**Proof:** By Lemma 11, $(st, prev\_anc(v_{2n+1}))$ is added to $V\_set(v_{2n+1})$. Since $prev\_anc(v_{2n+1}) \neq$ Detect_Ins_TSGD1 makes a forward state transition when $(st, prev\_anc(v_{2n+1}))$ is added to $V\_set(v_{2}$ However, just before $(st, prev\_anc(v_{2n+1}))$ is added to $V\_set(v_{2n+1})$, since $st$ is an accept state, $prev\_$ $(v_{2n+1}) \neq v_1$ and $v_{2n+1} \in set_1$, dependency $(prev\_anc(v_{2n+1}), v_{2n+1}) \rightarrow (v_{2n+1}, v_1)$ is added to $\Delta$. □

We now show that Detect_Ins_TSGD2 terminates in $O(n_G^2 m v_S)$ steps, for which we need to pr the following lemma.

**Lemma 12:** If during its execution, Detect_Ins_TSGD2 is in state $St_k$, then after a finite num 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 $O(n_G^3 m n_S)$ steps.

**Proof:** Detect_Ins_TSGD2 can be shown to terminate as a result of Lemma 12 using a sim argument as in Corollary 3.

The number of steps Detect_Ins_TSGD2 terminates in is equal to the product of the numbe times Detect_Ins_TSGD2 checks if an edge satisfies the conditions in Step 2 and the number of s

## Appendix -D- : TSGD Schemes

In this appendix, we prove Theorem 5. We begin by showing that Detect_Ins_TSGD1 and De-
tect_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 num
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 $O(n_G^2 m n_S)$ steps.

**Proof:** Detect_Ins_TSGD1 can be shown to terminate as a result of Lemma 10 using a sim
argument as in Corollary 3.

The number of steps Detect_Ins_TSGD1 terminates in is equal to the product of the numbe
times Detect_Ins_TSGD1 checks if an edge satisfies the conditions in Step 2 and the number of st
required to check if an edge satisfies the conditions in Step 2. Every time a transaction node is visi
the conditions in Step 2 need to be checked, on an average, for $v_S$ edges (the average number of s
a global transaction executes at is $v_S$), while every time a site node is visited, the conditions in Ste
need to be checked for at most $n_G$ edges (since the number of transaction nodes in the TSGD i
most $n_G$). Furthermore, every transaction node can be visited at most $v_S n_S$ times, while every
node can be visited at most $n_G n_S$ times (every node $v$ in the TSGD can be visited in a state $st$ o
at most once for every node $w$ such that edge $(v, w)$ is in the TSGD, and $F$ has at most $n_S$ stat
Since there are $m$ site nodes and at most $n_G$ transaction nodes in the TSGD, the number of ti
Detect_Ins_TSGD1 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 e
of the conditions in Step 2 can be checked in constant time and $v_S \ll n_G, v_S < m$, Detect_Ins_TSG
terminates in $O(n_G^2 m n_S)$ steps. □

We now show that Detect_Ins_TSGD1 traverses edges in the TSGD in a manner that ensure
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 dependen
$\Delta_F$. If the TSGD $(V, E, D \cup \Delta_F)$ contains a path $(v_1, v_2), (v_3, v_4), \ldots, (v_{2n-3}, v_{2n-2}), (v_{2n-1}, v_{2n})$, $v$
$v_3$, 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
fined, then during the execution of Detect_Ins_TSGD1, for all $i$, $i = 1, 2, 3, \ldots, n-1$, $(st, prev\_anc(v_{2i}$
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 for all $i$, $i = 1, 2, \ldots, n$
$(st, prev(v_{2i+1}))$ is added to $V\_set(v_{2i+1})$, where $st = state_F(init\_st_F, (v_3, v_4) \cdots (v_{2i-1}, v_{2i})(v_{2i+1}, v$
**Basis** ($i = 1$): In Step 1 of Detect_Ins_TSGD1, $(init\_st_F, v_1)$ is added to $V\_set(v_2)$. Since $v_2 =$
$prev\_anc(v_3) = v_1$, and $state_F(init\_st_F, (v_3, v_4)) = init\_st_F$, the lemma is true for $i = 1$ ($(init\_$
$prev\_anc(v_3))$ is added to $V\_set(v_3)$).
**Induction:** Let us assume that the lemma is true for $i = m$, $1 \leq m < n-1$. Thus, $(st_m, prev\_anc(v_2$
is added to $V\_set(v_{2m+1})$, where $st_m = state_F(init\_st_F, (v_3, v_4) \cdots (v_{2m-1}, v_{2m})(v_{2m+1}, v_{2m+2}))$.
show the lemma to be true for $i = m + 1$. Thus, we need to show that $(st_{m+1}, prev\_anc(v_{2m}$
is added to $V\_set(v_{2m+3})$, where $st_{m+1} = state_F(init\_st_F, (v_3, v_4) \cdots (v_{2m+1}, v_{2m+2})(v_{2m+3}, v_{2m+}$
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 $st_{m+}$
$st_F(st_m, \overline{L(v_{2m+1}, v_{2m+2})})$, if $v_{2m+1} = v_{2m+3}$.

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

We further use Lemma 3 to show that, for $F = FA(RT_2)$, $state_F(init\_st_F, edge(t_1)\cdots edge(t_{n-}$ $(sfirst(t_0), G_0))$ is an accept state. Let $edge(t_1)\cdots edge(t_{n-1})(sfirst(t_0), G_0) = (v_1, v_2)\cdots(v_{2m-1},$ 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 $g_i = \overline{L(v_{2i-1}, v_{2i})}$, and

$st_F(init\_st_F, g_1\cdots g_{m-1})$ is an accept state. We construct the sequence $g_1\cdots g_{m-1}$ with the ab properties as follows. For all $i = 1, \ldots, n-1$, let $f_i = \overline{(type(hdr(t_i)), type(first(t_i)))}$, if $arity(t_i) =$ else, $f_i = (type(hdr(t_i)), type(first(t_i)))(type(hdr(t_i)), type(last(t_i)))$. Since $type(t_1)\cdots type(t_{n-1}$ a string in $L(reg\_exp)$, by the construction of $FA(RT_2)$, it follows that $st_F(init\_st_F, f_1\cdots f_{n-1})$ i 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 defini 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 $arity(t_k) =$ then $g_i = L(v_{2i-1}, v_{2i})$, else $g_i = \overline{L(v_{2i-1}, v_{2i})}$.

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 s 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 $v_{2i-1} = v_{2i+1}$, t $g_i = \overline{L(v_{2i-1}, v_{2i})}$. We first show that if $v_{2i} = v_{2i+1}$, and $(v_{2i-1}, v_{2i}) \in edge(t_k)$ for some $k$, $1, 2, \ldots, n-1$, then $arity(t_k) = 2$. Suppose $arity(t_k) = 1$. Since $last(t_k)$ and $first(t_{(k+1)\text{m}}$ execute at the same site, $slast(t_k) = v_{2i-1}$, $sfirst(t_{(k+1)\text{mod}n}) = v_{2i+1}$, it follows that $v_{2i-1} = v_2$ which leads to a contradiction. Thus, $arity(t_k) = 2$, and $g_i = L(v_{2i-1}, v_{2i})$. Also, it can be sh that if $v_{2i-1} = v_{2i+1}$, and $(v_{2i-1}, v_{2i}) \in edge(t_k)$, then $arity(t_k) = 1$. Suppose $arity(t_k) = 2$ $v_{2i} = G_k$, then $v_{2i} = v_{2i+1} = G_k$, which leads to a contradiction. If $v_{2i-1} = G_k$, then since $last(t_k)$ $first(t_{(k+1)\text{mod}n})$ execute at the same site, $slast(t_k) = v_{2i}$, $sfirst(t_{(k+1)\text{mod}n}) = v_{2i+1}$, it follows $v_{2i} = v_{2i+1}$, which leads to a contradiction. Thus, $arity(t_k) = 1$, and, $g_i = \overline{L(v_{2i-1}, v_{2i})}$.

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. T by corollaries 4 and 6, during the execution of Detect_Ins_TSG?$((V, E, L), G_0, slast(t_0), set_1, set_2, R$ $sfirst(t_0)$ is added to $\Delta$, and thus $sfirst(t_0) \in \Delta_F$. However, this leads to a contradiction since showed earlier that $sfirst(t_0) \notin set_2 \cup \Delta_F$. Thus, every schedule $S$ is correct. $\square$

**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 $init_i$ every other transaction $G_i$ in $I$ is processed. By Lemma 1, since $R$ is complete, there exists a reg term $RT_2 = e_0 : reg\_exp$ and an instantiation $t_0 : t_1 t_2 \cdots t_{n-1}$ of $RT_2$ in $S$ such that $hdr(t_0) =$ 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 $first(t_{(j+1)\mathrm{mod}n})$ execute at the same site, and $last(t_j)$ is serialized a $first(t_{(j+1)\mathrm{mod}n})$ at the site, and
- $type(t_0) = e_0$ and $type(t_1) \cdots type(t_{n-1})$ is a string in $L(reg\_exp)$.

