Runtime verification

Sandor M Veres, Hongyang Qu and Paolo Izzo
ACSE, University of Sheffield

December 3, 2015

Why do we need runtime verification?

Our more limited focus in this talk

The agent reasoning cycle
- Definition of a simplified BDI agent
- Simplifications of agent operations

LISA abstractions to DTMC

LISA abstractions to MDP

Principles of applicable plan selection
- Constraints of plan selection

Purposes of runtime verification

- Monitoring own capabilities (physical and computational) so that plannning can be correctly applied.

Purposes of runtime verification

- Monitoring own capabilities (physical and computational) so that plannning can be correctly applied.
- Checking that a decision on a solution to a problem will not result in undesirable physical effects on the environment and other agents.

Purposes of runtime verification

- Monitoring own capabilities (physical and computational) so that plannning can be correctly applied.
- Checking that a decision on a solution to a problem will not result in undesirable physical effects on the environment and other agents.
- The agent learns new skills and capabilities - their use needs verification.

Purposes of runtime verification

- Monitoring own capabilities (physical and computational) so that plannning can be correctly applied.
- Checking that a decision on a solution to a problem will not result in undesirable physical effects on the environment and other agents.
- The agent learns new skills and capabilities - their use needs verification.
- The agent learns new ways of achieving goals by interaction with other agents - legal/moral/ethical verification is needed.

Runtime verification for decision making

- A limited instruction set agent (LISA) is introduced. to reduce the complexity and increase the verifiability of decision making.

Runtime verification for decision making

- A limited instruction set agent (LISA) is introduced. to reduce the complexity and increase the verifiability of decision making.
- The new decision method is structurally simpler and easily lends itself to both design time and runtime verification methods.

Runtime verification for decision making

- A limited instruction set agent (LISA) is introduced. to reduce the complexity and increase the verifiability of decision making.
- The new decision method is structurally simpler and easily lends itself to both design time and runtime verification methods.
- The process of converting a control agent in LISA into a model in a probabilistic model checker is described.

Runtime verification for decision making

- A limited instruction set agent (LISA) is introduced. to reduce the complexity and increase the verifiability of decision making.
- The new decision method is structurally simpler and easily lends itself to both design time and runtime verification methods.
- The process of converting a control agent in LISA into a model in a probabilistic model checker is described.
- Feasibility of runtime probabilistic verification is investigated and illustrated in the LISA agent programming system for verifying symbolic plans of the agent using a probabilistic model checker.

Limited instruction set agent - LISA

- *Initial Beliefs*.
  The initial beliefs and goals $B_0 \subset F$ are a set of literals that are automatically copied into the *belief base* B_t (that is the set of current beliefs) when the agent mind is first run.
- *Initial Actions*.
  The initial actions $A_0 \subset A$ are a set of actions that are executed when the agent mind is first run. The actions are generally goals that activate specific plans.
- *Logic rules*.
  A set of logic based implication rules L describes *theoretical* reasoning to improve the agent current knowledge about the world.

Limited instruction set agent - LISA

- *Executable plans*.
  A set of *executable plans* or *plan library* Π. Each plan π_j is described in the form:

$$p_j : c_j \leftarrow a_1, a_2, \ldots, a_{n_j} \tag{1}$$

  where $p_j \in B$ is a *triggering predicate*, which allows the plan to be retrieved from the plan library whenever it comes true, $c_j \in B$ is called the *context*, which allows the agent to check the state of the world, described by the current belief set B_t, before applying a particular plan, and $a_1, a_2, \ldots, a_{n_j} \in A$ is a list of actions.

