On the Computational Complexity of the Reachability Problem in UML Activity Diagrams

Xing Tan
Semantics Technology Laboratory
Department of Mechanical and Industrial Engineering
University of Toronto
Email: xtan@mie.utoronto.ca

Michael Gruninger
Semantics Technology Laboratory
Department of Mechanical and Industrial Engineering
University of Toronto
Email: gruninger@mie.utoronto.ca

***Abstract*—The reachability problem, the problem of checking whether certain node can be reached from some initial settings such that all restrictions on flow of control are satisfied, arises frequently in various applications of Unified Modeling Language activity diagrams. However, the complexity of the problem, or in fact the complexity of any general problem in UML activity diagrams, has yet to be investigated. Towards this initiative, we specify a class of diagrams where the reachability problem is PSPACE-complete. However, if one further condition is applied, it can be shown that the problem is NP-complete.**

I. Introduction

The Unified Modeling Language (UML), as a general purpose graphical language, has been widely used to describe the structural and behavioral information in various dynamical systems. It includes up to 13 different types of diagrams to meet different requirement in specifying different aspects of a system. Among them, activity diagrams are suitable to show the overall flow of control in a system and are applied in a variety of domains such as business processes modeling, workflow modeling, and software processes modeling.

One of the most essential problems associated with an activity diagram is the reachability problem: checking whether certain node can be reached from some initial settings such that all restrictions on flow of control are satisfied. Since it corresponds directly to checking the executability of the real world process that is specified visually by the diagram, dealing with the reachability problem is inevitably an important issue that has to be resolved properly to facilitate systematic design, and efficient use of activity diagrams. Consequently, a deeper understanding of the characteristics of the reachability problem is required. We approach the problem by taking a close look on the computational properties of deciding the reachability problem in two classes of activity diagrams and our results show that the problem in general is intrinsically hard to solve.

Although for each graphical notation in UML, the Object Management Group (OMG), the distributor of UML specifications, provides a textual statement to specify its syntax and semantics [6], the specifications, as observed and addressed by others ([1], for example), are quite general and subject to open interpretation. Given this, substantial efforts have been done in proposing semantic foundations for activity diagrams ([1] and [7], for example). Among them, the line of work [7], [**?**] and [9] provides Petri-nets-like semantics for activity diagrams by mapping them into Petri-nets. We adopt this approach in essence. However, as indicated in Section III, we apply the concepts of markings directly in the activity diagrams. In doing so, the dynamics of the activity diagrams can be illustrated in a clear and straightforward manner.

Since, at its complete level, UML activity diagrams support modeling traditional structured programming constructs such as sequences, loops and conditionals, being able to decide whether a node is reachable in these diagrams would imply being able to decide if a full programming language would halt, it is clear that the reachability problem is undecidable in general. Hence, our complexity analysis is carried out on the intermediate level activity diagrams where the line of tractability boundary will be drawn. At this level, only basic control flow, such as sequencing, branching, and concurrency, is allowed.

By simulating a Nondeterministic Linear Space Automaton ([5], page 175, as a reference, see [5] and [10] for definition of general Turing machine) as a k-bounded (i.e., at any time, any node of the diagrams can not contain more than k tokens; also called as 1-safe, if $k = 1$) activity diagram, we show that the reachability problem is PSPACE-hard for that class of activity diagrams. In addition, by transforming from the NP-complete One-in-Three 3SAT problem ([5], page 259), we show that the problem is NP-complete if certain form of acyclicity is further restricted to 1-safe diagrams.

The remainder of this paper is organized as follows. Definitions for UML activity diagrams are included in the subsequent Section III. Results on computational complexity of the reachability problems in two classes of activity diagrams are covered in the main part Section IV. Section V is a brief summary.

II. UML Activity Diagrams

Graph-theoretic definitions to describe UML activity diagrams and the concept of markings to capture the dynamics of activity diagrams are introduced in Section II.A and Section II.B, respectively.

A. Basic Definitions

Definition 1: An **UML activity diagram** is a pair (N, E), where N is a finite set and E is a binary relation on N. The elements in N are called **nodes**. Each node belongs to one and only one of the following types: Ini, $Final$, $Branch$, $Merge$, $Fork$, $Join$, or $Action$. The elements in E are called **edges**. The edge set E consists of ordered pairs of nodes. That is, an edge is a set $\{u, v\}$, where $u, v \in N$ and $u \neq v$. By convention, we use the notation (u, v) for an edge, rather than the set notation $\{u, v\}$.