When $init_0$ is processed, the procedure Detect_Ins_TSG? is invoked with arguments that include TSG $(V, E, L)$, $G_0$, $slast(t_0)$, $set_1$, $set_2$ and $RT_2$ since $type(G_0) = hdr(e_0)$ and $type(last(t_0)) = last($ Also, $sfirst(t_0) \in set_1$ (if $arity(t_0) = 1$, then since $sfirst(t_0) = slast(t_0)$, $sfirst(t_0) \in set_1$ $binary(t_0)$, then since $sfirst(t_0) \neq slast(t_0)$, and $type(first(t_0)) = first(e_0)$, $sfirst(t_0) \in se$ Furthermore, all the edges belonging to $G_0, \ldots, G_{n-1}$ are in the TSG when Detect_Ins_TSG? is invol In order to show this, we first show that $G_j$'s edges cannot be deleted from the TSG before $G_{(j+1)\mathrm{mo}}$ edges are deleted from the TSG, for all $j$, $j = 1, 2, \ldots, n-1$. Suppose, for some $j$, $j = 1, 2, \ldots, n-1$, edges are deleted from the TSG before $G_{(j+1)\mathrm{mod}n}$'s edges are deleted from the TSG. Let $slast(t_j) =$ Since $G_{jk}$ is serialized after $G_{((j+1)\mathrm{mod}n)k}$, at site $s_k$, $ser_k(G_{(j+1)\mathrm{mod}n})$ executes before $ser_k(G_j)$. T since $G_{(j+1)\mathrm{mod}n}$'s edges are inserted into the TSG before $ser_k(G_{(j+1)\mathrm{mod}n})$ executes, while $G_j$'s ec are deleted after $ser_k(G_j)$ executes, $G_{(j+1)\mathrm{mod}n}$'s edges must be in the TSG when $G_j$'s edges are dele (since we have assumed that $G_j$'s edges are deleted before $G_{(j+1)\mathrm{mod}n}$'s edges are deleted). Howe this leads to a contradiction, since edges belonging to $G_j$ and $G_{(j+1)\mathrm{mod}n}$ are deleted together w $fin_l$ for some transaction $G_l$ is processed (due to the sequence of edges between $G_j$ and $G_{(j+1)\mathrm{m}}$ $(G_j, s_k)(s_k, G_{(j+1)\mathrm{mod}n})$, if for every transaction $G_k \in V$ such that there is a sequence of edges f $G_j$ to $G_k$ in the TSG, $val_k$ has been processed, then for every transaction $G_k \in V$ such that there sequence of edges from $G_{(j+1)\mathrm{mod}n}$ to $G_k$, $val_k$ must also have been processed). Thus, $G_1$'s edges not deleted from the TSG before $G_2$'s edges are deleted, …, $G_{n-1}$'s edges are not deleted from TSG before $G_0$'s edges are deleted. By transitivity and since $G_0$'s edges are deleted only after $init_0$ 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 $(G_0, slast(t_0))edge(t_1) \cdots edge(t_{n-1})(sfirst(t_0), G_0)$ is a path in the TSG $(V,$ We begin by showing that any two consecutive edges in the path have a common node. Consecu edges in the path could be one of the following:

- $(sfirst(G_j), G_j)(G_j, slast(G_j))$, $j = 1, 2, \ldots, n-1$, where $arity(t_j) = 2$ ($G_j$ is the common no
- $(G_j, slast(t_j))(sfirst(t_{(j+1)\mathrm{mod}n}), G_{(j+1)\mathrm{mod}n})$, $j = 0, 1, \ldots, n-1$, where $arity(t_j) = 2$ or $j =$ $arity(t_{(j+1)\mathrm{mod}n}) = 1$ or $2$ (since for all $j$, $j = 0, 1, \ldots, n-1$, $last(t_j)$ and $first(t_{(j+1)\mathrm{m}}$ execute at the same site, $slast(t_j) = sfirst(t_{(j+1)\mathrm{mod}n})$ is the common node).
- $(sfirst(t_j), G_j)(sfirst(t_{(j+1)\mathrm{mod}n}), G_{(j+1)\mathrm{mod}n})$, $j = 1, 2, \ldots, n-1$, where $arity(t_j) = 1$, $ar$ $t_{(j+1)\mathrm{mod}n}) = 1$ or $2$ (since $arity(t_j) = 1$ implies that $sfirst(t_j) = slast(t_j)$, and $slast(t_j$ $sfirst(t_{(j+1)\mathrm{mod}n})$, it follows that $sfirst(t_j) = sfirst(t_{(j+1)\mathrm{mod}n})$ is the common node).

Also, for the sequence of edges $(sfirst(t_j), G_j)(G_j, slast(t_j))$ in the path, $j = 1, 2, \ldots, n-1$, it n or $St_j.v = prev\_anc(v_{2m+1})$, or due to Step 1). Thus, $St_k.v = v_{2m+1}$, $St_k.cur\_st = st_m$ and in s $St_k$, $head(St_k.anc(St_k.v)) = (prev\_anc(v_{2m+1}), v_j)$. Furthermore, it follows from Lemma 8 that a a finite number of steps, Detect_Ins_TSG2 is in a state $St'_k$ such that $St'_k \equiv St_k$ and no further forw transitions can be made from $St'_k$. Thus, in state $St'_k$,

- Since $prev\_anc(v_{2m+1}) \neq v_{2m+2}$ and $v_j \neq v_{2m+2}$, $head(St'_k.anc(St'_k.v))[1] \neq v_{2m+2}$, $head(St'_k.anc(St'_k.v))[2] \neq v_{2m+2}$,
- 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 $v_{2m+2} = v_{2n}$ 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 $st_{m+}$ $st_F(St'_k.cur\_st, \overline{L(St'_k.v, v_{2m+2})})$ 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 $v_{2m+2} \neq$

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

- if $v_{2m+2} = v_{2m+3}$, then either
  1. $St'_k.V\_set(v_{2m+2})$ already contains $(st_{m+1}, (St'_k.v, St'_k.v))$. Thus, since $St'_k.v = v_{2n}$ $v_{2m+1} \neq v_{2m+4}$, $prev\_anc(v_{2m+3}) = v_{2m+1}$, $(st_{m+1}, (prev\_anc(v_{2m+3}), v'_j))$ is added $V\_set(v_{2m+3})$ during the execution of Detect_Ins_TSG2, $v'_j \neq v_{2m+4}$.
  2. $St'_k.V\_set(v_{2m+2})$ already contains $(st_{m+1}, (St'_k.v, u_2))$ and $(st_{m+1}, (St'_k.v, u_3))$, $u_2 \neq$ Thus, since $St'_k.v = v_{2m+1}$, either $u_2 \neq v_{2m+4}$ or $u_3 \neq v_{2m+4}$ (since $u_2 \neq u_3$), $prev\_anc(v_{2m+3}) = v_{2m+1}$, $(st_{m+1}, (prev\_anc(v_{2m+3}), v'_j))$ is added to $V\_set(v_{2m+3})$ du the execution of Detect_Ins_TSG2, $v'_j \neq v_{2m+4}$.
- if $v_{2m+1} = v_{2m+3}$, then either
  1. $St'_k.V\_set(St'_k.v)$ already contains $(st_{m+1}, (prev\_anc(v_{2m+1}), v_{2m+2}))$. Thus, since $St'_k.$ $v_{2m+1}$, $v_{2m+2} \neq v_{2m+4}$, $prev\_anc(v_{2m+3}) = prev\_anc(v_{2m+1})$, $(st_{m+1}, (prev\_anc(v_{2m+3}),$ is added to $V\_set(v_{2m+3})$ during the execution of Detect_Ins_TSG2, $v'_j \neq v_{2m+4}$.
  2. $St'_k.V\_set(St'_k.v)$ already contains $(st_{m+1}, (prev\_anc(v_{2m+1}), u_2))$ and $(st_{m+1}, (prev\_anc(v_{2m+1}), u_3))$, $u_2 \neq u_3$. Thus, since $St'_k.v = v_{2m+1}$, either $u_2 \neq v_{2m+}$ $u_3 \neq v_{2m+4}$ (since $u_2 \neq u_3$), $prev\_anc(v_{2m+3}) = prev\_anc(v_{2m+1})$, $(st_{m+1}, (prev\_anc(v_{2m}$ $v'_j))$ is added to $V\_set(v_{2m+3})$ during the execution of Detect_Ins_TSG2, $v'_j \neq v_{2m+4}$. □

**Corollary 6:** Let Detect_Ins_TSG2($(V, E, L), v_1, v_2, set_1, set_2, RT$) return the set of site no $\Delta_F$. If the TSG $(V, E, L)$ contains a path $(v_1, v_2)(v_3, v_4)\cdots(v_{2n-1}, v_{2n})(v_{2n+1}, v_1)$, $v_2 = v_3$, consis with $set_2 \cup \Delta_F$, such that for the regular term $RT$, $F = FA(RT)$, $st = state_F(init\_st_F, (v_3, v_4)$ $\cdots(v_{2n-1}, v_{2n})(v_{2n+1}, v_1))$ is an accept state and $v_{2n+1} \in set_1$, then during the execution of tect_Ins_TSG2, $v_{2n+1}$ is added to $\Delta$.