Jason - a more complex agent language

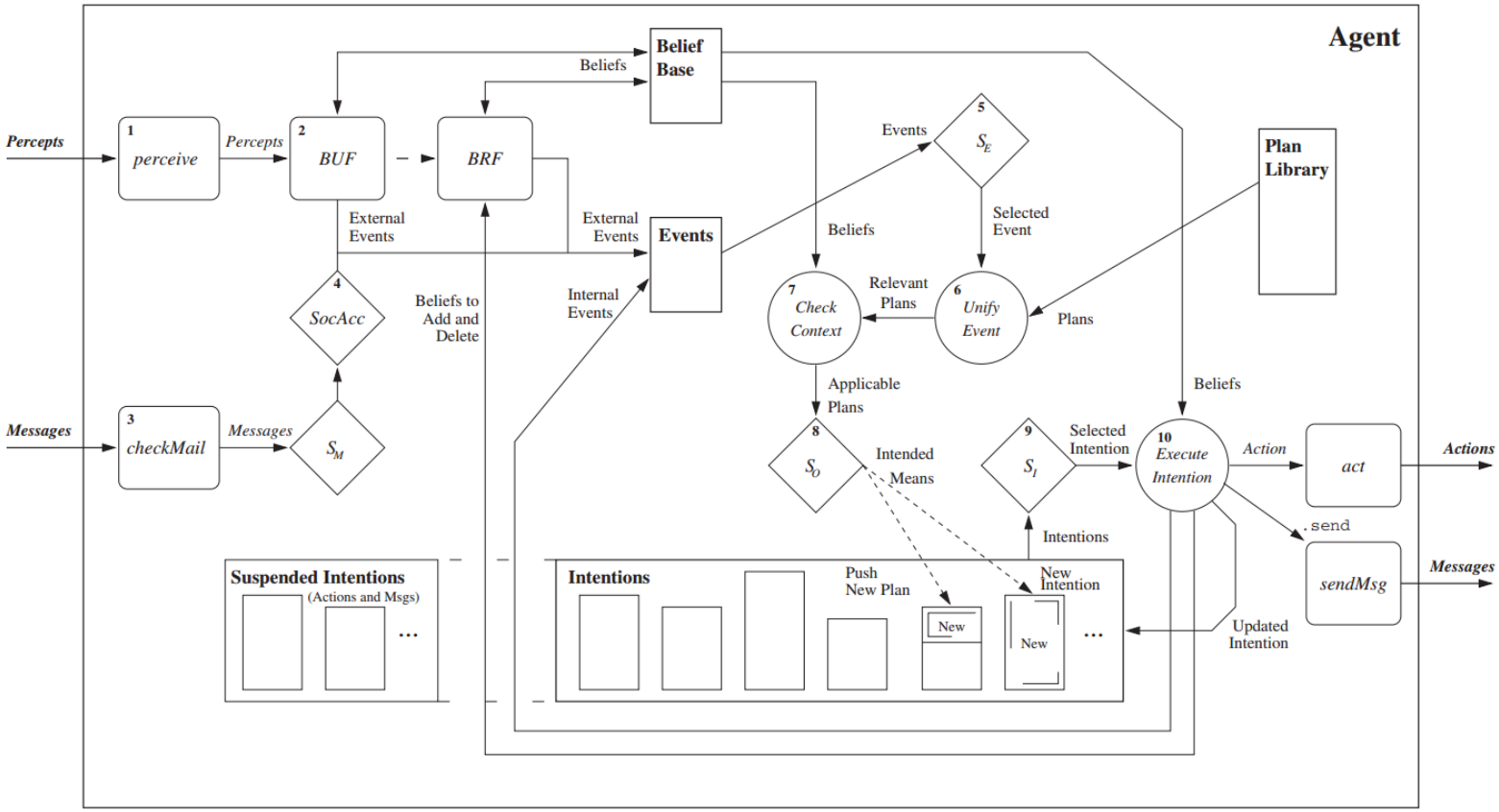

Figure: The *Jason* reasoning cycle, LISA simplifies this scheme while enabling limited complexity reasoning about future actions by model checking.

Reasons for simplfication

The purpose of simplifying the reasoning structures relative to AgenSpeak languages in use today, is threefold:

1. To make complexity of design time verification simpler.
2. To enable fitting of model checking for runtime verification of selected plans and actions.
3. To make logic based decisions more robust to mistakes in the a priori design of agent reasoning, and especially less fragile to badly timed appearance of predicates in the belief base.

Overview of simplifications

- *Beliefs and Goals*.
  In the Jason agent there is a distinction between beliefs and goals. Beliefs are what the agent knows about the world, and goals are special mental notes that the agent does not keep in the belief base when the plan they trigger is achieved. This distinction in a practical sense does not improve the implementation process. For this reason in LISA we drop the distinction between beliefs and goals.

