A Calculus for Shapes in Time and Space

Andreas Schäfer

Department of Computing Science,
University of Oldenburg, 26111 Oldenburg, Germany
schaefer@informatik.uni-oldenburg.de

Abstract. We present a spatial and temporal logic based on Duration Calculus for the specification and verification of mobile real-time systems. We demonstrate the use of the formalism and apply it to a case study. We extend a pure Duration Calculus specification for the controller by spatial assumptions to reason about spatial system properties. We prove that the formalism is undecidable in general for discrete and continuous domains and present a decidable fragment.

Keywords: Real-time systems, mobile systems, spatial logic, temporal logic, Duration Calculus.

1 Introduction

There are many well understood formal techniques for the specification and verification of real-time systems, among them the interval temporal logic Duration Calculus (DC) [ZHR91, HZ04] or Timed Automata [AD94]. But often problems of safety critical real-time systems have a spatial nature. Consider for example two trams driving on the same track. This situation itself is not dangerous as long as they do not "overlap" or as long as the distance between two trams is sufficiently large. Another example is a robot moving around in a restricted area. A desirable safety property might be that the robot moves at most 5 cm outside its area so that it does not endanger staff.

These properties cannot be expressed concisely and nicely in standard temporal logics. This led us to the idea to extend a well known formalism for real-time systems to be able to describe spatial properties when needed. The use of the formalism should be similar to the use of pure temporal logics when no spatial reasoning is required.

In this paper we describe how to extend the temporal interval logic Duration Calculus [ZHR91] to a spatio-temporal interval logic. Instead of having one dimension for time, we allow an arbitrary number of dimensions from which a subset is considered to be spatial and a subset to be temporal. From the point of view of our formalism, it does not matter whether a dimension is temporal or spatial. If we just chose one (temporal) dimension, we get the normal Duration Calculus.

In section 2 we introduce the formalism and show in section 3 how it can be used to describe spatial properties such as movement of objects in space. In

Z. Liu and K. Araki (Eds.): ICTAC 2004, LNCS 3407, pp. 463–477, 2005.

section 4 we show how a controller design done in pure Duration Calculus can be extended to reason about the spatial properties of the whole system.

In section 5 we prove that the formalism is undecidable for dense and discrete time/space-domains before giving a decidable fragment in section 6.

2 Shape Calculus

In this section we define the shape calculus and show for each feature that the extension is conservative and we do not add expressive power to Duration Calculus if we only consider one dimension.

In Duration Calculus the behaviour of a system is modelled by a set of time-dependant variables whose value change in time. The natural extension for a spatial and temporal logic is to choose the dimension for space and for time and to use flexible variables whose value depend on the point in time and space. According to DC terminology we will call these variables *observables* and let *Obs* denote the set of all observables. The semantics of an observable X is given by a function $\mathcal{I}$

$$\mathcal{I}[\![X]\!] : \mathbb{R}^n_{\geq 0} \to \{d_0, \ldots, d_m\}$$

where $\{d_0, \ldots, d_m\}$ denotes the range of the observable. To guarantee that the integral exisits for this function, we require $\mathcal{I}[\![X]\!]$ to be continuous almost everywhere. If we choose $n = 1$ we end up with the Duration Calculus definition.

State Expressions. Properties of a point in space and in time are expressed by *state formulae* denoted by π and build from comparison of observables and boolean connectors.

$$\pi ::= X = d \,|\, \neg\pi_1 \,|\, \pi_1 \wedge \pi_2$$

where d is in the range of the observable X. The semantics is then given by a function

$$\mathcal{I}[\![\pi]\!] : \mathbb{R}^n_{\geq 0} \to \{0, 1\}$$

which is defined in the expected way. The definition is exactly like in Duration Calculus. If the observable X is of boolean type, we will write X for $X = 1$.

Terms. Interval temporal logic assigns a real value to every interval and Duration Calculus introduces the integral operator to measure the duration of a certain state in a given interval.

Instead of intervals we use bounded polyhedra for the n-dimensional case. Alternatively we could have used hypercubes, but as we shall see in the examples, specifications can be done much more conveniently with polyhedra. The set of bounded polyhedra will be denoted by *Poly*. One dimensional polyhedra are intervals.

In many cases, we are not interested in the spatio-temporal volume, but only in a spatial or a temporal measure. For example, it might be important how much of a vehicle is outside its working area. In this case we need an integral

for the spatial part. On the other hand we might be interested in the amount of time the system is moving, which is a temporal integral.

To this end, we allow linear transformations T of the function $\mathcal{I}[\![X]\!]$ before applying the integral. With these transformations we can for example achieve projections on all axes or on hyperplanes. We define the set of *terms* as follows:

$$\theta ::= \int \boldsymbol{T}\pi \mid x \mid \boldsymbol{d}.i \mid f(\theta_1, \ldots, \theta_k)$$

where π is a state assertion, $\boldsymbol{T}$ is a $m \times n$ matrix of real numbers, x a rigid variable of type real, $\boldsymbol{d}$ a variable whose type is a n-dimensional vector of reals, and f a function symbol, which can be an arithmetic function like $+$. As usual, the value of the rigid variable is determined by a valuation $\mathcal{V}$ and the set of valuations is denoted by Val. The value of a vector $\boldsymbol{d}$ is also given by the valuation and the components of $\boldsymbol{d}$ are accessed by indexing. So $\boldsymbol{d}.i$ returns the ith component of $\boldsymbol{d}$.