**Proof:** By Lemma 9, $(st, (prev\_anc(v_{2n+1}), v_j))$ is added to $V\_set(v_{2n+1})$, where $v_j \neq v_1$. Si $prev\_anc(v_{2n+1}) \neq v_1$ and $v_j \neq v_1$, Detect_Ins_TSG2 makes a forward state transition when $(st, (p$ $(v_{2n+1}), v_j))$ is added to $V\_set(v_{2n+1})$. However, just before $(st, (prev\_anc(v_{2n+1}), v_j))$ is added $V\_set(v_{2n+1})$, since $st$ is an accept state, $prev\_anc(v_{2n+1}) \neq v_1$, $v_j \neq v_1$ and $v_{2n+1} \in set_1$, $v_{2n+}$ added to $\Delta$. □

**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 argum as in Corollary 3.

The number of steps Detect_Ins_TSG2 terminates in is equal to the product of the number of ti Detect_Ins_TSG2 checks if an edge satisfies the conditions in Step 2 and the number of steps requ to check if an edge satisfies the conditions in Step 2. Every time a transaction node is visited, the c ditions in Step 2 need to be checked, on an average, for $v_S$ edges (the average number of sites a glo transaction executes at is $v_S$), while every time a site node is visited, the conditions in Step 2 nee be checked for at most $n_G$ edges (since the number of transaction nodes in the TSG is at most $n$ Furthermore, every transaction node can be visited at most $2v_S n_S$ times, while every site node can 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 i edge satisfies the conditions in Step 2 is $2n_G^2 m n_S + 2n_G v_S^2 n_S$. Since each of the conditions in Step 2 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 ensure detects instantiations of regular terms, we define the following.

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

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

Note that, by the definition of path, it follows that for all $i$, $i = 1, 2, \ldots, n-1$, $v_{2i+2} \neq prev\_anc(v_2$ 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 w 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 $v_j$, $v_j \neq v_{2i+2}$, $(st, (prev\_anc(v_{2i+1}), v_j))$ is added to $V\_set(v_{2i+1})$, where $s$ $state_F(init\_st_F, (v_3, v_4)\cdots(v_{2i-1}, v_{2i})(v_{2i+1}, v_{2i+2}))$.

**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 $(st, (prev\_anc(v_{2i+1}), v_j))$ is added to $V\_set(v_{2i+1})$, where $s$ $state_F(init\_st_F, (v_3, v_4)\cdots(v_{2i-1}, v_{2i})(v_{2i+1}, v_{2i+2}))$.
**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 $i$ $((init\_st_F, (prev\_anc(v_3), v_j))$ is added to $V\_set(v_3)$, $v_j \neq v_4)$.
**Induction:** Let us assume that the lemma is true for $i = m$, $1 \leq m < n-1$. Thus, $(st_m, (prev\_anc(v_{2m+1}), v_j))$ is added to $V\_set(v_{2m+1})$, where $v_j \neq v_{2m+2}$, $st_m = state_F(init\_st_F, (v_3$ $\cdots(v_{2m-1}, v_{2m})(v_{2m+1}, v_{2m+2}))$. We show the lemma to be true for $i = m+1$. Thus, we n to show that $(st_{m+1}, (prev\_anc(v_{2m+3}), v'_j))$ is added to $V\_set(v_{2m+3})$, where $v'_j \neq v_{2m+4}$, $st_{m+}$ $state_F(init\_st_F, (v_3, v_4)\cdots(v_{2m+1}, v_{2m+2})(v_{2m+3}, v_{2m+4}))$. By the definition of $state_F$, $st_{m+1} = st_F($ $L(v_{2m+1}, v_{2m+2}))$, if $v_{2m+2} = v_{2m+3}$ and $st_{m+1} = st_F(st_m, \overline{L(v_{2m+1}, v_{2m+2})})$, if $v_{2m+1} = v_{2m+3}$.

**Corollary 4:** Let Detect_Ins_TSG1($(V,E,L), v_1, v_2, set_1, set_2, RT$). return the set of site no $\Delta_F$. If the TSG $(V,E,L)$ contains a path $(v_1,v_2)(v_3,v_4)\cdots(v_{2n-1},v_{2n})(v_{2n+1},v_1)$, $v_2 = v_3$, consis with $set_2 \cup \Delta_F$, such that for the regular term $RT$, $F = FA(RT)$, $st = state_F(init\_st_F, (v_3,v_4)$ $\cdots(v_{2n-1},v_{2n})(v_{2n+1},v_1))$ is an accept state and $v_{2n+1} \in set_1$, then during the execution of tect_Ins_TSG1, $v_{2n+1}$ is added to $\Delta$.

**Proof:** By Lemma 7, $(st, v_j)$ is added to $V\_set(v_{2n+1})$, where $v_j \neq foll(v_{2n+1})$. Since $foll(v_{2n+1}$ $v_1$, $v_j \neq v_1$ and Detect_Ins_TSG1 makes a forward state transition when $(st, v_j)$ is added to $V\_set(v_{2n}$ However, just before $(st, v_j)$ is added to $V\_set(v_{2n+1})$, since $st$ is an accept state, $v_j \neq v_1$ $v_{2n+1} \in set_1$, $v_{2n+1}$ is added to $\Delta$. $\Box$

We now show that Detect_Ins_TSG2 terminates in $O(n_G^2 m v_S)$ steps, for which we need to pr the following lemma.

**Lemma 8:** If during its execution, Detect_Ins_TSG2 is in state $St_k$, then after a finite numbe 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 $\{(st, v_1, v_2, v$ $(st$ is a state of $F) \wedge (v_1, v_2, v_3 \in V) \wedge ((st,(v_1,v_2)) \notin V\_set(v_3))\}$ 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 $s$ $st_F(St_k.cur\_st, \underline{L(St_k.v,u)})$ is defined, then $(st,(St_k.v, St_k.v)) \in St_k.V\_set(u)$ (alternatively, if $s$ $st_F(St_k.cur\_st, \overline{L(St_k.v,u)})$ is defined, then $(st', (head(St_k.anc(St_k.v))[1], u)) \in St_k.V\_set(St_k.v))$. no forward transition can be made from state $St_k$ (since every edge $(St_k.v, u)$ satisfies the third co tion in Step 2).

**Induction:** Let us assume the lemma is true for $num \leq m$, $m \geq 0$. We show that the lemma is if $num \leq m+1$ in state $St_k$. We show that after a finite number of moves, Detect_Ins_TSG2 is 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_TS 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 m 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 $(St_l.cur\_st, (St''_k.v, St''_k.v)) \notin St''_k.V\_set(u)$ and in state $St_l$, $(St_l.cur\_st, (St''_k.v, St''_k.v)) \in St_l.V\_se$ (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 t since before the transition is made, $(St_l.cur\_st, (St''_k.v, St''_k.v))$ does not belong to $V\_set(u)$ and $nu$ $m+1$ in $St''_k$, after the transition $St''_k \rightarrow St_l$ is made, $num \leq m$ in $St_l$. By IH, after a finite num of steps, Detect_Ins_TSG2 enters a state $St'_l \equiv St_l$, such that no forward transitions are possible f $St'_l$. Thus, Detect_Ins_TSG2 makes the reverse transition $St'_l \rightarrow St'''_k$ after a finite number of st where $St'''_k \equiv St''_k \equiv St_k$. Furthermore, in state $St'''_k$, $(St_l.cur\_st, (St''_k.v, St''_k.v)) \in St'''_k.V\_set(u)$ $St'''_k.v = St''_k.v$, 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 argumen can be shown that if Detect_Ins_TSG2 makes a forward transition $St''_k \rightarrow St_l$ due to edge $(St''_k.v, u)$ $\overline{L(St''_k.v,u)}$, then in a finite number of steps, Detect_Ins_TSG2 enters a state $St'''_k \equiv St''_k$ such that forward transitions are possible from $St'''_k$ due to edge $(St'''_k.v, u)$ and $\overline{L(St'''_k.v,u)}$.

