On Temporal Logic Programming Using Petri Nets

Abbas K. Zaidi, Member, IEEE

Abstract—A methodology for modeling temporal (time-sensitive) aspects of discrete-event systems is presented. A formalism of temporal logic which incorporates both point and interval descriptions of time is formulated, which is an extension of Allen’s interval logic [1]. A formal axiomatic system of this point-interval logic is presented. A graph model is shown to implement the axiomatic system of point-interval logic. This graph-based approach transforms the system’s specifications given by temporal statements into a graph structure. The graph-based temporal inference engine (TIE) identifies temporal ambiguities and errors (if present) in the system’s specifications, infers new temporal relations among system’s intervals, and identifies the user-defined intervals of interest.

I. INTRODUCTION

THE growing need for a formal logic of time for modeling and analyzing real world systems has led to an emergence of various kinds of representations and reasoning schemes for temporal information. The earliest attempts at formalizing a time calculus date back to 1941 by Findlay [2], and 1955 by Prior [3], [4]. Since then, there has been a number of attempts on issues related to this subject matter, like topology of time [5], first-order and modal approaches to time [6]–[18], treatments of time for simulating action and language, etc. Several attempts on mechanizing the temporal reasoning processes have also been reported in the literature [19]–[24]. The list of references provided in this paper is not at all exhaustive and may have missed some of the major contributions; for a comprehensive discussion on issues and an overview of temporal methodologies, the reader is referred to [4]. A majority of these approaches is formulated to handle issues relevant to a specific application area of interest to the scholar(s) who proposed the approach. Therefore, each approach comes with its advantages in addressing certain issues and disadvantages for lack of generality in handling others. Artificial Intelligence, software engineering, planning, cognitive sciences, time-sensitive databases, program verification and modeling of discrete-event systems are some of the examples of the application areas which have contributed mainly to the formulation of time calculi.

This paper presents an approach for the modeling and subsequent analysis of the temporal aspects of discrete-event systems. A system’s temporal aspects might be represented in terms of properties that hold for certain time intervals, processes taking some time to complete, and/or events/occurrences requiring virtually no time to take place. In order to model a system’s temporal aspects, one, therefore, needs to have a formalism capable of handling both interval and point descriptions of various components of the system.

An extension of Allen’s [1], [7]–[11] interval logic that incorporates both descriptions, point and interval, of time is presented. The new formalism, called point-interval temporal logic (PITL), extends the axiomatic system of interval logic by adding new axioms to it. A Petri net model is shown to represent this axiomatic system by transforming the system’s specifications given by temporal statements of point-interval logic into Petri net structures. A temporal inference engine (TIE) based on this Petri net representation infers new temporal relations among system intervals, identifies temporal ambiguities and errors (if present) in the system’s specifications, and finally identifies the intervals of interest defined by the user. The real strength of this inference engine is that it performs all these functions by completely avoiding the combinatorial nature of the problem inherent in the inference engines of its kind [20].

The paper is organized as follows. Section II presents the definitions of different notions and terms used in the approach together with an introduction of PITL. The section also presents an analytical model for the temporal relations. A transformation of this model to an equivalent Petri net representation is presented in Section III. The structure of Petri net representing temporal relations among intervals is shown to reveal inconsistencies and incompleteness present in the system. The graph-based inference engine, TIE, is also presented in this section. Section IV applies the result of the methodology to an example problem. Finally, Section V concludes the discussion by summarizing the contributions and identifying future extensions of the approach.

II. MATHEMATICAL MODEL

Definition 1: Interval: An interval is denoted by a symbol, i.e., a letter or a digit, and is defined as \(X = [sx, ex]\), where \(ex \geq sx\). In this definition \(sx\) denotes "start of \(x\)" and \(ex\) "end of \(x\)". The two bounds are abstract notions and may not have numerical values assigned to them. The intervals considered in this paper are all closed intervals.

Definition 2: Interval Length: The length of an interval \(X = [sx, ex]\), denoted by \(|X|\), is defined as \(|X| = ex - sx\).

Definition 3: Point: An interval \(X\) with \(|X| = 0\) is a point interval; \(X = [px, px]\) is a point interval or point, also denoted as \([px]\). In a system description, a point may be used to signify an occurrence or an event.

Definition 4: Temporal Relation \(R_t\): The temporal relation \(R_t\) is a truth functional binary relation defined as

\[R_t : I \times I \rightarrow \{T,F\}\]
where $I$ is the set of intervals and $T$ and $F$ are the Boolean values, true and false.

**Definition 5:** Composite Interval: Let $X = [sx, cx]$ and $Y = [sy, cy]$ be two intervals, where $sx = sy$ and $cx = cy$, then they are jointly represented by a composite interval $[X; Y] = [sx; sy, cx; cy]$. By definition $[X; X] = [X]$ or simply $X$.

**Definition 6:** Composite Point: Let $X = [px]$ and $Y = [py]$ represent a single point on the time line, i.e., $px = py$, then the composite point is represented as $[X; Y] = [px; py]$.

