Lightweight Formal Method for Robust Routing in Track-based Traffic Control Systems

Maryam Bagheri[1], Edward A. Lee[2], Eunsuk Kang[3], Marjan Sirjani[4], Ehsan Khamespanah[5], and Ali Movaghar[1]

[1] Sharif University of Technology
[2] University of California at Berkeley
[3] Carnegie Mellon University
[4] Mälardalen University, Reykjavik University
[5] University of Tehran

Abstract. In this paper, we propose a robust solution for the path planning and scheduling of the moving objects in a Track-based Traffic Control System (TTCS). The moving objects in a TTCS pass over pre-specified sub-tracks. Each sub-track accommodates at most one moving object in-transit. Due to the uncertainties in the context of a TTCS, we assign an arrival time interval to each moving object for each sub-track in its route, instead of an exact value. The moving object can safely enter into the sub-track in the mentioned time interval. To develop a safe plan, we adapt the tagged-signal model and provide a rigorous mathematical semantics for the actor model of a TTCS. To illustrate the applicability of the provided semantics, we provide a formal model of TTCSs in the Alloy language and use its analyzer to verify the developed model against system safety properties.

Keywords: Track-based Traffic Control Systems · Robustness · Alloy Language · Time Windows · Actor

1 Introduction

This paper proposes a lightweight formalism to model and solve the routing problem with time windows when traveling times of a set of moving objects are uncertain. The traveling time of a moving object can vary due to unforeseen events such as failures in the context of the system. For instance, because of the weather changes, it is not expected that an aircraft arrives at a part of its route exactly at a pre-determined time. Therefore, it is necessary to associate a *time window* with each moving object for each part of its route. This time interval shows the beginning and end of the time at which the moving object could arrive at that part. A robust routing solution finds stable routing plans for the moving objects despite the fact that the traveling times of the moving objects are uncertain. Although the proposed formalism in this paper is general enough to model different types of applications, the motivation for this work is inspired from our earlier study on a special class of traffic control systems

that is called Track-based Traffic Control Systems (TTCSs). A TTCS safely navigates a set of moving objects on a track-based traveling space, in which the smallest safe regions are called sub-tracks (described in Section 2). The air traffic control systems, rail traffic control systems, maritime transportation, unmanned vehicles, robotic systems, etc., are various applications of TTCSs. For example, the air traffic control system in North Atlantic is based on an Organized Track System that has a track-based structure [12]. In [1], we provided a formal model of TTCSs using coordinated actors, where each sub-track is modeled by an actor, and moving objects are considered as messages communicated among the actors.

In this paper, we use the actor model to develop robust routing plans for moving objects in a TTCS by finding time windows. To this end, we adapt the model of tagged signals proposed in [16, 15] to make it appropriate for TTCSs. A tagged-signal model is a rigorous mathematical formalism that was proposed for comparing actor-oriented models of computations in [16]. An actor has interface components that are called ports. Actors communicate with each other by exchanging events through their ports. In fact, an actor receives and produces events on its ports. In the tagged-signal model, a port is associated with a *signal* that is a set of events. A signal records the history of communications between two actors. This way, the behaviors of an actor can be viewed as constraints on the signals that affect the actor and are affected by the actor. In the tagged-signal model of a TTCS, an event is generalized to model a moving object that can arrive at a sub-track within an arrival time interval (described in Section 3). We use structural and behavioral constraints of a TTCS to define the behaviors of actors (described in Section 4). This way, given the departure time intervals of moving objects from their sources, a set of signals satisfying the behaviors of actors is obtained. These signals encapsulate routes and arrival time intervals of the moving objects.

To evaluate the applicability of the provided semantics, we propose a formal model of TTCSs based on the tagged-signal model in Alloy [13] (described in Section 5). Alloy is a declarative language based on a first-order relational logic. It is augmented with automatic bounded verification using the Alloy analyzer. We choose Alloy as our modeling language because: (1) It is an expressive-enough language that allows capturing the topological and behavioral aspects of a TTCS in a single model, (2) Since it combines first-order logic with a relational algebra, it is appropriate for describing tagged-signal models, (3) Since the Alloy analyzer is able to generate instances that satisfy constraints of the model, it is useful for synthesizing routing plans with time windows.

There is a vast literature on the problem of path planning and scheduling of moving objects in different application domains. For instance, Yu and Lavalle find collision-free paths for n robots such that a cost function is minimized [18]. The problem of finding conflict-free motions for the vehicles traveling across an intersection is studied by Gregoire [10] and Dasler and Mount [9]. With the same aim, Altché, et al. obtain optimal velocity profiles for the vehicles traveling along their paths in an intersection [3]. Kinsy, et al. find deadlock-free routes for transferring the data in a network on chip with channel bandwidth

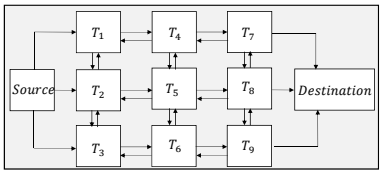


Fig. 1: A TTCS with a 3×3 mesh structure, a source, and a destination

constraints [14]. A formulation to obtain arrival time intervals of aircraft at sectors in their routes is proposed by Bertsimas, et al. [7]. Bansal, et al. use Hamilton-Jacobi reachability to obtain a set of collision-free configurations to transport the vehicles toward their destinations [5].

In addition to the above studies, there is a vast literature (i.e. [8, 17, 6, 2]) on the vehicle routing problem (VRP). A VRP with time windows is studied by Bettinelli, et al. [8], where routes are calculated such that the arrival time of each moving object at each node of its route is inside of the time window associated with the node. A robust VRP is studied by Ordóñez [17] and Agra, et al. [2]. The source of the uncertainty considered by Ordóñez in [17] can be the traveling time, costs, demands, etc.