The semantics of terms is a function

$$I[\![\theta]\!] : Poly \times Val \to \mathbb{R}$$

and in particular the semantics of the integral of a state assertion π after a transformation T is defined using the characteristic function χ. This idea is sketched in figure 1 (a)-(c).

$$I[\![\textstyle\int \boldsymbol{T}\pi]\!](\mathcal{M}, \mathcal{V}) = \displaystyle\int_{\mathcal{M}} \chi_{\boldsymbol{T}(\mathcal{M} \cap \mathcal{I}[\![\pi]\!]^{-1}(1))}$$

where

$$\chi_{\boldsymbol{T}(\mathcal{M} \cap \mathcal{I}[\![\pi]\!]^{-1}(1))} = \begin{cases} \mathbb{R}^m \to \mathbb{B} \\ \boldsymbol{x} \mapsto \begin{cases} 1 & \text{if } \exists \boldsymbol{x}' \in \mathcal{M} : \boldsymbol{x} = \boldsymbol{T}\boldsymbol{x}' \wedge \mathcal{I}[\![\pi]\!](\boldsymbol{x}') = 1 \\ 0 & \text{otherwise} \end{cases} \end{cases}$$

For the 1-dimensional case, the linear transformation $\boldsymbol{T}$ is only scaling and $\int \boldsymbol{T}\pi = \boldsymbol{T} \int \pi$ holds. Thus in this case the transformation does not add expressive power and we still have the Duration Calculus.

Formulae. In temporal interval logic, the intervals can be "chopped" into a leftmost and a rightmost subinterval. In our case we introduce the chop operator $\langle \boldsymbol{d} \rangle$ to split the polyhedron along a given hyperplane, which results in two new sub-polyhedra. This is illustrated in figure 1 (d).

The set of *formulae* is given by

$$F ::= F_1 \langle \boldsymbol{d} \rangle F_2 \mid p(\theta_1, \ldots, \theta_k) \mid \neg F_1 \mid F_1 \wedge F_2 \mid \exists x : F \mid \exists \boldsymbol{d} : F$$

where p is a predicate symbol, x a rigid variable and $\boldsymbol{d}$ a rigid vector. The other boolean connectives can be defined as the usual abbreviations.

The semantics is a function

$$\mathcal{I}[\![F]\!] : Poly \times Val \to \mathbb{B}.$$

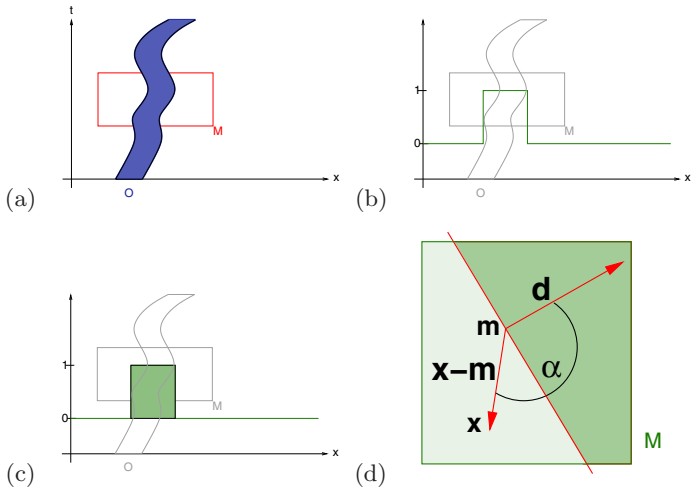

Fig. 1. (a) Function $\mathcal{I}[\![O]\!]$ with polyhedron $\mathcal{M}$, (b) the characteristic function $\chi_{\boldsymbol{T}(\mathcal{M}\cap\mathcal{I}[\![O]\!]^{-1}(1))}$ where $\boldsymbol{T} = e_x^T = (1,0)$ is the transposed unit vector for the spatial dimension. This is the projection onto the x-axis and (c) the integral $\int \chi_{\boldsymbol{T}(\mathcal{M}\cap\mathcal{I}[\![O]\!]^{-1}(1))}$. (d) illustrates the chop-operation

A formula $F_1 \langle \boldsymbol{d} \rangle F_2$ is valid if and only if there is a hyperplane orthogonal to the vector $\boldsymbol{d}$ such that the polyhedron $\mathcal{M}$ is split by this hyperplane into two polyhedra $\mathcal{M}_1$ and $\mathcal{M}_2$ which fulfil F_1 and F_2, respectively. So

$$\mathcal{I}[\![F_1 \langle \boldsymbol{d} \rangle F_2]\!](\mathcal{M}, \mathcal{V}) = true$$

iff there exists a $\boldsymbol{m} \in \mathcal{M}$ such that with $\mathcal{M}_1 \stackrel{df}{=} \{\boldsymbol{x} \in \mathcal{M} | \langle \boldsymbol{x} - \boldsymbol{m}, \boldsymbol{d} \rangle \leq 0\}$ and $\mathcal{M}_2 \stackrel{df}{=} \{\boldsymbol{x} \in \mathcal{M} | \langle \boldsymbol{x} - \boldsymbol{m}, \boldsymbol{d} \rangle \geq 0\}$ the following holds:

$$\mathcal{I}[\![F_1]\!](\mathcal{M}_1, \mathcal{V}) = true \text{ and } \mathcal{I}[\![F_2]\!](\mathcal{M}_2, \mathcal{V}) = true$$

Here $\langle \boldsymbol{x} - \boldsymbol{m}, \boldsymbol{d} \rangle$ denotes the scalar product of $\boldsymbol{x} - \boldsymbol{m}$ and $\boldsymbol{d}$ which is proportional to the cosine of the angle α between these vectors. Thus, it is negative iff α is greater than 180 degrees, i.e., the point $\boldsymbol{x}$ is below and positive otherwise. Note that the scalar product is bilinear. So scaling the vector $\boldsymbol{d}$ with positive reals does not change $\mathcal{M}_1$ or $\mathcal{M}_2$.

In the 1-dimensional case, when the vector used for the chop operation has only one dimension, there are only three different cases possible. All of them can be modelled in Duration Calculus using the DC chop operator ";" and conjunction.

$$\begin{aligned} F \langle 1 \rangle G &\iff F; G \\ F \langle 0 \rangle G &\iff F \wedge G \\ F \langle -1 \rangle G &\iff G; F \end{aligned}$$

So we have the conclusion.