Overview of simplifications

- *External Actions*.
  In LISA we introduce a new classification for external actions that can be either of type *runOnce*, if the action terminates itself after a single execution or *runRepeated*, if the action runs continuously until actively stopped by the agent. The latter implies that the agent is capable of monitoring the outcome of the *runRepeated* actions using its perception processes. In either case the external functions that execute actions can send predicates to the current belief base in the form of *action feedbacks*.

Overview of simplifications

- *Perception*.
  In LISA perception predicates can be of three types: *sensory perception*, *communication* and *action feedbacks*. As in Jason all the perception predicates generate events that can in turn trigger plans that are part of the plan library.
- *Logic rules*.
  In Jason the only way to internally add or remove beliefs from the belief base is with internal actions. In LISA predicates also arrive from action processes directly. Logic-based implication rules are not deeply implemented, and the main text itself advises against their use. In LISA we process logic rules until stable conclusions in each reasoning cycle.

LISA Operations

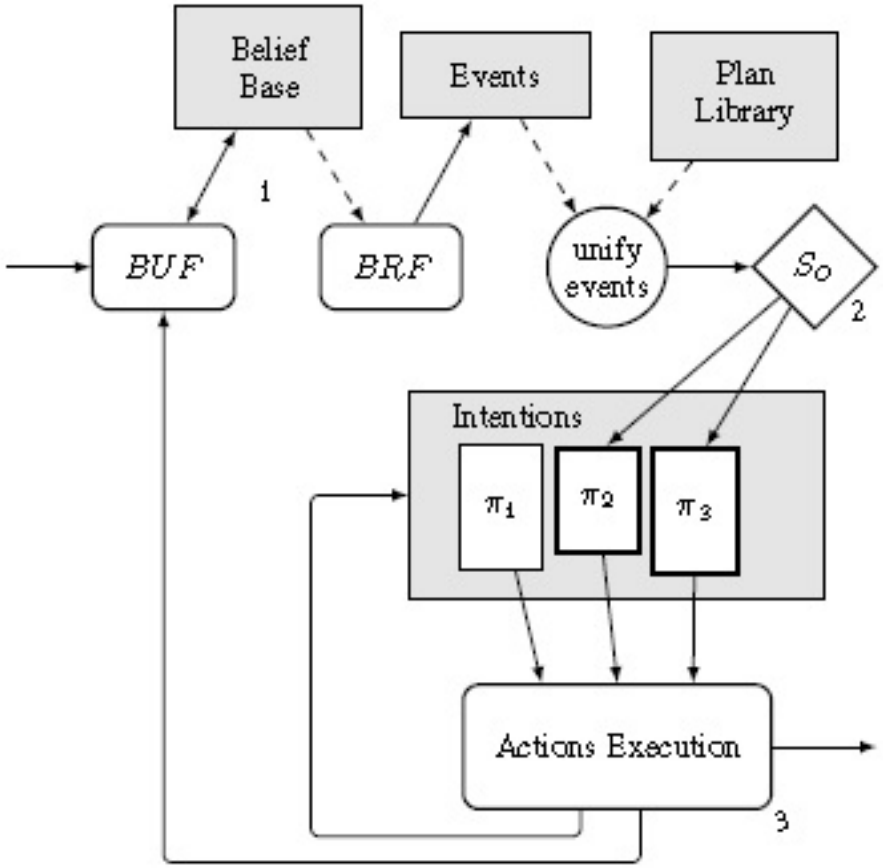

Figure: LISA operations (by Paolo Izzo, PGR Sheffield).

Description of the reasoning cycle of LISA - 1

LISA belief base update.
At the beginning of every reasoning cycle, the BUF checks for all the input coming from perception, action feedback and communications and updates the current belief set B_t. The BUF function also removes perception and action feedback predicates if they are not received persistently. For instance indication of a task completed may be a feedback sent only once while sensor based detection of dynamical instability can be a persistent feedback. The BRF generates a set of events $E_t = B_t \setminus B_{t-1}$ at every reasoning cycle. The LISA agent does not select a single trigger for intention but executes all plans with true context plans in a multi-threaded way.