**A. Point-Interval Temporal Logic**

The point-interval logic presented in this paper is an extension of Allen’s interval logic [1], [7]–[11]. The formalism considers a single time line. Any two intervals with nonzero lengths on this time line are related by one of the seven relationships presented in Fig. 1, case I. This set of seven temporal relations between two intervals was first presented by Allen [1]–[3] in his interval logic. The underlying assumption of the interval logic is the nonzero lengths of time intervals [4]. The present approach relaxes this assumption and allows intervals with zero lengths—points. Case II, Fig. 1, presents two possible temporal relations between two points. Finally, case III, Fig. 1, presents the possible set of temporal relations between a point and an interval. From now on, the term interval is used to refer both intervals and points if not explicitly stated otherwise.

**Definition 7:** Set of Temporal Relations, $R$: $R$ represents the set of temporal relations $Ri$ and is given as

$$R = \{\text{Before, Meets, Overlaps, Starts, During, Finishes, Equals}\}.$$  

**Proposition 1:** The temporal relations presented in Fig. 1 are mutually exclusive and exhaustive, i.e.,

1) if $(X Ri Y), Rit$ is an element of $R$, then there does not exist an $Rj$ in $R$ such that $(X Rj Y)$ holds true;
2) for any two intervals $X$ and $Y$ there must exist an $Ri$ in $R$ such that either $(X Ri Y)$ or $(Y Ri X)$ holds true (with the exception of Equals relation where $(X Equals Y) = (Y Equals X)$).

**B. Analytical Model**

An analytical representation of temporal relations among intervals is shown in Fig. 1. According to this representation, a temporal relation between two intervals can be described as a set of algebraic inequalities among points representing the start and end of these intervals.

Two points, $p1$ and $p2$, on a single time line can be related to each other by one of the following four relations:

1) $p1 > p2$;
2) $p1 = p2$;
3) $p1 < p2$;
4) $p1 ? p2$ (? represents “unknown”. This is added to incorporate incomplete information.)

With these four possibilities and the analytical model of Fig. 1, a temporal relation $Ri$ between two intervals $X$ and $Y$ can be represented as a four-digit string made of elements from the alphabet $\{<, =, >, ?\}$, where the first (left-most) digit represents the relation between $sx$ and $sy$, second digit between $sx$ and $cy$, third digit represents relation between $cx$ and $sy$, and fourth between $cx$ and $cy$. The representation does not take into account the point or interval nature of $X$ and $Y$. The information about the nature of intervals is assumed to be known; however, this four-digit representation can be extended to accommodate this feature as well.

The description of a temporal relation $Ri$ (known or unknown or partially known) between two intervals $X$ and $Y$, therefore, requires only 8 bits (for a computer implementation). This byte representation of a temporal relation between two intervals, and the nature of the intervals (which requires 1 bit for each interval) are all that is required to store (and maintain) complete, or partially/totally incomplete information about the two intervals. Even if one needs to store temporal relations between all possible combinations of intervals from a set of $n$ intervals, it will take $n + n(n - 1)/2$ bytes to store all.
this information. Temporal inconsistencies can be identified by checking inconsistent patterns of this 8-bit representation.

C. Axioms of Point-Interval Logic

1) Point Axioms:
   Let \( p_1, p_2, \) and \( p_3 \) be points defined on a single time line. The following four (1–4) axioms can be used to infer the temporal relation between two points on the time line. The remaining combinations of temporal relation among three points (L.H.S. of the axioms) result in an unknown, \( ? \), relation between the two points at the R.H.S. of the axioms; this is described by axiom 5, where underscore, \( _\_ \), is used to denote combinations of any of the four temporal relations \(<, =, >, ? \) not covered by axioms 1–4.

Together with the four-digit representation of a temporal relation and the nature of the intervals involved, point axioms implement the axiomatic system of the point-interval logic. The following example illustrates some of the axioms of point-interval logic.

Example 1: Let \( X, Y \) and \( Z \) be three intervals, with the following known relations among them:
1) \( X \text{ Overlaps } Y \);
2) \( Y \text{ Overlaps } Z \).

The corresponding string representations are given as follows:

\[
\begin{align*}
\text{sx} Vs \text{ sy} & \quad \text{sx} Vs \text{ ey} & \quad \text{ex} Vs \text{ sy} & \quad \text{ex} Vs \text{ ey} \\
\text{sx} Vs \text{ sz} & \quad \text{sy} Vs \text{ cz} & \quad \text{ey} Vs \text{ sz} & \quad \text{ey} Vs \text{ cz}.
\end{align*}
\]

The application of point axioms to these two strings yields

\[
\begin{align*}
\text{sx} Vs \text{ sz} & \quad \text{sx} Vs \text{ ez} & \quad \text{ex} Vs \text{ sz} & \quad \text{ex} Vs \text{ ez} \\
< & \quad < & \quad > & \quad <.
\end{align*}
\]

The resulting string corresponds to one of the following relations between \( X \) and \( Z \):
1) \( X \text{ Before } Z \) (obtained by replacing “?” with “<”);
2) \( X \text{ Meets } Z \) (obtained by replacing “?” with “=”);
3) \( X \text{ Overlaps } Z \) (obtained by replacing “?” with “>”).

