Verifying System-Level Properties of Neural-Network Robotic Controllers

Ziggy Attala, Ana Cavalcanti, Jim Woodcock

ROBOSTAR

robostar.cs.york.ac.uk

28th May 2024

Royal Academy of Engineering

Engineering and Physical Sciences Research Council

Outline

Overview

- Verifying learning-enabled robotic systems is challenging.
- Existing techniques and tools for verifying ANNs: component-level properties.
- Our work: Verifying robotic systems with ANN control components.
- Model and verify entire control software with system-level properties.
- Focus on trained, fully connected, ReLU neural networks for control.
- Combine behavioural models and ANN models.
- Combine traditional and ANN-specific verification tools.
- We use RoboChart: a domain-specific robot modelling and verification framework.
- Strategy for automated proof using Isabelle/HOL and Marabou.

The Paper and the Thesis

- Ziggy Attala, Ana Cavalcanti, Jim Woodcock.
  Modelling and Verifying Robotic Software that Uses Neural Networks.
  ICTAC 2023: 15-35. Springer LNCS.
- Ziggy Attala.
  Verification of RoboChart Models with Neural Network Components.
  PhD Thesis, University of York. October 2023.


Modelling and Verifying Robotic Software that Uses Neural Networks

1 Introduction

Outline

Motivation

- Robots are leaving their cages.
- Trustworthiness requires verification.
- Current approach to software engineering: ad hoc, code centric.
- Domain-specific modelling languages.
- Tractable mathematical models.
- Challenge: integrated reasoning.
- Systems engineering.
- Heterogeneous models.
- Verification tools: focused on ANN.
- Neural networks for control.

Example 1: Controller for Robot Motor

- Neural network controller. Single sensor input: floor gradient.
- Goal: adjust the motor to maintain robot speed.
- Input Layer: single neuron representing the sensor reading.
- Output Layer: single neuron converts gradient to motor input voltage.
- Gradient voltage (0–1V) needs to be scaled and mapped to motor voltage (0–6V).
- Multiply by scaling factor 5 to map into motor voltage range.
- Requirement: motor requires a minimum voltage of 1V to start moving.
- Add bias of 1V could be added to the scaled neuron output.
- Scaled and biased neuron output converted to actual voltage signal by DAC.

Example 2: Controller for Robotic Arm

- Neural network controller: robot arm for sorting objects based on their weight.
- Hidden layer: two neurons to capture different features of the input.
- The two neurons capture different weight ranges.
- Allows network to make more accurate sorting decisions.

Example 3: Controller for Robot with Autonomous Navigation

- Robot controller: steering angle based on distance to nearest sensed obstacle.
- Two hidden layers compute different features from the input data.
- Hidden Layer 1. Responsible for low-level feature distance to nearest obstacle.
- Identifying different distance ranges (e.g., near, medium, far).
- Recognising changes or gradients in the distance values.
- Extracting simple features related to the obstacle's proximity.
- Hidden Layer 2. Compute higher-level representations from low-level features.
- Map distance to angle ranges: sharp turn, moderate turn, slight turn, straight.
- Identify patterns that need obstacle avoidance or course correction.
- Learn non-linear map between distance and required angle adjustment.

Example 4: Neural Network with Probabilistic Output

- Robot arm Grasp and manipulate objects of different shapes, sizes, and materials.
- Predict probability distribution over different grasping strategies or configurations.
- Use input information about the object and its environment.
- Input Layer 3D point cloud data: depth sensors or cameras.
- Information about the robot's current state: arm joint angles, gripper position.
- Hidden Layers Extract spatial features and patterns.
- Output Layer Multiple neurons, each representing a different grasping strategy.
- Strategies: top grasp, side grasp, pinch grasp, etc.
- Output Predicted probability for corresponding grasping strategy.
- Activation Function: Softmax. Normalises scores.

Why use Neural Networks for Control?

