RoboChart & RoboSim

Modelling Robots and Collections

Alvaro Miyazawa

Department of Computer Science
University of York

January 23, 2019
Introduction
RoboChart
RoboSim
Collection modelling
Robotic platform modelling
INTRODUCTION
Motivation

1st phase: Abstract model

state machine

2nd phase: Simulation

controller code

| hardware simulation | discrete environment simulation |

3rd phase: Implementation

low-level code

| robot | environment |
State machines are often used to record, illustrate and explain

Usage is informal

Potential:
- Testing
- Code generation
- Verification
Objective

- Graphical notations
- Formal semantics
- Specialised, but comprehensive
- Supporting simulation, analysis and verification
**Approach**

- **RoboChart Models**
- **Requirements**
- **ARGoS**
- **RoboTool**
- **C++**
- **PRISM**
- **Storm**

- **Reactive Modules Formalism**
- **CSP and timed-CSP**

- **Qualitative Results**
- **Quantitative Results**

- **Simulation**
- Standard state machines + time + probability
- Formal semantics: untimed, timed and probabilistic
- Well-formedness conditions
- Tool support:
  - Modelling
  - Validation
  - Code generation: semantics and simulation
Module

- Models a single Robot
- 1 Robotic Platform
- 1+ Controllers
- Communication
  - Synchronous
  - Asynchronous
- Robotic Platform may provide shared variables
Robotic Platform

- Records assumptions about the robot hardware
  - which events the robot provides
  - which operations the robot supports
  - which variables are available
- Independent of controller and state-machines
- Single point of interaction with robot
Controller

- Models a specific behaviour
- Contains:
  - Behavioural state-machines
  - Operations
  - Variables
  - Events
- Supports multiple behavioural state-machines
- Communication between state-machines is synchronous
State-Machine

- Main behavioural specification construct
- Models both operations and behaviours
- Simple, Composite and Final states
- Initial and junction nodes
- Non-interlevel transitions
- Actions: entry, during, exit, transition
- Local variables
Types and Action Language

- Types based on Z Mathematical Toolkit
- Action language:
  - Assignment
  - Event signalling
  - Operation call
  - Sequential composition
- Control statements modelled using junctions and transitions
**Semantics**

- Formalised in CSP
- Coverage:
  - State-Machines
  - Controllers
  - Robotic Platforms
  - Modules
Semantics: Overview

Module = CSP Process
- Parallel composition of controllers
- Connections define synchronisation sets
- Asynchronous communication modelled through buffers
- Robotic platform incorporated via renaming

Controller = CSP Process
- Parallel composition of state-machines
- Connections define synchronisation sets
- External interactions via controller established via renaming
Semantics: Overview

- **State-Machine = CSP Process**
  - Parallel composition of states
  - Transitions are part of the source states
  - Junctions are part of the incoming transition
  - Initial nodes and final states are part of the parent state
  - States interact with each other to enter and exit
  - States synchronise on transition triggers to support top-down interruption

- **Action language**
  - Operation call = Process call
  - Event signalling = Communication on event channel
  - Assignment = Communication on setter channel

- **State components**
  - Isolated in memory process due to sharing
  - Help avoid polling for transition conditions
- Eclipse plugins
- Textual editor developed using Xtext
- Graphical editor developed using Sirius
- Code generator for the semantics
- Code generator for simulation
- Validation rules
module ChemicalDetector {
    robotic platform Rover {
        event alarm: Object
        event lightOn
        event lightOff
        event r
        move(v: Vector, speed: real): void
        LoadFlag(): void
        ReleaseFlag(): void
    }
    cref DF = DetectAndFlagC
    connection Rover on alarm to DF on found
    connection DF on flagged to LC on activate (async)
    connection Rover on l to DF on left
    connection Rover on r to DF on right
    cref LC = LightController
    connection LC on l on to Rover on lightOn
    connection LC on loff to Rover on lightOff
}

ChemicalDetector

-Rover
-move(v: Vector, speed: real): void
-LoadFlag(): void
-ReleaseFlag(): void

-Ref LightController
-lightOn
-lightOff

-Alarm

-LightController
-left
-right

-DetectAndFlagC

-stateMachine

-definiteEvent

-variable

-dataType

-field

-output

-operation

-event

-clock
Case studies:

- Alpha Algorithm (Single Robot and Collection);
- Chemical Detector;
- Autonomous Chemical Detector;
- Foraging;
- Transport; etc.

