UNIVERSITÀ DEGLI STUDI DI GENOVA

MANCHESTER 1824 The University of Manchester

RVPLAN: A General Purpose Framework for Replanning using Runtime Verification

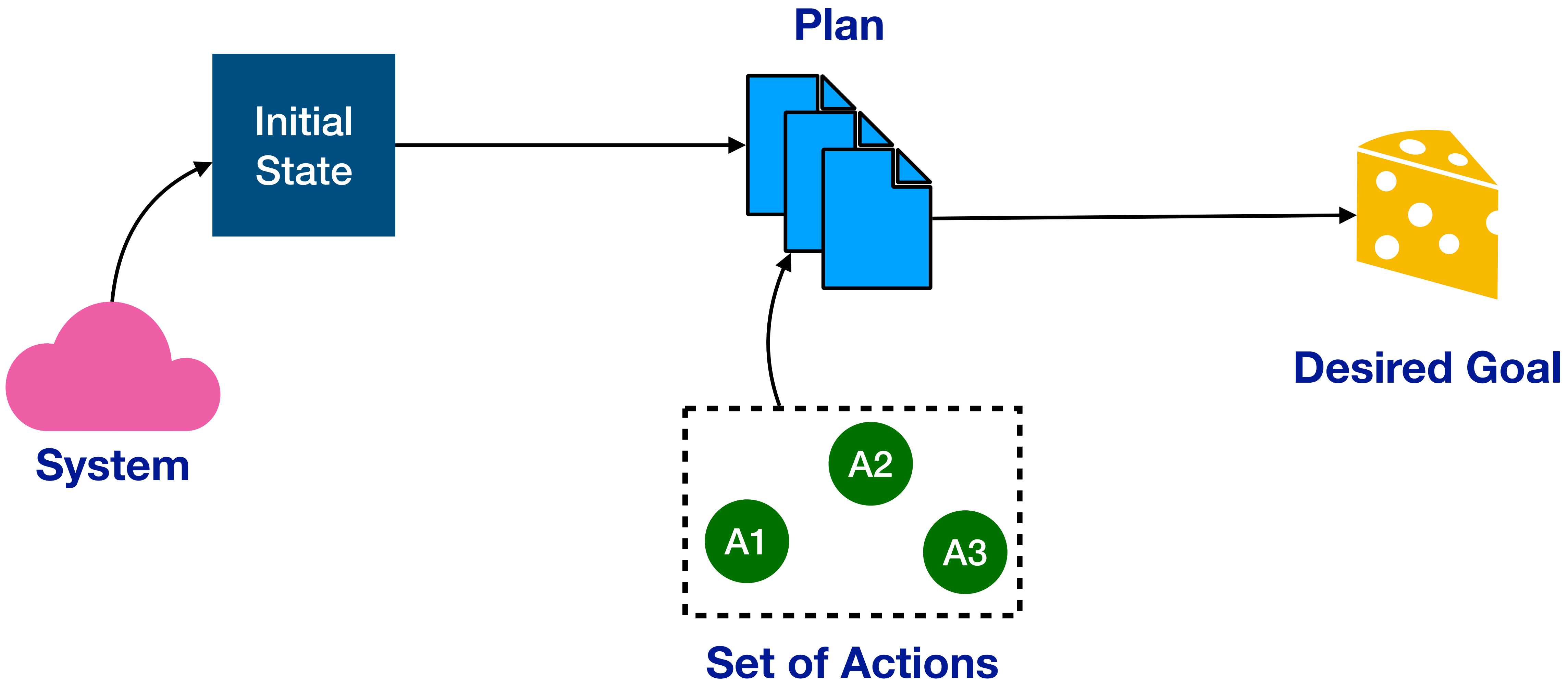
Angelo Ferrando[1] and Rafael C. Cardoso[2]
[1] *University of Genova*
[2] The University of Manchester

angelo.ferrando@unige.it
Research Fellow

Automated Planning

Automated planning and scheduling is a branch of Artificial Intelligence that concerns the realisation of strategies or action sequences, typically for execution by **intelligent agents**, **autonomous robots** and **unmanned vehicles**.

Automated planning techniques use models of the system to search the state-space for a sequence of actions that can achieve the goals of the application.

