Timed Spider Diagrams: a temporal visual specification language

P. Bottoni\textsuperscript{a,}\textsuperscript{*}, A. Fish\textsuperscript{b,}\textsuperscript{**}

\textsuperscript{a}Dipartimento di Informatica, “Sapienza” Università di Roma, Italy
\textsuperscript{b}School of Computing, Engineering and Mathematics, University of Brighton, UK

Abstract

Spider Diagrams (SDs) are a well-established visual language used to specify sets, their relationships, and constraints on their cardinalities. We propose Timed SDs (TSDs) as an extension enabling the expression of temporal constraints. We adopt an interval-based model of calendar time, permitting diagram elements to be specified to exist only over some interval. We introduce basic TSDs, where time constraints refer to an entire diagram rather than to individual elements, as a canonical form for TSDs, and enable the decomposition of complex TSDs into film-strip-like sequences of basic TSDs. Secondly, we introduce an innovative usage of SDs by specialising and adapting them to an OO-modelling context: in type-SDs a spider represents a type, whereas in instance-SDs a spider represents a specific object of a given type. A notion of conformance of an instance-SD to a type-SD ensues and we extend the concepts to instance-TSDs and type-TSDs. Finally, we extend the model to the application area of policy specification, introducing a notion of temporal policy, specifying the permissible states for instances of given types over a period without temporal gaps in it, and indicate when a sequence of time-annotated instances conforms to a policy.

Keywords:
Visual logic, Temporal constraint specification, Spider diagrams

\textsuperscript{*}Corresponding author: Email: bottoni@di.uniroma1.it. Tel: +39 06 4991 8426.
\textsuperscript{**}Principal corresponding author. Email: Andrew.fish@brighton.ac.uk. Tel: +44 1273 642534. Fax: +44 1273 642405.
1. Introduction

Spider Diagrams (SDs) are a diagrammatic logical specification and reasoning system built on top of Euler Diagrams (EDs). Among their many variants, one version was shown to be suitable for expressing monadic first order logic with equality statements [1]. Constraint Diagrams (CDs [2]) add expressivity to SDs (introducing explicit syntax for the expression of quantification and relations), but come with the trade-off of more complexity in interpretation [3]. CDs were proposed as a means of constraint expression within software system modelling, potentially as a replacement for the textual Object Constraint Language (OCL) within the Unified Modeling Language (UML). For example, they can be used to specify static constraints over the model such as a system invariant, or behavioural specifications in the form of pre and post-condition contracts. They can also be considered to be a modelling notation in their own right, independent of UML class diagrams for instance [4]. There are many perceived advantages of using diagrammatic constraint languages, particularly in terms of accessibility of the notation to stakeholders who may not be so comfortable with formal symbolic languages. An example of a useful feature that becomes available is the ability to display a logical proof as a sequence of diagrams, thereby enhancing confidence in the correctness of a statement (versus no proof displayed).

Prior to [5], no provision existed for modeling system evolution, as no notion of dynamics was incorporated within these diagrammatic languages. Indeed, time is not a primitive notion in SDs, so that time-dependent specifications are not directly definable. Timed SDs incorporate temporal constraints by annotating diagrams with interval specifications, with reference to a time model based on calendars. We specialise SDs for object-oriented modeling, restricting the interpretation of sets to denote collections of typed elements in a given state, and distinguishing between type-SDs and instance-SDs. In a type-SD, a spider represents a type and its habitat defines the permissible states for their instances (via the conformance relationship). In an instance-SD, a spider represents an instance and its habitat describes its current state. An instance-SD conforms to a type-SD if no spider is in a state not permissible according to the type-SD. We bring together the two extensions of SDs, and integrate them with a formal notion of policy.
1.1. Paper Overview

We provide a detailed overview of the paper contents, indicating the results and dependencies of material, enabling a selective reading, together with statements about the papers [5, 6] that we combine and extend.

Section 2 provides a formalised general form of Spider Diagrams (SDs). The very general form of SDs (Definition 2.1) improves on some of the presentations in the literature, retaining more intuition whilst being formal and permitting a variety of syntax (e.g. existential and constant spiders, and ties and strands), enabling specialisation to any restricted cases of interest; the interpretation and models for SDs are presented in Definitions 2.2-2.3.

The time model is described in Section 3, independent of SDs, enabling its usage with alternative constraint languages. Section 3.1 deals with intervals and their relations (Definitions 3.1 and 3.2) in a calendar based model, presenting a terminating, confluent rewriting system (Theorem 3.1) that enables the production of a non-overlapping cover of a given set of intervals (Definition 3.3 and Theorem 3.2). In Section 3.2, we introduce interval specifications, allowing variables and constraints on expressions, assignments and valuations of sets of interval specifications (Definitions 3.5 and 3.6), and the notion of satisfiability of interval specifications (Definition 3.7). The notion of non-overlapping covers lifts to intervals specifications in Theorem 3.3. This refines and extends the basic ideas presented in [5].

We present the Timed SD (TSD) system in Section 4. TSDs (Definition 4.1) consist of SDs with intervals specifications assigned to syntactic elements of the diagrams providing a timeline for their existence. These can provide complex temporal constraints within a single diagram, but we can translate them into sequences of basic timed SDs (Definition 4.2) which assign interval specifications to entire diagrams, as shown in Theorem 4.1, making use of the decomposition presented in Theorem 3.3.

In Section 5 we provide a specialisation and adaptation of SDs to an OO-modelling domain giving formal definitions of type-SD and instance-SDs, and a notion of the conformance of an instance-SD to a type-SD; collectively these diagrams are called Modelling SDs (MSDs). Theorem 5.1 enables a syntactic conformance check, making use of the notion of diagram projection (Definition 5.4). In [5], the intent of type-SDs and instance-SDs (previously named specification-SDs and snapshot-SDs) were presented without detail and results. We extend them to type-TSDs and instance-TSDs. The semantics (Definition 5.6) of a type-TSD is given as the set of all sequences of basic instance-SDs that conform to the type-SD, making use of the translation in
Theorem 4.1, and we see that consistent TSDs have non empty semantics in Theorem 5.2. This is a significantly improved version of the ideas in [5].

The use of TSDs within the OO-modelling context allows us to propose a notion of temporal policies, presented in Section 6. Policies (Definition 6.1) consist of a period of validity, a trigger, and a condition that must be satisfied for the policy to hold, together with examples of use (refined from the short paper [6]). Section 6.1 provides new semantics for policies, starting with the notion of an activator (Definition 6.3) firing the trigger of the policy for a sequence of instance SDs. Given an activator, Construction 6.1 computes the maximal time period over which the policy can be in force for the sequence, and then Definitions 6.4 and 6.5 define what it means for a sequence to be subject to and conform to a policy. The semantics of a policy is the set of sequences of basic timed instance-SDs that conform to it and Theorem 6.1 indicates that every policy is satisfiable. Sections 7 and 8 discuss related work and draw conclusions.

2. Spider Diagrams

We provide a detailed formalisation of the syntax and semantics of a general form of SDs. This system is more general than most occurrences of SDs in the literature, improves on existing formalisations or informal descriptions, specialising to different existing systems in the literature. First, we describe the concrete presentation of SDs, as distinct from the abstract syntax.

A (concrete, unitary) Euler diagram (ED) is a collection of (labelled) simple closed curves in the plane, called contours, decomposing it into connected minimal regions. A zone is a region inside one set of contours and outside all the remaining contours; zones may be shaded. We adopt the convention that all diagrams have a “boundary contour”, drawn as a rectangle and labelled by $U$; all regions are inside $U$. The intended semantics of EDs are that the interior of the curves represent sets, and this extends naturally to the intersection, containment and disjointness of sets, whilst shading is an additional means to specify that a set is empty.

A (concrete, unitary) SD is an ED together with extra syntax for spiders, i.e. trees whose vertices (called feet) are placed in zones, with no two vertices of the same tree lying within the same zone. The spiders are either constant (labelled, with square feet) or existential (unlabelled, with round feet). A pair of spiders’ feet within a zone can be joined by a tie (two parallel lines, similar to an equality sign) or a strand (a wiggly line). The intended semantics are
that: each spider represents either a specific individual (for a constant) or the existence of an element (for an existential), within the set determined by the region in which the spider has its feet (called its habitat); if two distinct spiders have feet within a common zone and the spiders represent elements within the set determined by that zone, then these elements are equal if the feet are connected by tie, and disjoint if the feet have no strand or tie connecting them; the only elements in the set represented by a shaded zone are those represented by spiders.

The left hand diagram in Figure 1 shows an SD with existential spiders, strands, ties and shading; removing the spiders, strands and ties leaves an ED. The diagram represents the following constraints: sets $A$ and $B$ are disjoint from set $C$ (since the interiors of their curves are disjoint); $|C| = 1$ due to the single footed spider and the shading within the zone that is inside $U$ and $C$ and outside $A$ and $B$; there are at least two elements in $A \cup B$ with at least one of them in $B \setminus A$ (where \ denotes set difference). This last constraint is deduced as follows: each of the three spiders within the region for $A \cup B$ indicates the existence of an element in the set corresponding to its habitat; the tie in the zone for $A \cap B$ means that if both of the involved spiders represent elements in $A \cap B$ then those elements are equal; the strand in the zone for $B \setminus A$ means that if both of the involved spiders represent elements in $B \setminus A$ then those elements may or may not be equal (if the strand was omitted then we would have that the elements are not equal).

The right hand diagram in Figure 1 shows an SD for use within a modelling context, having four curves, eight non-shaded zones, and a single con-
stant spider, labelled *Alice*, which inhabits two zones (inside of the curves *U* and *BritishCitizen* and outside of the curve *BornInUK*). Intuitively, *Alice* represents an entity which belongs to the set *BritishCitizen*, may also belong to the set *OfBritishDescent*, but does not belong to the set *BornInUK*.

More complex constraints can be expressed by permitting diagrams to be combined using standard logical connectives (conjunction, disjunction, negation). The abstract syntax of SDs is defined with reference to a set of labels; we first provide a notion of abstract zones (these are potential zones within a diagram with the correct set of curves) enabling interpretations of zones over a collection of diagrams, and then present the definition of SDs.

Let $L$ denote a fixed, countably infinite set of *labels* consisting of disjoint sets $C$ and $S$, such that: $C$ contains a designated label $U$; $S$ consists of disjoint sets $S_c$ and $S_e$. A pair $(X, Y)$, with $X, Y \subseteq P(C)$ disjoint, determines an abstract zone, and a region is a non-empty set of zones. We denote by $Z$ and $R$ the sets of all zones and regions, respectively. The symbols $P(X)$ and $P^2(X)$ denote the powerset of a given set $X$ and the set of subsets of cardinality 2 of $X$, respectively. We write $\tau \cap \upsilon$ and $\tau \cup \upsilon$ to denote the intersection and union, respectively, of two relations $\tau$ and $\upsilon$.

**Definition 2.1 (Spider Diagram).** A unitary spider diagram $d$ is a tuple $(C, Z, sh, S, h, \tau, \upsilon)$ such that:

- $C = C(d) \subseteq C$ is a finite set of curve labels with boundary curve label $U \in C$.
- $Z = Z(d) \subseteq \{(X, C \setminus X) \mid X \subseteq C, U \in X\}$ is a finite set of zones, s.t. $(\{U\}, C \setminus \{U\}) \in Z$, and $\forall c \in C \exists X \subseteq C[c \in X \land z = (X, C \setminus X) \in Z]$. If $z = (X, C \setminus X) \notin Z$ for some $X \subseteq C$, then $z$ is a missing zone of $d$. We denote the set of missing zones of $d$ by $Z_M(d)$.
- $sh : Z \to \mathbb{B}$ is a Boolean shading function on zones. A zone $z$ for which $sh(z) = \text{true}$ is said to be shaded. Let $Z^* = Z^*(d) = \{z \in Z \mid sh(z) = \text{true}\}$.
- $S = S(d) \subseteq S$ is a finite set of spider labels, comprised of constant spider labels $S_c \subseteq S_c$ and existential spider labels $S_e \subseteq S_e$.
- $h$ is the habitat function $h : S \to P(Z) \setminus \{\emptyset\}$ recording the set of zones that each spider touches. Each unique pair $(s, z) \in S \times Z$ s.t. $z \in h(s)$ is called a foot of $s$. Let $F = F(d)$ denote the set of all feet in $d$. 6
• \( \tau = \tau(d), \upsilon = \upsilon(d) \subseteq P^2(S) \times Z \) are two disjoint relations between pairs of spiders and zones which indicate if the spiders are connected by a tie or a strand within that zone, respectively. For every \( \{s_1, s_2\}, z \in \tau \cup \upsilon \) we have \( z \in h(s_1) \cap h(s_2) \).

A spider diagram (SD) is defined inductively: a unitary spider diagram is a spider diagram; if \( d_1 \) and \( d_2 \) are spider diagrams then so are \( \neg d_1, d_1 \land d_2 \) and \( d_1 \lor d_2 \). The collection of all SDs is denoted \( SD \).

Remark 1. The existential spiders (usually unlabelled) are provided with an identifier which is interpreted in a different way to the label of constant spiders. This provides us with the ability to reference the individual spiders and to distinguish between the two types of spiders. Constant spiders are interpreted as constants in the language, thereby binding together any two occurrences of a constant spider label in an interpretation, whereas existential spiders are simple constraints whose labelling is only a means of identification and thus these are interpreted within the context of a single diagram.