The corresponding axiom of the point-interval logic can now be described as

\[
(X \text{ Overlaps } Y) \land (Y \text{ Overlaps } Z) \\
\rightarrow (X \text{ Before } Z) \lor (X \text{ Meets } Z) \lor (X \text{ Overlaps } Z).
\]

The logical operator \( \lor \) is defined as follows:

\[
A_1 \lor A_2 \lor \cdots \lor A_n \\
= (A_1 \lor A_2 \cdots \lor A_n) \lor (A_1 \lor A_2 \cdots \lor A_n) \lor \cdots \\
\lor (A_{n-1} \lor A_{n-1} \lor A_n) \lor (A_{n-1} \lor A_{n-1} \lor A_n)
\]

where \( \overline{A} \) is Not \( A \).

D. Inference Engine

One may visualize an inference engine based on the axiomatic system presented in the previous section. Suppose we have a known temporal relation between two intervals \( X \) and \( Y \). Now, if a new temporal relation between two intervals \( Y \) and \( Z \) is discovered (or supplied by the user), the inference engine constructs an analytical representation of the temporal relation between \( X \) and \( Z \) with the help of known relations among \( X, Y \) and \( Z \), and the point axioms presented in the previous section. The resulting string representation of the relation between \( X \) and \( Z \) is pattern matched with the string representations of temporal relations to infer the applicable one(s) between \( X \) and \( Z \). However, the construction of the analytical representation of an unknown relation between two intervals, \( X \) and \( Z \), from the known temporal relation among system’s other intervals can be accomplished via a combinatorially large number of ways. The following example illustrates this issue.

Example 2: Let a system be described in terms of the following statements:
1) \( X \text{ Meets } Y \);
2) \( X \text{ Meets } Z \);
3) \( W \text{ Before } U \);
4) \( Y \text{ Finishes } U \).

The inference engine is required to infer temporal relation between intervals \( W \) and \( Y \). In the example statements, there are two possible alternatives to calculate the unknown relation: a) with the help of known relations \( (W \text{ Meets } X) \), and \( (X \text{ Meets } Y) \), infer the temporal relation between \( W \) and \( Y \); b) with the help of known relations \( (W \text{ Before } U) \), and \( (Y \text{ Finishes } U) \), infer the temporal relation between \( W \) and \( Y \). Both alternatives imply the same temporal relation between the two intervals—\( W \text{ Before } Y \).

The example illustrates the fact that an inference engine based on the axiomatic system of PITL has to search all possible combinations of the known temporal statements in order to explore the right combination(s) of statements that yields the result. The inference required to calculate an unknown relation may not be a single step inference (as illustrated in the example, where two statements are used to infer a third one.) Consider a situation where an unknown relation between \( X \) and \( Y \) is required to be inferred, and the known statements are \((X \text{ Meets } Y)((I_1 \text{ Meets } I_2), \ldots, (I_i \text{ Meets } I_j), \ldots, (I_n \text{ Meets } Y))\). In such a situation, a chain of inference is required to derive the unknown relation between \( X \) and \( Y \). The search for all such combinations of statements that (may) yield the required result among all possible combinations of known statements makes the inference process computationally intensive. (The combinations that may yield the required result are referred to as feasible combinations.) One may suggest that an inference engine does not have to search for all feasible combinations of statements; the search can be halted as soon as the first such combination is encountered, and a required relation is inferred. An inference engine with this approach can only be applied to a system of temporal statements, which is known to be consistent \textit{a priori}. A formal definition of consistency in temporal logic follows in the next section.
An inference engine for PITL requires an exhaustive enumeration of the result through all feasible combinations of statements, provided no knowledge of system’s correctness is available a priori. For a system with \( n \) intervals there could be different combinations to calculate the unknown relation between two intervals, where each combination with \( k \) intervals itself can have \( k! \) possible ways to perform the \( 2^{n-2} \) desired calculation. An inference engine that outputs the result as soon as it finds the first feasible set of inputs can only be applied to a known consistent system of temporal statements. This, in turn, requires a front-end verification mechanism for the inference engine that verifies the system’s correctness prior to applying the axioms of PITL. A detailed discussion and solution to this problem is presented in the next section.

### III. TL/PN METHODOLOGY

The complete formal axiomatic system presented in Section II for PITL can now be used to model temporal aspects of a system. This section presents a graph based approach, termed temporal logic/Petri net (TL/PN) formalism, for the implementation of the axiomatic system of PITL. The TL/PN approach transforms the system’s specifications given by temporal statements into a graph structure. The graph-based temporal inference engine (TIE) identifies temporal ambiguities and errors (if present) in the system’s specifications, infers new temporal relations among system’s intervals, and finally identifies the windows of interest to the user. The temporal inference engine, TIE, of the TL/PN methodology performs all these tasks by completely avoiding the combinatorial nature of the inference mechanism discussed in the previous section.