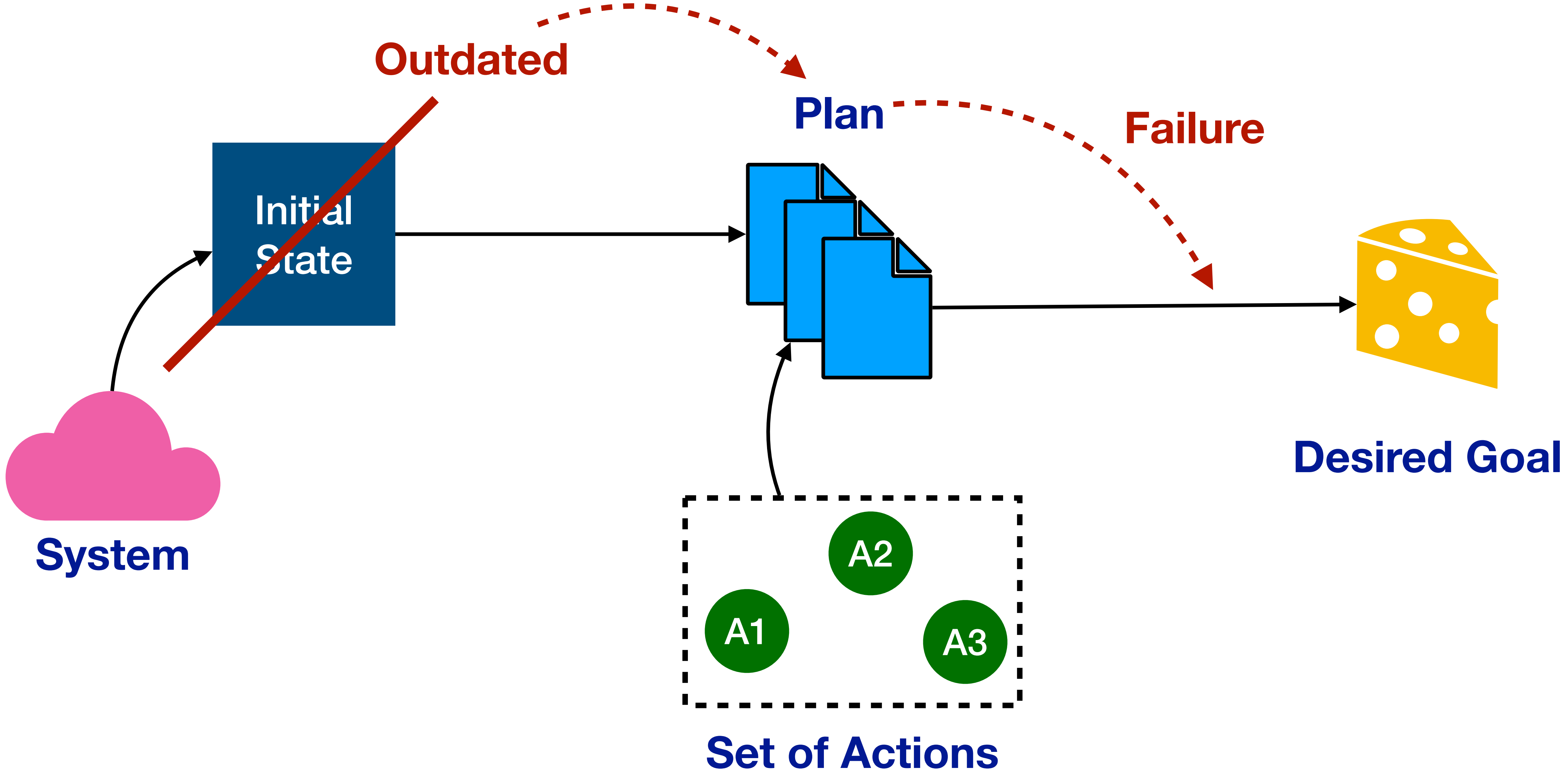


One of the problems of Automated Planning

Complex **real-world** applications can be **difficult to model**, above all in cyber-physical systems where the environment is dynamic and the information may be incomplete.

In these systems, plans can **fail** due to the use of **outdated** information or incorrect model abstraction.

When this happens, it is crucial for the system to be able to detect plan failure and trigger a **replanning mechanism** using up-to-date information.