Considering the aforementioned works, most of the works in this domain either employ optimization techniques (variants of liner programming formulations in [18, 3, 14, 7, 8, 17, 6, 2]) or do not deal with the scheduling problem with time windows. Even the work on the robust VRP assumes that time windows are inputs of the problem. To the best of our knowledge, this paper is the first work on scheduling/routing the moving objects with time windows that develops a rigorous mathematical semantics for TTCSs and implements it in Alloy.

2 Problem Definition

Track-based Traffic Control Systems (TTCSs) are large-scale safety-critical systems that consist of a set of moving objects which should be safely navigated [4]. A TTCS works based on the track-based design of the traveling space. In fact, to reduce the risk of collisions between the moving objects, the traveling space is divided into smaller safe regions that are called tracks. Based on the safe distance between the moving objects, each track is divided into a set of sub-tracks. Each sub-track is a critical section that accommodates only one crossing moving object. A TTCS with a 3×3 mesh structure, a source, and a destination is shown in Fig. 1. Each square denotes a sub-track, and interconnections between the squares denote pathways between the sub-tracks.

Each moving object in a TTCS starts its journey from a source and has a route. A route is a sequence of sub-tracks traveled by the moving object toward a destination. A route is a path on the structure of a TTCS (***appropriate routing*** constraint) and is free of any cycles (***avoiding circularity*** constraint). The route of the moving object is updated to a new route whenever the moving object

travels across a sub-track (***updating route***). In fact, the current sub-track is removed from the route of the moving object. Due to the uncertainty in traveling times of moving objects through the sub-tracks, we assign an arrival time interval to each moving object for each sub-track in its route. This time interval shows all the possible times at which the moving object may arrive at the sub-track. During a time interval where a sub-track is assigned to a moving object, arrivals of other moving objects at the sub-track are prohibited (***avoiding conflict*** constraint). The departure time interval of a moving object from a sub-track is its arrival time interval at the next sub-track in its route, and is affected by the uncertain traveling time of the moving object across the sub-track (***computing departure time interval***). In real-world applications of TTCSs, simultaneous entries of several moving objects into a sub-track are prevented by a policy in which the moving objects enter into the sub-track with some predefined priority. The priority in our approach is defined based on the entry direction. Our policy gives the highest priority to the moving object coming from the north. Then, the moving object coming from the west has the next highest priority, followed by an object coming from the south. Objects coming from the east have lowest priority. The policy could also be defined based on the features of the moving objects, e.g. their fuel level.

3 Time Interval Tagged Signals for TTCSs

The tagged signal model was introduced by Liu and Lee to provide a rigorous semantics for the actor model of a timed system [16]. In this section, we adapt the tagged signals to the domain of TTCSs. We explain how tagged signals are defined for the actor model of a TTCS. In a tagged signal model, each discrete communication between two actors is called an event that is a pair $(t, v) \in T \times V$, where $t \in T$ is a tag and $v \in V$ is a value. In a common used of the tagged signal model, the tag t marks the time at which the value v is communicated between two actors. A signal is a subset of $T \times V$ that represents a sequence of communications between two actors. In such a timed use of the tagged signal model, the set T is a totally-ordered set representing time.

In [1], we introduced coordinated actors to model a TTCS. Each actor models a sub-track, and the moving objects are modeled as messages communicated among actors. As actors communicate through their ports, the interconnections between actors represent pathways between the sub-tracks. In TTCSs, each sub-track is occupied with different moving objects in different times. Each message carries information such as the identifier and the route of a moving object (extra information such as the speed, fuel, etc., can also be carried by a message). Since we intend to derive time intervals for arrivals of the moving objects at sub-tracks, here instead we tag messages with time intervals. Hence, an event denotes a moving object with all possible times for its arrival at a sub-track. For instance, the event $e = ([2, 5], m)$ denotes that the moving object, modeled by the message m, enters into the sub-track corresponding to the receiver actor at a time within the interval $[2, 5]$.

The set T of tags in our model of a TTCS becomes a set of intervals. We assume that intervals in our model are closed. Let W be a set of non-negative integers with the usual numerical order. A closed interval $X = [\underline{x}, \overline{x}]$ is interpreted as the set $X = \{x \in W \mid \underline{x} \leq x \leq \overline{x}\}$ [11]. We define two different binary relations $\prec$ (*precede*), and $<_\delta$ (*before-with-δ-difference*) on the set T of intervals as follows.

Definition 1. *(Relation $\prec$):* $\forall X = [\underline{x}, \overline{x}], Y = [\underline{y}, \overline{y}] \in T,\ X \prec Y \Leftrightarrow \overline{x} \leq \underline{y}$. □

The relation $X \prec Y$ means that the entire time interval X precedes the start of the time interval Y.

Definition 2. *(Relation $<_\delta$):* $\forall X = [\underline{x}, \overline{x}], Y = [\underline{y}, \overline{y}] \in T,\ X <_\delta Y \iff \underline{x} + \delta \leq \underline{y} \wedge \overline{x} + \delta \leq \overline{y}$. □

The relation $X <_\delta Y$ means that the interval X starts and ends at least delta (δ) before Y starts and ends. In our model, the tag set T is defined as follows:

Definition 3. *(Tag): A set T of tags is defined as* $\{[\underline{x}, \overline{x}] \mid \underline{x}, \overline{x} \in W, \underline{x} \leq \overline{x}\}$. □

We define the set V of values in the model of a TTCS as a subset of all possible messages communicated between actors. Let I be the set of identifiers of sub-tracks, and M be the set of identifiers of moving objects. We use the route function $R : N \to I$ to define the route of a moving object. The function R is a total function for some N that is a lower subset of the natural numbers. A route is a set of tuples $(n, j), n \in N, j \in I$, such that n shows the position (index) of the sub-track with the identifier j in the route. We use $\mathscr{R}$ to denote the set of all route functions. A set of all messages is defined as follows.