In the present approach, all input statements are connected together with an implicit conjunction; they are all valid simultaneously. Therefore, they represent a system’s temporal description on a single time line: for any point on this time line there will exist only one future—a paradigm which will be referred to as single time line single future (STSF).

#### A. Petri Net Representation

The analytical model of a temporal relation \( R_h \) presented in Fig. 1 is transformed to a Petri Net (PN) representation by the following approach.

- A point \( t_1 \) is represented as a transition, labeled \( [t_1] \).
- Two points \( t_1, t_2 \) with \( t_1 = t_2 \) are represented as a single transition, labeled \( [t_1; t_2] \).
- Two points \( t_1, t_2 \) with \( t_1 < t_2 \) are represented as two transitions, \( [t_1] \) and \( [t_2] \), with a link from \( [t_1] \) to \( [t_2] \).

Readers not familiar with the theory and formalism of Petri nets are referred to \[26\]–\[28\] for both an introductory and advanced treatment of the subject matter.

**Definition 8: Link:** A place \( P \) is said to constitute a link from \( [t_j] \) to another \( [t_j] \) if the following condition holds:

\[
*P = \{[t_i]\} \quad \text{and} \quad P^* = \{[t_j]\}
\]

where \( *P \) and \( P^* \) represent the pre- and post-set of the place \( P \).

**Example 3:** Let \( t_1 \) and \( t_2 \) be two points on a single time line with \( t_1 < t_2 \); the Petri net representation of the temporal relation between the two points is shown in Fig. 2(a). For the sake of simplicity, the PN is transformed to a simpler graph representation with each transition in the PN represented as a node (vertex) and a link between two transitions as an arc (edge) between the two nodes representing the transitions. The temporal relation of the example is now represented as in Fig. 2(b).

The example presents a much simpler graphical representation for the temporal relation between two time points and is referred to as point graph (PG) representation. From now on, instead of using the Petri net representation, the PG representation is used for illustration purposes. However, the use of the Petri net formalism for modeling temporal statements will become necessary when the TL/PN methodology is extended to include alternative inputs. A discussion on this issue will appear in a future publication.

**Definition 9: Point Graph:** A point graph PG \((V,E)\) is a directed graph with each node or vertex \( v \in V \) representing a point on a time line and each edge \( e_{12} \in E \), between two vertices \( v_1 \) and \( v_2 \), representing a temporal relation “\( < \)” between the two vertices—\( (v_1 < v_2) \).

A temporal relation \( R_h \) between two intervals \( X \) and \( Y \) can now be represented by an equivalent Point Graph representation, as shown in Fig. 3.

The TL/PN approach takes statements in the point-interval logic and transforms each individual statement into an equivalent PG representation. The PG representing the entire system of temporal statement is then constructed by unifying individual PG’s to a (possibly) single connected graph. The unifying process employs a very simple approach, which merges two nodes into a single node if the labels of the two nodes contain at least one point in common. The following definition presents a formal description of this unification process.

**Definition 10: Unification:** Let \( v_1 = [p_i, \ldots, p_n] \) and \( v_2 = [p_j, \ldots, p_m] \) be two nodes in a PG representation. If there exist a point \( p_k \) such that: \( p_k \in [p_i, \ldots, p_n] \) and \( p_k \in [p_j, \ldots, p_m] \), then the two nodes are merge into a single node \( v_{12} \) s.t.

\[
\begin{align*}
&v_{12} = [p_i, \ldots, p_n] \cup [p_j, \ldots, p_m] \\
&v_{12}^* = v_1^* \cup v_2^*.
\end{align*}
\]

The following example illustrates the unification process.

**Example 4:** Let a system be described in terms of the following statements:

1. \( W \text{ Meets } X \);  
2. \( X \text{ Meets } Y \);  
3. \( W \text{ Before } U \);  
4. \( U \text{ Finishes } Y \).
CASE I—X and Y both intervals with non-zero lengths:
X = [sx, ex], Y = [sy, ey] with sx < ex and sy < ey

1. X Before Y
   \[ \text{[sx]} \rightarrow \text{[ex]} \rightarrow \text{[sy]} \rightarrow \text{[ey]} \]

2. X Meets Y
   \[ \text{[sx]} \rightarrow \text{[ex, sy]} \rightarrow \text{[ey]} \]

3. X Overlaps Y
   \[ \text{[sx]} \rightarrow \text{[sy]} \rightarrow \text{[ex]} \rightarrow \text{[ey]} \]

4. X Starts Y
   \[ \text{[sx, sy]} \rightarrow \text{[ex]} \rightarrow \text{[ey]} \]

5. X During Y
   \[ \text{[sx, sy]} \rightarrow \text{[ex]} \rightarrow \text{[ey]} \]

6. X Finishes Y
   \[ \text{[sy]} \rightarrow \text{[sx]} \rightarrow \text{[ex]} \rightarrow \text{[ey]} \]

7. X Equals Y
   \[ \text{[sx, sy]} \rightarrow \text{[ex, ey]} \]

Fig. 3. Point graph representations of temporal relations.

