A logic for robotics?

Alexandre Madeira[*], Renato Neves[*], Manuel A. Martins[†] and Luís S. Barbosa[*]

[*]*HASLab - INESC TEC & Universidade do Minho*
[†]*CIDMA, Dep. of Mathematics, Universidade de Aveiro*

Abstract. Dynamic logic combines logic with programs, which at a certain level of abstraction, can be regarded as behaviours changing the system state and, therefore, the truth value of formulas. This paper suggests a method for generating such logics for the domain of robot controllers and illustrates it with a logic for handling resource consumption.

Keywords: Logic in Computer Science, Dynamic logic; Robotics
PACS: 03B70

Motivation

One of the main challenges in robotics is the generation of controllers for autonomous, high-level robot behaviors comprising non-trivial sequences of actions. The standpoint of this paper is that these can be specified, and verified, in the classical framework of *dynamic logic* [4, 1], provided such logics are generated over a notion of *program* suitable to capture the sort of behaviours, or behavioural views, relevant for the robotics domain.

Dynamic logic has been used very successfully for conventional discrete programs defined by a (Kleene) algebra of “execution statements". On the other side of the spectrum, they are also used to reasoning about hybrid, or *cyber-physical* systems by considering an algebra of actions based on real numbers assignments, the standard Kleene operators and differential equations to specify continuous transitions from the “real" (physical) world (see, e.g., [3], where also probabilistic extensions are considered).

This paper sketches an approach to the specification of controllers for robotic engines based on a double use of (suitably generated) dynamic logics: one to deal with elementary programs or *methods*; another to reason about complex behaviours in robot missions. For illustration purposes, we focus on a very simple logic able to take care of resource management. Energy, for example, is a critical resource for deep sea robotics. In particular, the behaviour of a controller for this sort of robots deeply depends on the management of their energy resources. In this logic, $M \models_r^w \rho$ means that a model M can satisfy requirement ρ consuming at most r resources units.

A dynamic logic for resource management

We start from the observation that a controller for a robotic engine can be modelled by a *state space* and a number of *methods* defined upon it. The former captures possible robot configurations which may be related, for example, to its physical location or to a combination of context variables as measured by its sensors. In the example below, two such states are considered: *deep* and *base*, to distinguish between a robot staying underwater or docked. Methods, on the other hand, exhibit behaviour specified by a transition structure. An arrow between two states represents a possible transition labelled by an expression. The latter may be a single action, often annotated with some sort of weight (e.g., a cost, a probability, or even a condition on a continuous variable). But it may also be an expression on such actions or weights. Consider the method *read* in the example below. Its behaviour is given by a square matrix whose values are regarded, for illustration purposes, as operation costs. The first entry specifies that the cost of performing this method while remaining underwater is $a \cup a'$, i.e., a choice between two values a and a' (depending, for example, on contextual conditions abstracted here). On the other hand, performing *read* when docked is for free (entry $(2,2)$ marked as 0). Finally, annotation $\perp$ on the other two entries means that no data is gathered when the robot moves from one location to the other.

Modelling complex behaviours entails the need for reasoning at two levels: locally to a method, in order to combine local annotations, and globally to combine methods. In the tradition of dynamic logics, both local annotations and methods are composed through the operations of a Kleene algebra, which is the prototypical way of combining programs irrespectively of the different ways they can be defined. Technically, this is made possible by the well-known result that Kleene algebras are closed under the formation of matrices. Actually, we resort to action lattices [2] — an extension of Kleene algebras with a residuated lattice structure, which has the advantage of being equationally axiomatised and, consequently, a variety. This smooths formal reasoning but is relatively irrelevant for the introduction this paper aims at. Moreover, this lattice structure supports the truth space for a graded satisfaction.

On top of this two-staged model we define a specific dynamic logic. This is illustrated in the sequel through an example focused on resource consumption. Although the example is small enough to fit in the paper, the topic, related to energy saving, is a main concern in real-world submarine robotics.

Syntax. Signatures for a *resource dynamic logic* consists of pairs $\tau = (\text{Prop}, \Pi)$ of sets of propositions and atomic methods, respectively. In this running example we take, as explained above, $\tau = \big(\{deep, base\}, \{down, up, read\}\big)$ where propositions *deep* and *base* denote the location states and symbols $up, down$ and *read*, the methods for immersing, emerging and data gathering, respectively.