Corollary 1. *For $n = 1$ the shape calculus and the Duration Calculus coincide.*

Note 1. The chop operation is associative for the same vector $\boldsymbol{d}$ but not for different vectors

$$(F \langle \boldsymbol{d} \rangle G) \langle \boldsymbol{d} \rangle H \iff F \langle \boldsymbol{d} \rangle (G \langle \boldsymbol{d} \rangle H)$$

but in general if $\boldsymbol{d_1}$ is not a multiple of $\boldsymbol{d_2}$

$$(F \langle \boldsymbol{d_1} \rangle G) \langle \boldsymbol{d_2} \rangle H \not\iff F \langle \boldsymbol{d_1} \rangle (G \langle \boldsymbol{d_2} \rangle H).$$

2.1 Abbreviations

Duration Calculus defines a lot of abbreviations to ease the handling of specifications. We adopt them directly.

$$\ell \stackrel{df}{=} \int 1 \qquad \ell_{\boldsymbol{T}} \stackrel{df}{=} \int \boldsymbol{T} 1 \qquad \ell_{\boldsymbol{d}} \stackrel{df}{=} \int (\frac{1}{\|\boldsymbol{d}\|} \boldsymbol{d}) 1$$

can be used to measures the size or diameter of the polyhedron. We denote by e_x and e_t the unit vectors for the x- respectively time dimension, by e_x^T respectively e_t^T their transposition, and we write $\ell_t \stackrel{df}{=} \int e_t^T 1$ to measure the size along the time axis and $\ell_x \stackrel{df}{=} \int e_x^T 1$ for the size along the x-axis.

Like in Duration Calculus, the chop along the time axis is written

$$F; G \stackrel{df}{=} F \langle \boldsymbol{e_t} \rangle G.$$

The everywhere operator $\lceil \pi \rceil$ expresses that a state assertion π holds almost everywhere in the polyhedron. It can be augmented by a transformation $\boldsymbol{T}$.

$$\lceil \pi \rceil \stackrel{df}{=} \int \pi = \ell \wedge \ell > 0 \qquad \lceil \pi \rceil_{\boldsymbol{T}} \stackrel{df}{=} \int \boldsymbol{T} \pi = \ell_{\boldsymbol{T}} \wedge \ell_{\boldsymbol{T}} > 0.$$

A polyhedron of length zero is also denoted by

$$\lceil \rceil_{\boldsymbol{T}} \stackrel{df}{=} \ell_{\boldsymbol{T}} = 0.$$

The somewhere operator $\Diamond F$ allows the polyhedron to be chopped twice in the same direction such that in the middle polyhedron F holds.

$$\Diamond_{\boldsymbol{d}} F \stackrel{df}{=} true \langle \boldsymbol{d} \rangle F \langle \boldsymbol{d} \rangle true$$

with the dual globally operator $\Box$

$$\Box_{\boldsymbol{d}} \stackrel{df}{=} \neg \Diamond_{\boldsymbol{d}} \neg F.$$

Note 2. Although the chop operation is not associative for different directions, the $\Diamond$ operation commutes

$$\Diamond_{\boldsymbol{d_1}} \Diamond_{\boldsymbol{d_2}} F = \Diamond_{\boldsymbol{d_2}} \Diamond_{\boldsymbol{d_1}} F$$

and because of duality also the $\Box$ operation does

$$\Box_{\boldsymbol{d_1}} \Box_{\boldsymbol{d_2}} F = \Box_{\boldsymbol{d_2}} \Box_{\boldsymbol{d_1}} F.$$

3 Application

Using the proposed framework, we can easily describe movement and connectivity of objects in space. Apart from that, we can describe and reason about the control of these objects in this framework, using the Duration Calculus part, as this is just a conservative extention. At first we present some examples describing how objects move in space. Assume one spatial dimension x and one temporal dimension t. We consider an object O which is modelled by an observable O such that the observable has value 1 for a point in time and space iff the object occupies this point in space for this point in time.

3.1 STOP

The simplest expression is that an object O with size s does not move in space.

$$\mathsf{STOP}(O,s) \stackrel{df}{=} (\lceil \neg O \rceil \vee \lceil \rceil_{\boldsymbol{e_x}})\ \langle \boldsymbol{e_x} \rangle\ (\lceil O \rceil \wedge \ell_x = s)\ \langle \boldsymbol{e_x} \rangle\ (\lceil \neg O \rceil \vee \lceil \rceil_{\boldsymbol{e_x}})$$

The idea of this formula is sketched in figure 2 and it reads as follows. The polyhedron (here it is a square) can be partitioned into three parts along the x-direction. In the first part, everywhere is not object O or it has size zero in x-direction. In the second part, O is true everywhere and this part has size s in x-direction. So O has constant size s over time. And in the third part, there is again no object O or it has size zero. As we chop the square orthogonal to x-axis the position of O must be constant.

3.2 Continuous Movement

The STOP formula can be generalised to describe continuous movement with velocity v. Here we chop orthogonal to the velocity vector and obtain three sub-polyhedra such that only in the middle one O is true everywhere. This is sketched in figure 2 (b).

$$\mathsf{MOTION}(O,v) \stackrel{df}{=} (\lceil \neg O \rceil \vee \lceil \rceil_{\binom{1}{-v}})\ \langle \binom{1}{-v} \rangle\ \lceil O \rceil\ \langle \binom{1}{-v} \rangle\ (\lceil \neg O \rceil \vee \lceil \rceil_{\binom{1}{-v}})$$

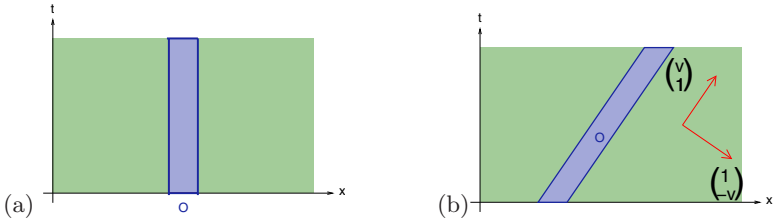

Fig. 2. (a) The object O remains immobile. (b) The object O moves continously, according to the pattern MOTION