Definition 4. *(Message): The set $M \times \mathscr{R}$ is the set of all messages.* □

Therefore, we have $V \subseteq M \times \mathscr{R}$. The same as [16], we define an event as a pair $(t, v) \in T \times V$. For example, the event $e = ([2, 5], (i, \{(1, j_1), (2, j_2)\}))$ denotes that the moving object i with the route $\{(1, j_1), (2, j_2)\}$ is able to enter into a sub-track at a time within $[2, 5]$. Two events are ordered by ordering of their tags; i.e. $\forall e = (t, v), e' = (t', v') \in T \times V, e \prec e' \iff t \prec t'$. In our model, signals are defined on a totally ordered set T. We use $\mathcal{S}$ to denote the set of all signals with the tag set T and the value set V.

4 Signal Model of TTCSs

In this section, we define the set of possible behaviors for an actor in the model of a TTCS. Based on [16], the behaviors of an actor can be seen as a set of constraints on the signals associated with its input and output ports. We use the constraints explained in Section 2 to define the behaviors of an actor. Therefore, a solution to our routing problem is found if, given the departure time intervals of moving objects from their sources, a signal is associated with each port of the model such that the behaviors of all actors are preserved.

Each actor in the model of a TTCS communicates with several actors. Suppose that $\mathcal{P}_{\mathcal{I}} = \{jin, liw, mis, kie\}$ and $\mathcal{P}_{\mathcal{O}} = \{ijn, ilw, ims, ike\}$ respectively denote the sets of input and output ports of the actor $i \in I$. The signal associated with the port jin is denoted by s_{jin}. The signal s_{jin} shows that the actor i receives a set of messages at different times from its northerly neighbor (n) that is actor j. The letters n, w, s, and e are symbols of the north, west, south, and east port of the actor. The set $\mathcal{S}^{\mathcal{P}_{\mathcal{I}}}$ (resp. $\mathcal{S}^{\mathcal{P}_{\mathcal{O}}}$) consists of all the sets in which a signal is associated with each port in the sort $\mathcal{P}_{\mathcal{I}}$ (resp. $\mathcal{P}_{\mathcal{O}}$). In fact, $\sigma \in \mathcal{S}^{\mathcal{P}_{\mathcal{I}}}$ (resp. $\mathcal{S}^{\mathcal{P}_{\mathcal{O}}}$) is a function mapping an input port (resp. output port) into a signal. We remind that $\mathcal{S}$ is the set of all signals, a signal is a set of events, and an event is a tag-value pair where the tag is an interval.

The actor i has a set B_i of behaviors, where $B_i \subseteq \mathcal{S}^{\mathcal{P}_{\mathcal{I}}} \times \mathcal{S}^{\mathcal{P}_{\mathcal{O}}}$. Before formally defining the possible behaviors of an actor, we define the functions and notations that are used in the formal definition of the constraints. For event $e = (t, v) \in T \times V$, where the tag $t = (\underline{t}, \overline{t})$ and the value $v = (m, R)$, we use $interval(e)$ to return t, $value(e)$ to return v, $\underline{x}(e)$ to return $\underline{t}$, $\overline{x}(e)$ to return $\overline{t}$, $id(e)$ to return m, $route(e)$ to return R, and $firstStrackId(e)$ to return the identifier of the first sub-track in the route of the moving object. The set of sub-tracks in the route $R : N \to I$ is denoted by $R \upharpoonright I = \{j | (n, j) \in R\}$. We use $|N|$ to denote the size of the set N. Furthermore, for some $j \in I$ and $m \in M$, $\delta_{m,j-tra}$ is the minimum traveling time of the moving object m across the sub-track j, and $\delta_{m,j-blo}$ is the maximum blocking time of the sub-track j for m. The two final notations are used for calculating the departure time interval of a moving object from a sub-track. For example, the departure time interval of m from the sub-track i for $[a, b]$ as the arrival time interval, $\delta_{m,j-tra} = 1$, and $\delta_{m,j-blo} = 4$ is $[c, d] = [a + 1, a + 4]$. This indicates that the object can spend between 1 and 4 time units in the track. In the following, for $\sigma \in \mathcal{S}^{\mathscr{P}}$ ($\mathscr{P} = \mathcal{P}_{\mathcal{I}}, \mathcal{P}_{\mathcal{O}}$), $Dom(\sigma)$ is the set of all tags in all signals $s_{abc} \in \sigma$, and $Events(\sigma)$ is the set of all events in all signals $s_{abc} \in \sigma$.

Avoiding conflict . Consider that the moving objects with the identifiers id_1 and id_2 cross over the sub-track i, while the arrival time interval of id_1 at the sub-track i precedes the arrival time interval of id_2 at the sub-track i. To avoid the conflict, the moving object id_2 can enter into the sub-track i at or after the time the moving object id_1 departs from it. Therefore, the departure time interval of the moving object id_1 from the sub-track i precedes the arrival time interval of the moving object id_2 at the sub-track i. In our notation,

$$\begin{aligned} \forall(\sigma_1, \sigma_2) \in B_i, \forall e_1, e_2 \in Events(\sigma_1), interval(e_1) \prec interval(e_2) \wedge \\ id_1 = id(e_1) \wedge id_2 = id(e_2), \\ \exists e_1' \in Events(\sigma_2), id(e_1') = id_1 \wedge interval(e_1') \prec interval(e_2). \end{aligned} \quad (1)$$

In other words, the moving objects cannot occupy the sub-track i at the same time. This implies the weaker condition that the intersection of their arrival time intervals is empty, i.e. $\forall t_1, t_2 \in Dom(\sigma_1), t_1 \neq t_2, t_1 \cap t_2 = \emptyset$.