If (u, v) is an edge in an activity diagram, we say that (u, v) is incident from or **leaves** node u and is incident to or **enters** node v. Given a node $u \in N$, the set ${}^\bullet u = \{v | (v, u) \in E\}$ is the **pre-set** of u, where each v is the input of u, and the set $u^\bullet = \{v | (v, u) \in E\}$ is the **post-set** of u, where each v is the output of u. It is required that

$$|{}^\bullet u| \begin{cases} = 0 & \text{if } n \text{ is the } Ini \text{ node} \\ = 1 & \text{if } n \text{ is the } Final \text{ node} \\ = 1 & \text{if } n \text{ is a } Branch \text{ node} \\ \geq 2 & \text{if } n \text{ is a } Merge \text{ node} \\ = 1 & \text{if } n \text{ is a } Fork \text{ node} \\ \geq 2 & \text{if } n \text{ is a } Join \text{ node} \\ = 1 & \text{if } n \text{ is an } Action \text{ node} \end{cases}$$

and

$$|u^\bullet| \begin{cases} = 1 & \text{if } n \text{ is the } Ini \text{ node} \\ = 0 & \text{if } n \text{ is the } Final \text{ node} \\ \geq 2 & \text{if } n \text{ is a } Branch \text{ node} \\ = 1 & \text{if } n \text{ is a } Merge \text{ node} \\ \geq 2 & \text{if } n \text{ is a } Fork \text{ node} \\ = 1 & \text{if } n \text{ is a } Join \text{ node} \\ = 1 & \text{if } n \text{ is an } Action \text{ node} \end{cases}$$

B. Markings and Reachability Problem

Definition 2: A **marking** of a diagram (N, E) is a mapping in the form $MK : N \to \mathcal{N}$, to indicate the distribution of tokens on the nodes of the diagram; it can be represented as an vector $MK(n_1), \ldots, MK(n_m)$ where $n_1, \ldots, n_m$ is an enumeration of the node set N and for all i such that $1 \leq i \leq m$, $MK(n_i)$ tokens are assigned to node n_i.

A node n is **marked** at the marking MK if $MK(n) > 0$. A marked node u is further **enabled** if it is **accepted** by every node $v \in u^\bullet$. Any non-$Join$ node v such that $v \in u^\bullet$ will accept u as long as u is marked. A type $Join$ node v such that $v \in u^\bullet$ will accept u if and only if, for every node t such that $t \in {}^\bullet v$, t is marked.

The firing of an enabled node u at MK leads to the successor marking MK' (Written as $MK \stackrel{u}{\Longrightarrow} MK'$). More precisely,

1) if u is a $Branch$ node, then for every node $n \in N$,

$$MK'(n) = \begin{cases} MK(n) - 1 & \text{if } n = u \\ \{MK(n) + 1, MK(n)\} & \text{if } n \text{ accepts } u \\ MK(n) & \text{otherwise} \end{cases}$$

and we also have, $\sum_{n_i \in u^\bullet} MK(n_i) = 1$.

2) if u is a non-$Branch$ node, then for every node $n \in N$,

$$MK'(n) = \begin{cases} MK(n) - 1 & \text{if } n = u \\ MK(n) + 1 & \text{if } n \text{ accepts } u \\ MK(n) & \text{otherwise} \end{cases}$$

In other words, after the firing of u, a token is removed from u and a token is added to the only node (if u is of type Ini, $Action$, $Merge$, $Join$), each node (if u is of type $Fork$), one and only one node (if u is of type $Branch$), in the post-set of u. There is no need to fire a node with type $Final$.

3) if a token fired by u is accepted by a $Join$ node n, it necessarily implies the simultaneous firings of all marked node u_i such that $u_i \in {}^\bullet n$. That is,
   a) their firings are atomic, and
   b) any two permutated firing sequences of the elements in ${}^\bullet n$ are equivalent,
   c) one token is assigned to the node $Join$,
   d) for each $u_i \in {}^\bullet n$, a token is removed.

   In this case, we write $MK \stackrel{\overrightarrow{u}}{\Longrightarrow} MK'$, where $\overrightarrow{u}$ stands for a linearization of all elements in ${}^\bullet u$.