Definitions 2.2 and 2.3 formalise the intuitive semantics, mentioned earlier. Interpretations provide a universe of discourse and a function which interprets curve labels and constant spider labels. This enables a consistent interpretation of these labels across a collection of diagrams. Then, an interpretation is a model for a diagram if it satisfies the semantics predicate, encapsulating the set of intended constraints imposed by the diagram.

Definition 2.2 (Interpretation). An interpretation is a pair \((U, \psi)\), where the set \( U \) is called the Universe of Discourse and the interpretation function \( \psi : C \cup S_c \rightarrow P(U) \) is such that \( \psi(U) = U \) and \( |\psi(s)| = 1 \) for every \( s \in S_c \). The domain of \( \psi \) extends to \( Z \cup R \) according to the following:

- For every \( z = (X, Y) \in Z \) we have
  \[
  \psi(z) = \bigcap_{c \in X} \psi(c) \cap \bigcap_{c' \in Y} (U \setminus \psi(c')) ,
  \]
  where by convention the second term evaluates to \( U \) if \( Y = \emptyset \).
- For every \( r \in R \) we have \( \psi(r) = \bigcup_{z \in r} \psi(z) \).
Definition 2.3 (Model). Let \( d \) be an SD. An interpretation \( m = (U, \psi) \) is a model for \( d \) if \( P_d(m) \), called the semantics predicate of \( d \), evaluates to \text{true}, where \( P_d(m) \) is defined as follows:

If \( d \) is a unitary SD then \( P_d(m) \) is the conjunction of the following two conditions.

1. Missing zones represent the empty set: \( z \in Z_M(d) \implies \psi(z) = \emptyset \).
2. There exists an extension of the domain of \( \psi \) to \( S = S(d) \), so that:
   - spiders represent singleton element sets: \( s \in S \implies |\psi(s)| = 1 \).
   - spiders represent sets contained within the interpretation of their habitat: \( s \in S \implies \psi(s) \subseteq \bigcup_{z \in h(s)} \psi(z) \).
   - if spiders represent (singleton sets of) elements within the set represented by a zone then these singleton sets are equal if the spiders were joined by a tie and not equal if they are not joined by either a tie or a strand:
     - if \( \psi(s_1), \psi(s_2) \subseteq \psi(z) \) for \( s_1, s_2 \in S \) and \( z \in Z \) then
       - \( \{s_1, s_2\}, z \in \tau \implies \psi(s_1) = \psi(s_2) \).
       - \( \{s_1, s_2\}, z \notin \tau \cup \nu \implies \psi(s_1) \neq \psi(s_2) \).
   - shading in a zone indicates that the only elements in the set represented by the zone are those represented by the spiders touching that zone: \( (z \in Z^* \text{ and } x \in \psi(z)) \implies x \in \psi(s) \) for some \( s \in S \) with \( z \in h(s) \).

Furthermore,

- If \( d = \neg d_1 \) then \( P_d(m) = \neg P_{d_1}(m) \).
- If \( d = d_1 \land d_2 \) then \( P_d(m) = P_{d_1}(m) \land P_{d_2}(m) \).
- If \( d = d_1 \lor d_2 \) then \( P_d(m) = P_{d_1}(m) \lor P_{d_2}(m) \).

Remark 2. Different logical systems can be obtained with the same syntax by modifications of the semantics predicate.

We partially specify an interpretation of curves and constant spider labels and then use it to consider models for \( d_1 \) and \( d_2 \), the left and right hand diagrams in Figure 1, respectively. Let \( U = \mathbb{Z} \), so that \( \psi(U) = \mathbb{Z} \). Let \( \psi(\text{BritishCitizen}) = 2\mathbb{Z} \) (all even integers), \( \psi(\text{OfBritishDescent}) = 3\mathbb{Z} \),
ψ(BornInUK) = 5\mathbb{Z}, \psi(A) = \{0, 1\}, \psi(B) = \{1, 2, 3\} and \psi(C) = \{4\}. Then \psi(\{U, OfBritishDescent, BritishCitizen\}, \{BornInUK\}) = 6\mathbb{Z} \setminus 5\mathbb{Z} (i.e. all integer multiples of 2 and 3 that are not multiples of 5), is an example of the interpretation of an abstract zone. We have that \( (\mathcal{U}, \psi) \) is a model for \( d_1 \); observe that spiders that touch a non-shaded zone indicate the minimal number of elements in the corresponding set (taking into account their habitats and the spider feet relationships), but there can be more elements in that set in any model. If instead we had taken \( \psi(B) = \emptyset \) in the interpretation then \((\mathcal{U}, \psi)\) would not have been a model for \( d_1 \) due to the presence of the single-footed spider in \( B \setminus A \). Now suppose further that \( \psi(Alice) = \{7\} \). Then \((\mathcal{U}, \psi)\) is not a model for \( d_2 \). However, if we had that \( \psi(Alice) = \{6\} \) then \((\mathcal{U}, \psi)\) is a model for \( d_2 \). Within a modelling context one may wish to take \( \mathcal{U} \) to consist of the world population, and then models for the diagram indicate that a specific person, identified by the label Alice, belongs to the set of British Citizen(s), is either of British descent or not, but is not born in the UK. The absence of shading indicates that we are not saying anything about people other than Alice.

Currently, there is no means to specify temporal information with SDs. The following citizenship example demonstrates the need for introducing a temporal extension. A person’s citizenship may change over time due to historical, legal, or personal reasons. A case in point is that of the inhabitants of South Tyrol, the German-Ladin speaking region of Italy. Until the end of World War I, these were members of the Austro-Hungarian Empire; then they became inhabitants of Italy, but towards the end of World War II the region was annexed by Hitler’s Third Reich. At the fall of Italian Fascism Italian rule was restored. The vicissitudes with respect to citizenship of Silvius Magnago, a prominent South Tyrol political leader, can be portrayed as in Figure 2, by associating temporal information with the presence of a foot in a zone over a certain interval. The spider for Magnago is constrained to...
exist between his birth and death dates (as indicated by the time annotation of the spider label), and one could be a citizen of the Third Reich only during a restricted time period (as indicated by the time annotation of the corresponding curve).

Remark 3. We view the time annotation as a specification of the temporality of diagram elements, and so we can consider a single diagram to be evolving as time progresses with diagram elements being born or dying. If the temporal annotations on a spiders’ feet do not overlap, one can interpret the spider as precisely identifying which set the corresponding element (e.g. the person SilviusMagnago in the previous example) is in during all of the specified times.

In the following sections we formalise the notion of temporal annotations, extending SDs to permit additional temporal specification and reasoning.

3. A model of time

Although alternative time-models are possible, and could be incorporated with SDs, we consider a discrete time model, aiming towards policy specification. As such we adopt an interval-based model of time, where intervals have finite duration. We introduce the notions of interval, defined by its inception and conclusion, and interval specifications, defining sets of intervals through constraints on their inceptions and/or conclusions. These will be used to specify the periods over which the elements of a timed-SD will be interpreted, permitting us to consider situations in which constraints expressed by SDs are in force over some time period.

3.1. Intervals

We assume that access to time information is given only through a primitive operator \texttt{NOW} returning a timestamp with every invocation, called \textit{time observation}. Timestamps are linearly ordered in such a way that if a time observation returns a timestamp $t_1$, any subsequent observation returns a timestamp $t_2$ with $t_2 \geq t_1$. However, we make an assumption of \textit{time progression}: for any sufficiently long sequence $(t_1, \ldots, t_k)$ of timestamps, produced by distinct consecutive time observations, we have $t_k > t_1$. Intervals are introduced with Definition 3.1. Let $\mathcal{T}$ denote the set of all timestamps.
Definition 3.1 (Interval). Given two timestamps $t_1$ and $t_2$, with $t_1 \leq t_2$, the time interval (or simply interval) $[t_1, t_2]$ represents the ordered set of timestamps which can be returned at any intermediate time observation, inclusive of $t_1$ and $t_2$. The timestamp $t_1$ is called the inception of the interval and $t_2$ its conclusion. The set of all time intervals is denoted by $I$.

We use timestamps structured according to calendars at varying levels of granularity and fix the lowest manageable granule (or unit) to be the second for the purposes of this paper (other granularities can be taken), so they are represented as $YYYY:MM:DD:hh:mm:ss$. The set of operators $Op = \{year, month, day, hour, minute, second\}$ provides cuts of the timestamp up to the relevant granule. For example, the operator $day$ yields the current year, month and day, whilst $second$ yields the complete current timestamp. We will use a variable $u$, ranging over the set of abbreviations $G = \{Y, M, D, h, m, s\}$, to indicate both a granularity level and the corresponding operator. The duration of an interval is the number of units which are spent between its inception and its conclusion.

As a notational shortcut, we may annotate intervals with the indication of the relevant granularity level and only consider a prefix of the timestamp up to that granularity. For example, $[2011 : 01 : 01, 2011 : 01 : 31]_D$ and $[2011 : 01 : 01, 2011 : 01 : 31 : 23]_h$ both encompass all the timestamps that occurred in January 2011. On the other hand, the notation $[2011 : 01, 2011 : 01 : 31]_Y$ would be incorrect, as only the prefix 2011 should appear in this case. We also make a requirement of unit significance, to rule out notations such as $[2011 : 01 : 01, 2011 : 01 : 31]_h$, where the granule associated with the interval is finer than those appearing in the interval. An interval $[t_1, t_1]_u$ is considered to have unitary duration with respect to $u$, as it represents the whole period during which each time observation will return a timestamp with the same prefix $u(t_1)$. Intervals may contain time expressions according to the syntax of equation (1), where $ArithmeticExp$ is an arithmetical expression involving constants and the operators $\{+, -, \times\}$.

\[
TimeExp := \text{timestamp} | TimeExp \text{ op } ArithmeticExp
\]

\[
op := \oplus | \ominus
\]  

(1)

Each constant represents a number of units, and hence so does an arithmetic expression. The operator $\oplus$ indicates that the whole duration of the number of units resulting from the arithmetic expression has to be added to the value of a time expression, to obtain the corresponding timestamp, at
the indicated level of granularity. Similarly, $\ominus$ indicates that the duration given by the expression on the right has to be subtracted from the value of the expression on the left. For an interval $[e_1, e_2]$, where $e_1$ and $e_2$ are time expressions, each $e_i$ must contain a single explicit timestamp $t_i$, for $i \in \{1, 2\}$.

For example $[2011 : 01 : 01, 2011 : 01 : 02 \oplus 30]_D$, $[2011 : 01 : 01 : 00, 2011 : 01 : 02 : 00 \ominus 720]_h$, and $[2011 : 01 : 01 : 00, 2011 : 01 : 02 : 00 \ominus (30 \times 24)]_h$ are alternative ways to indicate the interval corresponding to the month of January 2011. The values 30 or 720 above refer to the fact that during the whole of 30 days (or equivalently 720 hours), after the inception of January 2nd, 2011 each time observation will return timestamps with prefix 2011 : 01.

A calculus on calendar intervals can be introduced, following Allen's interval algebra [7], as shown in Definition 3.2, with some simple consequences discussed in Lemma 3.1 and Fact 3.1.

**Definition 3.2 (Interval relations).** Let $H^1 = [t^1_1, t^1_2], H^2 = [t^2_1, t^2_2] \in \mathcal{I}$. We define the following directed relationships:

- $H^1$ meets $H^2$, denoted $H^1 \prec H^2$, iff $t^1_1 = t^2_2 \oplus 1$.
- $H^1$ during $H^2$, denoted $H^1 \triangleright H^2$, iff $t^1_1 \geq t^2_1 \land t^1_2 \leq t^2_2$.
- $H^1$ overlaps $H^2$, denoted $H^1 \bowtie H^2$, iff $t^1_1 < t^2_1 \land t^1_2 \geq t^2_1 \land t^2_2 < t^2_2$.

If any of these relations (or their inverses) hold then the merge of $H^1$ and $H^2$, denoted $H^1 \odot H^2$, is the interval $[\min(t^1_1, t^2_1), \max(t^1_2, t^2_2)]$. Given a finite set of intervals $\mathcal{H} = \{H^1, \ldots, H^k\}$, with $H^i = [t^i_1, t^i_2]$, their merge exists if there exists a sequence $(H^{i_1}, \ldots, H^{i_k})$, with each $i_j \in \{1, \ldots, k\}$ distinct, s.t. $(\ldots (((H^{i_1} \odot H^{i_2}) \odot H^{i_3}) \cdots \odot H^{i_k}) \cdots) \exists$ and is equal to $[\min_{i \in \{1, \ldots, k\}}\{t^i_1\}, \max_{j \in \{1, \ldots, k\}}\{t^j_2\}]$. This is called the merge of $\mathcal{H}$ and is denoted by $\odot(\mathcal{H})$.

If $H^1 \bowtie H^2$ or $H^1 \bowtie H^2$ then their overlapping subinterval, $H^3$, is given by $H^3 = H^1 \bowtie H^3$ or $H^3 = [t^1_1, t^1_2]$, respectively. Given a set of intervals $\mathcal{H}$ the overlapping subinterval of $\mathcal{H}$ is the maximal interval $H$ s.t. for any two intervals $H^q, H^r \in \mathcal{H}$, we have $H \bowtie H^q \land H \bowtie H^r$, if such a $H$ exists, and is $\emptyset$ otherwise.

If none of meets, during or overlap (or their inverses) hold, then we say that $H^1$ and $H^2$ are disjoint, with $H^1 < H^2$ if $t^2_1 < t^1_2$ and $H^1 > H^2$ if $t^2_2 < t^1_1$.