Appropriate routing . A route is valid if for every pair of tuples (n, j) and $(n+1, k)$ in the route there is a connection from the sub-track j to the sub-track k in the structure of the traffic network. To guarantee that there is a connection between two subsequent sub-tracks in a route we require that:

$$\forall(\sigma_1, \sigma_2) \in B_i, \forall e \in Events(\sigma_1), (1, i) \in route(e). \tag{2}$$

To check the connection, it is sufficient to check whether $(1, i)$ belongs to the route.

Avoiding circularity . A route is free of any cycles if there is no repetitive sub-track in the route. In our notation,

$$\forall(\sigma_1, \sigma_2) \in B_i, \forall e \in Events(\sigma_1), \forall(n_1, j), (n_2, k) \in route(e), \\ n_1 \neq n_2 \Rightarrow j \neq k. \tag{3}$$

Updating route . Whenever a moving object enters into a sub-track, its route is updated to a new route. The new route does not contain the identifier of the current sub-track accommodating the moving object. To update the route of the moving object id traveling across the sub-track i, the first tuple of the route that is $(1, i)$ is removed from the route:

$$\forall(\sigma_1, \sigma_2) \in B_i, \forall e \in Events(\sigma_1), id = id(e), \exists e' \in Events(\sigma_2), \\ value(e') = (id, route) \wedge route : N' \rightarrow ((route(e) \upharpoonright I) \setminus \{i\}) \\ \wedge |N'| = |route(e)| - 1, \tag{4}$$

where N' is a lower subset of the natural numbers.

Computing departure time interval . The following constraints define how the departure time interval of the moving object departing from the sub-track i is determined. The arrival time interval of the moving object id_1 at the sub-track i is before (with-$\delta_{id_1,i-tra}$-difference) its departure time interval from the sub-track i, since the traveling time of the moving object across the sub-track is at least $\delta_{id_1,i-tra}$:

$$\forall(\sigma_1, \sigma_2) \in B_i, \forall e \in Events(\sigma_1), id_1 = id(e), \\ \exists e' \in Events(\sigma_2), id(e') = id_1 \wedge interval(e) <_{\delta_{id_1,i-tra}} interval(e'). \tag{5}$$

We can now specify some constraints on behaviors.

Constraint 1. Each possible departure time x belongs to an interval that is calculated based on the arrival time interval, the minimum traveling time, and the maximum blocking time as follows:

$$x \in [\underline{x}(e) + \delta_{id_1,i-tra}, \underline{x}(e) + \delta_{id_1,i-blo}] \tag{const.1}$$

Constraint 2. Consider that besides the moving object id_1, the moving object id_2 crosses over the sub-track i, while the arrival time interval of id_1 at the sub-track i precedes the arrival time interval of id_2. Since the departure time interval of a moving object shows all the possible departure times, none of the departure times of the moving object id_1 from the sub-track i belongs to the arrival time interval of the moving object id_2 at the sub-track i. Furthermore, each possible departure time of the moving object id_1 is less than each possible arrival time of the moving object id_2. In our notation,

$$\begin{aligned}\forall e_1, e_2 \in \mathit{Events}(\sigma_1), id_1 = id(e_1) \wedge id_2 = id(e_2) \wedge \\ \mathit{interval}(e_1) \prec \mathit{interval}(e_2), \exists e'_1 \in \mathit{Events}(\sigma_2), id(e'_1) = id_1 \wedge \\ \forall x \in \mathit{interval}(e'_1), x \notin \mathit{interval}(e_2) \wedge \forall x' \in \mathit{interval}(e_2), x < x' \quad \text{(const.2)}\end{aligned}$$

Constraint 3. To calculate the departure time interval of id_1 from the sub-track i, the arrivals of other moving objects at the next sub-track in the route of the moving object id_1 should be considered. To avoid the conflict, the order in which the moving objects can enter into the next sub-track is determined based on a policy. We use the same policy explained in Section 2. The arrival time interval of the moving object id_1 precedes the arrival time interval of a moving object id_2 if compared to id_2, id_1 has the highest priority for its arrival at the next sub-track. Let h be the next sub-track ($h = e'_1.value.route.first.strackId$), and $\mathcal{P}'_{\mathcal{I}} = \{ahn, ihw, bhs, che\}$ and $\mathcal{P}'_{\mathcal{O}} = \{han, hiw, hbs, hce\}$ be respectively the sets of input and output ports of h, where $\{a, b, c\} \subseteq I$. We assume that the moving object id_1 enters into the next sub-track in its route from the west. In our notation,