- Handling complex and non-linear environments: Robot control in dynamic, unstructured environments. Learn complex, non-linear mappings from data.
- Adaptability and generalisation: New situations not explicitly covered in training data. Operating environments with changing conditions and novel scenarios.
- Learning from Experience: Training with reinforcement learning to improve behaviour. Continuously adapt to changing conditions and new tasks.
- Handling High-dimensional Data: Process and integrate high-dimensional data from sensors. Extract relevant features. Challenging for traditional algorithms.
- End-to-End Control: Training maps raw sensor data directly to control outputs. Enables end-to-end control without feature engineering or state estimation.
- Parallel Processing: Real-time control tasks require low latency and high throughput. Use GPUs and specialised hardware accelerators.
- Scalability and Modularity: Modular and scalable ANNs. Integrate new sensors, control outputs, and task-specific modules. No control system redesign.

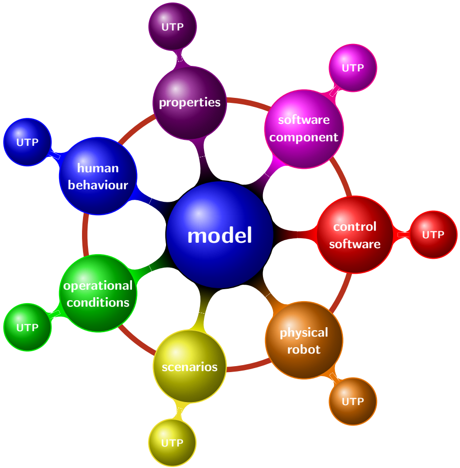
Why not use Neural Networks for Control?

- Replacing traditional controller with ANNs is challenging.
- It needs large amounts of training data.
- The controller is potentially unstable.
- There are correctness and safety concerns.
- There are difficulties in interpreting and explaining the learned control policies.
- In practice, many robotic systems use a hybrid approach.
- ANNs: specific tasks or modules. Perception, motion planning, low-level control.
- Traditional controllers handle higher-level decision-making.
- Usually task planning and safety-critical operations.
- Engineering decisions: choice between traditional and ANN controllers.
- Depends on specific robot application requirements, constraints, trade-offs.

Outline

RoboStar Vision

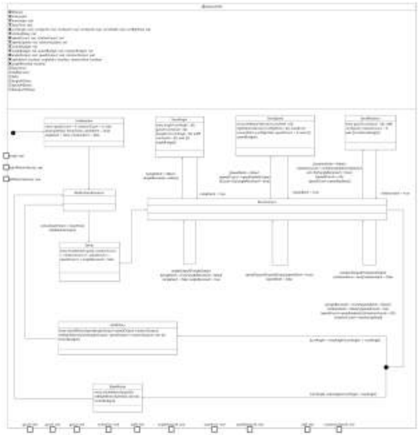
1. Simulation with commercial tools.
2. Coding in practical languages.
3. Tests: simulation, deployment.
4. Proof: model checking, theorem proving.
5. Evidence of properties.
6. Safety, security, more.
7. Significant asset: RoboTool.
8. Application agnostic.



Core Notation: RoboChart

RoboChart

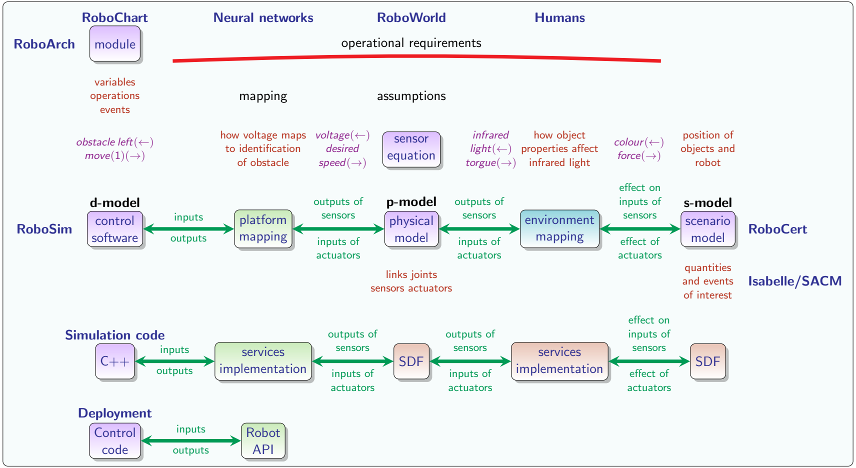
1. Statecharts for behaviour.
2. Parallel execution of statecharts.
3. Simple component model.
4. Synchronous or asynchronous.
5. Platform independent.
6. Capabilities: events and operations.
7. Timed behaviours.
8. Probabilistic choice.