CASE II—X and Y both points:
X = [px] and Y = [py] with sx = px = ex and sy = ey = py

1. X Before Y
   \[ \text{[px]} \rightarrow \text{[py]} \]

2. X Equals Y
   \[ \text{[px, py]} \]

CASE III—X is a point and Y is an interval:
X = [px] and Y = [sy, ey] with sx = ex = px and sy < ey

1. X Before Y
   \[ \text{[px]} \rightarrow \text{[sy]} \rightarrow \text{[ey]} \]

2. X Starts Y
   \[ \text{[px, sy]} \rightarrow \text{[ey]} \]

3. X During Y
   \[ \text{[sy]} \rightarrow \text{[px]} \rightarrow \text{[ey]} \]

4. X Finishes Y
   \[ \text{[sy]} \rightarrow \text{[px, sy]} \rightarrow \text{[ey]} \]

5. Y Before X
   \[ \text{[py]} \rightarrow \text{[px]} \rightarrow \text{[ey]} \]

Fig. 4. Point graph representation of temporal statements in Example 4.

A path between two nodes in a PG may not be a connected chain as is the case in Example 4. However, identification of connected chains in a PG may be a desirable feature for a computer implementation of this methodology. Filtering out all the chains that are nonconnected may result in a smaller graph with no loss of information. This issue is left for a later treatment of this subject.

B. System Verification

The PG representation of a system’s temporal aspects organizes the information contained in temporal statements into a graphical structure. This section presents an analysis of this graphical representation. The analysis applies certain graph-theoretic concepts on the structure of point graphs, identifies their structural properties, and finally interprets the results obtained in terms of the temporal aspects of the systems under consideration.

As mentioned in Section II, the inference engine of PITL requires a consistent system specification in order to infer temporal relation among system intervals. A verification methodology is presented in this section, which makes use of the structural properties of the PG to detect inconsistencies, if present in the system description.

Definition 11: Inconsistency [25]: A set of statements (inferences) is said to be inconsistent if they cannot all be true at the same time.

Definition 12: Inconsistency in Point-interval Logic: A system’s description contains inconsistent information if for some intervals X and Y both $X \preceq Y$ and $X \not\preceq j$, or $X \not\preceq Y$ and $Y \not\preceq X$ (with the exception of $\preceq i$) hold true.

In the point representation of intervals, the inconsistency is defined as follows.

Definition 13: Inconsistency in Point-interval Logic: A system’s description contains inconsistent information if for some points $t_1$ and $t_2$ (representing start and/or end of intervals) the following hold true: $t_1 < t_2$ and $t_1 = t_2$ or $t_1 < t_2$ and $t_1 > t_2$.

Proposition 3: A set of temporal statements is inconsistent if and only if the PG representation of the set contains self-loops and/or cycles.

Proof: Proof of the Proposition follows from the definition of inconsistency (Definition 12) and the construction of PG’s.

Definition 14: Self-Loop: In a Petri net, a place is said to constitute a self-loop if it is both input and output place of the same transition. In a point graph an arc forms a self-loop if it originates and ends in the same node.

A consistent set of temporal statements is, therefore, characterized by Proposition 4.

Proposition 4: A set of temporal statements is consistent if and only if the PG representation of the set is an acyclical graph.

The following example illustrates the results presented in Proposition 3.

Example 5: The following is a set of inconsistent statements. The cyclic structure of the unified PG, shown in Fig. 5,
Fig. 5. Inconsistent case.

reveals the inconsistency present in the system.

1) \( W \text{ Meets } X \);
2) \( X \text{ Meets } Y \);
3) \( Y \text{ Meets } Z \);
4) \( Z \text{ Meets } W \).

The self-loops present in a PG representation can be easily identified by individually scanning all the nodes in the graph. The discussion so far has characterized inconsistency in a set of temporal statements and has illustrated the ways in which an inconsistent set of statements reveals itself in the PG representation. The following are some graph-theoretic results that help calculate the cycles inside a point graph.

**Definition 15: Marked Graph:** A Marked graph is a connected Petri net in which each place has exactly one input and one output transition.

**Proposition 5:** The underlying Petri net of a connected point graph is a marked graph.

**Proof:** The proof follows from the definition of Marked graphs and the construction of point graphs.

**Proposition 5** allows the application of following results on PG representation.

**Theorem 1** [29]: The \( S \)-components of minimal support \( S \)-invariants of a Marked graph are exactly its directed elementary circuits.

(Note: For the definitions of \( S \)-component, and minimal support, refer to [30].)

**Definition 16: \( S \)-Invariant:** Given an incidence matrix \( C \) of a Petri net, an \( S \)-invariant is a nonnegative integer vector \( X \) of the kernel of \( C^T \), i.e.,

\[
C^T X = 0,
\]

**Definition 17: Connectivity Matrix** [31]: A Point Graph with \( n \) directed arcs and \( m \) nodes can be represented by a \( (n \times m) \) matrix \( J \), the Connectivity matrix. The rows correspond to arcs, the columns correspond to nodes.