$$\begin{aligned}\forall e_1 \in \mathit{Events}(\sigma_1), id_1 = id(e_1), \exists e'_1 \in \mathit{Events}(\sigma_2) \wedge id(e'_1) = id_1, \\ h = \mathit{firstStrackId}(e'_1), \forall (\sigma'_1, \sigma'_2) \in B_h, \\ \sigma'_1 = \{s'_{ahn}, s'_{ihw}, s'_{bhs}, s'_{che}\} \wedge e'_1 \in s_{ihe} = s'_{ihw} \wedge s_{ihe} \in \sigma_2 \wedge \\ \forall e \in s'_{ahn}, [\underline{x}(e_1) + \delta_{id_1,i-tra}, \underline{x}(e_1) + \delta_{id_1,i-blo}] \cap \mathit{interval}(e) \neq \emptyset \Rightarrow \\ \forall x \in \mathit{interval}(e'_1), x \notin \mathit{interval}(e) \wedge \forall x' \in \mathit{interval}(e), x' < x \wedge \\ \forall e \in s'_{bhs}, s'_{che}, [\underline{x}(e_1) + \delta_{id_1,i-tra}, \underline{x}(e_1) + \delta_{id_1,i-blo}] \cap \mathit{interval}(e) \neq \emptyset \Rightarrow \\ \forall x \in \mathit{interval}(e'_1), x \notin \mathit{interval}(e) \wedge \forall x' \in \mathit{interval}(e), x < x' \quad \text{(const.3)}\end{aligned}$$

The similar constraints are defined if the moving object id_1 enters into the next sub-track in its route from the other sides. As explained in the lines 4 and 5 of (const.3), the departure time interval of the moving object id_1 from the sub-track i is refined if it intersects with the arrival time interval of the moving object coming from the north at the sub-track h. After the refinement, the departure time interval of the moving object id_1 contains the times that are greater than the possible arrival times of the moving object coming from the north.

To sum up, the departure time interval of the moving object id_1 from the sub-track i is valid whenever the three constrains (const.1), (const.2), and (const.3)

are satisfied:

$$\forall e_1 \in Events(\sigma_1), id_1 = id(e_1), \exists e_1' \in Events(\sigma_2), \\ interval(e_1') = \{x | \text{const.1} \wedge \text{const.2} \wedge \text{const.3}\}. \quad (6)$$

5 Alloy Model

We implement the signal model of TTCSs for a small, proof-of-concept test case in Alloy and use the Alloy analyzer to generate safe routing plans with time windows. The goal of the test case is twofold: first, it makes concrete the abstract model by mapping it onto an executable constraint language, and second, it tests whether Alloy might provide a solution technique for finding feasible routes.

The Alloy language is based on a first-order relational logic that allows to describe the behaviors of a software system through a set of constraints. Like object-oriented languages, a system in Alloy is described as a set of types called signatures. A signature defines a set of objects and may contain several fields. Each field is a relation that relates objects of the signature to the field expression. Alloy allows to put some constraints on the objects of a signature to make sure that they behave as expected. These constraints are assumed to hold for every satisfying instance of the model.

The signature declaration in Alloy is provided by the *sig* keyword. A signature can *extend* another signature to become its subtype. An *abstract* signature only contains the elements that belong to its extensions. A signature that has only one element is marked *one*.

The types used in the Alloy model of a TTCS are declared in Listing. 1.1. We introduce a set of totally ordered elements as the signature *Time* to model the bounds of a time interval (line 2). The linear ordering over the objects of the Time signature is established by using the *ordering*[*Time*] expression (line 1). The *Event* signature models the set of events in the tagged-signal model (line 4). The *MovingObject* signature introduces a set of objects, each one representing a moving object (line 3). We assume that the system has only one source and one destination for the moving objects. The source and destination are defined using the signatures *Source* and *Dest*, respectively (lines 6 and 7). A separate signature is declared for each sub-track of the model, i.e. *T1* and *T9* respectively define sub-tracks with the identifiers 1 and 9, with assuming that the system has 9 sub-tracks (lines 8-10).

```
1  open util/ordering[Time]
2  sig Time {}
3  abstract sig MovingObject {}
4  sig Event {...}{...}
5  abstract sig SubTrack {...}{...}
6  one sig Source extends SubTrack {}{...}
7  one sig Dest extends SubTrack {}{...}
8  one sig T1 extends SubTrack {}{...}
9  ...
10 one sig T9 extends SubTrack {}{...}
11 one sig A1, A2, A3, A4 extends MovingObject {}
```

Listing 1.1: The signatures declared in the Alloy model of a TTCS

The signatures *A1*, *A2*, *A3*, and *A4* define four different moving objects in the model (line 11). The details of each signature are given in the following sections.

5.1 Events in the Alloy Model of a TTCS

The Alloy declaration of the events is presented in Listing. 1.2. Each event, belonging to the set Event, contains the fields a and b to define the beginning and the end of a time interval (line 2), a moving object (line 3), and a route that is a sequence of sub-tracks (line 4). Each event should satisfy a set of constrains on its time interval and route fields. These constraints are placed in curly braces immediately after the signature *Event*. A route is free of any cycles (lines 5-6). In fact, there are no two disjoint indexes with identical sub-tracks in a route. Also, the start point of an interval is less than its end point (line 7).

```
1 sig Event {
2   a, b : Time,
3   movingObject : MovingObject,
4   route : seq SubTrack,  }{
5   no disj i1, i2 : route.inds |
6     route[i1] = route[i2]
7     lt[a, b]  }
```