Thus, once a forward transition is made by Detect_Ins_TSG2 due to an edge $e$ and $L(e)/\overline{L(e)}$ f a state equivalent to $St_k$, then no further forward transitions can be made by Detect_Ins_TSG2 to $e$ and $L(e)/\overline{L(e)}$ from any state equivalent to $St_k$. Furthermore, everytime a forward transitio made from a state $St''_k$ that is equivalent to $St_k$ such that $num \leq m+1$ in $St''_k$, a reverse transi is made by Detect_Ins_TSG2 to a state $St'''_k$ equivalent to $St_k$ such that $num \leq m+1$ in $St'''_k$. S there are a finite number of edges incident on each node and in state $St_k$, $num \leq m+1$, eventua 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 $v_j$, $v_j \neq foll(v_{2i+1})$, such that $(st, v_j)$ is added to $V\_set(v_{2i+1})$, where $s$ $state_F(init\_st_F, (v_3, v_4) \cdots (v_{2i-1}, v_{2i})(v_{2i+1}, v_{2i+2}))$.

**Proof:** We prove the above lemma by induction on $i$. We prove that for all $i$, $i = 1, 2, \ldots, n$ there exists a $v_j \neq foll(v_{2i+1})$, such that $(st, v_j)$ is added to $V\_set(v_{2i+1})$, where $st = state_F(init\_$ $(v_3, v_4) \cdots (v_{2i-1}, v_{2i})(v_{2i+1}, v_{2i+2}))$.

**Basis** ($i = 1$): In Step 1 of Detect_Ins_TSG1, $(init\_st_F, v_1)$ is added to $V\_set(v_2)$. Since $v_2 =$ $v_1 \neq foll(v_3)$, and $state_F(init\_st_F, (v_3, v_4)) = init\_st_F$, the lemma is true for $i = 1$ ($(init\_st_F, v_j$ added to $V\_set(v_3)$, $v_j \neq foll(v_3)$).

**Induction:** Let us assume that the lemma is true for $i = m$, $1 \leq m < n - 1$. Thus, $(st_m, v_j)$ is ad to $V\_set(v_{2m+1})$, where $v_j \neq foll(v_{2m+1})$, $st_m = state_F(init\_st_F, (v_3, v_4) \cdots (v_{2m+1}, v_{2m+2}))$. We s the lemma to be true for $i = m + 1$. Thus, we need to show that $(st_{m+1}, v'_j)$ is added to $V\_set(v_{2m}$ where $v'_j \neq foll(v_{2m+3})$, $st_{m+1} = state_F(init\_st_F, (v_3, v_4) \cdots (v_{2m+3}, v_{2m+4}))$. By the definitio $state_F$, $st_{m+1} = st_F(st_m, L(v_{2m+1}, v_{2m+2}))$, if $v_{2m+2} = v_{2m+3}$ and $st_{m+1} = st_F(st_m, \overline{L(v_{2m+1}, v_{2m+}}$ 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 $V\_set(v_{2m+1})$ 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 s $St'_k$ such that $St'_k \equiv St_k$ and no further forward transitions can be made from $St'_k$. Thus, in state

- 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 $v_{2m+2} = v_{2m+3}$, t $st_{m+1} = st_F(St'_k.cur\_st, L(St'_k.v, v_{2m+2}))$ is defined, else if $v_{2m+1} = v_{2m+3}$, then $st_{m+1}$ $st_F(St'_k.cur\_st, \overline{L(St'_k.v, v_{2m+2})})$ 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 $v_{2m+2} \neq$

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

- if $v_{2m+2} = v_{2m+3}$, then $foll(v_{2m+1}) = v_{2m+2}$ and since $v_j \neq foll(v_{2m+1})$ (by the definitio 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 $St'_k.v = v_{2m+1}$, $v_{2m+}$ $foll(v_{2m+3})$, $(st_{m+1}, v'_j)$ is added to $V\_set(v_{2m+3})$ during the execution of Detect_Ins_TS $v'_j \neq foll(v_{2m+3})$.
  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 ei $u_2 \neq foll(v_{2m+3})$ or $u_3 \neq foll(v_{2m+3})$ (since $u_2 \neq u_3$), $(st_{m+1}, v'_j)$ is added to $V\_set(v_{2n}$ during the execution of Detect_Ins_TSG1, $v'_j \neq foll(v_{2m+3})$.

- 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 $St'_k.v = v_{2m+1}$, $foll(v_{2m+1}$ $foll(v_{2m+3})$, $v_j \neq foll(v_{2m+1})$, $(st_{m+1}, v'_j)$ is added to $V\_set(v_{2m+3})$ during the executio Detect_Ins_TSG1, $v'_j \neq foll(v_{2m+3})$.
  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 ei $u_2 \neq foll(v_{2m+3})$ or $u_3 \neq foll(v_{2m+3})$ (since $u_2 \neq u_3$), $(st_{m+1}, v'_j)$ is added to $V\_set(v_{2n}$ during the execution of Detect_Ins_TSG1, $v'_j \neq foll(v_{2m+3})$. □

## Appendix -C- : TSG Schemes

In this appendix, we prove Theorem 3. We begin by showing that Detect_Ins_TSG1 and tect_Ins_TSG2 detect instantiations of regular terms in $S$. States $St_k$ between the execution of any 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 $St_k$, then after a finite numbe 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 3:** Procedure Detect_Ins_TSG1 terminates in $O(n_G m n_S)$ steps.

**Proof:** We first show that Detect_Ins_TSG1 terminates in a finite number of steps. Let $St_1$ de the state immediately after the execution of Step 1 of algorithm Detect_Ins_TSG1. By Lemm after a finite number of steps, Detect_Ins_TSG1 is in a state $St'_1 \equiv St_1$ such that no further forw transitions can be made from $St'_1$. Detect_Ins_TSG1, thus executes Step 4 and since, in state $head(St'_1.F\_list(St'_1.v)) = (s*, G_i)$, 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 ti Detect_Ins_TSG1 checks if an edge satisfies the conditions in Step 2 and the number of steps requ 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 an average, for $v_S$ edges (the average number of sit global transaction executes at is $v_S$), while every time a site node is visited, the conditions in Ste need to be checked for at most $n_G$ edges (since the number of transaction nodes in the TSG is at m $n_G$). Furthermore, every transaction and site node can be visited at most $2n_S$ times (every node the TSG can be visited in a state $st$ of $F$ at most twice, and $F$ has at most $n_S$ states). Since there ar site nodes and at most $n_G$ transaction nodes in the TSG, the number of times Detect_Ins_TSG1 ch if an edge satisfies the conditions in Step 2 is $2n_G m n_S + 2n_G v_S n_S$. Since each of the condition Step 2 can be checked in constant time and $v_S < m$, Detect_Ins_TSG1 terminates in $O(n_G m n_S)$ steps

In order to show that Detect_Ins_TSG1 traverses edges in the TSG in a manner that ensure 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 $foll(v_{2i-1})$ as follows.

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

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

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

**Lemma 7:** Let Detect_Ins_TSG1$((V, E, L), v_1, v_2, set_1, set_2, RT)$ return the set of site nodes If the TSG $(V, E, L)$ contains a path $(v_1, v_2), (v_3, v_4), \ldots, (v_{2n-3}, v_{2n-2}), (v_{2n-1}, v_{2n})$, $v_2 = v_3$, consis with $set_2 \cup \Delta_F$ 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_TSG1, for all $i$, $i = 1, 2, 3, \ldots,$

- $(sfirst(G_j), G_j)(G_j, slast(G_j))$, $j = 1, 2, \ldots, n-1$, where $arity(t_j) = 2$ ($G_j$ is the common no

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

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

Also, for the sequence of edges $(sfirst(t_j), G_j)(G_j, slast(t_j))$ in the path, $j = 1, 2, \ldots, n-1$, it m be the case that $arity(t_j) = 2$, and thus $sfirst(t_j) \neq slast(t_j)$. Also, if for some $j, k$, $j = 0, 1, \ldots, n$ $j < k \leq n$, the sequence of edges $(G_j, slast(t_j))(sfirst(t_{(j+1)\bmod n}), G_{(j+1)\bmod n}), \ldots,$ $(sfirst(t_{k\bmod n}), G_{k\bmod n})$ is in the path, then it must be the case that for all $j < l < k$, $arity(t_l)$ = Thus, by Property 1, it follows that $slast(t_j) = sfirst(t_{(j+1)\bmod n}) = \cdots = sfirst(t_{k\bmod n})$, and for $r, s$, $j \leq r < s \leq k$,

- $G_r \neq G_{s\bmod n}$, and

- $G_r$ is serialized after $G_{s\bmod n}$ at site $sfirst(G_{s\bmod n})$. Thus, by Lemma 5, dependency $(G_r, sfirst(G_{s\bmod n}))\rightarrow(sfirst(G_{s\bmod n}), G_{s\bmod n})$ 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 $F = FA(RT_2)$, $state_F(init\_st_F, edge(t_1)\cdots edge(t_{n-}$ $(sfirst(t_0), G_0))$ is an accept state. Let $edge(t_1)\cdots edge(t_{n-1})(sfirst(t_0), G_0) = (v_1, v_2)\cdots(v_{2m-1},$ 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 $g_i = \overline{L(v_{2i-1}, v_{2i})}$, and

$st_F(init\_st_F, g_1\cdots g_{m-1})$ is an accept state. We construct the sequence $g_1\cdots g_{m-1}$ with the ab properties as follows. For all $i = 1, \ldots, n-1$, let $f_i = \overline{(type(hdr(t_i)), type(first(t_i)))}$, if $arity(t_i)$ = else, $f_i = (type(hdr(t_i)), type(first(t_i)))(type(hdr(t_i)), type(last(t_i)))$. Since $type(t_1)\cdots type(t_{n-1}$ a string in $L(reg\_exp)$, by the construction of $FA(RT_2)$, it follows that $st_F(init\_st_F, f_1\cdots f_{n-1})$ is 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 defini 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 $arity(t_k)$ = then $g_i = L(v_{2i-1}, v_{2i})$, else $g_i = \overline{L(v_{2i-1}, v_{2i})}$.