The set $Fm(\tau)$ of *formulas* for a signature $\tau = (\text{Prop}, \Pi)$, is given by the following grammar:

$$\rho \ni p \,|\, \rho \odot \rho \,|\, \langle \pi \rangle \rho \,|\, [\pi]\rho \tag{1}$$

$$\pi \ni \pi_0 \,|\, \pi;\pi \,|\, \pi \cup \pi \,|\, \pi^* \tag{2}$$

where $p \in \text{Prop}$, $\odot \in \{\wedge, \vee, \to\}$ and $\pi_0 \in \Pi$. Terms defined by the grammar (2) are called Π-programs.

Models. To handle resources a specific action lattice $([\mathbb{N}], max, +, \bot, 0, *, -_\bot, -_\bot, min)$, known as the *Floyd-Warshall* algebra, is used. Its carrier is the set $[\mathbb{N}] = \mathbb{N} \cup \{\bot, \top\}$. This means that non-deterministic choice is taken as the *max* function (applied to the relevant resources); composition (whenever defined) adds consumptions

$$a + b = \begin{cases} a+b, & \text{if } a, b \notin \{\bot, \top\} \\ \bot, & \text{if } a \text{ or } b \text{ equals to } \bot \\ \top, & \text{otherwise} \end{cases}$$

and the Kleene closure $a^* = \begin{cases} \top, & \text{if } a \in \mathbb{N} \setminus \{0\} \cup \{\top\} \\ 0, & \text{if } a \in \{\bot, 0\} \end{cases}$ represents the arbitrary repetition of a consume cost. Elements $\bot$ and 0 are the absorvent and the neutral elements for $+$ and *max*, respectively. Informally, $\bot$ and $\top$ represent undefined and infinite amount of resources, respectively. Finally, both left and right residuals are defined as truncated subtraction $a -_\bot b = \begin{cases} a-b, & \text{if } b \leq a \\ 0, & \text{otherwise} \end{cases}$ and, the meet lattice operation as the *min* function.

The behaviour of methods is encoded in $n \times n$-matrices, for n the cardinal of the state space, over this action lattice. Therefore, models for a signature $\tau = (\text{Prop}, \Pi)$, consists of triples $M = (W, V, M_\Pi)$ where W is the set of states, with cardinal n, $V : \text{Prop} \to \mathscr{P}(W)$ is a valuation, and $M_\Pi = \big(M_\pi \in M_n([\mathbb{N}])\big)_{\pi \in \Pi}$ the set of interpretations of methods available at the controller.

In our example, $M = \big(\{s_{deep}, s_{base}\}, V, M_\Pi\big)$, where valuation V is defined by $V(deep) = \{s_{deep}\}$ and $V(base) = \{s_{base}\}$. Methods (or atomic controller programs) M_Π consist of the following matrices:

$$M_{read} = \begin{bmatrix} a \cup a' & \bot \\ \bot & 0 \end{bmatrix} \qquad M_{up} = \begin{bmatrix} \bot & b \\ \bot & \bot \end{bmatrix} \qquad M_{down} = \begin{bmatrix} \bot & \bot \\ c & \bot \end{bmatrix}$$

$$\begin{tikzcd} s_{deep} \arrow[loop above, "a\cup a'"] & s_{base} \arrow[loop above, "0"] & s_{deep} \arrow[r, "b"] & s_{base} & s_{deep} & s_{base} \arrow[l, "c"'] \end{tikzcd}$$

for $a, a', b, c \neq \bot$. Note that $a \cup a'$ stands for $max\{a, a'\}$, the interpretation of the regular expression in the action lattice of base.