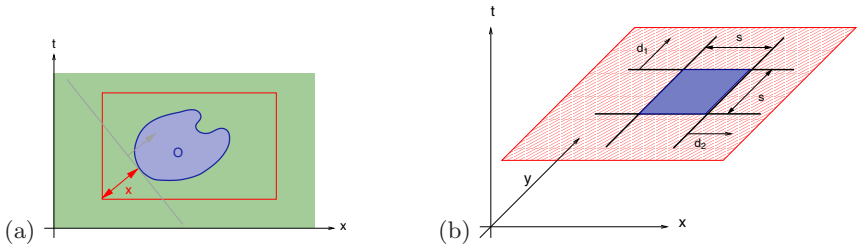

Fig. 3. (a) The object O has distance x in direction $\boldsymbol{d}$ from beginning of the polyhedron. (b) The object is a square

This does not specify where the object O starts its movement. A starting position can be defined by adding an additional constraint to MOTION, for example

$$\mathsf{MOTION}_0(O, v) \stackrel{df}{=} \mathsf{MOTION}(O, v) \wedge \neg(\lceil\neg O\rceil_x \langle e_x\rangle \mathit{true})$$

requires the object to start at point 0 on the x-axis.

3.3 Position of Objects

By taking the position of object O at a certain point of time, it is possible to define arbitrary movements. This is captured by

$$\mathsf{POS}_{\boldsymbol{d}}(O, x) \stackrel{df}{=} ((\lceil\neg O\rceil \vee \lceil\rceil_{\boldsymbol{d}}) \wedge \ell_{\boldsymbol{d}} = x) \langle \boldsymbol{d}\rangle \lceil O\rceil_{\boldsymbol{d}} \langle \boldsymbol{d}\rangle \mathit{true}.$$

In this formula we define x to be the maximal size in direction $\boldsymbol{d}$ of a sub-polyhedron in which O is not true. This is sketched in figure 3 (a).

3.4 Generalisation to More Than One Dimension

The MOTION formula given above easily generalises to more than one dimension. Assume that the movement of the object is defined by the vector $\boldsymbol{v} = (v_x, v_y, 1)^T$. Then

$$\mathsf{MOTION}_{3d}(O, \boldsymbol{v}) \stackrel{df}{=} \forall \boldsymbol{d}.\langle \boldsymbol{d}, \boldsymbol{v}\rangle = 0 \Rightarrow (\lceil\neg O\rceil \vee \lceil\rceil_{\boldsymbol{d}}) \langle \boldsymbol{d}\rangle (\neg \Diamond_{\boldsymbol{d}} \lceil\neg O\rceil) \langle \boldsymbol{d}\rangle (\lceil\neg O\rceil \vee \lceil\rceil_{\boldsymbol{d}})$$

describes the movement of an object in the plane, where $\langle \boldsymbol{d}, \boldsymbol{v}\rangle$ denotes the scalar product of $\boldsymbol{d}$ and $\boldsymbol{v}$.

3.5 Shape of Objects

The shape of objects at points in time can also be grasped. For example

$$\mathsf{Circle}(O, s) \stackrel{df}{=} \Box_t\big(\lceil\rceil_t \Rightarrow (\forall \boldsymbol{d}.(\lceil\neg O\rceil_{\boldsymbol{d}} \langle \boldsymbol{d}\rangle \lceil O\rceil_{\boldsymbol{d}} \wedge \ell_{\boldsymbol{d}} = s \langle \boldsymbol{d}\rangle \lceil\neg O\rceil_{\boldsymbol{d}}))\big)$$

requires that for every point in time, in the hyperplane which belongs to this point the object has size s in every possible direction. This is only true if the

object is a circle of diameter s and it does neither disappear nor does appear a second object of this kind.

The property that the object is a square with size s can be defined by the following formula. In figure 3 (b) we depict the idea of this definition.

$$\begin{aligned}\mathsf{Square}(O,s) \stackrel{df}{=} \Box_t\big(\lceil\rceil_t \Rightarrow (\exists \boldsymbol{d_1},\boldsymbol{d_2}.(\langle \boldsymbol{d_1},\boldsymbol{d_2}\rangle = \langle \boldsymbol{d_1},\boldsymbol{e}_t\rangle = \langle \boldsymbol{d_2},\boldsymbol{e}_t\rangle = 0 \wedge \\ (\lceil\neg O\rceil_{(e_x,e_y)^T} \\ \langle \boldsymbol{d_1}\rangle \\ \quad (\lceil\neg O\rceil_{(e_x,e_y)^T} \\ \quad \langle \boldsymbol{d_2}\rangle \lceil O\rceil_{(e_x,e_y)^T} \wedge \ell_{\boldsymbol{d_1}} = \ell_{\boldsymbol{d_2}} = s \\ \quad \langle \boldsymbol{d_2}\rangle \lceil\neg O\rceil_{(e_x,e_y)^T}) \\ \langle \boldsymbol{d_1}\rangle \\ (\lceil\neg O\rceil_{(e_x,e_y)^T}))))\big).\end{aligned}$$

4 Case Study: The Moving Robot (Roadrunner)

Now we apply the ideas of the previous section and specify a system using two spatial and one temporal dimensions. We consider a robot moving on an rectangular plateau. Assume that the robot is equipped with sensors all around to detect the edges. We do not consider the special task of the robot, but only model its movement and edge detection system. It is to be proven that the robot does not fall off the plateau.

Controller. A simple controller for the vehicle may consist only of two states and is sketched in figure 4. One state is *run*, in which the robot is moving forward. When the robot detects an edge, after a response time of δ_R ms, the robots starts to turn around for δ_T ms to rotate by angle π, before returning to state *run*.