Description of the reasoning cycle of LISA - 2

Plan selection.
Next the agent looks at the current event set E_t and retrieves all the plans from the plan library Π that are triggered by these events, it checks that the context meets the current beliefs, and starts executing the plan if the context is satisfied. To enable correct plan execution, predicates of mental notes can be used in the contexts of plans to enable the consistent execution symbolic plan generated by a plan dedicated to symbolic planning.

Description of the reasoning cycle of LISA- 3

Action Execution.
The agent retrieves the next action to be executed from each plan and calls for external or internal functions to execute the action. Similarly to Jason, a plan is suspended while waiting for a completion of an external action (to prevent runaway plan execution). Conditional actions hold up the execution of a plan until they are satisfied so they should be used with care to ensure eventual closure of plans.

Theorem 1

Theorem

Assuming an independent probability distribution of all action feedbacks and sensory events from the environment, the complete decision making of a LISA can be modelled by a DTMC.

Outline proof

As during the execution of any action(s) in some plan(s) is(are) represented by specific predicates in B, the power set $\wp(B)$ (the set of subsets of B) forms a state space for the agent as any $s \in \wp(B)$ provides complete information for the decision making of the agent. The state of the DTMC is initialised by its initial set of beliefs, goals and probabilities of feedback from its initial actions. Transition to a new state of the agent only happens at the end of each reasoning cycle. The new set of predicates for the new state is developed in two steps: (1) by deterministic application of reasoning rules, triggering or closure of new plans and elimination of predicates due to the semantics of LISA's execution (2) probabilistic appearance of perception predicates and action feedback predicates in B. The probability distribution of new sensory and action feedback predicates is well defined in any LISA program and hence modelling as a DTMC can be completed.

Theorem 2

Theorem

Assuming that both human communication and the physical environment are possible to model by MDPs, which are possibly inter-dependent by conditional probabilities, the complete decision making of a LISA in its environment can be modelled by an MDP.

Outline proof

The alteration relative to the proof of Theorem 1 is that the probabilistic appearance of perception and action feedback predicates is this time determined by the MDPs of human communication and interaction with the agent and predicates of action feedback from the physical environment. Due to the MDPs for both human responses as well as environmental responses well defined, all the state transition probabilities can be calculated by basic rules of conditional probabilities and the proof is hence completed.

Constraints of plan selection

- *Temporal order of actions*.
  The agent can be programmed so it has the ability to memorise and later check for past actions. Some actions must be preceded by others, for instance the agent is not allowed to pick up an instrument that it never deployed in the environment.
- *Actions performed in the context of operational modes*.
  Operational modes allow to give a general context to the status of the mission and can be used to reduce the number of associations between course of actions and possible outcomes .
- *Environmental dependency*.
  An autonomous vehicle is designed to operate autonomously in any condition. External events shape the way the system is going to act throughout the mission. For instance if the agent experiences a communication loss with other agents, etc.