Interpretation of global behaviours. The interpretation of a Π-behaviour π in M, denoted by M_π, is recursively defined from the methods in M_Π, taken as the atomic behaviours, as follows

- $M_{\pi;\pi'} = M_\pi \oplus M_{\pi'} = M$, where $M_{i,j} = max\{(M_\pi)_{i,k} + (M_{\pi'})_{k,j} | k \leq n\}$;
- $M_{\pi \cup \pi'} = M$ where $M_{i,j} = max\{(M_\pi)_{i,j}, (M_{\pi'})_{i,j}\}$, i.e., for any $i, j \leq n$;

- $M_{\pi^*} = (M_\pi)^*$, where $(M_\pi)^*$ is defined recursively by partitioning M in $\left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]$ for A and D square matrices, and taking $M^* = \left[\begin{array}{c|c} F^* & F^* \oplus B \oplus D^* \\ \hline C & max\{D^*, D^* \oplus C \oplus F^* \oplus B \oplus D^*\} \end{array}\right]$ with $F = max\{A, B \oplus D^* \oplus C\}$. In the base case, i.e. when the partitions are 1×1 matrices, we take $a^* = \begin{cases} \top, & \text{if } a \in \mathbb{N} \\ 0, & \text{if } a \in \{\bot, 0, \top\} \end{cases}$.

These global behaviours are exactly the programs running on the robot controller. To illustrate this construction consider a program controlling a data gathered mission. Informally, the robot makes an undetermined number of checks to the sensors; then it dives in the sea, collects data and returns to the dock state. Thus,

$$
\begin{aligned}
& M_{read^*;down;read;up} \\
= & \quad \{\text{ programs interpretation defn}\} \\
& M_{read^*} \oplus M_{down} \oplus M_{read} \oplus M_{up} \\
= & \quad \{\text{ programs interpretation defn}\} \\
& \begin{bmatrix} a \cup a' & \bot \\ \bot & 0 \end{bmatrix}^* \oplus \begin{bmatrix} \bot & \bot \\ c & \bot \end{bmatrix} \oplus \begin{bmatrix} a \cup a' & \bot \\ \bot & 0 \end{bmatrix} \oplus \begin{bmatrix} \bot & b \\ \bot & \bot \end{bmatrix} \\
= & \quad \{\text{ defn of } *\} \\
& \begin{bmatrix} f^* & f^* + \bot + 0^* \\ \bot & max\{0^*, 0^* + \bot + f^* + \bot + 0^*\} \end{bmatrix} \oplus \begin{bmatrix} \bot & \bot \\ c & \bot \end{bmatrix} \oplus \begin{bmatrix} a \cup a' & \bot \\ \bot & 0 \end{bmatrix} \oplus \begin{bmatrix} \bot & b \\ \bot & \bot \end{bmatrix} \\
= & \quad \{\text{ defn of } \oplus\} \\
& \begin{bmatrix} max\{\top + \bot, \bot + c\} & max\{\top + \bot, \bot + \bot\} \\ max\{\bot + \bot, 0 + c\} & max\{\bot + \bot, 0 + \bot\} \end{bmatrix} \oplus \begin{bmatrix} a \cup a' & \bot \\ \bot & 0 \end{bmatrix} \oplus \begin{bmatrix} \bot & b \\ \bot & \bot \end{bmatrix} \\
= & \quad \{\text{ defn of } \oplus\} \\
& \begin{bmatrix} max\{\bot + (a \cup a'), \bot + \bot\} & max\{\bot + \bot, \bot + 0\} \\ max\{c + (a \cup a'), \bot + \bot\} & max\{c + \bot, \bot + 0\} \end{bmatrix} \oplus \begin{bmatrix} \bot & b \\ \bot & \bot \end{bmatrix} \\
= & \quad \{\text{ defn of } \oplus\} \\
& \begin{bmatrix} \bot & \bot \\ c + (a \cup a') & \bot \end{bmatrix} \oplus \begin{bmatrix} \bot & b \\ \bot & \bot \end{bmatrix} = \begin{bmatrix} \bot & \bot \\ \bot & (a \cup a') + c + b \end{bmatrix}
\end{aligned}
$$

where $f = max\{a \cup a', \bot + \bot + 0^*\} = a \cup a'$. Note that, as expected, the unique transition captured by this behaviour goes from s_{base} to s_{base} (all the others are marked with $\bot$), and it consumes $max\{(a \cup a') + c + b\}$ resource units on its completion.