Generated semantics used for verification using FDR4

FDR4 compression functions highly effective
CURRENT DEVELOPMENTS

- Generation of simulations
- Generation of probabilistic semantics
- Generation of semantics for Isabelle/UTP
- Based on RoboChart
- Explicit cyclic pattern for simulation
- Related to RoboChart models via refinement
Collection Modelling
RoboChart

The focus of RoboChart is the modelling, analysis and simulation of individual robots.
RoboChart

The focus of RoboChart is the modelling, analysis and simulation of individual robots.

Other notations

Support in other notations tends to be concrete.
OBJECTIVE

- Support modelling, analysis and simulation of collections
- Reuse RoboChart models and semantics
Extensions

- new implicit type ID and module constant id;
- robotic platform events are broadcast and directional;
- broadcast events have implicit ID parameters: to and from;
- input events can restrict from and record its value;
- output events can restrict to parameter; and
- new diagram describes group of collections and how they communicate.
**Models**

\[ \pi \]

\[ N: \text{nat} \]

\[ i: \{1 \text{ to } N\} \text{ of AggregationRobot} \]
Semantics of collections

\[
\begin{align*}
& \{9\ i : \{1..N\} \rightarrow AggregationRobot(i)\} \\
& \quad J\{\text{report.in, report.out, ack.in, ack.out}\} \rightarrow K \\
& \bigg(\{9\ i : \{1..N\} \rightarrow 9\ j : \{1..N\} \setminus \{i\} \rightarrow Buffer(\langle\rangle, \text{report}, i, \text{report}, j)\}\bigg) \\
& \quad 9 \\
& \bigg(\{9\ i : \{1..N\} \rightarrow 9\ j : \{1..N\} \setminus \{i\} \rightarrow Buffer(\langle\rangle, \text{ack}, i, \text{ack}, j)\}\bigg)
\end{align*}
\]
Alpha Algorithm
**Alpha Algorithm (Old)**

- Communication
  - `ack: ID*ID`
- x: ID*ID, y: ID, neighs: Set(ID)
- RC: nat, id: ID
- RCC
- CommHw
- Internal

Report?
- `y[since(RCC)<RC]/ack!(|y, id|)`

Broadcast
- entry report!id

Receive
- `#RCC/neighs = {}`

Ack?
- `x[x[1]==id\since(RCC)<RC]/neighs = union(neighs, {x[2]})`
**Alpha Algorithm (new)**

### Communication

- **RC**: nat
- **x**: ID, **y**: ID, **neighs**: Set(ID)
- **RCC**
- **CommHw**
- **Internal**

**Report**: `report[|y = from|][since(RCC) < RC]/ack[|to = = y|]

**Broadcast**

- `entry report`

**Receive**

- `#RCC/neighs = {}`
- `ack[|x = from|][since(RCC) < RC]/neighs = union(neighs, {x})`
Events and their semantics

- $\text{ev}![\text{pred}]!e$
  - **semantics** $\text{ev.out.id?to} : \{x \mid x \leftarrow \text{ID}, \text{pred}\}!e \rightarrow \text{Skip}$
- $\text{ev}[\mid v = \text{from} \mid \text{pred} \mid]?u$
  - **semantics** $\text{ev.in?from} : \{x \mid x \leftarrow \text{ID}, \text{pred}\}.\text{id}?y \rightarrow \text{set_v!from} \rightarrow \text{set_u!y} \rightarrow \text{Skip}$
Current status

- Partial support for modelling
- Code generation for semantics
- Validation

Ongoing work

- Complete modelling support
- Extend simulation generation

Future work

- Optimise verification
- Investigate data abstraction and induction with FDR/two.osf/nine.osf
- Investigate theorem proving with Isabelle/UTP
Current status