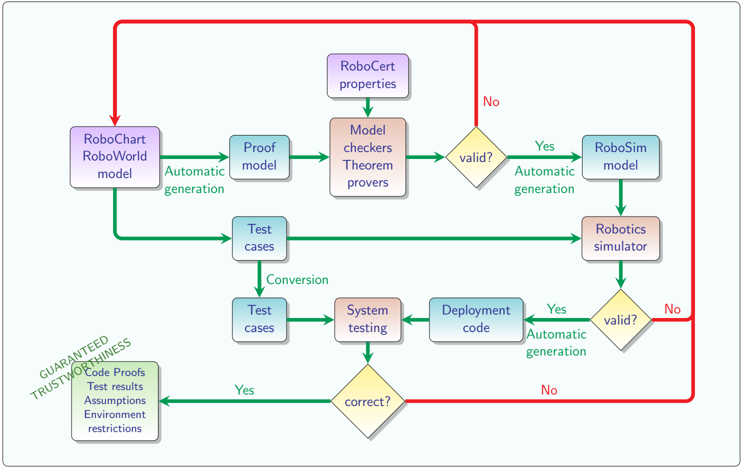
Deriving Value: RoboChart

1. Simulation model: cyclic mechanism.
2. Simulation code: CoppeliaSim, Gazebo, Drake, RT-Tester.
3. Deployment code.
4. Automatic test generation.
5. RoboWorld: operational requirements.
6. Model checking: FDR and PRISM.
7. Theorem proving: Isabelle/UTP.
8. RoboCert: property specification.
9. Ongoing work: neural networks, human behaviour, safety cases.