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 sh 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 $v_{2i-1} = v_{2i+1}$, t $g_i = \overline{L(v_{2i-1}, v_{2i})}$. We first show that if $v_{2i} = v_{2i+1}$, and $(v_{2i-1}, v_{2i}) \in edge(t_k)$ for some $k$, $1, 2, \ldots, n-1$, then $arity(t_k) = 2$. Suppose $arity(t_k) = 1$. Since $last(t_k)$ and $first(t_{(k+1)\bmod}$ execute at the same site, $slast(t_k) = v_{2i-1}$, $sfirst(t_{(k+1)\bmod n}) = v_{2i+1}$, it follows that $v_{2i-1} = v_2$ which leads to a contradiction. Thus, $arity(t_k) = 2$, and $g_i = L(v_{2i-1}, v_{2i})$. Also, it can be sho that if $v_{2i-1} = v_{2i+1}$, and $(v_{2i-1}, v_{2i}) \in edge(t_k)$, then $arity(t_k) = 1$. Suppose $arity(t_k) = 2$. $v_{2i} = G_k$, then $v_{2i} = v_{2i+1} = G_k$, which leads to a contradiction. If $v_{2i-1} = G_k$, then since $last(t_k)$ $first(t_{(k+1)\bmod n})$ execute at the same site, $slast(t_k) = v_{2i}$, $sfirst(t_{(k+1)\bmod n}) = v_{2i+1}$, it follows t $v_{2i} = v_{2i+1}$, which leads to a contradiction. Thus, $arity(t_k) = 1$, and, $g_i = \overline{L(v_{2i-1}, v_{2i})}$.

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. T 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 abor by the optimistic scheme. However, this leads to a contradiction since $G_0$ is a transaction in $I$

dependency is added during the execution of $act(ser_k(G_i))$, then $act(ser_k(G_j)$ must have already e cuted. On the other hand, if the dependency were added to the TSGD before $act(ser_k(G_i))$ execu then $act(ser_k(G_i))$ would not execute until $act(ack(ser_k(G_j)))$ completes execution (the depende $(G_j, s_k)\rightarrow(s_k, G_i)$ is deleted from the TSGD only after $ack(ser_k(G_j))$ is processed). Thus, in both ca $ser_k(G_j)$ executes before $ser_k(G_i)$, and thus, $G_{jk}$ is serialized before $G_{ik}$ at site $s_k$, which leads t contradiction. □

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

**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 reg term $RT_2 = e_0 : reg\_exp$ and an instantiation $t_0 : t_1t_2\cdots t_{n-1}$ of $RT_2$ in $S$ such that $hdr(t_0) =$ 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 $first(t_{(j+1)\bmod n})$ execute at the same site, and $last(t_j)$ is serialized a $first(t_{(j+1)\bmod n})$ at the site, and
- $type(t_0) = e_0$ and $type(t_1)\cdots type(t_{n-1})$ is a string in $L(reg\_exp)$.

When $val_0$ is processed, Detect_Ins_Opt is invoked with arguments that include the TSGD $(V', E$ $L')$, $G_0$, $slast(t_0)$, $set_1$, and $RT_2$ since $type(G_0) = hdr(e_0)$ and $type(last(t_0)) = last(e_0)$. A $sfirst(t_0) \in set_1$ (if $arity(t_0) = 1$, then since $sfirst(t_0) = slast(t_0)$, $sfirst(t_0) \in set_1$; if $arity(t_0) =$ then since $sfirst(t_0) \neq slast(t_0)$, and $type(first(t_0)) = first(e_0)$, $sfirst(t_0) \in set_1$). Furthermore the edges belonging to $G_0, \ldots, G_{n-1}$ are in the TSGD when Detect_Ins_Opt is invoked. In orde show this, we first show that $G_j$'s edges cannot be deleted from the TSGD before $G_{(j+1)\bmod n}$'s ed are deleted from the TSGD, for all $j$, $j = 1, 2, \ldots, n-1$. Suppose, for some $j$, $j = 1, 2, \ldots, n$ $G_j$'s edges are deleted from the TSGD before $G_{(j+1)\bmod n}$'s edges are deleted from the TSGD. $slast(t_j) = s_k$. Since $G_{jk}$ is serialized after $G_{((j+1)\bmod n)k}$, at site $s_k$, $ser_k(G_{(j+1)\bmod n})$ executes be $ser_k(G_j)$. Thus, since $G_{(j+1)\bmod n}$'s edges are inserted into the TSGD before $ser_k(G_{(j+1)\bmod n})$ execu while $G_j$'s edges are deleted after $ser_k(G_j)$ executes, $G_{(j+1)\bmod n}$'s edges must be in the TSGD w $G_j$'s edges are deleted (since we have assumed that $G_j$'s edges are deleted before $G_{(j+1)\bmod n}$'s ed are deleted). Furthermore, since $ser_k(G_j)$ and $ser_k(G_{(j+1)\bmod n})$ must have both executed when ( edges are deleted, $G_{(j+1)\bmod n}$ is serialized before $G_j$ when $G_j$'s edges are deleted. However, this le to a contradiction, since edges belonging to $G_j$ and $G_{(j+1)\bmod n}$ are deleted together when $act(f$ for some transaction $G_l$ executes (since $G_{(j+1)\bmod n}$ is serialized before $G_j$, if for every transac $G_k \in V$ serialized before $G_j$, $val_k$ has been processed, then for every transaction $G_k \in V$ seriali before $G_{(j+1)\bmod n}$ also, $val_k$ must have been processed). Thus, $G_1$'s edges are not deleted from TSGD before $G_2$'s edges are deleted, …, $G_{n-1}$'s edges are not deleted from the TSGD before ( edges are deleted. By transitivity and since $G_0$'s edges are deleted only after $val_0$ has been proces 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 proces before $val_0$ is processed).

We now show that $(G_0, slast(t_0))edge(t_1)\cdots edge(t_{n-1})(sfirst(t_0), G_0)$ is a path in the TSGD.

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 $st_{m+1}$ $st_F(st_m, \overline{L(v_{2m+1}, v_{2m+2})})$, 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 $V\_set(v_{2m}$ (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 due to Step 1). Thus, $St_k.v = v_{2m+1}$, $St_k.cur\_st = st_m$ and in state $St_k$, $head(St_k.$ $(St_k.v))[2] = prev(v_{2m+1})$. Furthermore, since Detect_Ins_Opt does not return abort, it follows f Lemma 2 that after a finite number of steps, Detect_Ins_Opt is in a state $St'_k$ such that $St'_k \equiv St_k$ 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 $(prev(v_{2m+1}), v_{2m}$ $(v_{2m+1}, v_{2m+2})$ in $D$; thus, there is no dependency $(head(St'_k.anc(St'_k.v)), St'_k.v) \rightarrow (St'_k.v, v_{2m}$ 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 $v_{2m+2} = v_{2n}$ 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 $st_{m+}$ $st_F(St'_k.cur\_st, \overline{L(St'_k.v, v_{2m+2})})$ 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 $St'_k.V\_set(v_{2m+2})$ 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 execu 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 exe tion of Detect_Ins_Opt. □

**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 $RT$, $F = FA(RT)$, $st = state_F(init\_st_F, (v_3, v_4)\cdots(v_{2n-1}, v_{2n})(v_{2n+1},$ 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 ab

**Proof:** Suppose Detect_Ins_Opt does not return abort. By Lemma 4, $(st, prev(v_{2n+1}))$ is ad to $V\_set(v_{2n+1})$. Since $prev(v_{2n+1}) \neq v_1$, Detect_Ins_Opt makes a forward state transition w $(st, prev(v_{2n+1}))$ is added to $V\_set(v_{2n+1})$. However, just before $(st, prev(v_{2n+1}))$ is added to $V\_set(v_{2n}$ 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. □

We are now in a position to prove Theorem 1. Before we present the proof, we introduce so 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 serial 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 co not have been added to the TSGD after $act(ser_k(G_i))$ has executed. Thus, dependency $(G_j, s_k$

The above definition of $state_F$ is recursive. In the following lemma, we show that an alterna 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 $e_1 e_2 \cdots, e_{n-}$ 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 $i = 1, \ldots, n$, $state_F(st, (v_1, v_2) \cdots (v_{2i-1}, v_{2i})$ $st_F(st, e_1 \cdots e_{i-1})$.
**Basis** ($i = 1$): $state_F(st, (v_1, v_2)) = st_F(st, \epsilon) = st$.
**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 $state_F(st, (v_1$ $\cdots (v_{2m+1}, v_{2m+2})) = st_F(st, e_1 \cdots e_m)$. 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 TSG/TSGD which $state_F$ ($F = FA(RT)$) with respect to the initial state $init\_st_F$ is an accept state. The follow lemma lays the groundwork for showing that Detect_Ins_Opt detects instantiation by detecting ap priate paths in the TSGD.