- Partial support for modelling
- Code generation for semantics
- Validation

Ongoing work

- Complete modelling support
- Extend simulation generation

Future work

- Optimise verification
- Investigate data abstraction and induction with FDR/two.osf/nine.osf
- Investigate theorem proving with Isabelle/UTP

/four.osf/two.osf
Current status

- Partial support for modelling
- Code generation for semantics
- Validation

Ongoing work

- Complete modelling support
- Extend simulation generation

Future work

- Optimise verification
- Investigate data abstraction and induction with FDR4
- Investigate theorem proving with Isabelle/UTP
Robotic platform modelling
Motivation

- RoboChart focuses on modelling controllers
- Robotic platform is abstracted as a set of events, variables and operations
- Existing XML-based notations: URDF, SDF, Collada
  - not convenient for modelling
  - not abstract enough
  - no facilities for modelling behaviour
Objectives

- Restructure and refactor SDF
- Provide graphical representation
- Extend with facilities to
  - model behaviours
  - map between operations, events and variables to sensors and actuators
- Formal semantics integrated with RoboSim
- Linked to RoboChart via abstraction
- Generate both SDF models and platform dependent simulation code
Simple Model
**Simple Model**

- Link: LWheel
  - Joint: LHinge: Revolute
    - Axis: (0.0, 1.0, 0.0)
      - Position: (-length/2, 0.0, height)
      - Orientation: (0.0, 0.0, π)

- Body: Wheel
  - Type: Cylinder(length:radius/2, radius:radius)

- Sensor: IR
  - Behaviour:
    - Inputs:
      - distance: real
    - Outputs:
      - voltage: real
    - Equations:
      - $voltage = 4 \times e^{(-0.028784 \times distance)}$
Simple Model

link
Base

joint
RHinge: Revolute
Axis: (0.0, 1.0, 0.0)

link
RWheel

position: (-length/4-radius/2, 0.0, -height/2-radius/4)
orientation: (0.0, 0.0, 0.0)

body
Chassis
Box(length, width, height)

body
CasterWheel
Sphere(radius:radius/2)

body
Wheel
Cylinder(length:radius/2, radius:radius)

behaviour
Inputs
das : real

Outputs
as : real

Local Variables
i, v : real

Constants
J, b, K, L, R : real

Equations
v = R*b*das/K + K*das
J*as' + b*as = K*i
L*i' + R*i = v - K*a
Simple Model

Platform Mapping

move (ls: real, as: real)

Local Variables

dsl, dsr : real

Constants

axisLength : real = width + 2*(radius/4 + 0.5 cm)

Equations

LHinge.das = dsl
RHinge.das = dsr
ls = radius*(dsl + dsr)/2
as = radius*(dsl - dsr)/axisLength

obstacle

(IR.voltage >= 3.0)

stop()

Action

move(0,0)
**Inputs**

*distance* : \( T \rightarrow \mathbb{R} 

**Outputs**

\( \text{las, ras} : T \rightarrow \mathbb{R} \)

**Behaviour Revolute**

\[
\begin{align*}
  v &= R \times b \times \text{das} / K + K \times \text{das} \\
  J \times \text{as}' + b \times \text{as} &= K \times i \\
  L \times i' + R \times i &= v - K \times \text{as}
\end{align*}
\]

**Behaviour IR**

\[
\text{voltage} = 4 \times e^{-0.028 \times \text{distance}}
\]
\[ A = (\text{Revolute}[\text{das} := \text{ldas}, \ldots]) \mid \text{Revolute}[\text{das} := \text{rdas}, \ldots]) \mid \text{IR} \]

\[ \text{Step}(l, r) = \mu X \bullet \left( (A \text{init das}, \text{rdas} = l, r) \right) \text{until} (\text{voltage} > 3); \text{obstacle} \rightarrow X \]

\[ M = \text{var } l, r : \mathcal{R} \bullet l, r := 0, 0; \mu X \bullet \text{Step}(l, r) \triangle \]