RoboChart Modelling Stack



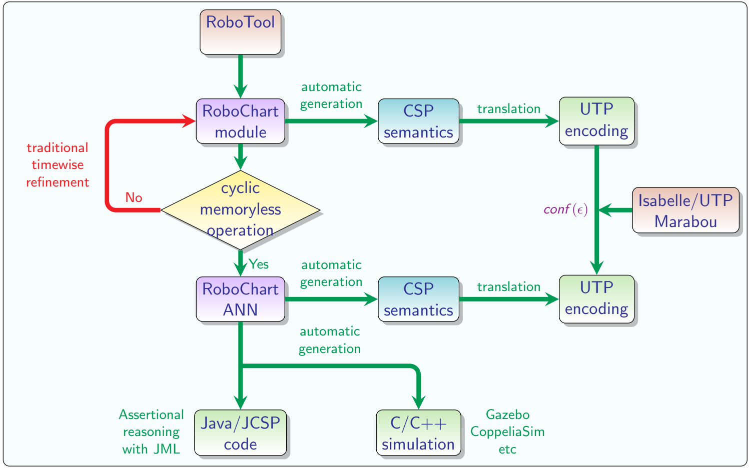
RoboStar: Comprehensive Support



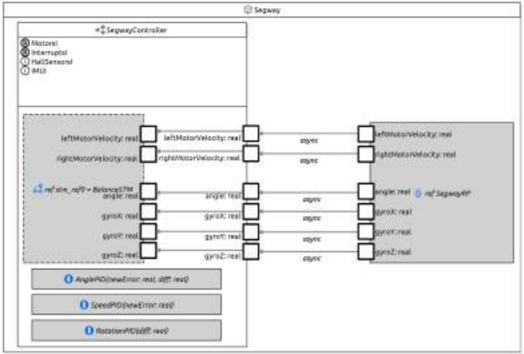
18/38

Neural Networks in RoboChart

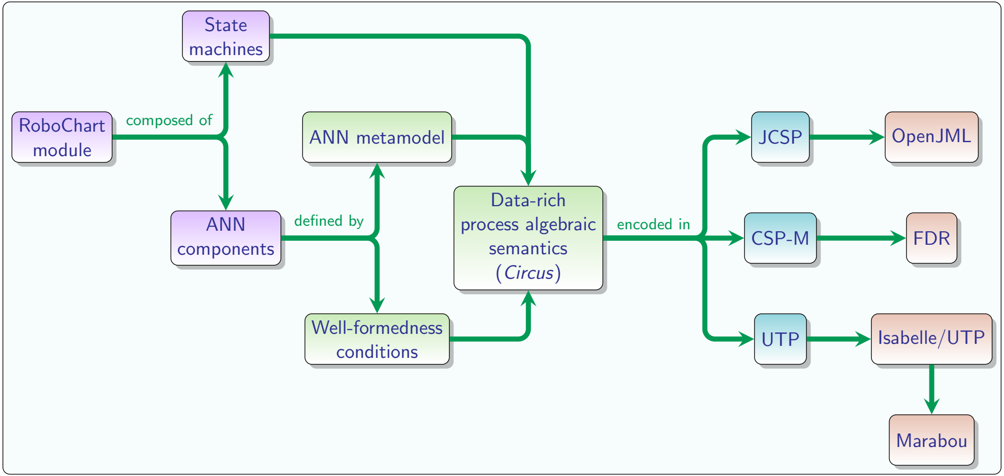
- Trained
- Feed forward
- Fully connected
- ReLU or linear activation



Example: A Segway

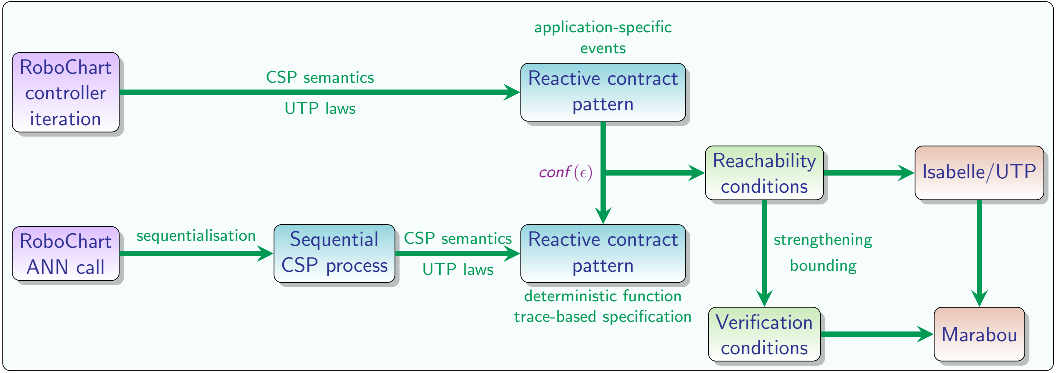


RoboChart with ANN: Our Language



Outline

RoboChart with ANN: Verification



CSP Models of ANNs

- Neurons as Processes Each neuron is represented as a concurrent process.
  Processes communicate through channels, representing weights between neurons.
- Communication and Synchronisation Modelled using CSP's primitives.
  This formalises information flow and computation within the neural network.
- Parallel and Distributed Computation Multiple neurons execute simultaneously.
- Formal Verification Theorem proving in Isabelle/UTP, model checking in FDR4.
  Check for convergence, stability, robustness, and specific properties.
- Compositionality Scaling analysis and verification of larger ANNs.
- Active research area to provide formal foundations for ANNs.