- \( J_{ij} = 1 \) if the directed arc in \( i \)-th row originates from the \( j \)-th node.
- \( J_{ij} = -1 \) if the directed arc in \( i \)-th row terminates in the \( j \)-th node.
- \( J_{ij} = 0 \) if the directed arc in \( i \)-th row is not connected to \( j \)-th node.

**Proposition 6:** The connectivity matrix of a PG is exactly the incidence matrix of its underlying Petri net.

**Proof:** Proof follows from the definitions of connectivity and incidence matrices, and the construction of a point graph from a Petri net.

Now, using results from Proposition 6 and Theorem 1, the cycles present in a PG can be identified as follows.

**Proposition 7:** A point graph contains cycles if and only if it has nonzero \( S \)-invariants calculated using the connectivity matrix.

Once cycles are detected in a PG by calculating nonzero \( S \)-invariants, the nodes responsible for these cycles can be easily identified. This will, in turn, identify intervals involved in these cycles. This information can be used to correct the system of temporal statements.

The PG model constructed for a system also helps identify the missing temporal relations among system intervals, thus provides means for elicitation of a complete specification of system’s temporal aspects. A complete system specification is characterized by the following definition.

**Definition 18: Completeness:** A system’s description is complete if for all intervals \( X \) and \( Y \), there exists a temporal relation \( R_{X \rightarrow Y} \) in \( R \), which is either provided explicitly or can be inferred through the axiomatic system.

The application of Definition 18 on the PG representation of a system’s intervals gives rise to the following definition of completeness.

**Definition 19: Completeness in the PG Representation:** A system’s description is complete if and only if the PG representing the system is an acyclical structure with

1) exactly one source node and one sink node;
2) one connected chain from the source node to the sink node that contains all the nodes in the system description.

**Example 6:** Let a system be described in terms of the following statements:

1) \( U \text{ Meets } V \);
2) \( V \text{ Meets } W \);
3) \( W \text{ Meets } X \).

The PG representing the system is shown in Fig. 6. According to Definition 19 and Proposition 8, the system specification is complete.

**Example 7:** Let a system be described in terms of the following statements:

1) \( U \text{ Meets } V \);
2) \( W \text{ Meets } V \).

The PG representing the system is shown in Fig 7.
The PG represents an incomplete system specification since no single temporal relation can be established between intervals $U$ and $W$ from the given input.

Although the PG in Example 7 does not establish a temporal relation between intervals $U$ and $W$, the application of axioms on this representation yields the only temporary relations that can hold between the two interval are “Equals” and “Finishes”—a fact also apparent in the PG representation because both $U$ and $W$ have the same end points. Introduction of any other relation between the two intervals will inevitably introduce loops in the PG representation, thus forcing inconsistencies in the system.

The TL/PN formalism, therefore, can be used to identify incompleteness and then to elicit a complete specification of a system’s temporal aspects by filling in the missing relations in a consistent manner.

Redundancy in a system of temporal statements refers to the presence of multiple copies of the same relation, which is an extension of the definition of redundancy in a set of decision rules [33].

**Definition 20: Redundancy:** A system’s description contains redundant information, if one or all of the following cases are present:

1) Multiple copies of a temporal relation, $X Ri Y$, between two intervals.
2) $X Ri Y$ both explicitly and implicitly (can be inferred by axioms) given.

Some of the redundant cases, as characterized by the definition, are automatically removed in the PG representation of the system; relations appearing as multiple arcs between two adjacent nodes are filtered out in the construction of the PG representation. A relation between two nodes $x$ and $y$ in a Point Graph, represented by an arc from $x$ to $y$, also introduces a redundancy if $y$ does not cover $x$. Example 4 presents such redundant cases. Fig. 8(a) reproduces the PG in Example 4. An equivalent representation of the PG in Fig. 8(a) is shown in 8(b), where the redundant relations among nodes are filtered out.

The redundant cases illustrated in Fig. 8 increase the size of the PG representation, and therefore require more storage than an equivalent PG representation without these cases. On the other hand, from a computational point of view, the presence of such cases may not directly affect the system performance in an adverse manner. However, an approach, although expensive in terms of computational requirements, is presented that can be used to filter out such redundancies.

The approach first constructs a virtual external node, $P_0$, in the PG representation and draws arcs from it to all the source nodes in the PG. Similarly, arcs are drawn from all the sink nodes in the PG to this external node. A connectivity matrix $J$ is then constructed for this new PG. The $S$-invariants are calculated for $J$. The calculated $S$-invariants correspond to all the directed paths from source nodes of the PG to sink nodes (assuming there are no internal loops in the PG representation). The maximal $T$-supports of the calculated $S$-invariant correspond to all the connected chains in the PG. A point graph formed by these connected chains (excluding the external node) would yield an equivalent representation of the PG without redundant arcs.

**Definition 21: T-Support:** If $X$ is an $S$-invariant, the set of input and output nodes of the arcs whose corresponding components in $X$ are strictly positive is the $T$-support of the invariant.

The $T$-support of an $S$-invariant is said to be maximal if and only if it is not contained in the $T$-support of another $S$-invariant but itself.