Listing 1.2: Definition of the events in Alloy

5.2 Sub-tracks in the Alloy Model of a TTCS

The Alloy declaration of sub-tracks is presented in Listing. 1.3. An actor corresponding to a sub-track communicates with a set of actors through its input and output ports (line 2). Each port is associated with a signal that is a set of events (lines 4-5). The *priority* field stores the priorities of the moving objects arriving simultaneously at the sub-track from different directions (line 3). It is assumed that for $(i_1, i_2) \in priority$, the moving object coming from i_1 has a higher priority to enter into the sub-track over the moving object coming from i_2. According to the appropriate routing constraint, the head element of the route encapsulated by each input signal event of an actor refers to that actor (line 8). The interconnections between the actors imply that each input signal event is an output signal event of a connected actor (line 9).

```
1 abstract sig SubTrack {
2   input, output : set SubTrack,
3   priority : input -> input,
4   inSignal : input -> Event,
5   outSignal : output -> Event,
6 }{
7   all i : input, ie : inSignal[i] {
8     ie.route[0] = this
9     ie in i.@outSignal[this]    }
10  this not in Source implies
11    all i : input, ie : inSignal[i] | some in2out[ie][this]
12  this not in Dest implies
13    all i : input, ie : inSignal[i] | some in2out[ie][this]
14 }
```

Listing 1.3: Definition of the sub-tracks in Alloy

Suppose that the actor is not source of the moving objects (11). In this case, there is a relation between the input and output signal events of an actor (line 12). The same argument is valid if the actor is not a destination (lines 13-14).

The function *in2out* is defined in Listing. 1.4. A function, denoted by the *fun* keyword, returns a value if inputs of the function satisfy all the constraints explained in its body. The *in2out* function describes the behaviors of a sub-track. According to this function, an input signal event *ie* of the sub-track *t* has a corresponding output event *oe* such that the predicate *nextEvent* is satisfied. A predicate, denoted by the *pred* keyword, defines a set of constraints. A predicate evaluates to true if its inputs satisfy all the constraints explained in its body. Otherwise, it evaluates to false. The predicate *nextEvent* describes the relation between two events *e1* and *e2*. The event *e2* is derived from *e1* by removing the first element in the route of *e1* (line 11). Furthermore, the beginning and the end of the time interval of e1 are respectively less than the beginning and the end of the time interval of *e2* (lines 12-13).

```
1  fun in2out :  Event -> SubTrack -> Event {
2    { ie : Event, t : SubTrack, oe : Event |
3      some i : t.input, o : t.output {
4        ie in t.inSignal[i]
5        oe in t.outSignal[o]
6        nextEvent[ie, oe]
7    }}}
8
9  pred nextEvent[e1, e2 : Event] {
10   e2.movingObject = e1.movingObject
11   e2.route = e1.route.rest
12   lt[e1.a, e2.a]
13   lt[e1.b, e2.b]
14 }
```

Listing 1.4: Constraints to define the behaviors of a sub-track

A TTCS is safe whenever no conflict happens on the sub-tracks of the system. The predicate *conflict*, declared in Listing. 1.5, describes when a conflict on a sub-track happens. Consider two different input signal events *e* and *e'* of a sub-track (line 2), where the time tag of *e'* starts before the time tag of *e* (line 3). A conflict happens when the time tag of the event corresponding to *e'* in the output signals of the sub-track does not precede the time tag of e (line 4).

```
1 pred conflict[t : SubTrack] {
2   some disj e, e' : t.inSignal[t.input] {
3     lte[e'.a, e.a]
4     not lt[in2out[e'][t].b, e.a]
5   }}
```

Listing 1.5: Definition of a conflict in the Alloy model of a TTCS

The *fixDuration* predicate in Listing. 1.6 determines how the departure time interval of a moving object from a sub-track is calculated based on the arrival time interval of the moving object at the sub-track, the minimum traveling time, and the maximum blocking time. The min and max arguments of this predicate are the relations from Time to Time that capture the minimum traveling time and the maximum blocking time, respectively (line 1). For instance, for $(j, k) \in$

min, $k - j$ is equal to the minimum traveling time. For the event i, $i.a = j$ results in $i.a.min = k$. This way, the possible values for the departure times of a moving object from a sub-track are calculated (lines 4-6).

```
1 pred fixDuration[min, max : Time -> Time] {
2   all t : SubTrack, i, o : Event |
3     i -> t -> o in in2out implies
4       o.a = i.a.min and
5       gte[o.b, i.b.min] and
6       lte[o.b, i.a.max]          }
```

Listing 1.6: Calculating the departure time interval from a sub-track

As explained in Section 2, based on traveling directions towards a sub-track, we define a policy to prevent the moving objects from arriving simultaneously at a sub-track. In our policy, the moving object coming from the north has the highest priority for entering into the sub-track. Then, the moving object coming from the west has priority over the moving objects coming from the south and the east. Finally, the moving object coming from the south has priority over one coming from the east. This policy is modeled by the *policy* predicate in Listing. 1.7. It explains that the incoming moving object from *i1* has priority over the moving object coming from *i2* (line 3) if *i1* is in the north side (line 4). The other directions are defined similarly (lines 5-6). It is notable that the example of Listing. 1.1 has only one source and consequently, has one flow direction from the west to the east. Therefore, no moving objects come from the east side.

```
1 pred policy {
2   all t : SubTrack, disj i1, i2 : t.input |
3     i1 -> i2 in t.priority iff {
4       i1 in t.N or
5       (i1 in t.W and i2 not in t.N) or
6       (i1 in t.S and i2 not in (t.N + t.W))
7     }}
```

Listing 1.7: Definition of the priorities for arriving into a sub-track in Alloy

5.3 Analysis

In this section, we first explain a particular topology that is used in our analysis. We then initialize the model and explain the predicates that should not be violated in the model. Afterward, we use the Alloy analyzer to generate the safe routing plans. We assume that the system has only one source and one destination for the moving objects, declared in Listing. 1.8 (lines 1 and 7). The *Source* and *Des* signatures are subtypes of the *Sub-track* signature.

```
1 one sig Source extends SubTrack {}{
2   no input
3   all a : MovingObject |
4     some o : output, oe : outSignal[o] {
5       oe.movingObject = a  }
6   }
7 one sig Dest extends SubTrack {}{
8   no output }
```

Listing 1.8: Definition of the source and destination in Alloy

The source does not have any input (line 2), and the destination does not have any output (line 8). Furthermore, every moving object appears at an output signal of the source (lines 4 and 5).