---

1The inverse relations are obtained by inverting the roles of $H^1$ and $H^2$. Here we do not consider the sub-cases finishes, starts or equal, of the during relationship from [7].
**Lemma 3.1.** Given a finite set of intervals $\mathcal{H} = \{H^1, \ldots, H^k\}$, with $H^i = [t^i_1, t^i_2]$, we can merge $\mathcal{H}$ iff for any pair $H^i, H^j \in \mathcal{H}$ s.t. $H^i \prec H^j$, there is at least one interval $H \in \mathcal{H}$ s.t. $[t^i_2 \oplus 1, t^j_2 \oplus 1] \bowtie H$.

**Fact 3.1.** The relation $\bowtie$ is reflexive and transitive. If $H^1 \bowtie H^2$ or $H^1 \bowtie H^2$ then their overlapping subinterval $H^3$ satisfies: $H^3 \bowtie H^1$, $H_3 \bowtie H^2$ and $H_3 \bowtie H^1 \bowtie H^2$.

**Example 3.1.** For $H^1 = [2011 : 07 : 17 : 10 : 38 : 05, 2011 : 07 : 17 : 10 : 59 : 59]$ and $H^2 = [2011 : 07 : 17 : 11 : 00 : 00, 2011 : 07 : 17 : 12 : 59 : 59]$, we have $H^1 \prec H^2$ and $H^1 \bowtie H^2 = [2011 : 07 : 17 : 10 : 38 : 05, 2011 : 07 : 17 : 12 : 59 : 59]$. For $H^3 = [2011 : 07 : 17 : 10 : 30 : 05, 2011 : 07 : 17 : 12 : 00 : 00]$, we have $H^1 \bowtie H^3$ and $H^3 \bowtie H^2$. For $H^4 = [2011 : 07 : 17 : 12, 2011 : 07 : 17 : 13]$ and $H^2 \bowtie H^4$, while $H^2 \bowtie H^4$. As $\mathcal{H} = \{H^1, \ldots, H^4\}$ satisfies the hypothesis of Lemma 3.1, $\bowtie(\mathcal{H})$ exists and is equal to $[2011 : 07 : 17 : 10 : 30 : 05, 2011 : 07 : 17 : 13 : 00 : 00]$. In general, any two intervals of the form $[t_1, 2011 : 01 : 31]_D$ and $[2011 : 02 : 01, t_2]_D$ meet.

If two intervals $H^1$ and $H^2$ are such that $H^1 \bowtie H^2$ or $H^1 \bowtie H^2$ then we can consider the natural decomposition of $\{H^1, H^2\}$ into a set of intervals including their overlapping subinterval. Such a set has cardinality between one and three, depending on whether some of the inceptions and conclusions of $H^1$ and $H^2$ are equal. Given a finite set of intervals $\mathcal{H}$, we use the rules from the set rewriting system $\mathcal{R}$ of Table 1 to produce the decomposition of $\mathcal{H}$ into a set $\mathcal{J}$ of intervals, no pair of which satisfy overlap or during. If $\bowtie(\mathcal{H})$ exists then so does $\bowtie(\mathcal{J})$ and these are equal, and so $\mathcal{J}$ can be considered as a sequence of intervals in which consecutive pairs meet.

| Table 1: Rewriting rules for pairs of intervals satisfying the overlap or during relationships. |
| For $H^1 = [t^1_1, t^1_2]$ and $H^2 = [t^2_1, t^2_2]$, we have: |
| $R_1$: $(\{H^1, H^2\}, \langle H^1 \bowtie H^2 \rangle) \rightarrow \{[t^1_1, t^2_1 \ominus 1], [t^2_1, t^1_2], [t^2_1 \oplus 1, t^2_2]\}$ |
| $R_2$: $(\{H^1, H^2\}, \langle H^2 \bowtie H^1 \rangle) \rightarrow \{[t^1_1, t^1_2 \ominus 1], [t^2_1, t^2_2], [t^2_1 \oplus 1, t^2_2]\}$ |

Assuming no equality amongst inceptions and conclusions, the application of a rule $R_i \in \mathcal{R}$ has the effect of taking $\overline{H} = \{H^1, H^2\} \subseteq H$, with $H^1 = [t^1_1, t^1_2]$ and $H^2 = [t^2_1, t^2_2]$ in the relation overlap or during, and replacing it with $\overline{J} = \{J^1, J^2, J^3\}$ with $J^1 \prec J^2 \prec J^3$. However, we do allow equality amongst
inceptions and conclusions, simply removing any ill-formed intervals that appear in the right hand side upon the application of the rule. For example, if $H^2 \bowtie H^1$ and $t^1_2 = t^2_2$, then $[t^2_2 \oplus 1, t^2_2]$ is ill-formed since $t^2_2 \oplus 1 > t^1_2$, and so we would obtain $J = \{[t^1_1, t^2_2 \oplus 1], [t^1_2, t^2_2]\}$ in this case. We present the rules in this manner in order to reduce the number of cases, and we assume that if intervals are annotated with a unit then it is the same unit. Figure 3 shows a graphical representation of the action of rule $R_1$ in the case that $t^1_2 = t^2_2$; here we obtain $J = \{[t^1_1, t^2_2 \ominus 1], [t^1_2, t^2_2], [t^1_2 \oplus 1, t^2_2]\}$.

Figure 3: A graphical representation of the action of rule $R_1$ in the case that $t^1_2 = t^2_2$.

The relation between the sets of intervals in rule $R_i$ is denoted by $H \xrightarrow{R_i} J$ and the application of $\mathcal{R}$ to a set of intervals $H$ is specified formally through the following set rewriting relation: Given two sets of intervals $H$ and $J$, we have $H \xrightarrow{\mathcal{R}} J$ iff there exist three sets of intervals $H$, $J$, and $Q$, with $Q \cap H = \emptyset$, and a rule $R_i \in \mathcal{R}$ such that: $H = Q \cup H$, $J = Q \cup J$, and $H \xrightarrow{R_i} J$. The reflexive and transitive closure of the rewriting relation is denoted by $\mathcal{R}^*$. Given a finite set of intervals, the iteration of $\mathcal{R}$ produces a unique collection of intervals, as a consequence of Theorem 3.1.

**Theorem 3.1 (Confluence).** The system $\mathcal{R}$ is terminating and confluent.

**Proof.** The system $\mathcal{R}$ is well-defined since the conditions in the rules are mutually exclusive and exhaustive of relations between non-identical, non-meeting, non-disjoint intervals. To prove that $\mathcal{R}$ is terminating we observe that for each rule application $\bigcirc(J) = \bigcirc(H)$ (viz. $[t^1_1, max(t^1_2, t^2_2)]$). Moreover, if the rule application to the two original intervals produces two intervals, then one is the same size as the original, and the other is strictly smaller; if it produces three intervals, then one of these is the overlapping
interval, $J^2$, and either: (i) $J^2$ is the same duration as $H^2$ (w.l.o.g.) and the other two intervals both have strictly shorter duration than $H^1$, or (ii) $J^2$ is of shorter duration than both $H^1$ and $H^2$, and we have that $J^1$ and $J^3$ are of strictly shorter duration than $H^1$ and $H^2$, respectively. Since the duration of any interval is at least 1, the process must terminate.

To prove confluence we first prove local confluence by applying the Critical Pairs Theorem, i.e. we show that all critical pairs are convergent. In this context, a critical pair is a pair of sets of intervals $\{H^1, H^2\}$, such that for both $\{H^1, H^2\}$ and $\{H^2, H^3\}$ there is some applicable rule in $R$. This can only happen if $H^2$ has overlapping subintervals with both $H^1$ and $H^3$ (where both subintervals can be the whole of $H^2$). By an exhaustive case analysis, one observes that by starting with either of the two conflicting applications, one terminates producing the same set of intervals, and so $R$ is locally confluent. The case $H^1 \propto H^2$, $H^2 \propto H^3$, $H^1 < H^3$ is shown graphically in Figure 4; details for the other cases are omitted. Since $R$ is also terminating it is confluent by Newman’s Lemma [8].

Each rule produces a non-overlapping cover for the pair of intervals in $H$. Definition 3.3 and Theorem 3.2 generalise this notion to sets of intervals.

**Definition 3.3 (Non-overlapping cover).** Let $\mathcal{H} = \{H^1, \ldots, H^n\}$ be a finite set of intervals and let $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$ be a partition of $\mathcal{H}$ whose
partite sets $Q_i$ are maximal with respect to the property that $\bigodot(Q_i)$ exists. A non overlapping cover of $Q_i$ is a finite set of intervals $J^i = \{J^1, \ldots, J^p\}$, with $J^k = [t^k_1, t^k_2]$, s.t. 1) $\bigodot(Q_i) = \bigodot(J^i)$; 2) for each $k \in \{1, \ldots, p - 1\}$, $J^k \prec J^{k+1}$ (i.e. $t^k_1 = t^k_2 \oplus 1$). A non overlapping cover of $H$ is the union of the non overlapping covers of each of the $Q_i$.

**Theorem 3.2 (Canonical non overlapping cover).** Let $H$ be a set of intervals. Then $R$ applied to $H$ produces a non overlapping cover. Such a cover is called canonical.

**Proof.** If $\bigodot(H)$ exists, then for each interval $H^1 \in H$, there is at least one interval $H^2$ s.t. either $H^1 \prec H^2$ or exactly one of the rules in $R$ is applicable to $H = \{H^1, H^2\}$, and its application generates a set $J$ which is a canonical non overlapping cover of $\bigodot(H)$. The process can now be iterated, by applying $R$ to $(H \setminus \bigodot(H)) \cup J$. From Theorem 3.1, the process terminates only when no rule is further applicable, and since only non-overlapping covers of subsets of $H$ can be created, the final result is a non-overlapping cover of $H$. In the general case, $H$ partitions into the partite sets $Q_i$, as in Definition 3.3. Then $R$ applies only to pairs of intervals within each of the $Q_i$, and it produces a non-overlapping cover for each of these. The union of these is a non-overlapping of $H$. $\square$

**Fact 3.2.** The set of timestamps appearing in the intervals of a set $J$ which is a canonical non overlapping cover of a set $H$ is such that: the set of inceptions and conclusions for $J$ contains all of inceptions and conclusions for $H$, plus at most timestamps of the form $t^k_1 \ominus 1$, $t^k_1 \oplus 1$, $t^k_2 \ominus 1$, $t^k_2 \oplus 1$ for $H_k = [t^k_1, t^k_2] \in H$, excluding $\min_{i \in \{1, \ldots, k\}} \{t^i_1\} \ominus 1$ and $\max_{j \in \{1, \ldots, k\}} \{t^j_2\} \oplus 1$. This property defines a necessary condition which can be used as a static check of non canonicity of some cover.


Intervals can be combined into *interval expressions*, according to the syntax of equation (2), to be interpreted as denoting sets of timestamps, where *interval* is a time interval, \(\sqcup\) gives the union of the sets denoted by the two expressions, and \(\cap\) their intersection. As an example, in Figure 2 the operator \(\sqcup\) is used to indicate the union of the two intervals during which Magnago was an Italian citizen.

\[
\text{IntExp} := \text{interval} \mid \text{IntExp} \sqcup \text{IntExp} \mid \text{IntExp} \cap \text{IntExp} \quad (2)
\]

### 3.2. Interval specifications

We need a notion of interval specification, presented in Definition 3.5. First, we extend time expressions to *time expressions with variables*, according to the syntax in equation (3); here \(\text{timevar}\) and \(\text{numvar}\) are variables ranging over timestamps and natural numbers, respectively, \(\text{prefop}\) is one of the unit operators in \(G\), \(\text{VarArithExp}\) is an expression over natural numbers and numerical variables and \(\text{op}\in\{\oplus,\ominus\}\).

\[
\text{VarTimeExp} := \text{StartVarExp} \mid \text{StartVarExp} \oplus \text{RestVarExp}
\]

\[
\text{StartVarExp} := \text{timestamp} \mid \text{timevar} \mid \text{prefop}(\text{VarTimeExp})
\]

\[
\text{RestVarExp} := \text{VarArithExp} \mid \text{numvar} \text{ op} \text{ RestVarExp}
\]

A *linear temporal constraint* has the form \(x \leq k\), or \(x < k\), with \(x\) a temporal variable and \(k\) a temporal expression. A *linear integer constraint* has the form \(a_1 \leq a_2\), or \(a_1 < a_2\) with \(a_1\) and \(a_2\) arithmetical expressions according to the syntax in equation (4).

\[
\text{ArithExp} := \text{integer} \mid \text{numvar} \mid \text{ArithExp} \text{ addop} \text{ MulExp}
\]

\[
\text{MulExp} := \text{integer} \times \text{ArithExp} \mid \text{ArithExp}
\]

\[
\text{addop} := + \mid -
\]

Let \(\mathcal{E}\) denote the set of all time expression with variables.
Definition 3.4 (Specification expression). A specification expression is defined by the following, with $e_1, e_2 \in E$:

$$\text{specExp} = [e_1, e_2] \mid \text{specExp} \cup \text{specExp} \mid \text{specExp} \cap \text{specExp}. \quad (5)$$

We call $SE$ the set of all specification expressions.

For $e \in E$, let $\text{Var}(e) = \text{NumVar}(e) \cup \text{TimeVar}(e)$ denote the set of (numerical and temporal) variables in $e$. For $e_1, e_2 \in E$ let $\text{Var}([e_1, e_2]) = \text{Var}(e_1) \cup \text{Var}(e_2)$, and for $SE \in SE$, let $\text{Var}(SE)$ be the set of all variables appearing in at least one interval of $SE$; similarly, we extend $\text{TimeVar}$ and $\text{NumVar}$ to interval expressions and specification expressions.