Action selections tree

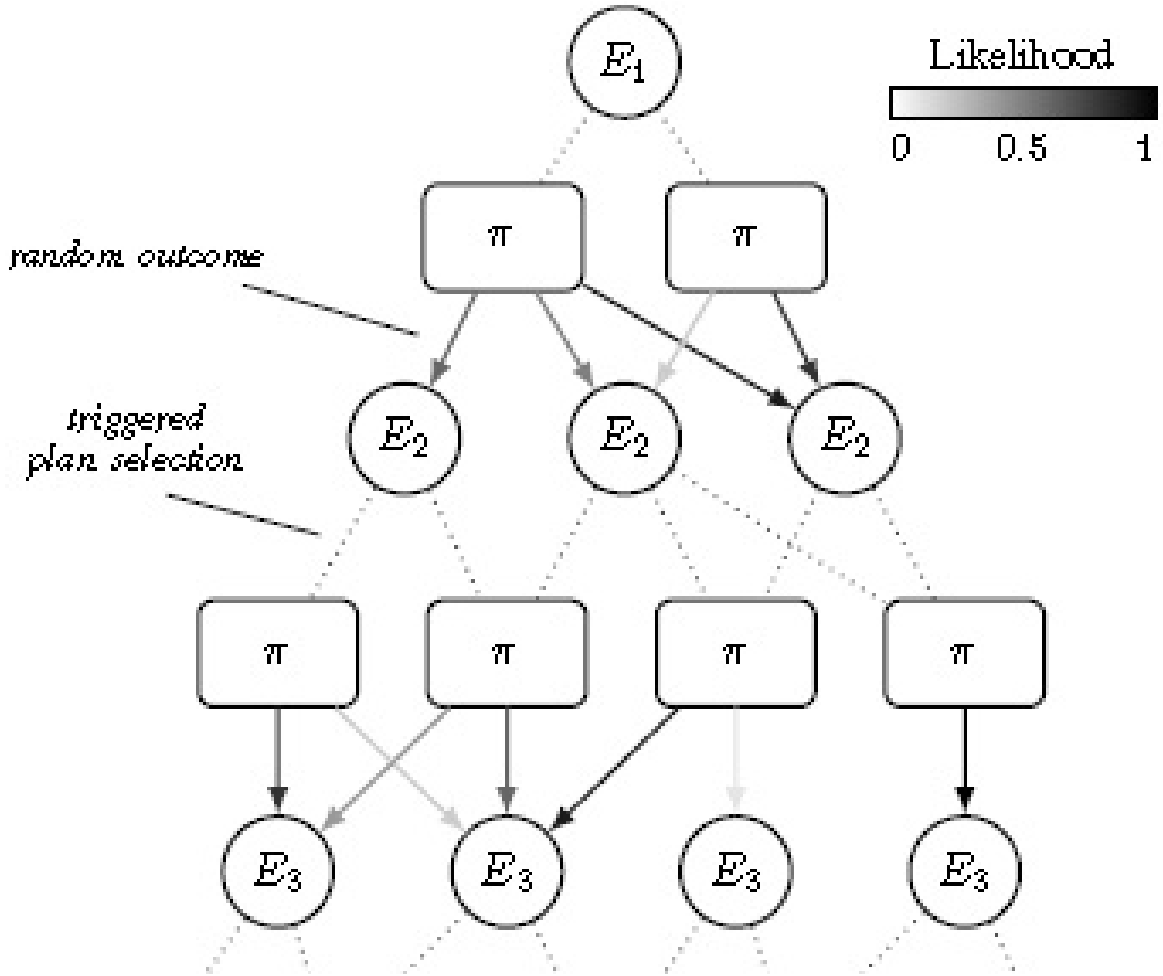

Figure: Decision tree of anticipated future possibilities (by Paolo Izzo, PGR Sheffield).

Possible course of plans tree

Let us define a "course of plans" as a sequence of plans $\hat{\Pi} \subset \Pi$, a branch of the tree in the previous figure associated with a particular goal represented by a set of events E_t that the agent is committed to pursue, and the likelihood that the course of action leads to the successful achievement of said goal:

$$c_{\pi}^{(j)} = \left\{ \hat{\Pi}^{(j)}, E_t^{(j)}, \lambda^{(j)} \right\} \tag{2}$$

where the index j numbers each branch of the three . At any moment in time the agent will have some knowledge of the environment in the form of belief predicates, it will know in what operational state it is operating at the moment and it will have knowledge of what has been done in the past, still in the form of predicates.

Possible course of plans tree

At any particular time a plan can have multiple course of actions associated with it. Using both pre-computed and runtime simulation results, it would be possible to predict the one with the highest likelihood of success to determine whether or not the plan is likely to lead to the achievement of the current goals. To do this "reward value" can be applied to every applicable plan in Π_t. Let us define a "reward function" $f_{\mathcal{R}}$ that uses these principles to assigns a *reward* value to every plan:

$$\mathcal{R}_t : \Pi_t \rightarrow \mathbb{R}^+ \tag{3}$$

Possible course of plans tree

This association is clearly time dependent in the sense that the same plan might have a variable likelihood of success value when applied at different points of the mission under different environmental conditions. For this reason, at run time a "reward update function"(which is defined at design time of the agent) needs to be evaluated each time the agent is required to make a choice between different plans, and so different course of actions:

$$\mathcal{RU} : \Pi_{t-1} \times \mathbb{R}^{+} \rightarrow \Pi_{t} \times \mathbb{R}^{+} \tag{4}$$

The above runtime schema and method enables the agent to consider it options for the future.