We consider a TTCS with a 3×3 mesh structure. A separate signature for each sub-track of the model is declared in Listing. 1.9. We only show the signatures of T_1 and T_7 (lines 1 and 4). The connections between the sub-tracks are determined by defining the input and output fields of the *SubTrack* signature (lines 2, 3, 5, and 6). For instance, the sub-track *T1* has connections from *T2*, *Source*, and *T4* (line 2). The definition of the structure of a TTCS is completed by declaring the functions *N*, *S*, *W*, and *E* (lines 7-12). These functions respectively define the north, south, west, and east neighbors of a sub-track. As an instance, the sub-track *T1* is a north neighbor of *T2* (line 8), and so on.

```
1  one sig T1 extends SubTrack {}{
2    input = T2 + Source + T4
3    output = T2 + T4                }
4  one sig T7 extends SubTrack{} {
5    input = T4 + T8
6    output = T4 + T8 + Dest       }
7  fun N : SubTrack -> SubTrack {
8    T2 -> T1 + T5 -> T4 + T8 -> T7 +
9    T3 -> T2 + T6 -> T5 + T9 -> T8      }
10 fun S : SubTrack -> SubTrack {...}
11 fun W : SubTrack -> SubTrack {...}
12 fun E : SubTrack -> SubTrack {... }
```

Listing 1.9: The 3×3 mesh map of a TTCS in Alloy

The initial conditions of the Alloy model are defined in Listing. 1.10. The elements *A1*, *A2*, *A3*, and *A4* are four different moving objects. The events, referring to these moving objects, are assigned to the output ports of the source. The departure time interval of a moving object from its source is defined by accessing the totally ordered elements of the *Time* signature. The *first* keyword in Listing. 1.10 returns the first element of the Time signature and *first.next* returns the element placed after the first element. Based on this definition, the departure time interval of A1 precedes the departure time interval of A2, the departure time interval of A2 precedes the departure time interval of A3, and the departure time interval of A3 precedes the departure time interval of A4 (lines 3-6).

```
1 pred initConditions {
2   all t : Source.output, e : Source.outSignal[t] {
3     e.movingObject = A1 implies (e.a = first and e.b = e.a.next)
4     e.movingObject = A2 implies (e.a = first.next.next and e.b = e.a.next)
5     e.movingObject = A3 implies (e.a = first.next.next.next.next and e.b =
      ↪  e.a.next)
6     e.movingObject = A4 implies (e.a = first.next.next.next.next.next.next
      ↪  and e.b = e.a.next)  }
7 }
```

Listing 1.10: The initial conditions of the Alloy model

As mentioned in Section 2, whenever several moving objects arrive simultaneously at a sub-track, we use a policy to order the moving objects for entering into the sub-track. The predicate *prioritiesHold* defined in Listing. 1.11, returns

true if there is no sub-track whose input signals violate the priorities defined for the incoming moving objects. Suppose that the moving object coming from *i1* has priority over the moving object coming from *i2*, i.e. $(i1, i2) \in priority$ for the sub-track t (line 6). The priorities are violated if the time tag of *e1* in *i1* does not begin before the time tag of the event *e2* in *i2* (as explained in line 8, the time tags overlap, where overlapping is defined in line 1).

```
1 pred overlap[e1, e2 : Event] {
2   lte[e1.a, e2.a]
3   lt[e2.a, e1.b]     }
4 pred prioritiesHold {
5   no t : SubTrack, i1, i2 : t.input |
6     i1 -> i2 in t.priority and
7     some e1 : t.inSignal[i1], e2 : t.inSignal[i2] |
8       overlap[e2,e1]    }
```

Listing 1.11: Holding priorities over each sub-track

As explained in Listing. 1.12, a TTCS is unsafe if a conflict happens on a sub-track (line 2) or the destination of the moving objects (line 3). There is a conflict on the destination if it has two input signal events with the overlapping time intervals (lines 4-5). The system is safe if the predicate *unsafe* evaluates to false (line 7).

```
1 pred unsafe {
2   (some t : SubTrack | conflict[t]) or
3   some disj e, e' : Dest.inSignal[Dest.input] {
4     lte[e'.a, e.a]
5     not lt[e'.b, e.a]
6   }}
7 pred safe { not unsafe }
```

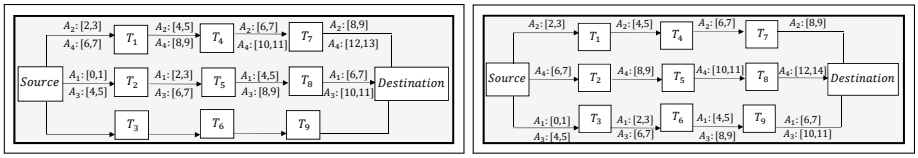
Listing 1.12: Defining the situations in which the system is unsafe

The only fixed constants in our Alloy model are the departure time intervals of the moving objects from the source, the minimum traveling time, and the maximum blocking time. The Alloy analyzer generates all the possible plans for the moving objects that satisfy the constraints on the model. A moving object plan includes a route from the source to the destination along with the arrival time interval of the moving object at each sub-track in its route. The properties of our interest are defined in Listing. 1.13.

```
1 run findSafePlan {
2   safe
3   policy
4   initConditions
5   fixDuration[next.next, next.next.next.next]
6   prioritiesHold
7 } for 1 but 20 Time, 16 Event, 4 seq
```

Listing 1.13: Generating the safe solutions to the model

The Alloy analyzer tries to find a solution to the model by executing the *run* command. It exhaustively searches the state space up to a bound to generate a counterexample to an assertion or to generate a satisfying instance to a predicate. The *findSafePlan* run command tries to find the safe solutions to the model. This command returns the solutions in which there are up to 20 *Time* objects,



(a) (b)

Fig. 2: Robust plans obtained by the Alloy analyzer. The connections are marked with the names and time intervals of the moving objects passing through them.