Definition 3.5 (Interval specification). An interval specification is a pair $\delta = (SE, \mathcal{Y})$ where: (1) $SE \in SE$; (2) $\mathcal{Y} = \mathcal{Y}_1 \cup \mathcal{Y}_u$ and for each expression of the form $[e_1, e_2]$ in $SE$, we have: (2a) $\mathcal{Y}$ contains the interval condition constraint $e_1 \leq e_2$; (2b) $|\text{TimeVar}([e_1, e_2])| \leq 1$; (2c) $\mathcal{Y}_1$ contains at most one linear temporal constraint over the variable in $\text{TimeVar}([e_1, e_2])$, if it exists; (2d) $\mathcal{Y}_u$ contains a set of linear integer constraints over variables in $\text{NumVar}([e_1, e_2])$. We define: $\text{Var}(\delta) = \text{Var}(SE)$, $\text{NumVar}(\delta) = \text{NumVar}(SE)$, $\text{TimeVar}(\delta) = \text{TimeVar}(SE)$.

Let $I'$ denote the set of interval specifications, and $I_0'$ the set of (simple) interval specifications $(SE, \mathcal{Y})$ where $SE$ is of the form $[e_1, e_2]$.

Remark 4. These definitions present the case of the lowest level granule, considering expressions in terms of timestamps; this extends in the natural manner to take into account time units. Interval specifications over $u$ are such that all of the specifications in $SE$ are of the form $[e_1, e_2]_u$ and we let $I'_u$ denote the set of interval specifications over $u$, while $I' = \bigcup_{u \in G} I'_u$.

Definition 3.6 (Assignments and Valuations). Let $\delta \in I'$. Then an assignment for $\delta$ is $\nu_\delta = (a_t, a_n)$, with $a_t : \text{TimeVar}(\delta) \to T$, $a_n : \text{NumVar}(\delta) \to \mathbb{N}$. An assignment $\nu_\delta$ induces a valuation function $\hat{\nu}_\delta$, whose application to $e \in E$ is called the valuation of $e$ according to $\hat{\nu}_\delta$.

Let $K = \{\delta_i = ([e_1^i, e_2^i], \mathcal{Y}^i) \mid i = 1, \ldots, l\} \in \mathcal{P}(I'_0)$ with $\mathcal{Y} = \bigcup_{1 \leq i \leq l} \mathcal{Y}^i$ the set of constraints on variables in $K$. An assignment for $K$ is $\nu_K = (a_t, a_n)$, with $a_t : \bigcup_{i \in \{1, \ldots, l\}} \text{TimeVar}(\delta_i) \to T$ and $a_n : \bigcup_{i \in \{1, \ldots, l\}} \text{NumVar}(\delta_i) \to \mathbb{N}$. Then $\nu_K$ induces a valuation function $\hat{\nu}_K$. 

18
Definition 3.7 (Valuations and Satisfiability). Let \( \delta = (SE, \mathcal{V}) \in \mathcal{I}' \), and \( \mathcal{K} = \{\delta^i = ([e^i_1, e^i_2], \mathcal{V}^i) \mid i = 1, \ldots, l\} \in \mathcal{P}^f(\mathcal{I}'_0) \). Then \( \mathcal{V}_\delta \) satisfies \( \mathcal{V} \), denoted by \( \mathcal{V}_\delta \models \mathcal{V} \), if no assignment in \( \mathcal{V}_\delta \) violates a constraint in \( \mathcal{V} \). Furthermore, \( \mathcal{V}_\mathcal{K} \models \mathcal{V} \), if \( \mathcal{V}_\delta \models \mathcal{V}^i \) for \( i = 1, \ldots, l \), where \( \mathcal{V}_\delta^i \) is the restriction of \( \mathcal{V}_\mathcal{K} \) to \( \delta^i \). If there is at least one \( \mathcal{V}_\mathcal{K} \) s.t. \( \mathcal{V}_\mathcal{K} \models \mathcal{V} \), then \( \mathcal{K} \) is satisfiable.

- If \( SE = [e_1, e_2] \), then \( \delta \) is satisfiable iff \( int(\delta) = \{[t_1, t_2] \in \mathcal{I} \mid \exists \mathcal{V}_\delta | \mathcal{V}_\delta \models \mathcal{V} \wedge [t_1, t_2] = \mathcal{V}_\delta([e_1, e_2])] \} \) is non-empty.

- If \( SE = se_1 \sqcup se_2 \), then \( \delta \) is satisfiable iff either \( (se_1, \mathcal{V}) \) or \( (se_2, \mathcal{V}) \) is satisfiable.

- If \( SE = se_1 \sqcap se_2 \), then \( \delta \) is satisfiable iff there is an assignment \( \mathcal{V}_\delta \) such that \( \mathcal{V}_\delta \models \mathcal{V} \) and \( \mathcal{V}_\delta(se_1) \sqcup \mathcal{V}_\delta(se_2) \) is not empty.

Given an interval specification of the form \( ([e, e \oplus z], \mathcal{V}) \), if \( z \) does not appear in \( e \) and the only constraint for \( z \) in \( \mathcal{V} \) has the form \( z \geq 0 \), then we say that \( e \oplus z \) is unlimited and write \( ([e, 0], \mathcal{V}') \), where \( \mathcal{V}' \) is \( \mathcal{V} \) with \( z \geq 0 \) removed. In this paper, we deal only with satisfiable specifications \( \delta \), ruling out ill-formed specifications such as \( ([x, x \ominus 1], \mathcal{V}) \). We may omit \( \mathcal{V} \) when we are only concerned with the interval component of an interval specification.

Example 3.3. Let \( \delta = ([x \oplus 1, x \oplus (1 + 2 \times y)]_D, \{0 \leq y, y \leq 5\}) \in \mathcal{I}'_D \) with \( \text{TimeVar}(x \oplus 1) = \{x\} \), \( \text{TimeVar}(x \oplus (1 + 2 \times y)) = \{x\} \), \( \text{NumVar}(x \oplus (1 + 2 \times y)) = \{y\} \). Let \( \mathcal{V}_\delta = \{\{x \mapsto 2011 : 07 : 17\}, \{y \mapsto 1\}\} \). The valuation for these assignments are given by: \( \mathcal{V}_\delta(x \oplus 1) = 2011 : 07 : 18 \), \( \mathcal{V}_\delta(x \oplus (1 + 2 \times y)) = 2011 : 07 : 20 \). Then \( \mathcal{V}_\delta = \{2011 : 07 : 18, 2011 : 07 : 20\} \). We see that \( \delta \) is satisfiable since \( int(\delta) \) is the set of all intervals \( [t_1, t_2]_D \) with conclusion at the end of an even number of days (between 0 and 10) after the day marking the inception of the interval. Note that because of the operator \( D \), for the case \( y \mapsto 0 \), we would still have the whole duration of one day in the interval \( [2011 : 07 : 18, 2011 : 07 : 18]_D \).

Lemma 3.2. Let \( \mathcal{K} = \{\delta^i = ([e^i_1, e^i_2], \mathcal{V}^i) \mid i = 1, \ldots, l\} \in \mathcal{P}^f(\mathcal{I}') \) and \( \mathcal{V}_\delta^i \), for \( i = 1, \ldots, l \), be a set of assignments. Then \( \mathcal{V}_\mathcal{K} = \bigcup_{i=1}^{l} \mathcal{V}_\delta^i \) is an assignment for \( \mathcal{K} \) if each \( \mathcal{V}_\delta^i \) assigns the same value for each occurrence of the same variable.
Theorem 3.3 (Covering of specifications). Let \( \mathcal{K} = \{ \delta_1, \ldots, \delta_n \} \in \mathcal{P}^{f}(I'_0) \). Then, there exists a process to compute the finite collection \( \overline{\mathcal{K}} = \{ \mathcal{K}^1, \ldots, \mathcal{K}^l \} \) of sequences of interval specifications such that for each \( \mathcal{K}^h \in \overline{\mathcal{K}} \) with \( h \in \{1, \ldots, l\} \): (i) \( \text{Var}(\mathcal{K}^h) = \text{Var}(\mathcal{K}) \); (ii) every valuation function \( \hat{V}_{\mathcal{K}} \) of \( \mathcal{K} \) is a valuation function of \( \mathcal{K}^h \); (iii) \( \hat{V}_{\mathcal{K}}(\mathcal{K}^h) \) is a non-overlapping cover of \( \hat{V}_{\mathcal{K}}(\mathcal{K}) \). Hence, \( \mathcal{K} \) is satisfiable iff \( \overline{\mathcal{K}} \) is non-empty.

By extension, we call each set \( \mathcal{K}^h \in \overline{\mathcal{K}} \) from Theorem 3.3 a cover of \( \mathcal{K} \).

Proof. We first show how, given \( \mathcal{K} \), one can compute the collection \( \overline{\mathcal{K}} \) of all sequences of interval specifications meeting conditions (i)-(iii).

Let \( Y_{\mathcal{K}} \) be the union of the constraints in the interval specifications of \( \mathcal{K} \). We first rewrite any interval specification \( \delta_j \in \mathcal{K} \) to an interval specification \( \delta_j' \) by replacing each occurrence of a timestamp \( t \) in \( \delta_j \) with a variable \( \bar{t} \) and replacing in \( Y_{\mathcal{K}} \) the constraints resulting from the interval condition for these variables, forming the set of constraints \( Y_{\hat{\mathcal{K}}} \). Then, for each \( j \), we let \( e_1^j \) or \( e_2^j \) be the two expressions in \( \delta_j' \) and generate the corresponding fresh variables \( X_1^j \) and \( X_2^j \). Then we generate all linear orderings on variables in \( \bigcup_{j \in \{1, \ldots, n\}} (\text{NumVar}(\delta_j') \cup \{X_1^j, X_2^j\}) \) compatible with the constraints in \( Y_{\hat{\mathcal{K}}} \cup \bigcup_{j \in \{1, \ldots, n\}} \{X_1^j \leq X_2^j\} \). For each ordering, we consider the possible symbolic relations between interval specifications compatible with the ordering, i.e., if we have \( Y_1 \leq \cdots \leq Y_{i-1} \leq Y_i \leq Y_{i+1} \leq Y_{i+2} \leq \cdots \leq Y_k \), we consider the possible relations \([Y_{i-1}, Y_i] < [Y_{i+1}, Y_{i+2}], [Y_i, Y_{i+1}] \bowtie [Y_{i-1}, Y_{i+2}], [Y_{i-1}, Y_i] < [Y_{i+1}, Y_{i+2}], \) etc. For each choice of relation, we apply the symbolic version of the rules in \( \mathcal{R} \); for example, for the case \([Y_i, Y_{i+1}] \bowtie [Y_{i-1}, Y_{i+2}] \), we obtain the new set of intervals \{\([Y_{i-1}, Y_i \ominus 1], [Y_i, Y_{i+1}], [Y_{i+1} \ominus 1, Y_{i+2}]\)\}. Finally, we substitute the new variables with the corresponding expressions and \( \bar{t} \) with the original timestamp \( t \), to generate a sequence of interval specifications, ruling out those which cannot satisfy the interval condition. By construction, each valuation of this sequence generates a non-overlapping cover of the interval induced by the valuation of the original set of specifications (it has no \( \bowtie \) or \( \bowgeq \) relations between any two intervals in the sequence and merges to a single interval if there were no disjoint relations considered in the construction). \( \overline{\mathcal{K}} \) is then the set of interval specifications in the constructed sequence.

The proof of the equivalence follows: For the only if direction: since \( \mathcal{K} \) is satisfiable, \( \overline{\mathcal{K}} \) is non-empty. For the if direction: if \( \overline{\mathcal{K}} \) is non-empty, then there is at least one sequence \( \mathcal{K}^h \) for which some valuation function exist, say
\( \mathcal{V}_{K^h} \), which makes \( K^h \) satisfiable. Since \( Var(K^h) = Var(K) \) by construction, \( \mathcal{V}_{K^h} \) is also a valuation function for \( K \), which makes \( K \) satisfiable. \( \square \)

**Example 3.4.** Consider the set \( \mathcal{K} = \{ H^1, H^2 \} \), with \( H^1 = [2011 : 01, x]_M \) and \( H^2 = [2011 : 01 \oplus y, x \oplus y]_M \), and \( \mathcal{Y}_{K_t} = \{ 2011 : 01 \leq x, x \leq 2013 : 12 \} \), \( \mathcal{Y}_{K_a} = \{ y \leq 10 \} \). An interval satisfying \( H^2 \) will have the same duration as one satisfying \( H^1 \), but will occur at an offset of no more than 10 months from the latter. Hence, we replace \( H^1 \) with \( \overline{H^1} = [\overline{t^1}, x] \) and \( H^2 \) with \( \overline{H^2} = [\overline{t^1} \oplus y, x \oplus y] \) and substitute \( \overline{t^1} \) for the first constraint in \( \mathcal{Y}_{K_t} \). Now we introduce the variables \( X^2_1 \) and \( X^2_2 \) corresponding to the expressions \( \overline{t^1} \oplus y \) and \( x \oplus y \) and the constraint \( X^2_1 \leq X^2_2 \). The possible orderings are:

1. \( \overline{t^1} \leq x \leq X^2_1 \leq X^2_2 \), with cases \( \overline{H^1} < \overline{H^2} \) and \( \overline{H^1} \prec \overline{H^2} \), both of which, after symbolic rewriting and substituting back timestamps and expressions for the new variables, generate the sequence \( \langle H^1, H^2 \rangle \).