Runtime model checking example in LISA

Consider AV on an exploration mission. The vehicle has to explore two areas, that we will call “Area A” and “Area B”.
The system is trying to carry out two sets of actions:

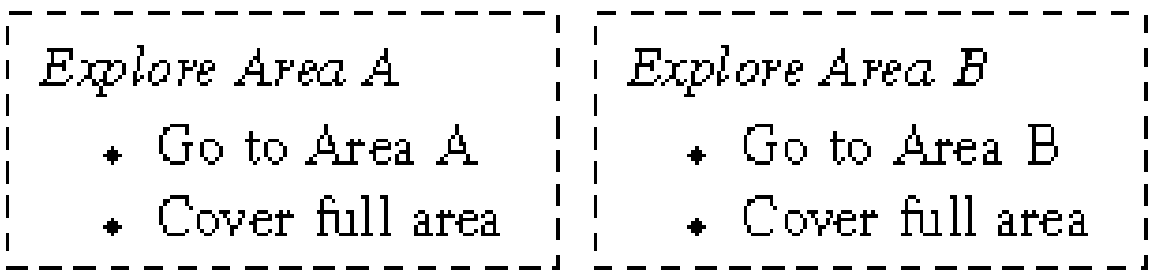

Figure: Exploration problem.

Runtime model checking example in LISA

Each area is partitioned into a number of blocks: $A_1, A_2, \ldots, A_{N_A}$ for Area A and $B_1, B_2, \ldots, B_{N_B}$ for Area B. The exploration of the areas can be done in the same number of steps accordingly, one block for each step. The vehicle consumes one unit of fuel to explore one block. During the exploration, the weather can change: with probability p, it becomes bad and with $1 - p$ it becomes good. When the former happens, the vehicle consumes twice as much fuel to explore each block. When the weather in the other area is good, it can move to that area with probability $1 - q$ of a successful passage. When one area is fully explored, the vehicle goes to the other area, assuming there is still enough fuel. Moving between the two areas consumes one unit of fuel, as well as going to each area from the base. At any point during the mission, if the fuel tank becomes almost empty, e.g. when the fuel in the tank is one unit, the vehicle has to go back to the base. Once both areas are explored, the agent will head back to the base.

PRISM code generated by a LISA

```
1   dtmc
2
3   const int No = 15;
4   const int Na = 5;
5   const int Nb = 5;
6
7   const double Pa = 0.1; // probability of bad weather in Area A
8   const double Pb = Pa;  // probability of bad weather in Area B
9   const double Pi = 0.5; // probability of initial choice between Area A and
        Area B
10  const double Ps = 0.6; // probability of switch between Area A and Area B in
        case of bad weather
11
12  module robot1
13      a1 : [0..Na] init 0; // Area A
14    b1 : [0..Nb] init 0; // Area B
15    oil : [0..No] init No; // oil level
16    s  : [0..3] init 0; // 0: base, 1: Area A, 2: Area B, 3: mission aborted
17
18    // initial choice
19      [] (s = 0) & (a1 = 0)& (oil > 0)
20      -> Pi: (s'=1)&(oil'=oil-1) + (1-Pi): (s'=2)&(oil'=oil-1);
21
```

Figure: Part 1 of the runtime RPISM code.