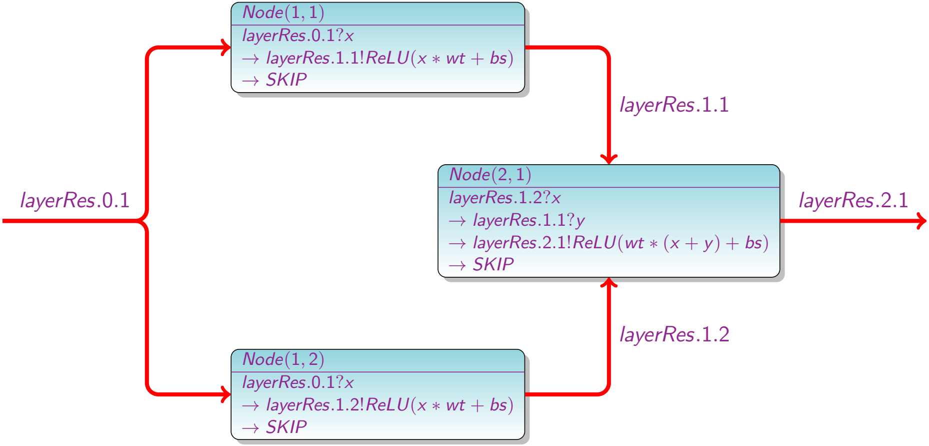
CSP Dataflow Architecture for ANNs

- Model an ANN as a recurrent dataflow network with transforming-buffer nodes.
- Implement this model in CSP. Analyse it in Isabelle/UTP and FDR4.
- Transformation totality ensures network totality.
- Dataflow architecture ensures network deadlock-freedom.
- Dataflow architecture ensures network divergence-freedom.
- Architecture and transformations ensure network determinism.

Simple Example: Generic ANN

- Consider an ANN with one input layer, N_h hidden layers, and one output layer.
- Layers are indexed between $0 \mathinner{\ldotp\ldotp} layerNo$, where $layerNo = N_h + 2$.
- Nodes are connected with communication channels.
- Layer l, node n has inputs on $layerRes.(l-1).n$ and outputs on $layerRes.l.n$.
- Consider one input node, one hidden layer with two nodes, and one output node.
- There are four channels: $layerRes.0.1$, $layerRes.1.1$, $layerRes.1.2$, $layerRes.2.1$.
- Three processes: $Node(1,1)$, $Node(1,2)$, $Node(2,1)$, two hidden, one output.
- There is no material behaviour in the input node.
- Process behaviour: $Inputs \mathbin{;} Outputs$. Network is recurrent, left implicit.
- $layerRes.1.2?x \rightarrow layerRes.1.1?y \rightarrow layerRes.2.1!ReLU(wt * (x + y) + bs) \rightarrow SKIP$

CSP Model of an ANN: 1–1–1 Layers



CSP Model for ANN

- $ANN = ((HiddenLayers \; [\![\{\!| layerRes.(layerNo - 1) |\!\}]\!] \; OutputLayer) \setminus HiddenEvts \; \triangle_{end} \; Skip) \; ; \; ANN$
- $HiddenEvts = \Sigma \setminus \{\!| layerRes.0, layerRes.layerNo, end |\!\}$
- $HiddenLayers =$
  $\big\|\; i : 1 \,..\, layerNo - 1 \bullet [\{\!| layerRes.(i-1), layerRes.i |\!\}] HiddenLayer(i, layerSize(i), layerSize(i-1))$
- $HiddenLayer(l, s, inpSize) = \big\|\; i : 1 \,..\, s \bullet [\{\!| layerRes.(l-1) |\!\}] \; Node(l, i, inpSize)$
- $Node(l, n, inpSize) =$
  $(|||\; i : 1 \,..\, inpSize \bullet NodeIn(l, n, i)) \; [\![\{\!| nodeOut.l.n |\!\}]\!] \; Collator(l, n, inpSize)) \setminus \{\!| nodeOut |\!\}$
- $NodeIn(l, n, i) = layerRes.(l-1).i?x \rightarrow nodeOut.l.n.i!(x * weight) \rightarrow Skip$
- $Collator(l, n, inpSize) = \mathbf{let}\; C(l, n, 0, sum) = layerRes.l.n!(ReLU(sum + bias)) \rightarrow Skip$
  $C(l, n, i, sum) = nodeOut.l.n.i?x \rightarrow C(l, n, (i-1), (sum + x))$
  $\mathbf{within}\; C(l, n, inpSize, 0)$
- $OutputLayer =$
  $\big\|\; i : 1 \,..\, layerSize(layerNo) \bullet$
  $[\{\!| layerRes.(layerNo - 1) |\!\}] \; Node(layerNo, i, layerSize(layerNo - 1))$

Marabou

- SMT-based neural network verification tool from Stanford University and Galois.
- Gives formal guarantees about properties and outputs.
- Robustness Verify behaviour wrt input perturbations and adversarial attacks.
  Determine maximum perturbation for unchanged output wrt specified threshold.
- Output Range Analysis Possible output values for given input range.
- Input-Output Check if input patterns always lead to specific output patterns.
  Check if certain output classes are never produced for a given set of inputs.
- Safety Properties Ensure output doesn't exceed certain thresholds.
  Ensure certain inputs never lead to unsafe outputs.
- Can be used as part of end-to-end verification. RoboStar!