**Lemma 4:** Consider a TSGD $(V, E, D, L)$ containing a path $(v_1, v_2) \cdots (v_{2n-3}, v_{2n-2}), (v_{2n-1}, v$ $v_2 = v_3$, 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 fined. If Detect_Ins_Opt$((V, E, D, L), v_1, v_2, set_1, RT)$ does not return abort, then during the execu of Detect_Ins_Opt (before it returns commit), for all $i$, $i = 1, 2, 3, \ldots, n-1$, $(st, prev(v_{2i+1}))$ is ad 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 d not return abort, then for all $i$, $i = 1, 2, \ldots, n-1$, $(st, prev(v_{2i+1}))$ is added to $V\_set(v_{2i+1})$, wh $st = state_F(init\_st_F, (v_3, v_4) \cdots (v_{2i-1}, v_{2i})(v_{2i+1}, v_{2i+2}))$.
**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 $i = 1$ (($init\_st_F, prev($ is added to $V\_set(v_3)$).
**Induction:** Let us assume that the lemma is true for $i = m$, $1 \leq m < n-1$. Thus, if tect_Ins_Opt does not return abort, then $(st_m, prev(v_{2m+1}))$ is added to $V\_set(v_{2m+1})$, where $st_r$ $state_F(init\_st_F, (v_3, v_4) \cdots (v_{2m-1}, v_{2m})(v_{2m+1}, v_{2m+2}))$. We show the lemma to be true for $i =$ 1. Thus, we need to show that if Detect_Ins_Opt does not return abort, then

node can be visited at most $n_G n_S$ times (every node $v$ in the TSGD can be visited in a state $st$ o at most once for every node $w$ such that edge $(v, w)$ is in the TSGD, and $F$ has at most $n_S$ stat Since there are $m$ site nodes and at most $n_G$ transaction nodes in the TSGD, the number of ti Detect_Ins_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 e of the conditions in Step 2 can be checked in constant time and $v_S \ll n_G, v_S < m$, Detect_Ins_ terminates in $O(n_G^2 m n_S)$ steps. □

Before we show that Detect_Ins_Opt detects instantiations, we define the notion of a path in or to capture the notion of instantiations in the TSGD. Corresponding to every instantiation, there path, defined below, in the TSGD (paths are similarly defined for a TSG; the requirement tht there no dependencies between certain edges is trivially satisfied in a TSG).

**Definition 9:** Consider a TSG/TSGD containing the sequence of edges $(v_1, v_2)(v_3, v_4)\cdots(v_{2n-1}$ $n > 1$. 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 $v_{2i} = v_2$ or $v_{2i-1} = v_{2i+1}$, and
- if for some $j, k = 1, 2, \ldots, n$, $j \le k$, $v_{2j-1} = v_{2j+1} = v_{2j+3} = \cdots = v_{2k-1}$, then
  1. if $j < k$, then $v_{2j} \neq v_{2j+2} \neq v_{2j+4} \neq \cdots \neq v_{2k}$, and for all $l, m$, $j \le l < m \le k$, there is 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 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 $(v_{2i-1}, v_{2i})(v_2$ $v_{2i+2})$, $i = 1, \ldots, n-1$, 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 $prev(v_2$ as follows.

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

Note that, by the definition of path, $prev(v_{2i+1}) \neq v_{2i+2}$ and there is no dependency $(prev(v_{2i}$ $v_{2i+1}) \rightarrow (v_{2i+1}, v_{2i+2})$ in the TSGD. Only certain paths in the TSG/TSGD in which the sequenc transaction types are a string in $L(reg\_exp)$ correspond to instantiations of $RT = e_0 : reg\_exp$ in $S$ order to ensure that transaction type information can be taken into account when detecting path the TSG/TSGD, we define $state_F$ below.

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

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

where $st' = state_F(st, (v_1, v_2)\cdots(v_{2i-3}, v_{2i-2}))$. □

**Basis** ($num = 0$): If $num = 0$ in state $St_k$, then in state $St_k$, for every edge $(St_k.v, u)$, if $s$ $st_F(St_k.cur\_\overline{st, L(St_k.v}, u))$ is defined, then $(st, St_k.v) \in St_k.V\_set(u)$ (alternatively, if $st' =$ $(St_k.cur\_st, \overline{L(St_k.v, u)})$ is defined, then $(st', u) \in St_k.V\_set(St_k.v)$). Thus, no forward transi 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 lemma is true if $num \leq m+1$ in state $St_k$. We show that if Detect_Ins_Opt does not return abort, t after a finite number of moves, Detect_Ins_Opt is in a state $St'_k$ such that $St'_k \equiv St_k$ and no forw 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_ 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 m 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 $(St_l.cur\_st, St''_k.v) \notin St''_k.V\_set(u)$ and in state $St_l$, $(St_l.cur\_st, St''_k.v) \in St_l.V\_set(u)$ (since the tra tion $St''_k \rightarrow St_l$ causes $(St_l.cur\_st, St''_k.v)$ to be added to $V\_set(u)$). Note that, since before the transi is made, $(St_l.cur\_st, St''_k.v)$ does not belong to $V\_set(u)$ and $num \leq m + 1$ in $St''_k$, after the tra tion $St''_k \rightarrow St_l$ is made, $num \leq m$ in $St_l$. By IH, since Detect_Ins_Opt does not return abort, a a finite number of steps, Detect_Ins_Opt enters a state $St'_l \equiv St_l$, such that no forward transit are possible from $St'_l$. Thus, since it does not return abort, Detect_Ins_Opt makes the reverse tr sition $St'_l \rightarrow St'''_k$ after a finite number of steps, where $St'''_k \equiv St''_k \equiv St_k$. Furthermore, in state $(St_l.cur\_st, St''_k.v) \in St'''_k.V\_set(u)$ and $St'''_k.v = St''_k.v$, and thus, no forward transition can be m 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 condi in Step 3(c)). Using a similar argument, it can be shown that if Detect_Ins_Opt makes a forward tr sition $St''_k \rightarrow St_l$ due to edge $(St''_k.v, u)$ and $\overline{L(St''_k.v, u)}$, then in a finite number of steps, Detect_Ins_ enters a state $St'''_k \equiv St''_k$ such that no forward transitions are possible from $St'''_k$ due to edge $(St'''_k.v$ and $\overline{L(St'''_k.v, u)}$.

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

**Corollary 1:** Procedure Detect_Ins_Opt terminates in $O(n_G^2 m n_S)$ steps.

**Proof:** We first show that Detect_Ins_Opt terminates in a finite number of steps. Let $St_1$ de the state immediately after the execution of Step 1 of algorithm Detect_Ins_Opt. If Detect_Ins_ does not return abort, then by Lemma 1, after a finite number of steps, Detect_Ins_Opt is in a s $St'_1 \equiv St_1$ such that no further forward transitions can be made from $St'_1$. Detect_Ins_Opt, t executes Step 4 and since, in state $St'_1$, $head(St'_1.F\_list(St'_1.v)) = (s*, G_i)$, Detect_Ins_Opt termin in a finite number of steps. If, on the other hand, Detect_Ins_Opt returns abort, then it trivi 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 ti Detect_Ins_Opt checks if an edge satisfies the conditions in Step 2 and the number of steps requ 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 an average, for $v_S$ edges (the average number of sit global transaction executes at is $v_S$), while every time a site node is visited, the conditions in Ste need to be checked for at most $n_G$ edges (since the number of transaction nodes in the TSGD i

## Appendix -B- : Optimistic Scheme

Before we prove Theorem 1, we need to prove certain lemmas. In the following lemma, we s
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
in the global schedule $S$, and $G_0$ be a transaction in $I$. If $R$ is complete, then there exists a reg
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 $RT_1 = e'_0 : reg\_exp_1$ and $I = t'_0 : t'_1 \cdots t'_{n-1}$, $n > 1$. Since $I$ is an instantiation of
in $S$,

- for all $j$, $j = 0, 1, \ldots, n-1$,
  1. $t'_j \in \Sigma_S$, and
  2. $last(t'_j)$ and $first(t'_{(j+1) \bmod n})$ execute at the same site, and $last(t'_j)$ is serialized a $first(t'_{(j+1) \bmod n})$ at the site, and
- $type(t'_0) = e'_0$ and $type(t'_1) \cdots type(t'_{n-1})$ is a string in $L(reg\_exp_1)$.