PRISM code generated by a LISA

```
22      // decision in Area A
23    [tick2] (s = 1) & (t = 0) & (a1 < Na) &
24    (oil > 0) & (w1 = 0) -> (a1'=a1+1) & (oil'=oil-1);
25    [tick2] (s = 1) & (t = 0) & (a1 = Na)
26    & (b1 < Nb) & (oil > 0) -> (s'=2) & (oil'=oil-1);
27    [tick2] (s = 1) & (t = 0) & (a1 < Na) & (b1 < Nb)
28        & (oil > 1) & (w1 = 1) & (w2 = 1)
29                -> (a1'=a1+1) & (oil'=oil-2);
30    [tick2] (s = 1) & (t = 0) & (a1 < Na)  & (b1 < Nb)
31        & (oil > 1) & (w1 = 1) & (w2 = 0)
32          -> Ps: (a1'=a1+1) & (oil'=oil-2) +                                n
33                (1-Ps): (s'=2) & (oil'=oil-1);
34
35      // decision in Area B
36    [tick2] (s = 2) & (t = 0) & (b1 < Nb) & (oil > 0)
37    & (w2 = 0) -> (b1'=b1+1) & (oil'=oil-1);
38    [tick2] (s = 2) & (t = 0) & (b1 = Nb) & (a1 < Na)
39    & (oil > 0) -> (s'=1) & (oil'=oil-1);
40    [tick2] (s = 2) & (t = 0) & (b1 < Nb) & (a1 < Na)
41    & (oil > 1) & (w2 = 1) & (w1 = 1)
42              -> (b1'=b1+1) & (oil'=oil-2);
43    [tick2] (s = 2) & (t = 0) & (b1 < Nb) & (a1 < Na)
44    & (oil > 1) & (w2 = 1) & (w1 = 0)
45            -> Ps: (b1'=b1+1) & (oil'=oil-2) + (1-Ps): (s'=2) & (oil'=oil-1);
46     [tick2] (s > 0) & (s < 3) & (oil = 0) -> (s'=3);
47
```

Figure: Part 2 of the runtime RPISM code.

PRISM code generated by a LISA

```
48  endmodule
49
50  module environment
51    t: [0..1] init 1; // control weather change
52
53    [tick1] (s > 0) & (s < 3) & (t = 1) -> (t' = 0);
54    [tick2] (s > 0) & (s < 3) & (t = 0) -> (t' = 1);
55  endmodule
56
57  module weather1
58      w1 : [0..1];
59
60    [tick1] (s > 0) & (s < 3) & (t = 1) -> Pa:(w1' = 1) + (1-Pa):(w1'=0);
61  endmodule
62
63  module weather2
64      w2 : [0..1];
65
66    [tick1] (s > 0) & (s < 3) & (t = 1) -> Pb:(w2' = 1) + (1-Pb):(w2'=0);
67  endmodule
```

Figure: Part 3 of the runtime RPISM code.

Conclusions

Restricted Instruction Set Agent (LISA) has been proposed. The architecture originates from previous AgentSpeak implementations such as Jason and Jade.
We reduced complexity by simplifying the reasoning cycle with by simple methods of triggering a plan and by the use of asynchronous multi-threaded processing and proposed the use of model checking techniques on two levels:
1) Design-time model checking: we prove that it is possible to abstract the agent/environment to a DTMC/MDP and in turn verify the decision making process with existing model checking techniques,
(2) Runtime verification of executable plan selection: the agent is able to assess a tree of possible future outcomes and select a plan which is most likely to succeed in reaching the mission goals.
Automated generation of PRISM models for runtime verification in complex environments still remains an open research area.

References

[1] S. M. Veres, L. Molnar, N. K. Lincoln, and C. Morice, Autonomous vehicle control systems - a review of decision making, vol. 225, no. 2, pp. 155 195, 2011.
[2] N. K. Lincoln and S. M. Veres, Natural language programming of complex robotic BDI agents, Intelligent and Robotic Systems, vol. 71, no. 2, pp. 211-230, 2013.
[3] S. M. Veres, Knowledge of machines: review and forward look, Proc. IMechE Vol. 225 Part I: J. Systems and Control Eng., pp. 1-8, 2015.
[4] S. Tantrairatn and S. M. Veres, A rational agent framework for adaptive flight control of UAV, ICUAS-15, International Conference on Unmanned Aircraft Systems June 9-12, Denver Marriott Tech Center, 2015.

References

[5] S.Veres, A class of BDI agent architectures for autonomous control, CDC 2004, 14-17 Dec. Paradise Islands, Bahamas, 2004.
[6] Autonomous Asteroid Exploration by Rational Agents, IEEE Computational Intelligence Magazine Vol. 8, No 4, pp 25-38, 2013
[7] On Efficient Consistency Checks by Robots, by Hongyang Qu and Sandor M Veres , The European control Conference, 2014
[8] *Natural Language Programming of Agents and Robotic Devices* (book), S M Veres, SysBrain, London, 2008
[9] R. V. et al., Ed., CLARAty:Coupled Layer Architecture for Robotic Autonomy, company report ed. Jet Propulsion Laboratory and California Institute of Technology, 2000.

Thank you for your attention.

Any questions?