16 *Event* objects, and one object for each one of the other signatures (line 7). Furthermore, the length of the allowed sequences in the model (length of the routes) is bounded to 4. In this example, the minimum traveling time and the maximum blocking time, which are arguments of the *fixDuration* predicate, are set to 2 and 4, respectively (lines 5 and 12). We use the Alloy analyzer 4.2 on a windows machine with 4 GB RAM and Intel(R) Core(TM) i5-2410M CPU @ 2.30GHz for our experiments. When the *findSafePlan* run command is executed, an instance of the model is generated in around 4 minutes and 37 seconds. The solution provided by this instance is briefly shown in Fig. 2(a), where the routes of the moving objects are determined by marking the connections with the names and arrival time intervals of the moving objects. The Fig. 2(b) illustrate another instance of the *findSafePlan* run command.

In this section, we used the Alloy modeling language to implement the tagged-signal model of TTCSs. We then used the Alloy analyzer to evaluate the applicability of the proposed model. The results illustrate that the proposed formalism successfully contributes in finding safe routing plans with time windows for a set of moving objects. However, the Alloy analyzer has some limitations: it analyzes the model up to a scope consisting of a user-defined number of objects, and the time required to find instances of a complex model using this analyzer is overwhelming. Therefore, the Alloy analyzer due to its scalability limitations may not be a useful tool for obtaining routing plans in practice.

6 Conclusion and Future Work

We developed a semantics for the actor model of a TTCS based on the tagged-signal model. The behaviors of an actor in the tagged-signal model are explained through a set of constraints on the signals associated with input and output ports of the actor. The set of signals that satisfy the behaviors of all actors gives us routes of the moving objects along with their arrival time windows for each sub-track in their routes. We also provided a formal model of TTCSs in Alloy. We leveraged the capability of Alloy as a constraint solver to enumerate all possible instances and find a solution to the planning problem. In the future, we will develop a heuristic algorithm to reduce the time required for obtaining routing plans using the proposed tagged-signal model.

References

1. Coordinated actor model of self-adaptive track-based traffic control systems. Journal of Systems and Software **143**, 116 – 139 (2018)
2. Agra, A., Christiansen, M., Figueiredo, R., Hvattum, L.M., Poss, M., Requejo, C.: The robust vehicle routing problem with time windows. Computers and Operations Research **40**(3), 856 – 866 (2013), transport Scheduling
3. Altché, F., Qian, X., de La Fortelle, A.: Time-optimal coordination of mobile robots along specified paths. In: IROS. pp. 5020–5026 (2016)
4. Bagheri, M., Akkaya, I., Khamespanah, E., Khakpour, N., Sirjani, M., Movaghar, A., Lee, E.A.: Coordinated actors for reliable self-adaptive systems FACS 2016
5. Bansal, S., Chen, M., Fisac, J.F., Tomlin, C.J.: Safe sequential path planning under disturbances and imperfect information. In: 2017 American Control Conference (ACC). pp. 5550–5555 (May 2017)
6. Belenguer, J.M., Benavent, E., Prins, C., Prodhon, C., Calvo, R.W.: A branch-and-cut method for the capacitated location-routing problem. Computers and Operations Research **38**(6), 931 – 941 (2011)
7. Bertsimas, D., Lulli, G., Odoni, A.: The Air Traffic Flow Management Problem: An Integer Optimization Approach, pp. 34–46 (2008)
8. Bettinelli, A., Ceselli, A., Righini, G.: A branch-and-cut-and-price algorithm for the multi-depot heterogeneous vehicle routing problem with time windows. Transportation Research Part C: Emerging Technologies **19**(5), 723 – 740 (2011)
9. Dasler, P., Mount, D.M.: On the Complexity of an Unregulated Traffic Crossing, pp. 224–235. Springer International Publishing, Cham (2015)
10. Gregoire, J.: Priority-based coordination of mobile robots. CoRR **abs/1410.0879** (2014)
11. Hickey, T., Ju, Q., Van Emden, M.H.: Interval arithmetic: From principles to implementation. J. ACM **48**(5), 1038–1068 (Sep 2001)
12. International Civil Aviation Organization (ICAO): North atlantic operations and airspace manual (2016)
13. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. The MIT Press (2006)
14. Kinsy, M.A., Cho, M.H., Wen, T., Suh, E., van Dijk, M., Devadas, S.: Application-aware deadlock-free oblivious routing. SIGARCH Comput. Archit. News **37**(3), 208–219 (Jun 2009)
15. Lee, E.A., Sangiovanni-Vincentelli, A.: A framework for comparing models of computation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **17**(12), 1217–1229 (Dec 1998)
16. Liu, X., Lee, E.A.: Cpo semantics of timed interactive actor networks. Theoretical Computer Science **409**(1), 110 – 125 (2008)
17. Ordóñez, F.: Robust Vehicle Routing, chap. Chapter 7, pp. 153–178
18. Yu, J., LaValle, S.M.: Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. IEEE Transactions on Robotics **32**(5), 1163–1177 (Oct 2016)