Let $G_0 = hdr(t'_k)$, for some $k$, $k = 0, 1, \ldots, n-1$. Since $R$ is complete, there exists a regular t
$RT_2 = type(t'_k) : reg\_exp_2$ such that

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

is a string in $\in L(reg\_exp_2)$. Thus,

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

is the required instantiation of $RT_2$ in $S$. $\Box$

We next show that the manner in which Detect_Ins_Opt traverses edges in the TSGD ensures t
it detects instantiations of regular terms in the TSGD. We first introduce the following additi
notation.

Between the execution of any two steps[3] of Detect_Ins_Opt, the contents of $v$, $cur\_st$, $\Delta$, $anc($
$V\_set(v_i)$ and $F\_List(v_i)$ for all $v_i \in V$ constitute a state $St_k$ of Detect_Ins_Opt. We denote the cont
of $v$, $cur\_st$, $\Delta$, $anc(v_i)$, $V\_set(v_i)$, and $F\_List(v_i)$ for any $v_i \in V$ in state $St_k$ by $St_k.v$, $St_k.cur$
$St_k.\Delta$, $St_k.anc(v_i)$, $St_k.V\_set(v_i)$ and $St_k.F\_List(v_i)$ respectively. State changes in Detect_Ins_Opt
caused by steps 1, 3 and 4. We refer to state transition $St_j \rightarrow St_k$ due to Step 3 as a *forward transit*
while a state transition $St_j \rightarrow St_k$ due to Step 4 is referred to as a *reverse transition*. Also, two st
$St_j$ and $St'_j$ are said to be equivalent (denoted by $St_j \equiv St'_j$) if $St_j.v = St'_j.v$, $St_j.cur\_st = St'_j.cur$
and for all $v_i \in V$, $St_j.anc(v_i) = St'_j.anc(v_i)$, $St_j.F\_List(v_i) = St'_j.F\_List(v_i)$. Detect_Ins_Opt has
following interesting property: if it makes a forward transition $St_j \rightarrow St_k$ and for a state $St'_k \equiv$
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_
is in state $St_k$, then after a finite number of steps, it enters a state $St'_k \equiv St_k$ such that no forw
transitions from $St'_k$ are possible.

**Proof:** We prove the lemma by induction on *num*, the number of elements in $\{(st, v_1, v$
$(st$ is a state of $F) \wedge (v_1, v_2 \in V) \wedge ((st, v_1) \notin V\_set(v_2))\}$ 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 $F\_list(v) = []$, $anc(v) = []$, $V\_set(v) = \emptyset$. $v = s_k$, $F\_list(s_k) = [(st*, G_i)]$, $anc(s_k) = [(G_i, G_i)]$, $F = FA(RT)$, $V\_set(s_{$ $\{(init\_st_F, (G_i, G_i))\}$ and $cur\_st = init\_st_F$. Set $\Delta = \emptyset$.

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 $(head(anc(v))[1], v) \rightarrow (v, u)$ in $D \cup \Delta$.
   - There is a dependency $(head(anc(v))[2], v) \rightarrow (v, u)$ in $D \cup \Delta$.
   - If $st = st_F(cur\_st, L(v, u))$ is defined then $(st, (v, v)) \in V\_set(u)$, and if $st' = st_F(cur\_st, \overline{L(v, u)})$ is defined then $(st', (head(anc(v))[1], u)) \in V\_set(v)$.

   then go to Step 4.

3. Choose an edge $(v, u)$ such that
   (a) $head(anc(v))[1] \neq u$, and
   (b) $head(anc(v))[2] \neq u$, and
   (c) there is no dependency $(head(anc(v))[1], v) \rightarrow (v, u)$ in $D \cup \Delta$, and
   (d) there is no dependency $(head(anc(v))[2], v) \rightarrow (v, u)$ in $D \cup \Delta$, and
   (e) $st = st_F(cur\_st, L(v, u))$ is defined and $(st, (v, v)) \notin V\_set(u)$, or $st' = st_F(cur\_st, \overline{L(v, u)})$ is defined and $(st', (head(anc(v))[1], u)) \notin V\_set(v)$.

   If $st$ is defined and $(st, (v, v)) \notin V\_set(u)$, then do
   - If $st$ is an accept state, $u \in set_1$ and $v \neq G_i$, then $\Delta := \Delta \cup \{(v, u) \rightarrow (u, G_i)\}$.
   - $F\_list(u) := (cur\_st, v) \circ F\_list(u)$, $anc(u) := (v, v) \circ anc(u)$, $cur\_st := st$, $V\_set($ $V\_set(u) \cup \{(st, (v, v))\}$, $v := u$. Go to Step 2.

   If $st'$ is defined and $(st', (head(anc(v))[1], u)) \notin V\_set(v)$ then do
   - If $st'$ is an accept state, $v \in set_1$, $u \neq G_i$ and $head(anc(v))[1] \neq G_i$, then $\Delta \cup \{(head(anc(v))[1], v) \rightarrow (v, G_i)\}$.
   - $F\_list(v) := (cur\_st, v) \circ F\_list(v)$, $anc(v) := (head(anc(v))[1], u) \circ anc(v)$, $cur\_st :$ $V\_set(v) = V\_set(v) \cup \{(st', (head(anc(v))[1], u))\}$. Go to Step 2.

4. If $head(F\_list(v)) \neq (st*, G_i)$, then $temp1 := head(F\_list(v))[1]$, $temp2$ $head(F\_list(v))[2]$, $F\_list(v) := tail(F\_list(v))$, $anc(v) = tail(anc(v))$, $cur\_st := te$ $v := temp2$ and go to Step 2.

5. **return**($\Delta$).

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 $F\_list(v) = []$, $anc(v) = []$, $V\_set(v) = \emptyset$. 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 $\Delta = \emptyset$.

2. If, for every edge $(v, u)$ one of the following is true:

   - $head(anc(v)) = u$.
   - There is a dependency $(head(anc(v)) \rightarrow (v, u))$ in $D \cup \Delta$.
   - If $st = st_F(cur\_st, \underline{L(v, u)})$ is defined then $(st, v) \in V\_set(u)$, and if $st' = st_F(cur\_st, \overline{L(v, u)})$ is defined then $(st', head(anc(v))) \in V\_set(v)$.

   then go to Step 4.

3. Choose an edge $(v, u)$ such that

   (a) $head(anc(v)) \neq u$, and

   (b) there is no dependency $(head(anc(v)) \rightarrow (v, u))$ in $D \cup \Delta$, and

   (c) $st = st_F(cur\_st, \underline{L(v, u)})$ is defined and $(st, v) \notin V\_set(u)$, or $st' = st_F(cur\_st, \overline{L(v, u)})$ is defined and $(st', head(anc(v))) \notin V\_set(v)$.

   If $st$ is defined and $(st, v) \notin V\_set(u)$, then do

   - if $st$ is an accept state, $u \in set_1$ and $v \neq G_i$, then $\Delta := \Delta \cup \{(v, u) \rightarrow (u, G_i)\}$.
   - $F\_list(u) := (cur\_st, v) \circ F\_list(u)$, $anc(u) := v \circ anc(u)$, $cur\_st := st$, $V\_set($ $V\_set(u) \cup \{(st, v)\}$, $v := u$. Go to Step 2.

   If $st'$ is defined and $(st', head(anc(v))) \notin V\_set(v)$, then do

   - if $st'$ is an accept state, $v \in set_1$ and $head(anc(v)) \neq G_i$, then $\Delta :=$ $\{(head(anc(v)), v) \rightarrow (v, G_i)\}$.
   - $F\_list(v) := (cur\_st, v) \circ F\_list(v)$, $anc(v) := head(anc(v)) \circ anc(v)$, $cur\_st :=$ $V\_set(v) = V\_set(v) \cup \{(st', head(anc(v)))\}$. Go to Step 2.

4. If $head(F\_list(v)) \neq (st*, G_i)$, then $temp1 := head(F\_list(v))[1]$, $temp2$ $head(F\_list(v))[2]$, $F\_list(v) := tail(F\_list(v))$, $anc(v) = tail(anc(v))$, $cur\_st :=$ te $v := temp2$ and goto Step 2.

5. **return**($\Delta$).