\[ \left( \text{move.ls.as} \rightarrow \{l, r\} : \left[ \begin{array}{ll}
\text{true}, & \text{ls} = \text{rd} \times (l + r)/2 \land \\
\text{as} = \text{rd} \times (l - r)/aL & \end{array} \right] ; X \right) \]
\[ A = (\text{Revolute}[\text{das} := \text{l das}, \ldots]) \mid \text{Revolute}[\text{das} := \text{r das}, \ldots]) \mid \text{IR} \]

\[
\text{Step}(l, r) = \mu X \bullet \left( (A \text{ init } \text{ldas}, \text{rdas} = l, r) \right. \\
\left. \text{until} (\text{voltage} > 3); \text{obstacle} \rightarrow X \right)
\]

\[
\mathcal{M} = \text{var } l, r : \mathcal{R} \bullet l, r := 0, 0; \mu X \bullet \text{Step}(l, r) \triangle \\
\left( \text{move.ls.as} \rightarrow \{l, r\} : \left[ \begin{array}{l}
\text{true, } ls = rd \times (l + r)/2 \land \\
\text{as} = rd \times (l - r)/aL \end{array} \right] ; X \right)
\]
\[ A = (\text{Revolute}[\text{das} := \text{ldas}, \ldots]) \mid \text{Revolute}[\text{das} := \text{rdas}, \ldots]) \mid \text{IR} \]

\[ \text{Step}(l, r) = \mu X \bullet \left( \text{\texttt{\textbf{init} l\text{das}, r\text{das} = l, r}} \right) \text{ until } (\text{voltage} > 3); \text{ obstacle } \rightarrow X \]

\[ M = \text{\texttt{var} } l, r : \mathcal{R} \cdot l, r := 0, 0; \mu X \bullet \text{Step}(l, r) \bigtriangleup \]

\[ \left( \text{move.l.s.as } \rightarrow \{l, r\} : \begin{cases} \text{true, } & l\text{s} = r\text{d} \times (l + r)/2 \land \text{as} = r\text{d} \times (l - r)/aL \end{cases} ; X \right) \]
$A = (\text{Revolute}[\text{das} := \text{ldas}, \ldots] \mid \text{Revolute}[\text{das} := \text{rdas}, \ldots] \mid \text{IR})$

$\text{Step}(l, r) = \mu X \bullet (\text{A init l\text{das}, r\text{das} = l, r})$

$\text{M} = \text{var} \ l, r : \mathcal{R} \bullet l, r : 0, 0; \mu X \bullet \text{Step}(l, r) \triangleright$

$(\text{move.ls.as} \rightarrow \{l, r\} : \left[\begin{array}{c}
\text{true, } \text{ls = rd } \times (l + r)/2 \wedge \\
\text{as = rd } \times (l - r)/aL \end{array}\right] ; X)$
\[ A = (\text{Revolute}[\text{das} := \text{ldas}, \ldots] | \text{Revolute}[\text{das} := \text{rdas}, \ldots] | \text{IR}) \]

\[ \text{Step}(l, r) = \mu X \bullet \left( (A \text{ init } \text{ldas}, \text{rdas} = l, r) \left[ \begin{array}{l} \text{Local Variables} \\ l, v : \text{real} \\ \text{Constants} \\ J, b, K, L, R : \text{real} \\ \text{Equations} \\ v = R^2 b^2 \text{das} / K + K \text{das} \\ J a^2 + b a = K t \\ t^2 + R^2 = v^2 K a \end{array} \right) \right) \]

\[ \text{A} = \text{var } l, r : \mathbb{R} \bullet l, r := 0, 0; \mu X \bullet \text{Step}(l, r) \triangleq \left( \text{move.ls.as} \rightarrow \{l, r\} : \left[ \begin{array}{c} \text{true, } \text{ls} = \text{rd} \times (l + r)/2 \\ \text{as} = \text{rd} \times (l - r)/\alpha L \end{array} \right] ; X \right) \]
Simple Model

\[
A = (\text{Revolute}[\text{das} := \text{ldas}, \ldots] \mid \text{Revolute}[\text{das} := \text{rdas}, \ldots] \mid \text{IR})
\]

\[
\text{Step}(l, r) = \mu X \bullet \left( \text{A init (ldas, rdas = l, r)} \right)
\]