2. \( \overline{t^1} \leq X^2_1 \leq x \leq X^2_2 \), with case \( \overline{H^1} \preceq \overline{H^2} \), giving rise to the sequence \( \langle [2011 : 01, 2011 : 01 \oplus y \ominus 1]_M, [2011 : 01 \oplus y, x]_M, [x \ominus 1, x \oplus y]_M \rangle \).

3. \( \overline{t^1} \leq X^2_1 \leq X^2_2 \leq x \), with \( \overline{H^2} \asymp \overline{H^1} \). This would give rise to the sequence \( \langle [2011 : 01, 2011 : 01 \oplus y \ominus 1]_M, [2011 : 01 \oplus y, x \oplus y]_M, [x \oplus y \ominus 1, x]_M \rangle \). As the last specification cannot satisfy the interval condition, this case does not contribute to the construction of the set of sequences \( \mathcal{K} \).

4. **Timed Spider Diagrams**

   We introduce Timed-SDs by assigning an interval of existence to each diagram element (e.g. curves, spiders, feet, etc), thus allowing a diagram creator to express complex temporal constraints. Since from a reader’s perspective the interpretation of these diagrams may be challenging, we introduce the notion of a basic timed-SD, which assigns a single interval of existence to an entire diagram. Then we show how to transform a timed-SD into a temporally-contiguous sequence of basic timed-SDs. As an example, Figure 2 in Section 2 presents an example of timed-SD, while Figure 5 gives its corresponding sequence of basic timed-SDs, where the interval presented at the bottom is associated with a whole diagram. The rest of this Section provides a formal account of these notions.

---

\( ^2 \)Using \( t^1 \) here to refer to the inception of \( H^1 \).
Diagrams with temporal aspects have to satisfy some notion of time-consistency; that is, elements are present only within a proper context (e.g. a foot can be in a zone only during the existence interval for that zone). Definition 4.1 formally presents the notions of a timed SD and time-consistency.

**Definition 4.1 (Timed SD).** A timed SD is a construct \( d_T = (d, \mathcal{K}, \omega) \), where \( d = (C, Z, sh, S, h, \tau, \nu) \) is a unitary-SD, \( \mathcal{K} \) is a finite set of interval specifications, and \( \omega : C \cup Z \cup (Z \times B) \cup S \cup F \cup \tau \cup \nu \rightarrow \mathcal{K} \) is a surjective function on \( \mathcal{K} \) assigning an interval specification to every curve, zone, shaded zone, spider, spider foot, tie or strand in \( d \). With \( \mathcal{Y} \) the set of constraints for \( \mathcal{K} \), let \( \mathcal{V}_K \) be an assignment for \( \mathcal{K} \), s.t. \( \mathcal{V}_K \models \mathcal{Y} \). We say that \( d_T \) is time-consistent w.r.t. \( \mathcal{V}_K \) (or that \( \mathcal{V}_K \) is time-consistent for \( d_T \)) iff the following hold:

- \( f = (s, z) \in F \implies \hat{\mathcal{V}}_K(\omega(f)) \bowtie (\hat{\mathcal{V}}_K(\omega(s)) \cap \hat{\mathcal{V}}_K(\omega(z))) \)
- \( z \in Z \implies \hat{\mathcal{V}}_K(\omega(z), sh(z)) \bowtie \hat{\mathcal{V}}_K(\omega(z)) \)
- \( z \in Z, c \in C \implies \hat{\mathcal{V}}_K(\omega(z)) \bowtie \hat{\mathcal{V}}_K(\omega(c)) \)
- \( t = (s_1, s_2, z) \in \tau \cup \nu \implies \hat{\mathcal{V}}_K(\omega(t)) \bowtie ((\hat{\mathcal{V}}_K(\omega(s_1)) \cap \hat{\mathcal{V}}_K(\omega(s_2))) \cap \hat{\mathcal{V}}_K(\omega(z))) \)

Figure 5: The sequence of basic timed-SDs corresponding to Figure 2.
This extends naturally to compound SDs by considering the union of the constraints from each unitary component and requiring time consistency within each unitary component.

We consider only timed SDs associated with time-consistent valuations (i.e. we assume that \( \mathcal{Y} \) contains all of the induced constraints). A timed SD \( d_T \) constrains diagram elements in \( d \) to be present during the specified time-intervals; that is, for any element \( x \) of \( d \) and any interval \( I \) disjoint from \( \hat{V}_K(\omega(x)) \), we have that \( x \) is not present during any interval \( J \) such that \( J \preceq I \). Note that associating an interval with a pair in \( Z \times \mathcal{B} \) allows for modifications over time of the shading of a zone.

From a technical point of view, one assumes that all diagram elements (curves, zones, shading, spiders, feet, ties or strands) have an associated interval in timed-SD, but from a user point of view, one wishes to allow abbreviations, only annotating the important parts that change and automatically deriving others. A timed SD \( d_T \) can represent a complex time-based set of constraints. A basic timed SD is a simplified version in which the entire diagram is associated with a single interval specification, and a sequence of them are contiguous with respect to an assignment if the valuations of the interval specifications of consecutive diagrams meet.

**Remark 5.** In the following we consider only interval specifications in \( \mathcal{I}_0' \). One can extend to deal with specification expressions of the form \( se_1 \sqcup se_2 \) by splitting into two alternative cases utilising the same underlying diagrams involved, but with the expression replaced by \( se_1 \) in one, and \( se_2 \) in the other. For \( se_1 \cap se_2 \) one may increment \( \mathcal{Y} \) with the constraints ensuring the interval condition for any valuation.

**Definition 4.2 (Basic timed SD).**

- A basic timed-SD is a pair \( B = (d, \delta) \), where \( d \) is an SD and \( \delta \in \mathcal{I}_0' \).
- Let \( \langle d \rangle = \langle (d_1, \delta_1), \ldots, (d_n, \delta_n) \rangle \) be a sequence of basic timed-SDs. Let \( \mathcal{V}_B \) be an assignment of \{\( \delta_1, \ldots, \delta_n \)\}. Then \( \langle d \rangle \) is contiguous w.r.t. \( \mathcal{V}_B \) iff \( \forall j \in \{1, \ldots, n-1\} \left[ \mathcal{V}_B(\delta_j) \prec \mathcal{V}_B(\delta_{j+1}) \right] \).

Given a time-consistent \( d_T \), with an assignment \( \mathcal{V}_K \), we can consider contiguous sequences of basic timed SDs \( \langle d' \rangle \) which essentially capture the same set of constraints. In particular, post-valuation, the lifetime of the \( d_T \) is
the same as that of \( \langle d' \rangle \) by the first two conditions in Definition 4.3. The third condition ensures that the timeline of \( \langle d' \rangle \) is decomposed so that no diagram element of \( d \) should exist during only part of the interval of existence of a diagram in the sequence. This permits the fourth condition to unambiguously define the set of diagram elements that must exist during the time period of each diagram in \( \langle d' \rangle \). The canonical time-decomposition is the natural decomposition of the timeline so that intervals start or end exactly at the set of timestamps where diagram elements are born or die. For simplicity we assume that \( \bigcirc(\mathcal{V}_K(\omega(x))) \) exists in the following, and we write \( x \in d \) to denote a diagram element (i.e. as shorthand for \( x \in C(d) \cup Z(d) \cup S(d) \cup Z(d) \times \mathbb{B} \cup F(d) \cup \tau(d) \cup v(d) \)).

**Definition 4.3 (Time decomposition).** Let \( d_T = (d,K,\omega) \) be a timed SD with \( \mathcal{V}_K \) a time-consistent assignment, \( \langle d' \rangle = \langle (d'_1, \delta'_1), \ldots, (d'_m, \delta'_m) \rangle \) a sequence of basic timed SDS, and \( \mathcal{V}_{K'} \) an assignment for \( K' = \{ \delta'_i | 1 \leq i \leq m \} \). Suppose that: (i) \( \langle d' \rangle \) is contiguous w.r.t. \( \mathcal{V}_{K'} \); (ii) \( \bigcirc_{x \in d} \mathcal{V}_K(\omega(x)) = \bigcirc_{i \in \{1, \ldots, m\}} \mathcal{V}_{K'}(\delta'_i) \); (iii) no interval \( \mathcal{V}_K(\omega(x)) \), for \( x \in d \), overlaps or is during any interval in \( \mathcal{V}_{K'}(\delta'_i) \), unless they are equal; (iv) \( x \in d'_i \Leftrightarrow (x \in d \land \mathcal{V}_{K'}(\delta'_i) \bowtie \mathcal{V}_K(\omega(x))) \). Then \( \langle d' \rangle \) is a time-decomposition of \( d_T \) w.r.t. \( \mathcal{V}_K \) and \( \mathcal{V}_{K'} \) (or of \( d_T \) w.r.t \( \mathcal{V}_K \) if \( \mathcal{V}_K = \mathcal{V}_{K'} \)). If \( m \) is minimal then \( \langle d' \rangle \) is a canonical time-decomposition of \( d_T \) w.r.t. \( \mathcal{V}_K \) and \( \mathcal{V}_{K'} \).

Theorem 4.1 shows that timed SDS possess time-decompositions into contiguous sequences of basic timed SDs, indicating a constructive method. Figure 6 illustrates the procedure for some orderings in a generic case.

**Theorem 4.1 (Covering of timed SDs).** Let \( d_T = (d,K,\omega) \) be a timed SD, and \( \mathcal{V}_K \) a time-consistent assignment for \( K \). Then there exists a non-empty collection, \( \{ \langle d^1 \rangle, \ldots, \langle d^k \rangle \} \), of sequences \( \langle d^i \rangle = \langle (d^i_1, \delta^i_1), \ldots, (d^i_{m_i}, \delta^i_{m_i}) \rangle \) of basic timed SDs s.t. each \( \mathcal{V}_K(\langle d^i \rangle) = \langle (d^i_1, \mathcal{V}_K(\delta^i_1)), \ldots, (d^i_{m_i}, \mathcal{V}_K(\delta^i_{m_i})) \rangle \) is a time-decomposition of \( d_T \) w.r.t. \( \mathcal{V}_K \).

**Proof.** To prove the theorem, we apply the construction in the proof of Theorem 3.3, filtering out interval specifications which would violate the conditions for time-consistency. For each admissible sequence, to each interval \([e^i_1, e^i_2]\) we associate the diagram \( d_i \) consisting of all of the elements of \( d \) s.t. the interval specifications are associated with the variables defining the interval (i.e. \( X^i_1, X^i_2 \) in the construction given in Theorem 3.3). □
In Section 6 we will consider policy specification, requiring the use of sequences as per Definition 4.4 and we will lift the notion of contiguity from intervals (arising as the valuation of interval specifications) to interval specifications (independent of assignment).

**Definition 4.4 (Bounded contiguity).** A sequence of basic timed-SDs $\mathcal{B} = \langle B_1, \ldots, B_n \rangle$ is bound to be contiguous if for each subsequence $\langle B_i, B_{i+1} \rangle$, $1 \leq i < n$, the associated interval specifications satisfy $e_i \oplus 1 = e_{i+1}^1$.

5. Spider Diagrams for modelling

We present an SD-variant which integrates SDs into an object-oriented modelling setting. Two diagrams types are considered, called *instance*-SDs and *type*-SDs, collectively referred to as *Modelling Spider Diagrams* (MSDs). In both of these kinds of diagram, curves represent states, whilst spiders
represent individuals. However, spiders represent instances in the instance-SD and types in the type-SD; this is enforced by restricting the possible universes for the interpretations/models of the diagrams. The two kinds of diagram are related via the habitat of spiders in the type-SD: this defines the permissible states for instances of that type, appearing in the instance-SD. We provide motivating examples before presenting the theory.

Figure 7 shows two type-SDs $d_1$ (left) and $d_2$ (right). For $d_1$, let $U$ be the set of all account types within a bank at a certain configuration of the system. Then each spider determines a single account type, and each spiders’ habitat indicates the set of permissible states for any instance of that type (i.e. any actual account of that type).

![Diagram](image)

Figure 7: Two examples of type-SDs for bank accounts and insurance policies.

An interpretation of $d_1$ is a pair $(U, \psi)$, where $\psi$ maps: (i) each curve to the set of account types whose instances can be in the corresponding state $Opened$, $Closed$, (in the) $Black$ or (in the) $Red$; (ii) the two spiders to the types $PremiumAccount$ and $SavingsAccount$. In order that this interpretation is a model, the following conditions must hold: (i) missing zones in $d$ represent the empty set; i.e. the states $Opened$, $Closed$, (in the) $Black$ and (in the) $Red$ are mutually disjoint; (ii) the interpretation of each spider is contained within the interpretation of its habitat, so the type $PremiumAccount$ is associated with one of the four states and the type $SavingsAccount$ cannot be associated with the state (in the) $Red$. We will then use such a notion of model to support a notion of conformance of instances to a type: e.g. no instance of a saving account can be overdrawn (in the $Red$), and each instance of a
premium account must be found in exactly one of these states.

The use of shaded zones is illustrated by $d_2$ in Figure 7 which presents a type-SD for insurance policies, which can be Normal or Premium. The possible states of policy instances are: Covering (the damages covered by the policy will be payed), Overdue (the premium has not been payed), Closed (the policy is extinct), or certain conjunctions of these. The diagram specifies that no policy type admits instances which are both covering and closed (since the corresponding zone is not present), only instances of PremiumPolicy can be both overdue and covered (since the Premium policy spider is the only spider with a foot in the corresponding shaded zone), while instances of policies of both types can be both overdue and closed (e.g. the person died while the policy was overdue, so the policy was archived with an indication of this fact). An instance-SD depicting a particular configuration of policy instances, such as those managed by a branch of the insurance company, will conform to this type-SD only if the spiders in the instance-SD inhabit regions permitted by the habitat of the corresponding spider in the type-SD; the correspondence between spiders is given by a typing function on the spiders in the instance-SDs. Finally, since there are no strands and ties between feet, the types specified by the spiders are disjoint; thus no policy can be both Normal and Premium.