**C. TL/PN Inference Engine**

Once a consistent (error-free) description of a system is derived and represented in terms of a PG structure, the calculation of unknown temporal relations among system intervals and windows of interest becomes a mere search problem in the net. The advantage of the TL/PN formalism is that it not only verifies system correctness prior to any inference making, but also overcomes the combinatorial problem, discussed in Section II, associated with inferring new temporal relations.

The TL/PN inference engine (TIE) infers temporal relation between two intervals $X (= \{sx, cx\})$ and $Y (= \{sy, cy\})$ by constructing the string representation of the temporal relation between the two intervals by searching for the directed paths between the nodes representing the intervals in the PG representation and using the facts that “start of an interval” is less than or equal to “end of the interval”. The search for the directed paths between two nodes in a PG requires the use of two algorithms called FPSO and FPSI [33], two variants of an earlier FindPath algorithm [32]. The FPSO($p$) algorithm, when applied to a node $p$ in a PG, collects all the nodes that have directed paths to $p$. The FPSI($p$) algorithm, on the other hand, collects all the nodes to which $p$ has a directed path. The existing implementation of the two algorithms uses a depth-first search strategy to calculate the result. The search terminates as soon as it encounters the desired node $x$ during its search or encounters all the sink nodes without hitting the desired node. Tables I and II present the formal descriptions of the two techniques. The notation $x \gg p$, used in the tables, represents the binary relation that a directed path exists from node $x$ to node $p$.

The following example illustrates the mechanism of TIE in calculating the unknown temporal relation between intervals.

**Example 8:** Consider the system of temporal statement introduced in Example 4.

![Diagram](image-url)
The PG representing the set of temporal statements is shown in Fig. 8. The inference engine is then asked to calculate the temporal relation between the intervals \( W \) and \( Y \), represented as "?-\( R(W,Y) \)" in TIE formalism. The inference engine tries to build the string representation for \( R(W,Y) \) by first searching for the relation between points "\( cw \)" and "\( sy \)". Since \( sy \in \text{FPSI}(cw) \), the engine (TIE) calculates "\( cw < sy \)". Using the fact that "\( sw < cw \)" and "\( sy < cy \)"; the entire string representation is calculated to be

\[
\text{swVs sySwVs cy cwVs sy cwVs cy} < < < < .
\]

The string corresponds to "\( W \) Before \( Y \)"—returned by TIE.

Table III presents the general form of queries and their return values as implemented in the inference engine TIE.

The following example illustrates the inference engine’s processing for identifying an interval of interest to user.

**Example 9:** Consider the temporal system in Example 8. Let the interval representing the overlap between intervals \( U \) and \( X \) be desired to be calculated, denoted as "?-\( \text{window}(U,X) \)". The inference engine processes this query by the following algorithm:

\( \text{-window}(U,X) \)

**Step 1:** Input statements in PITL (i.e., \( X \) Before \( Y \)).

**Step 2:** Construct a unified point graph.

**Step 3:** Verify system correctness by applying the \( S \)-invariant algorithm to the connectivity matrix of the PG. Report the inconsistent cases and halt.

**Step 4:** Invoke TIE for calculating temporal relation and windows of interest.

### IV. APPLICATION

Suppose a system’s description is given as follows:

1) Drill allocated to Robot \( P \) at 06:00/Day 1.
2) Drill freed by Robot \( P \) at 11:00/Day 1.
3) Drill moved from Robot \( P \) to Robot \( Q \) at 09:00/Day 1.
4) Drill bit #1 moved from Robot \( P \) to Robot \( Q \) at 07:00/Day 1.
5) Drill bit #1 freed by Robot \( P \) at 10:00/Day 1.

**Step 1:**

The temporal aspects of the system are formally represented in terms of statements of point-interval temporal logic, PITL, given as follows.

1) \( X \) Starts \( Y \)

where \( X = [6,6] \) is a point specifying the event of allocating Drill to Robot \( P \), and \( Y \) is the interval during which Robot \( P \) is in possession of Drill.

2) \( U \) Finishes \( Z \)

where \( U = [11,11] \) is a point specifying the event of Drill being taken from Robot \( Q \), and \( Z \) is the interval during which Robot \( Q \) is in possession of Drill.

3) \( W \) Finishes \( Y \) and \( W \) Starts \( Z \)

where \( W = [9,9] \) is a point specifying the event of reallocation of Drill from Robot \( P \) to Robot \( Q \), and \( Y \) and \( Z \) are the intervals specified before.

4) \( V \) Finishes \( K \) and \( V \) Starts \( R \)

where \( V = [7,7] \) is a point specifying the event of reallocation of Drill bit #1 from Robot \( Q \) to Robot \( P \), \( K \) is the interval during which Robot \( Q \) is in possession of Drill bit #1, and \( R \) is the interval during which Robot \( P \) is in possession of Drill bit #1.