\[
\mathcal{M} = \var l, r : \mathcal{R} \bullet l, r := o, o; \mu X \bullet \text{Step}(l, r) \triangle
\]

\[
\text{move.ls.as} \rightarrow \{l, r\} : \begin{cases}
\text{true}, & ls = rd \times (l + r)/2 \land \\
\text{as} = rd \times (l - r)/aL & \end{cases} ; X
\]
$$A = (\text{Revolute}[\text{das} := \text{ldas}, \ldots] \mid \text{Revolute}[\text{das} := \text{rdas}, \ldots] \mid \text{IR})$$

$$\text{Step}(l, r) = X \bullet (A \text{ init} \text{ldas}, \text{rdas} = l, r)$$

$$\text{M} = \text{var} \ l, r : \mathcal{R} \bullet l, r := 0, 0; X \bullet \text{Step}(l, r)$$

$$\begin{align*}
\text{move.l.s.as} & \rightarrow \{l, r\} : \begin{cases}
\text{true, } & \text{ls} = \text{rd} \times (l + r) / 2 \\
\text{as} = \text{rd} \times (l - r) / \text{aL} \end{cases} ; X
\end{align*}$$
Behaviours

$$\mathcal{A} = (\text{Revolute}[\text{das} \leftarrow \text{ldas}, \ldots] \mid \text{Revolute}[\text{das} \leftarrow \text{rdas}, \ldots] \mid \text{IR})$$
Behaviours

\[ A = (\text{Revolute}[\text{das} := \text{ldas}, \ldots] \mid \text{Revolute}[\text{das} := \text{rdas}, \ldots] \mid \text{IR}) \]

\[ \text{Step}(l, r) = \mu X \bullet \left( (A \text{ init } \text{ldas, rdas} = l, r) \text{ until } (\text{voltage} > 3); \right) \]

\[ \text{obstacle} \rightarrow X \]
Behaviours

$$\mathcal{A} = (\text{Revolute}[\text{das} := \text{ldas}, \ldots] \mid \text{Revolute}[\text{das} := \text{rdas}, \ldots] \mid \text{IR})$$

$$\text{Step}(l, r) = \mu X \bullet \left( (\mathcal{A} \text{ init } \text{ldas}, \text{rdas} = l, r) \text{ until } (\text{voltage} > 3); \right)$$

$$\text{obstacle } \rightarrow X$$

$$\mathcal{M} = \text{var } l, r : \mathcal{R} \bullet l, r := 0, 0;$$

$$\mu X \bullet \left( \text{Step}(l, r) \triangle \text{move.ls.as} \rightarrow \right.$$

$$\{l, r\} : \left[ \begin{array}{l}
\text{true, } ls = rd \times (l + r)/2 \land \\
\text{as} = rd \times (l - r)/aL
\end{array} \right] ; X \left. \right)$$
Semantics

- \( A \): behaviours of the platform model.
- \( Step \): behaviours in \( A \) until input events are true.
- \( M \): behaviours in \( Step \) interrupted by variables assignments, operation calls and output events
Conclusions

- RoboChart supports modelling including time and probability
- Formal semantics specified in CSP
- Tool support for modelling, verification and simulation
- RoboSim models can be
  - derived from RoboChart models
  - related to RoboChart models formally
- Partial support for modelling collections and robotic platforms
Modelling support for platform modelling
Case studies in platform modelling
Generation of
  ▶ SDF models
  ▶ simulation code
  ▶ formal semantics
Integration with RoboChart models via abstraction
Ana Cavalcanti, Alvaro Miyazawa, Augusto Sampaio, Wei Li, Pedro Ribeiro, and Jon Timmis.

**Modelling and Verification for Swarm Robotics.**
DOI: 10.1007/978-3-319-98938-9_1.

Alvaro Miyazawa, Pedro Ribeiro, Wei Li, Ana Cavalcanti, Jon Timmis, and Jim Woodcock.

**Robochart: Modelling and Verification of the Functional Behaviour of Robotic Applications.**
DOI: 10.1007/s10270-018-00710-z (To Appear).