The **firing sequence** $\sigma = n_1, ..., n_m$ is a sequence of nodes in N (again, note that some subsequence of σ might be inseparable and enabled by one $Join$ node). For particular σ and MK, σ is **legal** at MK if there are marking sequence $MK_0, MK_1, ..., MK_m$ such that $MK = MK_0$, $M_0 \stackrel{n_1}{\Longrightarrow} MK_1$, ..., $MK_{m-1} \stackrel{n_m}{\Longrightarrow} MK_m$ (written as $MK \stackrel{\sigma}{\Longrightarrow} MK_m$). Also, We write $MK \stackrel{*}{\Longrightarrow} MK'$ if there exists a firing sequence σ such that $MK \stackrel{\sigma}{\Longrightarrow} MK'$.

The **reachability set** of an activity diagram instance in the form of a 3-tuple (N, E, MK_0), where (N, E) is the diagram and MK_0 is its initial marking, is the set of all MK' such that $MK_0 \stackrel{*}{\Longrightarrow} MK'$ in (N, E).

Definition 3: The **reachability problem (REACH)** for an (N, E, MK_0) is to decide whether some arbitrary final marking MK_{final} is in the reachability set of the instance.

An instance (N, E, MK_0) is **cyclic**, or **reversible**, if there exists a firing sequence σ in the reachability set such that some node, say n_i, appears more than once in σ. It is **acyclic**, or **irreversible**, otherwise. An instance (N, K, MK_0) is **k-bounded** if the number of tokens of any node $n \in N$ at any MK in the reachability set is bounded by k; it is **1-safe** if $k = 1$. Without loss of generality, we can assume that REACH is to test, in an activity diagram with an unique Ini node and $Final$ node where the initial MK_0 contains a unique token in Ini, if the token can reach $Final$. Hence,

Definition 4: **OSafe-REACH** is the reachability problem for 1-safe activity diagrams.

Definition 5: **KBounded-REACH** is the reachability problem for k-bounded activity diagrams.

Definition 6: **A-OSafe-REACH** is the reachability problem for acyclic, 1-safe activity diagrams.

Figure 1 is a pictorial representation of an instance of (N, E, MK_0), where
$N = \{Ini, Fin, A, B, C, D, F1, F2, B, M, J\}$,

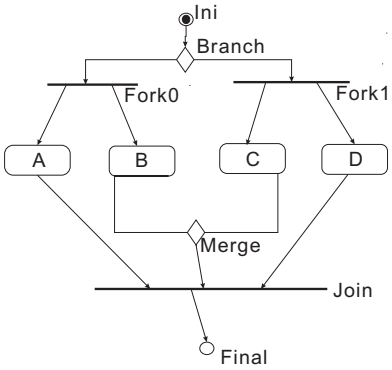

Fig. 1. An example of an intermediate UML activity diagram

$E = \{(Ini, B), (B, F0), (B, F1), (F0, A), (F0, B), (F1, C), (F1, D), (A, J), (B, M), (C, M), (D, J), (M, J), (J, Fin)\}$, and $\{MK_0(Ini) = 1, MK_0(n_i) = 0$ for all other nodes$\}$.

In this example, the REACH problem involves finding a firing sequence from node Ini that will deliver a token into the node Fin. Note that Fin is not reachable from Ini, as no matter which branch Ini chooses, flow of control can only arrive at two of the three branches entering the node J, making the occurrence of J impossible. However, Fin is certainly reachable if the node J is replaced with a node of type $Merge$.

III. Computational Complexity

In this section, the computational complexity of OSafe-REACH and A-OSafe-REACH are presented. The PSPACE-hard part of the proof in Theorem 1 is particularly inspired by a similar construction showing that the reachability problem in a 1-safe Petri nets is PSPACE-hard [4].

Theorem 1: The OSafe-REACH problem is PSPACE-complete.