**Remark 6.** We consider only SDs with constant spiders in the following, although the discussion is readily extendible to existential spiders. An existential spider in a type-SD would assert the existence of some type, not further specified, whose instances can be in any state in its habitat. Similarly, an existential spider in an instance-SD would represent the existence of some unspecified instance in a given state.

Figure 8 is taken from the domain of library system regulations. It shows a type-SD $d$ (top) and an instance-SD $d'$ (bottom left). An instance-SD has more restrictions placed on its syntax, i.e. it admits no shading, and each spider has a single foot. Moreover, with each spider we associate the name of the type of instance with which it can be interpreted, by concatenating the name of the type to the label for the spider.

An interpretation of an instance-SD takes $U$ to be the set of all instances and spiders to be instances which are in the state determined by their habitat. Thus $d'$ indicates that the instance John of type reader is suspended, whilst the instance Susan of type admin is not suspended. We consider an instance-SD, $d'$, to conform to a type-SD, $d$, if the constraints imposed on each of the
In order to easily compare the constraints imposed by $d$ on $d'$ we project $d$ onto a new diagram $d^{\{U,Suspended\}}$ (shown in the bottom right of Figure 8) which has the same set of curves as $d'$; this is essentially the application of multiple remove curve rules (to those curves in $d$ which are not in $d'$), with the appropriate syntactic modification to ensure that we obtain the correct inference. Since the diagram $d'$ has correctly typed individuals ($John$ and $Susan$) in admissible zones for each type in the projection $d^{\{U,Suspended\}}$ (i.e. their habitats are contained within the habitat of the spider for the corresponding type) we have that $d'$ conforms to $d$.

We provide a formalisation of type-SDs, instance-SDs and relate these via the notion of conformance. To simplify the presentation, we consider a strict type system where instances are singly typed; one could extend to permit multiple types, and then make use of strand and ties in type-SDs. Let $\mathbb{T}$ and $\mathbb{S}$ denote the set of all types and states of the modelled system, respectively, and $\sigma : \mathbb{T} \rightarrow \wp(\mathbb{S})$ a function associating each type with a set of states, indicating that instances of $t \in \mathbb{T}$ can be in any state in $\sigma(t)$.

We present the concepts for unitary diagrams, but they extend naturally
to compound diagrams. We assume that each curve label corresponds to the name of a state, (so we can consider the curve label as the state itself, and write $c \in S$) and that zones formed from overlapping curves correspond to overlapping states, as defined by the AND operator on states from [9].

**Definition 5.1 (Type-SD).** Let $d$ be an SD s.t.: (i) all spiders are constant spiders, $S(d) = S_c$; (ii) there are no strands or ties, $\tau \cup \nu = \emptyset$. Then $d$ is a type-SD if the interpretations of $d$ are restricted to be $(U, \psi)$ s.t.: (i) $U = \mathbb{T}$; (ii) $\psi(c) = \{ t \in \mathbb{T} \mid c \in \sigma(t) \}$, for all $c \in C(d)$.

Let $I$ denote the set of instances present in the current configuration of the modeled system and let $\theta : I \rightarrow \mathbb{T}$ be a function mapping an instance to its type; we extend the domain of $\theta$ to subsets of $I$. Moreover, a function $\rho : I \rightarrow \wp(S)$ indicates the set of states that each instance is in, for that configuration.

**Definition 5.2 (Instance-SD).** Let $d' = (C', Z', sh', S', h', \tau', \nu')$ be a unitary SD s.t.: (i) all spiders are constant spiders, $S' = S'_c$; (ii) there are no strands or ties, $\tau' \cup \nu' = \emptyset$; (iii) there is no shading, $\forall z' \in Z' \ [sh'(z') = \text{false}]$; (iv) every spider has exactly one foot, $\forall s' \in S' \ [ \ | h'(s') | = 1 ]$. Let $\chi : S' \rightarrow \mathbb{T}$ be a function that assigns a type to each spider. Then, $(d', \chi)$ is an instance-SD if the interpretations of $d'$ are restricted to be $(U', \psi')$ s.t.: (i) $U = I$; (ii) $\forall c' \in C' \ [\psi(c') = \{ x \in I \mid c' \in \rho(x) \}]$; (iii) $\forall s' \in S' \ [\theta(\psi'(s')) = \chi(s')]$.

We may write $d = (C, Z, sh, S, h)$ and $d' = (C', Z', S', h')$ for type- and instance-SDs, respectively, omitting the irrelevant components, and treating the type function $\chi$ as implicit; note that we display $\chi$ graphically using the second component of the spider labels (e.g. in Figure 8 one of the spider labels indicates that the instance John is of type reader). Satisfactory of SDs means that there exists a model, and this concept applies to instance-SDs and type-SDs in the natural way. However, we require a notion of conformance relating an instance-SD $d'$ to a type-SD $d$ so that a model for $d'$ is consistent with a model for $d$: if a spider is mapped to an instance of a given type, then it inhabits a zone corresponding to permissible state for that type. We first define conformance for unitary type-SDs, and indicate how to extend to compound type-SDs later.
Definition 5.3 (Conformance). Let \( d \) be a unitary type-SD and \( d' \) an instance-SD. Then \( d' \) conforms to \( d \), denoted \( d' \models d \), if \( C(d') \subseteq C(d) \) and there exists a map \( \Theta : S(d') \rightarrow S(d) \) s.t. for any model \( m' = (I, \psi') \) of \( d' \), there is a model \( (T, \psi) \) of \( d \) with: (i) \( \forall s' \in S(d')[\theta(\psi'(s')) = \psi(\Theta(s'))] \); (ii) \( \forall s' \in S(d'), \psi'(s') \subseteq \psi'(c') \implies \psi(\Theta(s')) \subseteq \psi(c') \).

Figure 9 gives an overview of the morphisms involved in interpretation and conformance (left), and the relations between components of type-SDs and instance-SDs (right). Arrows without a label represent immersions of the source into the target: arrows from \( C(d) \) and \( C(d') \) to \( S \) relate curves to simple states with the same name; arrows from a set to its powerset indicate the natural inclusion as a singleton set; arrows from \( C(d') \) to \( C(d) \) indicate inclusion (extended to their powersets); arrows from \( Z(d') \) to \( Z(d) \) indicate correspondence according to the projection operation; arrows from \( \varphi(C(d)) \) to \( Z(d) \) associate a set of curves \( X \) to the zone \((X, C \setminus X)\). The conditions imposed on type-SDs, instance-SDs and conformance are highlighted with reference to the numbered boxes in the figure: (1) for Definition 5.1 (ii); (2) for Definition 5.2 (ii); (3) for Definition 5.2 (iii), crossing the two regions; (4) for Definition 5.3 (i); (5) for Definition 5.3 (ii), via a subset relationship.

It is beneficial to be able to provide simple criteria to decide if an instance-SD \( d' \) conforms to a type-SD \( d \). Such a check may require a manipulation of the diagrams. For example, \( d \) may have extra curves indicating some further subdivision of states not expressed in \( d' \). Therefore we provide a projection operation on SDs, according to Definition 5.4, which can be used to remove
the set of curves in \( d \) that do not occur in \( d' \), redefining zones, their shading, and the habitat of spiders accordingly. We can then define a syntactic check of conformance, according to Theorem 5.1, based on the comparison of \( d' \) with the projection of \( d \), checking that instance spiders only inhabit zones corresponding to those inhabited by spiders of the correct type.

**Definition 5.4 (Projection).** Let \( C_p, C \subseteq \mathcal{C} \) be sets of curve labels, with \( U \in C_p \cap C \), and let \( z = (X, Y) \in \varphi(C) \times \varphi(C) \). Then \( z^C_{C_p} = (X \cap C_p, Y \cap C_p) \) is the projection of \( z \) with respect to \( C_p \), and \( Z^C_{C_p} = \{ z^C_{C_p} \mid z \in Z \} \) is the projection of a set of zones \( Z \) with respect to \( C_p \). Let \( d \) be a SD with \( C_p \subseteq C \equiv C(d) \). The projection of \( s \in S(d) \) is the spider \( s^C \) with habitat defined by: (i) \( h(s^C) \subseteq Z^C_{C_p} \); (ii) \( z \in h(s) \implies z^C_{C_p} \in h(s^C) \); and (iii) \( z' \in h(s^C) \implies \exists z \in h(s) \text{ s.t. } z' = z^C_{C_p} \). The projection of \( d \) with respect to \( C_p \), denoted \( d^C_{C_p} \), is the diagram obtained as follows: (1) replace \( Z \) by \( Z^C_{C_p} \); replace each \( s \in S(d) \) by \( s^C \); (2) if \( z \in Z(d^C_{C_p}) \) then \( z \in Z^*(d^C_{C_p}) \) iff \( \forall z' \in Z(d)[(z')^C_{C_p} = z \implies z' \in Z^*(d)] \).

An instance-SD \( d' \) can have multiple spiders of the same type corresponding to a single spider in a type-SD \( d \), but they all must satisfy the constraints imposed by \( d \) if conformance is to hold. In particular: (i) the curves of \( d' \) are present in \( d \) and the zones of \( d' \) are present in the projection of the zones of \( d \); (ii) for \( s' \in S(d') \), the type of \( s' \) is equal to the interpretation of its corresponding spider (through \( \Theta \)) in \( d \) and the projection of the habitat of \( s \) must contain the habitat of \( s' \).

**Theorem 5.1 (Static check of conformance).** Let \( d \) be a unitary type-SD and \( d' \) an instance-SD. Then \( d' \) conforms to \( d \) if the following conditions hold:

1. \( C(d') \subseteq C(d) \land Z(d') \subseteq Z(d)^C(d') \).
2. \( \forall s' \in S(d') \exists s \in S(d) [(s = \Theta(s')) \land (h'(s') \cap Z(d)^C(d') \subseteq h(s)^C(d'))] \).

**Proof.** For conformance to hold, due to Definition 5.3 there must be a map from the spiders of \( d' \) to the spiders of the \( d \) such that models for \( d' \) give rise to models of \( d \), where the type of the interpretations of the instance spider is the interpretation of the corresponding type spider, and if the interpretation of a spider is contained within the interpretation of a curve in \( d' \) then the same holds for \( d \). Since \( d \) is a type-SD and \( d' \) is an instance-SD: (1) by Definition 5.1 curves in \( d \) are interpreted as the set of types whose instances
can be in the given state determined by the curve; (2) by Definition 5.2 curves in \(d'\) are interpreted as the set of instances in a given simple state, zones are interpreted as the set of instances in the conjunction of the simple states of the curves containing the zone, and spiders are interpreted as instances that are typed correctly (according to their label).

Condition 1 states that: the projection of \(d\) onto the set of curves of \(d'\) has the same set of curves as \(d\) and every zone in \(d'\) is present in \(d^{C(d')}\). Thus there are no zones missing from \(d'\) which are present in the projection of \(d\). This ensures that the models of the underlying Euler diagrams are compatible. Condition 2 states that: for each spider \(s'\) in the instance-SD \(d'\) there is a spider \(s\) in the type-SD \(d\) with the correct typing, such that the habitat of \(s'\) when restricted to the zones of \(d^{C(d')}\) is contained within the projection of the habitat of \(s\). Thus every spider \(s'\) in \(d'\) inhabits a zone that is contained within the habitat of the corresponding spider \(s\) in the type-SD \(d\). This, together with the constraints imposed of the models by Definition 5.1 and Definition 5.2 ensures the condition on the interpretation of spiders and curves in Definition 5.3 holds.

The conformance relationship extends to permit conformance of a (unitary) instance to a compound type-SD in the natural way, as indicated in Table 2. The instance-SD \(d'\) (shown in header column) conforms to the compound type-SD (shown in header row) according to the expression in the relevant cell. For example, \(d'\) conforms to \(d_1 \lor d_2\) if \(d\) conforms to both \(d_1\) and \(d_2\). The relationship between a type-SD and an instance-SD provides a standard connection between types and their instances.

**Table 2: Extending the conformance relationship to compound type-SDs.**

<table>
<thead>
<tr>
<th>(\neg d_1)</th>
<th>(d_1 \lor d_2)</th>
<th>(d_1 \land d_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d') (\not\models d_1)</td>
<td>((d' \models d_1) \lor (d' \models d_2))</td>
<td>((d' \models d_1) \land (d' \models d_2))</td>
</tr>
</tbody>
</table>

5.1. *Timed Modelling Spider Diagrams*

We extend MSDs to *Timed Modelling Spider Diagrams* (TMSDs). A motivating example is shown in Figure 10, where a temporal aspect is added to the modelling of insurance policies, by associating interval specifications with the feet of the *PremiumPolicy* spider. The diagram \(d\) is the disjunction
of two unitary diagrams, and the temporal constraints are evaluated over \( d \). From \( d \) we can infer that such a policy cannot be simultaneously \textit{Overdue} and \textit{Covering} for more than 14 days, and afterwards the policy becomes either just \textit{Covering} (left hand diagram) or just \textit{Overdue} (right hand diagram). In particular, the time annotation for the feet in the shaded regions constrains any policy to be both \textit{Overdue} and \textit{Covering} for at most 14 days, indicated by the constraint on the variable \( k \); the variable \( x \) expresses the day in which the policy expired. The interval specifications for the other two feet (in the left and right hand diagrams, respectively) indicate that the policy can subsequently either be in the state \textit{Covering} or in the state \textit{Overdue} starting from the next day (i.e. day \( x \oplus k + 1 \) or \( x \oplus 15 \), respectively). If, at the inception of the 15th day, the policy instance is in the state \textit{Overdue} it can remain in this state indeterminately, as indicated by the '*' symbol. If, on the contrary, on the inception of the day ending the period in which it is both \textit{Overdue} and \textit{Covering}, the instance is in the state \textit{Covering}, it will remain in that state until the expiry period is reached (one year after the inception of the assignment for \( x \)).