In order to derive an implementation from the specification, a subset of DC called the DC-Implementables introduced by Ravn [Rav95] is important. We use the implementables to specify the controller and thus need to introduce the following abbreviation:

$$F \longrightarrow \lceil\pi\rceil \stackrel{df}{=} \neg\Diamond_t(F;\lceil\neg\pi\rceil)$$

With this abbreviation we can define the controller in the following way. Initially, the controller is in state *run*.

$$\lceil run\rceil \vee \lceil\rceil_t \qquad \text{(init-control)}$$

From state *run*, the controller can evolve to state *turn*.

$$\lceil run\rceil \rightarrow \lceil run \vee turn\rceil \qquad \text{(successor-run)}$$

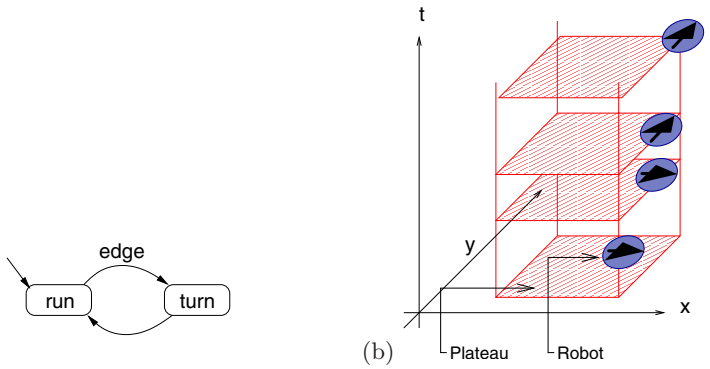

Fig. 4. (a) A simple controller for the roadrunner. (b) A possible unsafe run

From state *turn* it can evolve to state *run*.

$$\lceil turn \rceil \rightarrow \lceil run \vee turn \rceil \qquad \text{(successor-run)}$$

State *run* must be left if an edge is detected after at most δ_R time units.

$$\lceil run \wedge edge \rceil \wedge \ell = \delta_R \rightarrow \lceil \neg run \rceil \qquad \text{(progress-run)}$$

State *turn* must be left after δ_T time units.

$$\lceil turn \rceil \wedge \ell = \delta_T \rightarrow \lceil \neg turn \rceil \qquad \text{(progress-turn)}$$

Spatial Behaviour. To specify the spatial behaviour of the robot we introduce two boolean oberservables RR – for roadrunner – and P – for plateau –. The assumptions are defined using the abbreviations given in the last paragraph.

1. In state *run*, the robot moves with constant velocity v in one direction:

$$\Box_t(\lceil run \rceil \Rightarrow \exists \boldsymbol{d}.(\mathsf{MOTION}_{3d}(RR, d) \wedge \boldsymbol{d}.1^2 + \boldsymbol{d}.2^2 = v \wedge \boldsymbol{d}.3 = 1))$$

2. In state *turn* the robot turns around its axis. We assume that the function f computes the new direction $\boldsymbol{d_2}$ given the old direction $\boldsymbol{d_1}$ and the time which is spent in state *turn*.

$$\Box_t((\lceil run \rceil \wedge \mathsf{Motion}_{3d}(RR, \boldsymbol{d_1}); \lceil turn \rceil \wedge \ell_t = t; \lceil run \rceil \wedge \mathsf{Motion}_{3d}(RR, \boldsymbol{d_2})) \Rightarrow \boldsymbol{d_2} = f(\boldsymbol{d_2}, t))$$

3. The shape of the robot is a circle with unit diameter.

$$\mathsf{Circle}(RR, 1)$$

4. The edge detection is activated as soon as robot leaves the plateau.

$$\Box_t \lceil RR \wedge \neg P \rceil \rightarrow \lceil edge \rceil$$

4.1 Environment

We need to specify that the plateau P is rectangular and does neither change its shape nor moves beneath the robot.

$$\lceil \neg P \rceil \langle \boldsymbol{e_x} \rangle (\lceil \neg P \rceil \langle \boldsymbol{e}_y \rangle \lceil P \rceil \langle \boldsymbol{e}_y \rangle \lceil \neg P \rceil) \langle \boldsymbol{e}_x \rangle \lceil \neg P \rceil$$

Initially, the roadrunner is in the centre of the plateau and the length of the plateau is 21 units. For better readability we omitted the parentheses.

$$
\begin{aligned}
&\lceil \rceil_t \wedge \\
&\lceil \neg(P \wedge RR) \rceil_{(\boldsymbol{e}_x, \boldsymbol{e}_y)^T} \\
&\langle \boldsymbol{e_x} \rangle \\
&\quad \lceil \neg(P \wedge RR) \rceil_{(\boldsymbol{e}_x, \boldsymbol{e}_y)^T} \\
&\quad \langle \boldsymbol{e_y} \rangle \\
&\quad\quad \lceil P \wedge \neg RR \rceil_{(\boldsymbol{e}_x, \boldsymbol{e}_y)^T} \wedge \ell_x = 10 \\
&\quad\quad \langle \boldsymbol{e_x} \rangle \\
&\quad\quad\quad \lceil P \wedge \neg RR \rceil_{(\boldsymbol{e}_x, \boldsymbol{e}_y)^T} \wedge \ell_y = 10 \\
&\quad\quad\quad \langle \boldsymbol{e_y} \rangle \\
&\quad\quad\quad \lceil P \wedge RR \rceil_{(\boldsymbol{e}_x)^T} \wedge \lceil P \wedge RR \rceil_{(\boldsymbol{e}_y)^T} \wedge \ell_x = 1 \wedge \ell_y = 1 \\
&\quad\quad\quad \langle \boldsymbol{e_y} \rangle \\
&\quad\quad\quad \lceil P \wedge \neg RR \rceil_{(\boldsymbol{e}_x, \boldsymbol{e}_y)^T} \wedge \ell_y = 10 \\
&\quad\quad \langle \boldsymbol{e_x} \rangle \\
&\quad\quad \lceil P \wedge \neg RR \rceil_{(\boldsymbol{e}_x, \boldsymbol{e}_y)^T} \wedge \ell_x = 10 \\
&\quad \langle \boldsymbol{e_y} \rangle \\
&\quad \lceil \neg(P \wedge RR) \rceil_{(\boldsymbol{e}_x, \boldsymbol{e}_y)^T} \\
&\langle \boldsymbol{e_x} \rangle \\
&\lceil \neg(P \wedge RR) \rceil_{(\boldsymbol{e}_x, \boldsymbol{e}_y)^T}) \\
&; \mathsf{true}
\end{aligned}
$$