Proof: We first prove OSafe-REACH is in PSPACE by showing that it is in NPSPACE. Suppose we have an instance of OSafe-REACH in the form of 4-tuple $(G, MK_0, Ini, Final)$, where $G = (N, E)$ is an activity diagram, MK_0 is the initial marking of G, and Ini and $Final$, respectively, is the initial and final node in G. Since G is 1-safe, the maximal number of different markings of G is 2^m, where $m = |N|$. Hence, if there ever exists a firing sequence σ such that $MK_0 \stackrel{\sigma}{\Longrightarrow} MK'$ and MK' marks $Final$, we must have $|\sigma| \leq 2^m$, i.e., MK' shall be reached in 2^m steps. Otherwise, some marking must be repeated in σ and the subsequence between any two repeated markings can be removed in σ to obtain a shorter σ' whereas $MK_0 \stackrel{\sigma'}{\Longrightarrow} MK'$ also holds.

As such, we propose a nondeterministic algorithm that decides OSafe-REACH as follows:

On input $(G, MK_0, Final)$:

1) Nondeterministically select a firing sequence σ, where $|\sigma| \leq 2^m$.
2) Firing σ from MK_0 to obtain a MK', i.e., $MK_0 \stackrel{\sigma}{\Longrightarrow} MK'$. (Simply *reject* if σ is not completely firable from MK_0.)
3) *Accept* if the $Final$ node is marked in MK', *reject* otherwise.

The algorithm hence is bounded by nondeterministic linear space. As it precedes, a size m stack is required to maintain the current marking. But by virtue of Savitch's theorem, OSafe-REACH is also in PSPACE.

To show that OSafe-REACH is PSPACE-hard, suppose we have an NLBA T in the form $(Q, \Gamma, \Sigma, \delta, q_{ini}, Q_{accept}, Q_{reject}, \$)$ and is bounded by the size of its input string, whereas $\{Q = q_1, q_2, \ldots, q_m\}$ is the set of states and $q_1 = q_{ini}$ and $q_{accept}, q_{reject} \in Q$, Σ is the set of input symbols, $\Gamma = \{a_1, a_2, \ldots, a_p\}$ is the set of tape symbols, $\Gamma \supseteq \Sigma$, and $\$ = a_1$. Given a sentence $w = \$w_1 w_2 \ldots w_n\$$ in $\$\Sigma^*\$$, we will construct in polynomial time an instance of activity diagram $OSafe(T) = (N, E, MK_0, Ini, Final)$ as follows:

1) $N_{table} \subseteq N$ and $N_{table} = \{LocationSymbol_{i,j} | 0 \leq i \leq n+1, 1 \leq j \leq p\} \cup \{LocationState_{i,j} | 0 \leq i \leq n+1, 1 \leq j \leq m\} \cup \{Ini, Final\}$, that is,
   a) The 2-dimensional table $LocationSymbol$ marked with w tokens is used to maintain the current content in the tape of T and the 2-dimensional table $LocationState$ with one token is used to indicate the current state. In other words, the automaton T is in state q_j scanning location i with the symbol k on i if and only if there are one token at $LocationSymbol_{i,k}$ and $LocationState_{i,j}$, respectively.
   b) a token at node $LocationSymbol_{0,1}$, and at $LocationSymbol_{n+1,1}$, corresponds to the symbol "\$" at the left end, and the right end of the input string of T.
   c) each $n_i \in N_{table}$ is an action node.
2) $E_{ini} \subseteq E$ and $E_{ini} = \{(Fork_{ini}, LocationSymbol_{i,j}) |$input has symobl j at $i\}$ $\cup$ $\{(Fork_{ini}, LocationSymbol_{0,1}\}$ $\cup$ $\{(Fork_{ini}, LocationSymbol_{n+1,1}\} \cup \{(Ini, Fork_{ini}\}$, that is, a token in Ini will be forked into elements of the table, reflecting the input for T.
3) The initial marking $MK_0(T)$ simply assigns one token at Ini.
4) If a state is in Q_{accept}, then, it will enter the node $Final$. (See Fig. 2 for an example of the first four steps).
5) For each transition function $\delta_i \in \delta$ such that $\delta_i = \{(q_s \times a_t) \longrightarrow (q_u \times a_v \times L), (q_s \times a_t) \longrightarrow (q_w \times a_r \times R)\}$ and for each location j such that $1 \leq j \leq n$, we construct
   a) a subgraph $CpntLSS_{i,j,s,t} = (CpntN_i, CpntE_i)$, where $CpntN_i \subseteq N$ such that $CpntN_i = \{Fork_i, Join_i\}$ and $CpntE_i \subseteq E$ such that $E_i = \{(Fork_i, LocationState_{j,s}), (Fork_i, LocationSymbol_{j,t}), (LocationState_{j,s}, Join_i), (LocationSymbol_{j,t}, Join_i)\}$, constructs a component corresponding to: T is at