Outline

Reactive Contracts in UTP

- Contract extension for semantics of state-rich CSP processes.
- Provides a rich set of algebraic laws for process verification.
- Observational variables:

st, st'	:	$Var \nrightarrow Val$	program state
ok, ok'	:	$Bool$	initiation and termination
tr, tr'	:	$\mathrm{seq}\ Event$	event traces
tt'	:	$\mathrm{seq}\ Event$	process's event trace $tr' - tr$
$wait, wait'$	:	$Bool$	quiescence
ref, ref'	:	$\mathbb{P}\ Event$	refusal sets

Reactive Contracts

- Syntax: $[\, P[st] \vdash Q[tt', st, ref'] \mid R[tt', st, st'] \,]$.
- Semantics: $ok \land P[tt, st] \Rightarrow ok' \land (Q[tt', st, ref'] \lhd wait' \rhd R[tt', st, st'])$.

- Precondition P: condition on pre-state st.
- Postcondition R: relation on state st, update st', event trace tt'.
- Pericondition Q: relation on quiescent but not final observations.

  Relation on pre-state st, event trace tt', refusals ref'.

Reactive Contracts

- Simple pattern for contracts: $\textsc{Peri}[t, E]$ and $\textsc{Post}[t]$.
- CSP processes without state variables.
- Pericondition $\textsc{Peri}[t, E]$: Event trace t observed. Event set E not refused.

  $$\textsc{Peri}[t, E] \mathrel{\widehat{=}} tt' = t \land ref' \cap E = \emptyset$$

- Postcondition $\textsc{Post}[t]$: Event trace t has been observed.

  $$\textsc{Post}[t] \mathrel{\widehat{=}} tt' = t.$$

- Channel set $\{\!|c|\!\}$: all events communicable on channel c.

Conformance

$$\begin{array}{l} Q \ \mathit{conf}(\epsilon) \ P \ \Leftrightarrow \\ \quad \exists \, s : \mathrm{seq}\, \mathit{Event}; \ a : \mathbb{P}\, \mathit{Event} \mid \\ \quad\quad \mathit{tt} \ \mathit{seqapprox}(\epsilon) \ s \wedge (\alpha P \setminus \mathit{ref}') \ \mathit{setapprox}(\epsilon) \ a \bullet \\ \quad\quad\quad P[s, (\alpha P \setminus a) \ / \ \mathit{tt}, \mathit{ref}'] \sqsubseteq Q \end{array}$$

- s: approximation of traces
- a: approximation of acceptances

Only outputs are approximated.

Outline

Contributions

- Method for robotic software with reliable, white-box ANN components.
- Deductive guarantees on the behaviour of system-level properties.
- Platform-independent models for validation, simulation, and verification.
- Metamodel: trained, feed-forward, fully connected ANNs. Any size or shape.
- General, extensible, formal representation of ReLU ANNs.
- Validation using FDR4 model checker. Simulation using JCSP.
- Reactive contract theory enables verification using Isabelle/UTP.

Contributions

- ANN property proof method based on refinement.
- Numerical instability of ANNs. Provides worst-case error bound.
- Substitutability: ANN for RoboChart controller. Guaranteed error bound.
- Example case study: inverted pendulum PID controller.
- Translate reactive contract to multiple input/output reachability properties.
- Integrated approach to reason about ANN, using a variety of techniques
- Simulation: Java and standard tools. Proof: Isabelle/UTP + Marabou.

Future Work

- More case studies.
- Challenge problems.
- Timed models.
- Probabilistic models.
- Simulation models.
- Perception.

ROBOSTAR