As in any other dynamic logic, behaviours are first class citizens in formulas expressing operational requirements over controllers. Therefore, the satisfaction relation for formulas is presented as follows;

Satisfaction. Let $M = (W, V, M_\Pi)$ be a model for a signature $\tau = (\text{Prop}, \Pi)$. The graded satisfaction relation in M consists of a function

$$\models : W \times Fm(\tau) \to [\mathbb{N}]$$

defined as follows:

- for any $p \in$ Prop, $(w \models \rho) = \begin{cases} 0, & \text{if } w \in V(p) \\ \bot, & \text{otherwise} \end{cases}$
- $(w \models \rho \wedge \rho') = min\{w \models \rho, w \models \rho'\}$
- $(w \models \rho \vee \rho') = max\{w \models \rho, w \models \rho'\}$
- $(w \models \rho \to \rho') = \begin{cases} (w \models \rho') - (w \models \rho) & \text{if } (w \models \rho') \geq (w \models \rho) \\ 0 & \text{otherwise} \end{cases}$

- $\big(w \models \langle\pi\rangle\rho\big) \;=\; max_{w' \in W}\Big\{M_\pi(w,w') + w' \models \rho\Big\}$
- $\big(w \models [\pi]\rho\big) \;=\; min_{w' \in W}\Big\{M_\pi(w,w') + w' \models \rho\Big\}$

Thus, satisfaction is defined by

$$w \models_r \rho \;\text{ iff }\; \big(w \models \rho\big) \leq r$$

We have now all ingredients to express and check properties about behaviours. Back to the example we are able to check for which battery costs one may safely gather data at deep sea:

$$\begin{array}{ll}
& s_{base} \models_r \langle read^*; down; read; up\rangle base \\
\equiv & \quad \{\text{ defn } \models_r\} \\
& \big(s_{base} \models \langle read^*; down; read; up\rangle base\big) \leq r \\
\equiv & \quad \{\text{ defn } \models\} \\
& \Big(max\big\{M_{read^*;down;read;up}\big(s_{base}, w'\big) + \big(w' \models base\big) \,\big|\, w' \in W\big\}\Big) \leq r \\
\equiv & \quad \{\text{ since } (w \models base) = \bot \text{ for any } w \neq s_{base}\} \\
& \big(max\{M_{read^*;down;read;up}\big(s_{base}, s_{base}\big) + \big(s_{base} \models base\big)\}\big) \leq r \\
\equiv & \quad \{\text{ by programs interpretation defn.}\} \\
& \big(max\{(a \cup a') + c + b, \top\}\big) \leq r \\
\equiv & \quad \{\text{ since } (a \cup a') + c + b \leq \top\} \\
& (a \cup a') + c + b \leq r \\
\equiv & \quad \{\text{ choice in the action lattice } [\mathbb{N}]\} \\
& max\{a + c + b, a' + c + b\} \leq r
\end{array}$$

Therefore, we conclude that the implementation of this behaviour is successful whenever the level r of robot autonomy is greater than $max\{a + c + b, a' + c + b\}$ units.

Concluding

The method illustrated above can be instantiated in different engineering contexts, i.e., for distinct interpretations of local and global behaviours of robot engines. In each case, however, a standard dynamic logic is generated indeed. The interested reader may want to check some of its axioms; for example, it is no difficult task to proof that $\langle\pi;\pi'\rangle\rho = \langle\pi\rangle\langle\pi'\rangle\rho$ still holds for the resource management logic discussed here.

Acknowledgements. This work is funded by ERDF - European Regional Development Fund through the COMPETE Programme (Operational Programme for Competitiveness) and by National Funds through FCT, the Portuguese Foundation for Science and Technology, within projects FCOMP-01-0124-FEDER-028923, project PEst-OE/MAT/UI4106/2014; and by project NORTE-07-0124-FEDER-000060, co-financed by the North Portugal Regional Operational Programme (ON.2 – O Novo Norte), under the National Strategic Reference Framework (NSRF), through the European Regional Development Fund (ERDF).

REFERENCES

1. Dexter Kozen David Harel and Jerzy Tiuryn. *Dynamic Logic*. MIT Press, 2000.
2. Dexter Kozen. On action algebras. manuscript in: Logic and Flow of Information, Amsterdam, 1991.
3. André Platzer. *Logical Analysis of Hybrid Systems - Proving Theorems for Complex Dynamics*. Springer, 2010.
4. Vaughan R. Pratt. Semantical considerations on floyd-hoare logic. In *FOCS*, pages 109–121. IEEE Computer Society, 1976.