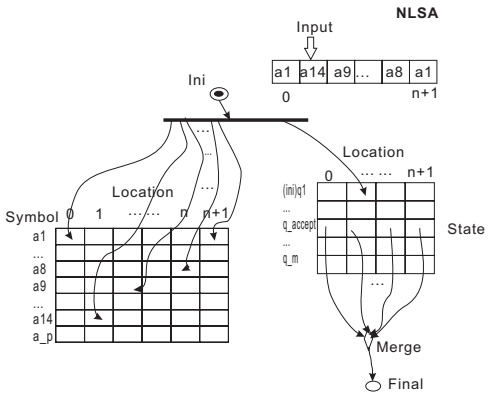

Fig. 2. An Example of an initial marking in an $OSafe(T)$

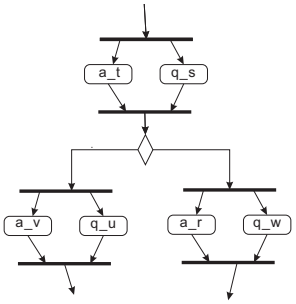

Fig. 3. An example of a nondeterministic transition function in $OSafe(T)$

state s, reading symbol t at location j.

b) The subgraphs $CpntLSS_{i,j-1,s,t}$, and $CpntLSS_{i,j+1,s,t}$, are constructed similarly.

c) A branch node $Branch_i \in N$ is introduced to simulate the nondeterministic transition of δ_i. More specifically, $CpntLSS_{i,j,q_u,a_v}$ enters $Branch_i$ and $Branch_i$ leaves for $CpntLSS_{i,j-1,q_s,a_t}$ and $CpntLSS_{i,j+1,q_w,a_r}$ (see Fig. 3 for an example).

6) Transition functions dealing with boundary cases can be treated similarly as in the step above.

7) In T, transition function for a state q_i might include reading different symbols, say a_j or a_k, for example. Accordingly, the construction will have to deal with two components, say $CpntLSS_{i,j,k,l}$ and $CpntLSS_{i,j,k,m}$, which share the node $LocationState_{j,k}$. In this case, a $Merge$ node is added before the node to take a token from either component and a $Branch$ node is added after the node to return the token to the component. Since any T can not in the same time at different state, or location, it is impossible for this $Merge$ or $Fork$ node to have multiple tokens thus the 1-safe property is maintained. (see Fig. 4 for an illustration).

8) N (and E) does not contain any other nodes (or edges, respectively).

Since any nondeterministic move in T corresponds precisely to a nondeterministic firing of a token in $OSafe$ and vice versa, it is obvious that T accepts w if and only if $OSafe(T)$ can reach the $Final$ node. Finally, we complete this proof by showing

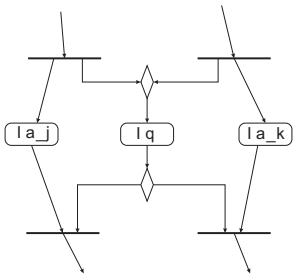

Fig. 4. An example of a $LocationState$ node corresponding to multiple Components in $OSsafe(T)$

that the transformation is bounded by $O(n|Q \cdot \Gamma|^2)$. Since in the worst case, each location involves a nondeterministic transition function that deals with every symbol in Γ for every state in Q and the number of nodes created is bounded thus by $n \cdot |Q| \cdot |\Gamma| \cdot |Q| \cdot |\Gamma| \cdot 2$. ■

Similarly we can prove

Corollary 2: The KBounded-REACH problem is PSPACE-complete.

Proof: Based on the fact that, the total number of markings in a k-bounded activity diagram is bounded by $\left[\left(\sum_{i=1}^{k} i\right) + 1\right]^m$, a nondeterministic algorithm can be proposed accordingly, ensuring that the problem is also in PSPACE. The same PSPACE-hard proof from Theorem 1 can be applied here as a 1-safe diagram is certainly k-bounded. ■

Theorem 3: The A-OSafe-REACH problem is NP-compete.