![Figure 10: An example of temporal constraints for the condition of a premium policy being \textit{Covering} and \textit{Overdue}, with all intervals specified at the granularity of days.](image)

MSDs extend to TMSDs in the same manner as SDs extend to TSDs; compound timed type-SDs are permitted, but only unitary timed instance-SDs. Moreover, timed instance-SDs are annotated only with intervals and not with interval specifications. We extend the notion of conformance to TMSDs, as follows: (i) timed type-SDs and timed instance-SDs are translated into contiguous sequences of basic TMSDs, adapting the construction of Theorem 4.1; (ii) sequences are checked for conformance (Definition 5.5). A sequence of contiguous basic timed instance-SDs conforming to a contigu-
ous sequence of basic timed type-SDs is called a *story*. The semantics of a timed type-SD is given as the set of all of its stories, i.e. all possible timed instance-SD sequences that satisfy all of the constraints for the corresponding intervals (Definition 5.6).

![Diagram of coverage](image1)

**Figure 11:** Alternative contiguous basic timed type-SDs for the timed type-SD of Figure 10.

![Diagram of coverage](image2)

**Figure 12:** Two contiguous sequences of basic timed instance-SDs. Top: an SD-story for the timed type-SD of Figure 10. Bottom: not an SD-story for the timed type-SD of Figure 10.

Figure 11 shows two examples of contiguous sequences of basic timed type-SDs, encapsulating the alternative possible cases permitted by the timed type-SD of Figure 10: the top (resp. bottom) sequence is the scenario arising from the left (resp. right) unitary diagram. Figure 12 (top) shows an example of a contiguous sequence of basic timed instance-SDs conforming to the contiguous basic timed type-SDs at the top of Figure 11. Hence, it conforms to the type-SD in Figure 10, and so is an SD-story. Figure 12 (bottom) shows an example which does not conform to either of the contiguous sequence of basic timed type-SDs in Figure 11, and hence is not an SD-story for Figure 10.
Definition 5.5 (Conformance of TMSDs). Let \( d_T = (d, \mathcal{K}, \omega) \) be a timed type-SD, \( \langle d \rangle = \langle (d_1, \delta_1), \ldots, (d_p, \delta_p) \rangle \) a time-decomposition of \( d_T \) w.r.t. an assignment \( \mathcal{V}_K \), and \( \langle d' \rangle = \langle (d'_1, \delta'_1), \ldots, (d'_m, \delta'_m) \rangle \) a sequence of contiguous basic timed instance-SDs. We say that \( \langle d' \rangle \) conforms to \( \langle d \rangle \), and hence to \( d_T \), w.r.t. \( \mathcal{V}_K \) if \( d'_j \models d_i \) for any \( i, j \) s.t. \( \delta'_j \triangleleft \mathcal{V}_K(\delta_i) \lor (\delta'_j) \propto \mathcal{V}_K(\delta_i) \) and the inceptions of \( \mathcal{V}_K(\delta_1) \) and \( \delta'_1 \) are equal. We say that \( \langle d' \rangle \) conforms to \( d_T \) if there exists one such assignment \( \mathcal{V}_K \). The basic timed instance-SD sequence \( \langle d' \rangle \) is called a \( d_T \)-story w.r.t. \( \mathcal{V}_K \). Any \( d_T \)-story w.r.t. some \( \mathcal{V}_K \) is called a \( d_T \)-story.

Definition 5.6 (Semantics). Let \( d_T = (d, \mathcal{K}, \omega) \) be a timed type-SD. The semantics of \( d_T \) is the set of all the \( d_T \)-stories for all possible (time-consistent) valuations of \( d_T \).

We refer to SD-stories when \( d_T \) is implicit. The construction in Theorem 4.1 is applicable to both type-SDs and instance-SDs, leading to Theorem 5.2, which enables the construction of an SD story for any time consistent timed type-SD.

Theorem 5.2 (Non-emptiness of semantics). Let \( d_T = (d, \mathcal{K}, \omega) \) be a timed type-SD. If \( \mathcal{K} \) admits a time-consistent valuation then \( d_T \) has non-empty semantics.

6. Policy specification

Time-based policies specify constraints on instances by using sequences of (bound to be contiguous) timed type-SDs (see Definition 6.1). These are enforced within a validity interval, meaning that a policy cannot become active before a certain timestamp and that it ceases to hold after a certain timestamp (specified with respect to some external event or by a time expression). A policy is activated by a trigger, which is an occurrence of an event (within the validity interval) causing the policy to become active. Upon this occurrence, a time observation provides a timestamp which is assigned as the value of a designated variable \( WHEN \).

Definition 6.1 (Timed policy specification). A timed SD policy specification, or policy for short, is a construct \( \Pi = (validity, trigger, condition) \) (also denoted \( \Pi = (V_a, T_r, C_o) \)) s.t.
1. validity is an interval specification \([P, Q]\), where each of \(P\) and \(Q\) is either a fixed timestamp or is a name for an event, the occurrence of which fires a time observation.

2. trigger is a pair \((d, W)\), where \(d\) is a type-SD and \(W\) is either the variable \(WHEN\) or a fixed timestamp.

3. condition is a set \(B = \{B^1, \ldots, B^n\}\) of sequences of basic timed type-SDs, where each \(B^i = \{B^i_1, \ldots, B^i_{h_i}\}\) is bound to be contiguous, and \(TimeVar(B) \subseteq \{WHEN\}\).

We give an example of policy with reference to a model of vehicles interacting with an automated parking meter system, covering various parking strips in a city centre, presented informally in [6]. Cars can be in one of three disjoint states: \(Running\), \(FreeParking\), and \(TollParking\), where \(Running\) indicates that the car is in motion, \(FreeParking\) that the car is situated in a free parking zone, and \(TollParking\) that the car is situated within a toll parking zone. For each vehicle, the system records the time-stamp of its entering and exiting either a free parking or a toll parking zone, and automatically charges an appropriate amount (or fine) to the credit card of the registered owner of the vehicle. When in a toll parking zone, vehicles may be within the permitted period of time for parking (i.e. in the state \(WithinTime\)), or their permitted time may have expired (and they are in the state \(Expired\)).

![Figure 13: A timed SD policy specification for the parking meter model.](image-url)

Figure 13 shows an example of a policy specification, where the validity expression is shown textually at the top of the figure. Here, \(meterInPlace\)
and policyChange denote external events and express that the policy is valid from the activation of the parking meter system until some change in the policy will take place. The type-SD in the middle of Figure 13 presents the trigger for the policy: it fires when a Car changes its state to WithinTime, at a timestamp within the validity period for the policy. We assume that the trigger would also fire for an instance which at the start of the validity period is in the state WithinTime. The policy requirement stated by the condition is that if a car stays parked for the whole period of 60 minutes, then, starting with the 61st minute, it is in the state Expired (possibly leading to administrative measures), where it can remain indefinitely. This condition would typically be ended by the occurrence of some external event (e.g. the car is moved away), rather than by meeting some temporal deadline.

Policies can define complex conditions, and can express some forms of negative constraint. For example, the policy in Figure 14 can be interpreted as stating that after leaving a free parking zone, where it cannot stay for more than 60 minutes, a car cannot return to a free parking zone before 120 additional minutes have elapsed.

Figure 14: Specifying a forbidden period; constraints on variables in the condition are shown next to the trigger.

In the example above, the type-SD in the trigger happens to coincide with the type-SD at the initial time step of the sequence in the condition, since the policy is related to a constraint with an immediate effect. However, the use of a trigger permits a time-delay before the effect.
The next example, derived from a model of Internet billing, illustrates this feature as well as the use of sets of alternative sequences of timed type-SDs in the policy condition. Moreover, we introduce a notion of role in Definition 6.2, allowing modelers to specify different aspects of a policy.

**Definition 6.2 (Role annotation).** Let \( R \) be a set of role names, \( d \) a MSD, and \( s \in S(d) \) a spider. A role-annotation of \( s \) in \( d \) is a function \( \text{role} : \{s\} \rightarrow R \). The definition of conformance is then restricted to consider only spiders with matching role annotations.

We use textual annotation to indicate the roles in the figures. Figure 15 shows an example policy specification which is valid for the duration of a contract, starting when the \text{contractSubscribed} event occurs and ending upon the occurrence of the \text{newContract} event. The policy is triggered when the user, in the role of \text{Traffic}, is in state \text{Exceeded} (having exceeded usage allowance). Constraints express that the billing day is the middle of the next calendar month\(^3\) and that the payment deadline is 15 days later than the billing day. The condition consists of two basic timed type-SD sequences, expressing the possible alternative type-SDs that an instance sequence must conform to in order for it to satisfy the policy. The top sequence covers the case in which the user pays before the deadline without having exceeded usage limit: in this case the user’s allowance in his/her role as \text{Traffic} is returned to normal. The alternative scenario is presented by the bottom sequence: if the user has not paid by the deadline then he/she has to pay a fine.

**6.1. Synchronisation semantics**

In principle, a policy, when activated for a specific instance in a model, can hold indefinitely for that instance (i.e. the policy holds over an unbounded time period). For example, according to Figure 15, a \text{User as Payer} can remain in the \text{HasToPayFine} state at all subsequent timestamps past the deadline. However, specific instances will cease to be in that state because the fine is payed or the contract is discontinued at some moment.

Since these changes of state occur not on a time basis, but due to external events, we can define conformance to a policy only as long as an instance is

\(^3\)The operator \text{fifteenOfMonth} returns, for each valuation \( \tau \) of a temporal expression \( \nu \), the timestamp associated with the inception of the interval \( [M(\tau) \oplus 14, M(\tau) \oplus 14]_D \).
subject to that policy. Hence, we define the semantics of a policy via a
procedure for deciding if a contiguous sequence of basic timed instance-SDs
conforms to the policy. For simplicity, we assume that we have evaluated
any variables in the time expression for the validity interval of the policy
in Definition 6.3. Note that the annotation of an instance-SD by a single
timestamp \( t \) is equivalent to its annotation by \([t, t]\).

**Definition 6.3 (Activator).** Let \( \Pi = (V_a, T_r, C_o) \) be a policy, and let \( T_r \)
denote the SD in \( T_r \). Let \( \langle d \rangle = \langle (d_1, \delta_1), \ldots, (d_n, \delta_n) \rangle \) be a sequence of bound
to be contiguous basic timed instance-SDs, with \( \delta_i = [e_i^1, e_i^2] \). \( T_r \) can be
annotated with:

1. a fixed timestamp \( t_w \). Then \( t_w \) is a potential activator of \( \Pi \) for \( \langle d \rangle \),
w.r.t. \( V_{\delta_i} \), if \( d_i \models T_r \), where \( t_w \) is within \( \overline{V_{\langle d \rangle}}(\delta_i) \).
2. the variable WHEN. Then timestamp \( t_w \) is a potential activator of \( \Pi \)
   for \( \langle d \rangle \), w.r.t. \( V_{\langle d \rangle} \), if \( t_w = \overline{V_{\langle d \rangle}}(e_i^1) \) for some \( i \in \{1, \ldots, n\} \) and: (i)
   \( i = 1 \Rightarrow d_1 \models T_r \); (ii) \( i \neq 1 \Rightarrow d_{i-1} \not\models T_r \land d_i \models T_r \).

We say that \( d_i \), hence \( \langle d \rangle \), satisfies the trigger of \( \Pi \) at a potential activator
\( t_w \). The earliest potential activator of \( \Pi \) for \( \langle d \rangle \) that lies within the validity
interval of \( \Pi \) is called the activator of \( \Pi \) for \( \langle d \rangle \), w.r.t \( V_{\langle d \rangle} \).
Given a sequence of bound to be contiguous basic timed instance-SDs and an assignment, we consider the sequence generated from the corresponding valuation and adopt the terminology in Definition 6.3 omitting reference to the assignment. We can check if such a sequence of basic timed instance-SDs annotated with contiguous intervals conforms to the conditions of a policy specification; we use the abbreviated notation $d_{t_w}$ for diagram $d$ at timestamp $t_w$ (i.e. over $[t_w, t_w]$) within a sequence of basic timed instance-SDs.

A policy could be activated many times within a particular instance sequence (e.g. a car parking in a toll parking zone, moving away and returning after several hours, re-triggers the policy activation). On the other hand, a subsequent potential activator might not trigger a new activation, but be required by the condition of the policy itself (e.g. a yearly contract might establish that a User as Payer is in the state HasToPay on 15th of each month). For the sake of simplicity, we assume that a policy is triggered once by a sequence (hence the use of the singular term activator above).

Suppose we have a sequence $\langle d \rangle$ of basic timed instance-SDs evaluated so that its annotations are contiguous intervals, and that $t_w$ is the activator of $\Pi = (V_a, T_r, C_o)$ for $\langle d \rangle$, and a valuation of the variables in $V_a$ of $\Pi$. We wish to know if $\langle d \rangle$ triggers $\Pi$ and if so, whether it satisfies any of the conditions of the policy for their entire duration. Therefore, we construct $[t_w, M]$, where $M$ is obtained by a valuation of the variables in any interval specification in $C_o$, and we choose the valuation so that $M$ is maximal. For simplicity, in Construction 6.1 we suppose that $C_o$ consists of a single type-SD sequence.