4.2 Requirement

If we assume, that the robot is safe if at most 50 % of its area is outside the plateau, the safety-requirement is specified by

$$\Box_t \textstyle\int (RR \wedge \neg P)_{(\boldsymbol{e}_x, \boldsymbol{e}_y)^T} \leq 0.5.$$

Verification. Trying to verify the safety property, it turns out that the system is not safe even if we chose the velocity v to be low enough such that the robot can stop the motors in its response time. The erroneous situation is sketched in

figure 4. If we assume the upper bound for the response time δ_B to be 500 ms and the velocity v to be $v = 1m/s$ then less than half of the robot is not on the plateau, after having detected an edge. If this just happens in a corner of the plateau and the robot turns 90 degree and continuous to move in the wrong direction, after the response time of 500 ms $\frac{3}{4}$ of the robot will be off the plateau.

Such a behaviour could for example be avoided by driving backwards for δ_B before turning around.

5 Undecidability of Satisfiability

As Duration Calculus with dense time domain is undecidable in general [HZ97] and the shape calculus with a zero dimensional space is exactly the Duration Calculus, it is not decidable either.

However, a restricted subclass of Duration Calculus is decidable with discrete time domain [HZ97], but this does not hold for the shape calculus with more than 1 dimensions. We prove this by reduction from the emptiness problem for 2-dimensional tiling systems [GR97].

5.1 Tiling-Systems and Encoding of Turing Machine Computations

A tile is a 2×2 matrix whose elements are taken from a given alphabet $\Sigma \cup \{\#\}$ where $\#$ is a special symbol which is not included in the set Σ. For a set of tiles Θ the local language $L(\Theta)$ is the set of all $m \times n$ matrices M such that every 2×2 block of M is in Θ. Note, that the blocks are overlapping. Additionally all four borders of this matrix must only consist of the $\#$ symbol. So the matrix must be framed by $\#$. In [GR97] it is shown that the following problem is undecidable: "Given a set of tiles Θ, is $L(\Theta)$ empty?".

For the proof, a Turing Machine computation is simulated in a 2-dimensional array as sketched in figure 5. Each row represents a configuration of the Turing Machine and the row below its successor configuration.

5.2 Encoding Tilings in Shape Calculus

For a set of tiles $\Theta = \{p_1, \ldots, p_k\}$ we can give a formula F_Θ in shape calculus, such that $L(\Theta) \neq \emptyset$ iff F_Θ is satisfiable. To this end, every tile

$$p_i = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

is mapped to a formula

$$\begin{aligned} F_{p_i} = &((\lceil a \rceil \wedge \ell_{\boldsymbol{e_1}} = 1 \wedge \ell_{\boldsymbol{e_2}} = 1) \langle \boldsymbol{e_2} \rangle (\lceil b \rceil \wedge \ell_{\boldsymbol{e_1}} = 1 \wedge \ell_{\boldsymbol{e_2}} = 1)) \langle \boldsymbol{e_1} \rangle \\ &((\lceil c \rceil \wedge \ell_{\boldsymbol{e_1}} = 1 \wedge \ell_{\boldsymbol{e_2}} = 1) \langle \boldsymbol{e_2} \rangle (\lceil d \rceil \wedge \ell_{\boldsymbol{e_1}} = 1 \wedge \ell_{\boldsymbol{e_2}} = 1)). \end{aligned}$$

$\# \ldots \ \# \ \# \ \# \ \ \# \ \ \# \ \ \# \ \ldots \ \#$	
$\# \ \sqcup \ \ldots \ \sqcup \ q_0 \ \ w_1 \ \ w_2 \ \ w_3 \ \ldots \ \#$	(initial configuration)
$\# \ \sqcup \ \ldots \ \sqcup \ w_1^1 \ \ q^1 \ \ w_2^1 \ \ w_3^1 \ \ldots \ \#$	(2nd configuration)
$\# \ \sqcup \ \ldots \ \sqcup \ q^2 \ \ w_1^2 \ \ w_2^2 \ \ w_3^2 \ \ldots \ \#$	(3rd configuration)
$\# \ldots$	
$\# \ \sqcup \ \ldots \quad q_f \ \ w_1^m \ \ w_2^m \ \ w_3^m \ \ldots \ \#$	(final configuration)
$\# \ldots \ \# \ \# \ \# \ \ \# \ \ \# \ \ \# \ \ldots \ \#$	

Fig. 5. Encoding of Turing Machine computation in a 2-dimensional picture

With these sub-formulae we define F_Θ to be

$$\begin{aligned} F_\Theta = & \Box_{\boldsymbol{e_1}} \Box_{\boldsymbol{e_2}} ((\ell_{\boldsymbol{e_1}} = 2 \wedge \ell_{\boldsymbol{e_2}} = 2) \Rightarrow \bigvee_{i=1}^{k} F_{p_i}) \wedge \\ & \lceil \# \rceil \wedge \ell_{\boldsymbol{e_1}} = 1 \\ & \quad \langle \boldsymbol{e_1} \rangle \left(\lceil \# \rceil \wedge \ell_{\boldsymbol{e_2}} = 1 \langle \boldsymbol{e_2} \rangle \lceil \neg \# \rceil \langle \boldsymbol{e_2} \rangle \lceil \# \rceil \wedge \ell_{\boldsymbol{e_2}} = 1 \right) \langle \boldsymbol{e_1} \rangle \\ & \lceil \# \rceil \wedge \ell_{\boldsymbol{e_1}} = 1 \end{aligned}$$