Proof: It is easy to see that A-OSafe-REACH is in NP. Since we know that, if the $Final$ node in such an acyclic diagram is reachable, the firing sequence starting from Ini and reaching it is bounded by the size of the nodes in the diagram $|N|$. A nondeterministic algorithm need only guess a firing sequence and check in polynomial time if it satisfies all constraints and delivers the token to the $Final$ node.

To show that A-OSafe-REACH is NP-hard, we will transform One-In-Three 3SAT to A-OSafe-REACH. Suppose we have an arbitrary instance of One-In-Three 3SAT, where $U = \{u_1, u_2, ..., u_n\}$ is a set of variables and $C = \{c_1, c_2, ..., c_m\}$ such that $|c_j| = 3$ for $1 \leq j \leq m$ is the set of clauses. We construct an activity diagram $AOSafeReach(OneInThree3sat) = (N, E, MK_0, Ini, Final)$ with the following steps.

1) For each variable u_i and its negation $\bar{u_i}$ we construct an auxiliary variable component in which a$Branch$ node called B_{u_i} enters the $Action$ node u_i and the $Action$ node $\bar{u_i}$ before merging into a $Merge$ node M_{u_i}.

2) For each clause $C_j = [p_{j1}, p_{j2}, p_{j3}]$ we construct an auxiliary clause component in which a $Branch$ node called B_{C_j} enters three branches corresponding to actions p_{j1}, p_{j2}, p_{j3} before merging into a Merge node called M_{C_j}.

3) The diagram contains a $Fork$ node $Fork$ and a $Join$ node $Join$, where Ini enters $Fork$, $Join$ enters $Final$. Also, $Fork$ enters every node B_{u_i} for $1 \leq i \leq n$ and every node B_{C_j} for $1 \leq j \leq m$, whereas every M_{u_i} for

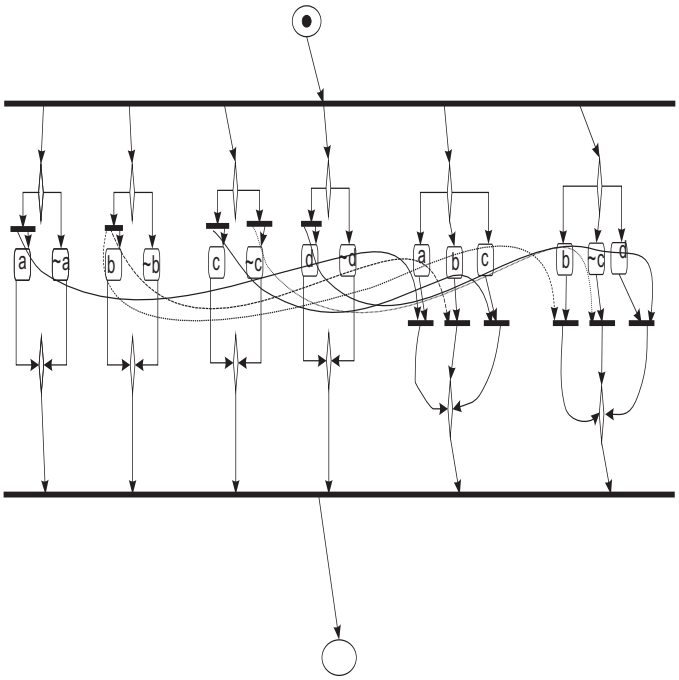

Fig. 5. Polynomial time transformation from an One-In-Three 3SAT Instance to an activity diagram

$1 \leq i \leq n$ and every M_{C_j} for $1 \leq j \leq m$ enters $Join$.

4) In the constructed diagram, we also need to indicate which literals occur in which clauses in the One-In-Three 3SAT instance. Assume literal p_{ij} represents a negation of a variable $\bar{u_k}$ (or a variable u_k, but the construction is similar), we now add an extra fork node $Fork_{\bar{u_k}}$ such that B_{u_k} enters $Fork_{\bar{u_k}}$ instead (not directly to $\bar{u_k}$) and $Fork_{\bar{u_k}}$ enters $\bar{u_k}$. Additionally, on the corresponding clause component where p_{ij} belongs to, $Fork_{\bar{u_k}}$ and p_{ij} enters a new $Join$ node $Join_{p_{ij}}$ and $Join_{p_{ij}}$ enters M_{C_j}.
5) The initial marking $MK(OneInThree3SAT)$ simply assigns one token at Ini.