**Construction 6.1 (Maximal duration).** Let $\Pi = ([P, Q], T_r, C_o)$ be a policy, where any variables in $[P, Q]$ have been evaluated, $C_o$ contains only $\langle d \rangle = (\langle d_1, \delta_1 \rangle, \ldots, \langle d_n, \delta_n \rangle)$ a sequence of basic timed type-SDs, and let $t_w$ be a timestamp. Then $M$ is calculated as follows:

1. If $Q = t_v$, a timestamp, then set $y = t_v$.
2. Otherwise set $y = *$ (i.e. the conclusion is given as an event).
3. If all of the time expressions $e_i^j$ in $\langle d \rangle$ contain only constants and variables with upper bounds in $\mathbb{N}$, then $mx$ is the conclusion of the interval $\bigotimes_{i \in \{1 \ldots n\}} \hat{V}_\Pi(\delta_i)$, where $\hat{V}_\Pi$ is a valuation of all variables in $\Pi$, consistent with the chosen valuation of $[P, Q]$, with $\hat{V}_\Pi(\delta_i) = [t_w, h_i]$ for a set $\{h_1, \ldots, h_n\}$ that maximises the duration of the merge.
4. Otherwise $mx = *$.
5. $M = \min(y, t_w \oplus mx)$
where we assume $t < *$ and $t \oplus * = *$ for all $t$.

A sequence of instance-SDs, $\langle d' \rangle$, is subject to a policy $\Pi = (V_a, T_r, C_o)$, if: it has an activator, $t_w$; $\langle d' \rangle$ lies within the interval of maximal extent $[t_w, M]$; there are no spiders’ feet within curves in $\langle d' \rangle$ if those curves do not appear in a diagram in $C_o$. This last requirement allows us to consider any states which are not modelled in the policy, but which are in the instance-SD sequence, as interrupt states, because the policy only holds for as long as the instances remain in one of the states modelled in the policy (e.g. if the state of the car is BurntOut, then the instance is not subject to the ParkingMeter policy, but entering that state might trigger a different policy for disposing of the car). Thus we restrict the instances to be relative to the policy (i.e. they only refer to objects/states in the policy specification).

**Definition 6.4 (Sequence subject to policy).** Let $\Pi = (V_a, T_r, C_o)$ be a policy, and let $\langle d' \rangle = \langle(d'_1, [t_1, s_1]), \ldots, (d'_n, [t_n, s_n])\rangle$ be a contiguous sequence of basic timed instance-SDs s.t. $[t_i, s_i] \in \mathcal{I}$ for $i \in \{1, \ldots, n\}$, and $d'_i \neq d'_{i+1}$ for $1 \leq i \leq n$. Suppose that $M$ is obtained as in Construction 6.1. We say that $\langle d' \rangle$ is subject to policy $\Pi$ over $[t_w, t_1]$, if there is a timestamp $t_i$ s.t.: 

1. $t_w$ is the activator of $\Pi$ for $\langle d' \rangle$;
2. $t_w \leq t_i \leq M$;
3. any curve in $\{d'_{i+1}, \ldots, d'_i\}$ which is not in $C_o$ has no spiders’ feet inhabiting it, where $t_w \in [t_j, s_j]$ and $j < i$.

**Definition 6.5 (Sequence conforming to policy).** Let $\Pi = (V_a, T_r, C_o)$ be a timed SD policy, and let $\langle d' \rangle = \langle(d'_1, [t_1, s_1]), \ldots, (d'_n, [t_n, s_n])\rangle$ be a contiguous sequence of basic timed instance-SDs s.t. $[t_i, s_i] \in \mathcal{I}$ for $i \in \{1, \ldots, n\}$, and $d'_i \neq d'_{i+1}$ for $1 \leq i \leq n$. Let $\langle d \rangle = \langle(d_1, \delta_1), \ldots, (d_n, \delta_n)\rangle$ be a prefix of a sequence of contiguous basic type-SDs in $C_o$. We say that $\langle d' \rangle$ conforms to $\Pi$ w.r.t a valuation $\mathcal{V}_\Pi$ if:

1. $t_1$ is an activator for $\Pi$ and $\langle d' \rangle$ is subject to $\Pi$ over $[t_1, s_n]$.
2. $d'_i \models d_i$ for all $1 \leq i \leq n$.
3. For $1 \leq i \leq n$, $t_i = \widehat{\mathcal{V}_\Pi(e^1_i)}$, where $e^1_i$ is the inception of $\delta_i$, and $s_n \leq \widehat{\mathcal{V}_\Pi(e^n_n)} \leq M$, where $e^n_n$ is the conclusion of $\delta_n$, and $M$ is obtained as in Construction 6.1.
Then $\langle d' \rangle$ conforms to $\Pi$ if there exists a valuation function $V_\Pi$ for which $\langle d' \rangle$ conforms to $\Pi$ w.r.t $V_\Pi$.

Definition 6.6 described the semantics of a policy in terms of the instance-SDs that conform to it. Following the arguments of Theorem 5.2, we give conditions under which a policy has non empty semantics in Theorem 6.1, whose proof is a straightforward extension of the proof of Theorem 5.2.

**Definition 6.6 (Semantics of policy).** The semantics of a policy $\Pi$ is the set of all sequences of basic timed instance-SDs that conform to $\Pi$.

**Theorem 6.1 (Non-empty semantics of policy).** Let $\Pi$ be a policy. Then $\Pi$ has non-empty semantics.

### 7. Related work

Several models of time have been proposed for formal specifications, both in relation to real-time [10, 11] or hybrid [12] behaviours. Time-based extensions have been also proposed for calculi or specification languages of concurrent processes (see [13] or [14]). In general, these models deal with intervals to model uncertainty about the actual occurrence of an event. In Statemate, a clock-synchronous semantics is provided where events can only occur when a clock ticks [15]. This view was adopted also in [16] to integrate time in graph transformations, by introducing a specific attribute updated by clock messages to processes.

In general, however, we are interested in the persistence in a state over a period as dictated by time-dependent policies, rather than in modeling the occurrence of specific transitions triggered by any type of events. As a consequence, the model of time adopted is connected to the notion of calendar time, as adopted in the area of temporal databases and temporal rule based access control. In particular, we exploit a model analogous to that Bertino et al. [17]. In Bertino’s model a calendar is a set of contiguous intervals, each with its own duration, containing all the instants between its extreme granules, from the start of the first granule to the end of the second one. Based on this, they introduce periods, to express that some roles have to be granted specific access rights at recurring times. Ning et al. exploit the notions of calendars and granules to define a calendar algebra, where operations allow the grouping of intervals or the subdivision of granules.
and the combination of different calendrical structures [18]. Of particular interest is the notion that instants belonging to different granules cannot be interleaved. Kurt and Özsoyoglu consider calendar times as corresponding to an instant, rather than a granule [19]. An examination of the problems related to the use of different granularities is in [20].

Starting with the examination of the relations in Allen’s algebra, Halpern and Shoam have provided the foundations for a modal logic of time intervals, called $HS$ considering only the operators $after$, $begins$ and $ends$ and their inverse versions [21]. Different fragments have been studied, most of which are undecidable. Among the decidable fragments is $AA$, i.e. the one considering the $after$ and $before$ (inverse of after) relations. Note that in our model the $meets$ relation is the primary one in the definition of policies. However, results obtained in the framework of $HS$ theory are not immediately transferrable to our model for two main reasons: first, in $HS$ two intervals are in the $after$ relation if they have the middle endpoint in common, whereas we require them to be contiguous; second, intervals are reduced to their point-based definition, whereas we consider granules with a finite duration.

The different possible choices for the foundation of interval logics are discussed in [22]. While we do not deal with temporal logic issues here, they can be addressed within the framework of layered temporal logics [23], where layers correspond to our granularities. Also, our two-sided semantics (i.e. considering conformance with type-SDs within the constraints set by time expressions) is related to the proposal of propositional logics associated with a duration calculus which assigns durations to states and combines propositions on states with propositions on durations (see [24] for a complete treatment in the context of real-time systems).

Modeling of time is also relevant to the field of multimedia, since sequential media (typically audio and video) need to be synchronised with the presence of static documents over some length of time. In many cases one is therefore interested in durations of intervals which can start at any point in time, rather than at specific instants. For example, Bowman et al. have defined a formalism in which, once a starting point for the system is set, reasoning can be performed on the occurrence, within the current interval, of a state, based on the lengths of the current and previous intervals [25].

In terms of the theory of SDs, in [26], a SD reasoning system containing strands and ties was introduced, although the paper was not completely formalised. Subsequently, in [27], a more thorough formalised reasoning system was presented, without strands and ties, but which presented spiders in
a slightly unnatural manner. The expressiveness of SDs (with ties but not strands) is known to be equivalent to that of monadic first order logic with equality, and in [28] it was shown that introducing constants into the system does not change the expressiveness. Although several interesting SD-variants have been considered (e.g. they were integrated with conceptual graphs to yield Conceptual Spider Diagrams in [29]), to the best of our knowledge the work started in [5], and developed in [6] and in this paper, is the first attempt to integrate time-related aspects into the formalism. Constraint diagrams are a more expressive generalisation of SDs, intended for use within an object oriented modelling context, and [4] parallel models in Z and constraint diagrams were developed. The idea of representing system dynamics through sequences of EDs was introduced in [30], to follow the evolution of sets (rather than the state of individuals) under the effect of Reaction Systems [31].

In principle, as we introduce the time component as annotations over diagrams or their components, the same mechanism could be adopted to enrich other visual reasoning formalisms which do not already have a notion of time. For example, the notion of conformance (of instances to a model or of models to a metamodel) is currently associated with checking invariants defined at the level of the upper model [32]. One could annotate fragments of the upper model, or the invariants themselves, with interval specifications and check conformance with its instances over specific intervals. In general, temporal constraints can be set over structural properties of a configuration as well or the transformations the configuration can undergo. As an example, one could associate two different architectural styles [33] with a same system, one to be followed during work-hours and one during night, adding a temporal aspect to adaptation processes [34]. In all these cases, the verification of conformance with respect to a model remains a process independent from the check of the satisfaction of the temporal constraints, so that it does not require a modification of the underlying logic.

8. Discussion and conclusions

Spider Diagrams (SDs) are a well-established diagrammatic notation for describing relations between sets and conditions on their cardinality. Prior to this work, they: (a) applied only to static constraints with no means of expressing any temporal information; (b) were logically focussed without formal means of integration within a modelling framework. The applicability of SDs had limitations, accordingly. We have made significant advance in
addressing both of these avenues, culminating in the ability to specify policies within the developed notation. We detail here the progress made.

We presented a formalisation of a general version SDs, encompassing existing variants in the literature, and providing us with a standard form on which to base this, and future, work. Next, we developed a calendar-based model aiming towards the application to time-based policies, specifying admissible evolutions of a system state. The model is particularly suitable to expressing requirements and constraints on system configurations, rather than real-time behaviours. The model of time is developed independently from the SD extensions/variations so that: (i) the time model could be used with other notations; (ii) alternative time based models could be used within this SD-based framework, depending upon domain requirements. We integrate the time model and the SD notation, defining TimedSDs which permit the expression of time dependent constraints. We provide a general version enabling complex constraints to be specified within a single diagram, by permitting the expression of temporal constraint on the lifetime of the syntax within the diagram. Then we provide a simplified (basic) version in which the only temporal specifications apply to the existence of an entire diagram; sequences of these basic TimedSDs whose timelines are contiguous provide a ‘comic-strip’ like view. We provide a procedure to decompose (time-consistent) TimedSDs into sequences of contiguous basic TimedSDs. Thus we have addressed limitation (a).

Subsequently, we adapt SDs to fit into a modelling framework, defining type-SDs (giving constraints on types and the admissible states of instances of that type) and instance-SDs (providing particular states of instances), together with the key concept of an instance-SD conforming to a type-SD. Thus, the traditional model-based semantics of SDs is translated into a state-based semantics, using the notion of conformance of type-SDs by instance-SDs. Next, we consider typed systems presenting some form of temporal evolution, and enrich type-SDs and instance-SDs with the temporal model, extending the notion of conformance with respect to temporal constraints, enabling instance evolutions (timed instance-SDs) to be checked for conformance with time-based specifications (timed type-SDs). Finally, this SD adaptation is extended further to enable the specification of policies within this SD-based framework, utilising the machinery developed within this paper.

Within this framework, we plan to pursue different directions of research. In particular, the current work only considers conformance to policies, but does not address the problem of policy enforcement. We plan to attack this
problem by defining temporal events, for example associated with the first observation of some timestamp, typically marking the passage from one basic timed type-SD to the next. Moreover, by integrating different event types, rule-based systems can be devised to generate sequences of instance-SDs conforming to some policy, to be used for simulations of system behaviours, integrating event-based and time-dependent specifications, as proposed in [16] for the case of graph transformations.

We have not currently dealt with logical reasoning, and temporal logics based on the adopted calendar model can be combined with inference systems typically associated with SDs. For example one may use diagrammatic reasoning to describe which logical inferences can be drawn for SDs within a single interval, or temporal logic to prove that some specification must be met for a policy to be satisfied. In general, standard notions and results from the theory of (static) SDs have to be reviewed and lifted to timed SDs, both at the type-SD and the instance-SD levels. As the time annotation mechanism is relatively independent both of the type of diagrammatic model and of the adopted model of time, a general framework for time annotation of diagrammatic systems can be investigated.

References