---

**TABLE I**

<table>
<thead>
<tr>
<th>Find Path-To-Sources (FPSO) Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>For a PG ( (V,E) )</td>
</tr>
<tr>
<td>( p ) an element in ( V )</td>
</tr>
<tr>
<td>( \text{FPSO}(p) = S )</td>
</tr>
<tr>
<td>where</td>
</tr>
<tr>
<td>( S = {x \mid x \in V, x &gt;&gt; p } )</td>
</tr>
<tr>
<td>( \forall x \in S, x &lt; p )</td>
</tr>
</tbody>
</table>

**TABLE II**

<table>
<thead>
<tr>
<th>Find Path-To-Sinks (FPSI) Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>For a PG ( (V,E) )</td>
</tr>
<tr>
<td>( p ) an element in ( V )</td>
</tr>
<tr>
<td>( \text{FPSI}(p) = S )</td>
</tr>
<tr>
<td>where</td>
</tr>
<tr>
<td>( S = {x \mid x \in V, p &gt;&gt; x } )</td>
</tr>
<tr>
<td>( \forall x \in S, p &lt; x )</td>
</tr>
</tbody>
</table>

---
TABLE III
QUERIES FOR CALCULATING TEMPORAL RELATIONS

<table>
<thead>
<tr>
<th>Example Queries</th>
<th>Return Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\texttt{?-R(X,Y)}$</td>
<td>$\text{X Ri Y}$</td>
<td>The inference engine returns a specific $Ri$ if it finds one, otherwise it outputs all possible relations that might exist without creating an inconsistency, given incomplete information available.</td>
</tr>
<tr>
<td>&amp; $\text{where } R$ is an element in $R \cup {\text{unknown}}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\texttt{?-X Ri Y}$</td>
<td>$\text{Yes</td>
<td>No</td>
</tr>
<tr>
<td>&amp; $\text{where } R$ is an element in $R \cup {\text{unknown}}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\texttt{?-R([p1, p2], [p3, p4])}$</td>
<td>$\text{X Ri Y}$</td>
<td>This query presents a generalized format of the first query. This syntax helps a user in constructing custom intervals by selecting arbitrary points in the PG as start and end of these user-defined intervals, with $'p1 &lt; p2'$ and $'p3 &lt; p4'$ for intervals, and $'p1 = p2'$ or $'p3 = p4'$ for points.</td>
</tr>
<tr>
<td>&amp; $\text{where } R$ is an element in $R \cup {\text{unknown}}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\texttt{?-[p1, p2] Ri [p3, p4]}$</td>
<td>$\text{Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

V. CONCLUSION

The TL/PN formalism provides a basic engine for system description and modeling. The Petri net model of the system is not merely a different representation of the same information present in the statements of point-interval temporal logic, but it provides an analytical tool for temporal reasoning, validation, verification, and calculation of windows of interest. The current approach offers the following:

1) A general, complete and sound formalism of point-interval temporal logic (PITL). This extends the approach of Allen’s temporal logic.

2) An inference engine (TIE) that infers the temporal relationship among intervals without running into combinatorial problem.

Now if it is required to know the precise period (interval) during which Drill and Drill bit #1 are both in the possession of Robot $P$—the window associated with intervals $Y$ and $R$—the inference engine is asked the following query: $\texttt{?-window(Y,R)}$. Since (S Finishes R) returns a value ‘yes’, the found window is: $[sr; ey]$ or $[ek; sr; pu; pu; ey; s2]$ in the composite representation.
3) A verification mechanism for the temporal formalism that identifies inconsistencies and incompleteness in the system of temporal statements.

4) A temporal information elicitation tool that identifies the present incompleteness in the input. It builds the system knowledge base incrementally and, therefore, new temporal statements can be added to the system at any stage of the TL/PN methodology; however, the statements that are already input cannot be deleted without restarting the whole process from the beginning.

The input statements to the methodology are all connected together with logical AND. Therefore, they represent a system’s description on a single time line. Due to incompleteness we may not map these intervals on a single line, but a complete system will have all its intervals on a single time line (a PG with one connected chain). For any point on this time line there will exist only one future.

The approach presented does not take into consideration the lengths of intervals and the actual time (clock time) of occurrence of events. An extension of the approach will incorporate these features into the methodology. The inference engine would be extended to answer queries regarding actual time of an event or length of an interval.

REFERENCES


Abbas K. Zaidi (M’90) was born in 1965 in Karachi, Pakistan. He received the B.E. degree in electrical engineering from NED University of Engineering and Technology, Karachi, in 1989, and the M.S. degree in electrical engineering and Ph.D. degree in information technology and engineering from George Mason University (GMU), Fairfax, VA, in 1991 and 1994, respectively.

He has taught at NED University, FAST Institute of Computer Science, and the University of Karachi. He has also worked as a Research Assistant Scientist at the Center of Excellence in C3I, GMU. At present, he is an Associate Professor, Department of Computer Science, Mohammad Ali Jinnah University, Karachi, and is a Consultant, Systems Architectures Laboratory, Center of Excellence in C3I, GMU. His current research interests include distributed intelligence systems, temporal control of discrete event systems, V&V of rule-based Systems, and colored Petri nets.

Dr. Zaidi was elected a full member of Sigma Xi in 1995.