An illustration of this construction is given in figure 5, where the One-In-Three 3SAT instance is $C = \{[a, b, c], [b, \neg c, d]\}$.

We now show that the node $Final$ is reachable from Ini in the diagram $AOSafeReach(OneInThree3SAT)$, if and only if, $OneInThree3SAT$, the instance of the problem is one-in-three satisfiable. Suppose first that $t : U \rightarrow \{T, F\}$ is a one-in-three truth assignment satisfying C. Based on t, we can construct a firing sequence δ such that, 1) for each variable u_i, if $t(u_i) = True$ (or $t(\bar{u_i}) = True$), the node u_i (or $\bar{u_i}$) should be included in the first segment of δ (relative order between u_is does not matter); 2) for each literal p_{ij}, if $p(i_j) = True$, the node p_{ij} should be included in the second segment of δ (relative order between u_is does not matter). It is easy to verify that firing of δ constructed in this way can reach $Final$ from Ini in the constructed diagram. Conversely, if δ is a reachable firing sequence in the diagram, for each i such that $1 \leq i \leq n$, B_{u_i} and M_{u_i} will ensure that only one of u_i or $\bar{u_i}$ is chosen where as B_{C_i} and M_{C_i} will ensure that only one of p_{ij} in C_j is chosen , but this constructs a One-In-Three truth assignment t for the 3SAT problem. It is required that each of the clause component should merge to the $Fork$ node, which means that t in the same time is actually a satisfiable assignment. The transformation can be verified as a polynomial time transformation without difficulty, since the size of constructed diagram is bounded by a polynomial mn, which is the size of the 3SAT instance. ■

IV. Summary

Starting by defining activity diagrams as directed graphs with typed nodes, and introducing the concept of markings directly into the UML activity diagrams, we have shown that the reachability problem in activity diagrams is at least PSPACE-hard. This result is significant since the problem is one of the most important problems associated with the design and use of activity diagrams. Given the popular use of the diagrams, an extensive study of the complexity of all typical problems associated with activity diagrams is obviously necessary. In the future, we are interested in continuing this line of work by defining subclasses of the diagrams where the reachability problem is tractable and investigating new problems other than the reachability problems on these subclasses.

Finally, we emphasis that, although there exist extensive complexity results of reachability problems for 1-safe Petri-nets (see [2] for example), those results can not, by any means directly, be used to obtain the complexity results reported in this paper. Since there does not exist a trivial way to correctly map a Petri net into an activity diagram even if the reversed direction is obviously practical (see [7] again).

References

[1] Börger, E., Cavarra, A., Riccobene, E.: An ASM semantics for UML activity diagrams. AMAST 2000, Iowa City, Iowa, USA (2000) 293–308
[2] Cheng, A., Esparza, J, Palsberg, J: Complexity results for 1-safe nets. Theoretical Computer Science **147** (1995) 117–136
[3] Esparza, J: Decidability and complexity of Petri Net problems- an introduction, Problems-An Introduction, Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, Lecture Notes in Computer Science. **1491** (Springer-Verlag, 1998) 374-428.
[4] Jones, N.D., Landweber, L.H., Lien, Y.E.: Complexity of some problems in Petri nets. Theoretical Computer Science **4** (1977) 277–299
[5] Garey, M.R., Johson, D.S.: Computers and intractability - a guide to NP-completeness. W.H. Freeman and Company, Cambridge, MA, USA (1979)
[6] OMG: Unified Modeling Language (OMG UML), Superstructure, V2.1.2. (2007)
[7] Störrle, H.: Semantics of control-flow in UML 2.0 Activities. VLHCC '04: Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing, Washington, DC, USA (2004) 235–242
[8] Störrle, H.: Semantics and verification of data flow in UML 2.0 Activities. Electronic Notes in Theoretical Computer Science **127** (2005) 35–52
[9] Störrle, H., Hausmann, J.H.: Towards a formal semantics of UML 2.0 Activities. In: Liggesmeyer, P., Pohl, K., Goedicke, M. (eds.) Software Engineering. LNI., GI, **64**, (2005) 117–128
[10] Sipser, M.: Introduction to the theory of computation, Second Edition. PWS Publishing Company, Boston, MA, USA (2006)