The first part describes, that each 2×2 block must be in Θ. Note that we consider the discrete shape calculus and therefore we may only chop at discrete positions. The second part defines that the picture must be framed by #. As we consider a discrete time and space domain, the formula F_Θ is satisfiable if and only if the the local language $L(\Theta)$ is not empty. Therefore the satisfiability problem for shape calculus with discrete domain and more than one dimension is undecidable. □

6 Decidable Fragment

If we restrict the set of formulae to

$$F ::= \lceil P \rceil \mid \lceil P \rceil_{\boldsymbol{e}_x} \mid \lceil P \rceil_{\boldsymbol{e}_t} \mid F \wedge G \mid \neg F \mid F \langle \boldsymbol{e}_x \rangle G \mid F \langle \boldsymbol{e}_t \rangle G$$

and the set of models to one discrete infinite temporal dimension and consider all other spatial dimensions to be finite, the satisfiability problem is decidable.

We follow the lines of the decidability proof for discrete time Duration Calculus [HZ04] and extend it to one temporal and one finite spatial dimension with cardinality n, but it can be easily generalised to more than one finite spatial dimension. Assume we use boolean observables $X_1, \ldots, X_o$. Then for a point in space and time the vector $(x_1, \ldots, x_o) \in \{true, false\}^o$ describes which observables are true. We use $o \times n$ matrices to describe the space for a point of time and take the set $\Sigma^n = \{[w_1, \ldots, w_n] | w_i \in \{true, false\}^o\}$ of all $o \times n$ matrices as our alphabet.

We use $\Sigma^{\leq k} = \bigcup_{i=0}^{k} \Sigma^i$ as an abbreviation. For a state assertion P and $1 \leq i \leq n$ we define $\Sigma^n_{i,P} = \{[w_1 \ldots w_n] \in \Sigma^n | w_i \models P\}$ as all possible space configurations such that at point i in space the state assertion P is true. These

sets can be computed as we allow only propositional formulae for the state assertions.

Additionally we define $\Sigma^n_{\Box P} = \bigcap_{i=1}^n \Sigma^n_{i,P}$ and $\Sigma^n_{\Diamond P} = \bigcup_{i=1}^n \Sigma^n_{i,P}$. We define for every $1 \leq i, j \leq m \leq n$ the functions $h_{i,j,(m)}([w_1, \ldots, w_m]) = [w_i, \ldots, w_j]$ where we assume $h_{i,j,(m)}([w]) = \varepsilon$ if $j < i$. These functions yield the middle part from i up to j of the matrix $[w_1, \ldots, w_m]$.

For the construction of our languages, we use these functions as homomorphisms for languages $L \subseteq (\Sigma^m)^*$ and consider its inverse $h^{-1}_{i,j,(m)}$.

The construction of $\mathcal{L}_n(F)$ over the alphabet Σ^n proceeds inductively as follows:

$$\mathcal{L}_n(\lceil P \rceil) = (\Sigma^n_{\Box P})^+ \qquad \mathcal{L}_n(\lceil P \rceil_{\boldsymbol{e}_t}) = (\Sigma^n_{\Diamond P})^+$$

$$\mathcal{L}_n(\lceil P \rceil_{\boldsymbol{e}_x}) = \bigcap_{i=1}^{n} ((\Sigma^n)^* \circ \Sigma^n_{i,P} \circ (\Sigma^n)^*) \qquad \mathcal{L}_n(F \wedge G) = \mathcal{L}_n(F) \cap \mathcal{L}_(G)$$

$$\mathcal{L}_n(F \langle \boldsymbol{e}_t \rangle G) = \mathcal{L}_n(F) \circ \mathcal{L}_n(G) \qquad \mathcal{L}_n(\neg F) = \overline{\mathcal{L}_n(F)}$$

For the spatial chop operation $F \langle \boldsymbol{e}_x \rangle G$, we consider every possible position i at which the chop can be applied. For every $0 \leq i \leq n$ we construct the languages $\mathcal{L}_i(F)$ and $\mathcal{L}_{n-1}(G)$. Every letter in $\mathcal{L}_i(F)$ is a $o \times i$ matrix which models a space with cardinality i. We have to consider every possible extention to the right of this matrix to an $o \times n$ matrix. This is done using the inverse homomorphism $h^{-1}_{1,i,(n)}$. The same is done for $\mathcal{L}_{n-1}(G)$. We end up with

$$\mathcal{L}_n(F \langle \boldsymbol{e}_x \rangle G) = \bigcup_{i=0}^{n} \left(h^{-1}_{1,i,(n)}(\mathcal{L}_i(F)) \cap h^{-1}_{i+1,n,(n)}(\mathcal{L}_{n-i}(G)) \right).$$

Using this construction for a formula F the following holds:

Theorem 1. *$\mathcal{L}_n(F)$ is empty if and only if F has no model.*

7 Conclusion

In this paper, we presented how a well known real-time formalism can be enriched to allow spatial verification. To this end, we generalised the Duration Calculus to more than one dimension and showed how spatial properties like continuous movement can be formalised. We gave an example to show that specifications in pure Duration Calculus can be used directly in the extension, spatial requirement just need to be added. We proved undecidability results and identified a decidable subset.

Related Work. There is a lot of related work around in the field of spatial and temporal logics. The *Region Connection Calculus* (RCC) proposed by Randell, Cui and Cohn [RCC92] is widely used in the AI community and fundamental for many extensions. It handles regions and their connectivity, for example whether

one region intersects another or whether it is a proper part. It does not handle time or quantitative measures and so it is not well suited for the description of mobile safety-critical real-time systems. It has been used in [Gal95] to develop a *qualitative* theory of movement. Additionally one should be able to encode the RCC in the shape calculus. This is still to be investigated.