Figure 13: Procedure Detect_Ins_TSGD1

**procedure** Detect_Ins_TSG2($(V,E,L), G_i, s_k, set_1 set_2, RT$):

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

2. If, for every edge $(v,u)$ one of the following is true:

   - $head(anc(v))[1] = u$ or $head(anc(v))[2] = u$.
   - If $st = st_F(cur\_st, L(v,u))$ is defined then

     (a) there exist nodes $u_2, u_3$, $u_2 \neq u_3$, such that $(st, (v, u_2)) \in V\_set(u)$, $(st, (v, u_3)) \in V\_s$ or

     (b) $(st, (v, v)) \in V\_set(u)$,

     and if $st' = st_F(cur\_st, \overline{L(v,u)})$ is defined then

     (a) there exist nodes $u_2, u_3$, $u_2 \neq u_3$, such that $(st', (head(anc(v))[1], u_2)) \in V\_s$ $(st', (head(anc(v))[1], u_3)) \in V\_set(v)$, or

     (b) $(st', (head(anc(v))[1], u)) \in V\_set(v)$.
   - $v \in (set_2 \cup \Delta)$ and $u = G_i$.

   then go to Step 4.

3. Choose an edge $(v,u)$ such that

   - $head(anc(v))[1] \neq u$ and $head(anc(v))[2] \neq u$, and
   - $st = st_F(cur\_st, L(v,u))$ is defined and

     (a) there do not exist nodes $u_2, u_3$, $u_2 \neq u_3$, such that $(st, (v, u_2)) \in V\_set(u)$, $(st, (v, u$ $V\_set(u)$, and

     (b) $(st, (v, v)) \notin V\_set(u)$,

     or $st' = st_F(cur\_st, \overline{L(v,u)})$ is defined and

     (a′) there do not exist nodes $u_2, u_3$, $u_2 \neq u_3$, such that $(st', (head(anc(v))[1], u_2)) \in V\_s$ $(st', (head(anc(v))[1], u_3)) \in V\_set(v)$, and

     (b′) $(st', (head(anc(v))[1], u)) \notin V\_set(v)$, and
   - $v \notin (set_2 \cup \Delta)$ or $u \neq G_i$.

   If $st$ is defined, 3(a) and 3(b) then do

   - If $st$ is an accept state, $u \in set_1$ and $v \neq G_i$, then $\Delta := \Delta \cup \{u\}$.
   - $F\_list(u) := (cur\_st, v) \circ F\_list(u)$, $anc(u) := (v, v) \circ anc(u)$, $cur\_st := st$, $V\_set(u) = V\_se$ $\{(st, (v, v))\}$, $v := u$. Go to Step 2.

   If $st'$ is defined, 3(a′) and 3(b′) then do

   - If $st'$ is an accept state, $v \in set_1$, $u \neq G_i$ and $head(anc(v))[1] \neq G_i$, then $\Delta := \Delta \cup \{v\}$.
   - $F\_list(v) := (cur\_st, v) \circ F\_list(v)$, $anc(v) := (head(anc(v))[1], u) \circ anc(v)$, $cur\_st :=$ $V\_set(v) = V\_set(v) \cup \{(st', (head(anc(v))[1], u))\}$. Go to Step 2.

4. If $head(F\_list(v)) \neq (st*, G_i)$, then $temp1 := head(F\_list(v))[1]$, $temp2 := head(F\_list($ $F\_list(v) := tail(F\_list(v))$, $anc(v) = tail(anc(v))$, $cur\_st := temp1$, $v := temp2$ and go to Step

5. **return**($\Delta$).

Figure 12: Procedure Detect_Ins_TSG2

**procedure** Detect_Ins_TSG1($(V, E, L), G_i, s_k, set_1, set_2, RT$):

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

2. If, for every edge $(v, u)$ one of the following is true:

   - If $st = st_F(cur\_st, L(v, u))$ is defined then either
     (a) $head(anc(v)) = u$ or
     (b) $(st, v) \in V\_set(u)$ or
     (c) there exist two distinct nodes $v_1, v_2$ such that $(st, v_1) \in V\_set(u)$ and $(st, v_2) \in V\_set$

     and if $st' = st_F(cur\_st, \overline{L(v, u)})$ is defined then either
     (a) $(st', head(anc(v))) \in V\_set(v)$, or
     (b) there exist two distinct nodes $v_1, v_2$ such that $(st', v_1) \in V\_set(v)$ and $(st', v_2) \in V\_se$
   - $v \in (set_2 \cup \Delta)$ and $u = G_i$.

   then go to Step 4.

3. Choose an edge $(v, u)$ such that

   - $st = st_F(cur\_st, L(v, u))$ is defined and
     (a) $head(anc(v)) \neq u$, and
     (b) $(st, v) \notin V\_set(u)$, and
     (c) there do not exist two distinct nodes $v_1, v_2$ such that $(st, v_1) \in V\_set(u)$ and $(st,$ $V\_set(u)$,

     or $st' = st_F(cur\_st, \overline{L(v, u)})$ is defined and
     (a′) $(st', head(anc(v))) \notin V\_set(v)$, and
       – there do not exist two distinct nodes $v_1, v_2$ such that $(st', v_1) \in V\_set(v)$ and $(st',$ $V\_set(v)$, and
   - $v \notin (set_2 \cup \Delta)$ or $u \neq G_i$.

   If $st$ is defined, 3(a), 3(b) and 3(c), then do

   - If $st$ is an accept state, $u \in set_1$ and $v \neq G_i$, then $\Delta := \Delta \cup \{u\}$.
   - $F\_list(u) := (cur\_st, v) \circ F\_list(u)$, $anc(u) := v \circ anc(u)$, $cur\_st := st$, $V\_set(u) = V\_set$ $\{(st, v)\}$, $v := u$. Go to Step 2.

   If $st' = st_F(cur\_st, \overline{L(v, u)})$ is defined, 3(a′) and 3(b′), then do

   - If $st'$ is an accept state, $v \in set_1$ and $head(anc(v)) \neq G_i$, then $\Delta := \Delta \cup \{v\}$.
   - $F\_list(v) := (cur\_st, v) \circ F\_list(v)$, $anc(v) := head(anc(v)) \circ anc(v)$, $cur\_st := st'$, $V\_set$ $V\_set(v) \cup \{(st', head(anc(v)))\}$. Go to Step 2.

4. If $head(F\_list(v)) \neq (st*, G_i)$, then $temp1 := head(F\_list(v))[1]$, $temp2 := head(F\_list($ $F\_list(v) := tail(F\_list(v))$, $anc(v) = tail(anc(v))$, $cur\_st := temp1$, $v := temp2$, and go to Step

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), $anc(v) = []$, $V\_set(v)$
   Set $v = s_k$, $F\_list(s_k) = [(st*, G_i)]$ ($st*$ is a special termination state), $anc(s_k) =$
   $F = FA(RT)$, $V\_set(s_k) = \{(init\_st_F, G_i)\}$ and $cur\_st = init\_st_F$.

2. If, for every edge $(v, u)$ one of the following is true:

   - $head(anc(v)) = u$.
   - There is a dependency $(head(anc(v)), v) \rightarrow (v, u)$ in $D$.
   - if $st = st_F(cur\_st, L(v, u))$ is defined then $(st, v) \in V\_set(u)$, and if $st' = st_F(cur\_st, \overline{L(v, u)})$ is defined then $(st', u) \in V\_set(v)$.

   then go to Step 4.

3. Choose an edge $(v, u)$ such that

   (a) $head(anc(v)) \neq u$, and

   (b) there is no dependency $(head(anc(v)), v) \rightarrow (v, u)$ in $D$, and

   (c) $st = st_F(cur\_st, L(v, u))$ is defined and $(st, v) \notin V\_set(u)$, or $st' = st_F(cur\_st, \overline{L(v, u)})$ 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 $(v, u) \rightarrow (u, G$
     $D$, then **return**(abort).
   - $F\_list(u) := (cur\_st, v) \circ F\_list(u)$, $anc(u) = v \circ anc(u)$, $cur\_st := st$, $V\_set($
     $(st, v) \cup V\_set(u)$, $v := u$. 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 $(u, v) \rightarrow (v$
     then **return**(abort).
   - $F\_list(v) := (cur\_st, v) \circ F\_list(v)$, $anc(v) = u \circ anc(v)$, $cur\_st := st'$, $V\_set($
     $(st', u) \cup V\_set(v)$. Go to Step 2.

4. If $head(F\_list(v)) \neq (st*, G_i)$, then $temp1 := head(F\_list(v))[1]$, $temp2$
   $head(F\_list(v))[2]$, $anc(v) = tail(anc(v))$, $F\_list(v) := tail(F\_list(v))$, $cur\_st := te$
   $v := temp2$ and go to Step 2.

5. **return**(commit).

Figure 10: Procedure Detect_Ins_Opt