A more recent approach by Merz et al [MWZ03] considers an extension of TLA to describe mobile systems, but this approach uses the π-Calculus [Mil99] notion of mobility, namely mobility by changes of links between processes. Also using this notion of mobility, Cardelli give a spatial logic tailored for the ambient calculus in [CG00] or the π-Calculus in [CC03] which allows to reason about freshness of names, name restriction and composition.

By contrast, the Real Space Process Algebra proposed by Baeten and Bergstra [BB92] uses the 3-dimensional space and in an relativistic extension the 4-dimensional space and describes actions occurring at certain points in space.

Closer to our approach are the recent multi-dimensional modal logics introduced for example in [BC02, AvB01, Mul98, ADN97] but they do not allow quantitative reasoning, neither spatial nor temporal. A logic specially tailored to reason about distances can be found in [WZ03] but this approach does not consider time.

Coming from Duration Calculus there are various extensions. The extension for hybrid systems by Zhou, Ravn and Hansen [ZRH93] introduces the diffential operator instead of the integral. But this formalism is still a temporal logic and does not take space into account.

Finally, a 2-dimensional extension of Duration Calculus by Pandya and Van Hung [PH98] uses 2-dimensional time, one dense time line (macro time) and orthogonal a discrete time line (stepped time) to model superdense computations.

Perspectives. For future work, we like to give a set of rules to make spatial reasoning more convenient and apply these rules on more case studies. The question whether there is a relative complete and sound calculus and finding richer decidable subsets are also things to tackle in the future.

Acknowledgements. The author thanks E.-R. Olderog and the members of the "Correct System Design" group for fruitful discussions and draft-reading earlier versions.

References

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[ADN97] S. N. Artemov, J.M. Davoren, and A. Nerode. Modal logics and topological semantics for hybrid systems. Technical Report 97-05, Cornell University Ithaca, 1997.

[AvB01] M. Aiello and H. van Benthem. A Modal Walk Through Space. Technical report, Institute for Logic, Language and Computation, University of Amsterdam, 2001.

[BB92] J.C.M. Baeten and J.A. Bergstra. Asynchronous Communication in Real Space Process Algebra. In Jan Vytopil, editor, *FTRTFT '92, Nijmegen, The Netherlands*, volume 571 of *LNCS*, pages 473–492. Springer, 1992.

[BC02] B. Bennett and A.G. Cohn. Multi-Dimensional Multi-Modal Logics as a Framworf for Spatio-Temporal Reasoning. *Applied Intelligence*, 17(3):239–251, 2002.

[CC03] L. Caires and L. Cardelli. A Spatial Logic for Concurrency. *Information and Computation*, 186(2):194–235, 2003.

[CG00] L. Cardelli and A. D. Gordon. Anytime, Anywhere: Modal Logics for Mobile Ambients. In *POPL 2000,*, pages 365–377. ACM Press, 2000.

[Gal95] A. Galton. Towards a qualitative theory of movement. In *Spatial Information Theory*, pages 377–396, 1995.

[GR97] D. Giammarresi and A. Restivo. *Handbook of Formal Languages – Beyond Words*, volume 3, chapter Two-Dimensional Languages, pages 215–267. Springer, 1997.

[HZ97] M. R. Hansen and Zhou Chaochen. Duration Calculus: Logical Foundations. *Formal Aspects of Computing*, 9:283–330, 1997.

[HZ04] M. R. Hansen and Zhou Chaochen. *Duration Calculus: A Formal Approach to Real-Time Systems*. EATCS: Monographs in Theoretical Computer Science. Springer, 2004.

[Mil99] R. Milner. *Communicating and mobile systems: the π-calculus*. Cambridge University Press, 1999.

[Mul98] P. Muller. A Qualitative Theory of Motion Based on Spatio-Temporal Primitives. In A. G. Cohn, L. K. Schubert, and S. C. Shapiro, editors, *KR'98. Principles of Knowledge Representation and Reasoning, Trento, Italy*, pages 131–143. Morgan Kaufmann, 1998.

[MWZ03] S. Merz, M. Wirsing, and J. Zappe. A Spatio-Temporal Logic for the Specification and Refinement of Mobile Systems. In M. Pezzè, editor, *FASE 2003, Warsaw, Poland*, volume 2621 of *LNCS*, pages 87–1014. Springer, 2003.

[PH98] P. K. Pandya and Dang Van Hung. Duration Calculus of Weakly Monotonic Time. In A. P. Ravn and H. Rischel, editors, *FTRTFT'98, Lyngby, Denmark*, number 1998 in LNCS, pages 55–64. Springer, 1998.

[Rav95] A. Ravn. Design of embedded real-time computing systems. Technical report, Dept. Comp. Science, Technical University of Denmark,Bld. 344, DK-2800 Lyngby, 1995.

[RCC92] D. A. Randell, Z. Cui, and A. Cohn. A Spatial Logic Based on Regions and Connection. In B. Nebel, C. Rich, and W. Swartout, editors, *KR'92. Principles of Knowledge Representation and Reasoning*, pages 165–176. Morgan Kaufmann, San Mateo, California, 1992.

[WZ03] F. Wolter and M. Zakharyaschev. Reasoning about distances. In G. Gottlob and T. Walsh, editors, *IJCAI-03, Acapulco, Mexico, August 9-15, 2003*, pages 1275–1282. Morgan Kaufmann, 2003.

[ZHR91] Zhou Chaochen, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, 1991.

[ZRH93] Zhou Chaochen, A.P. Ravn, and M.R. Hansen. An Extended Duration Calculus for Hybrid Real-Time Systems. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, volume 736 of *LNCS*, pages 36–59. Springer, 1993.