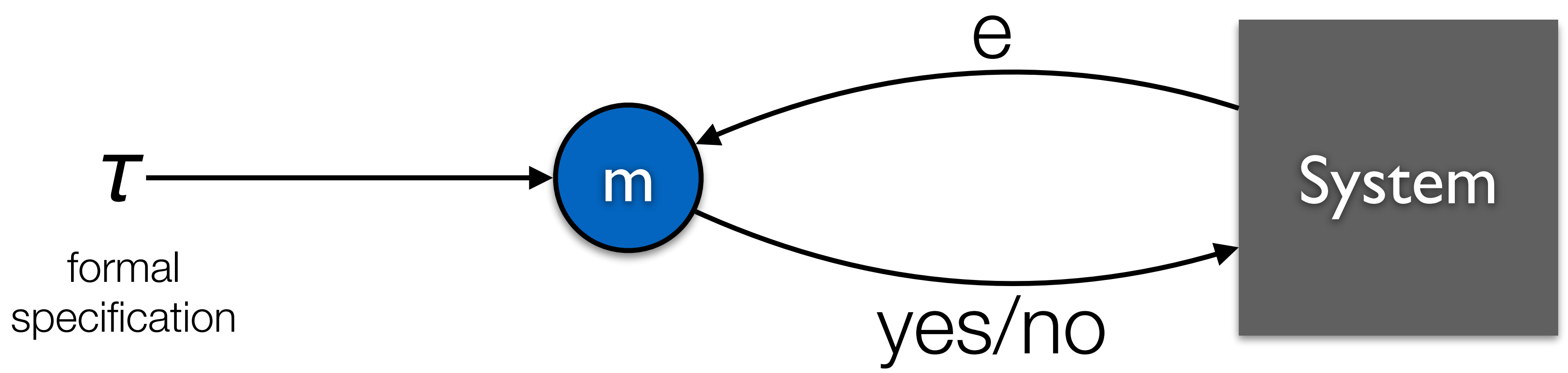


Runtime Verification

Complement of

- formal static verification (such as *Model Checking*)
  pros: formal, exhaustive **cons: suffers from scalability**
- testing.
  pros: scales well **cons: not formal, not exhaustive**

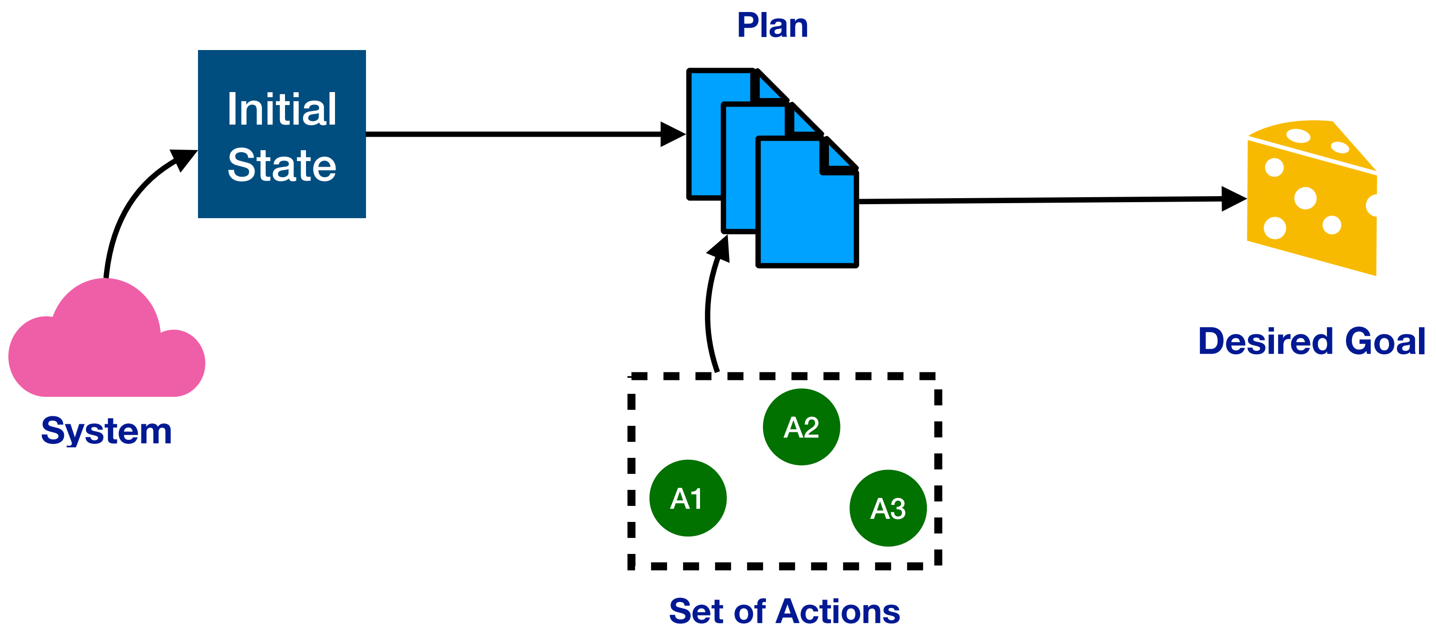
Dynamic checking of system behaviour using one (or multiple) **Monitor(s)**
pros: formal, scales well, can be done after deployment
cons: not exhaustive



RV can help detecting failures

If the model used by an automated planner is imprecise, or the system is dynamic and continuously changes, the system needs to **detect plan failure** and to consistently **trigger replanning** with an updated model.

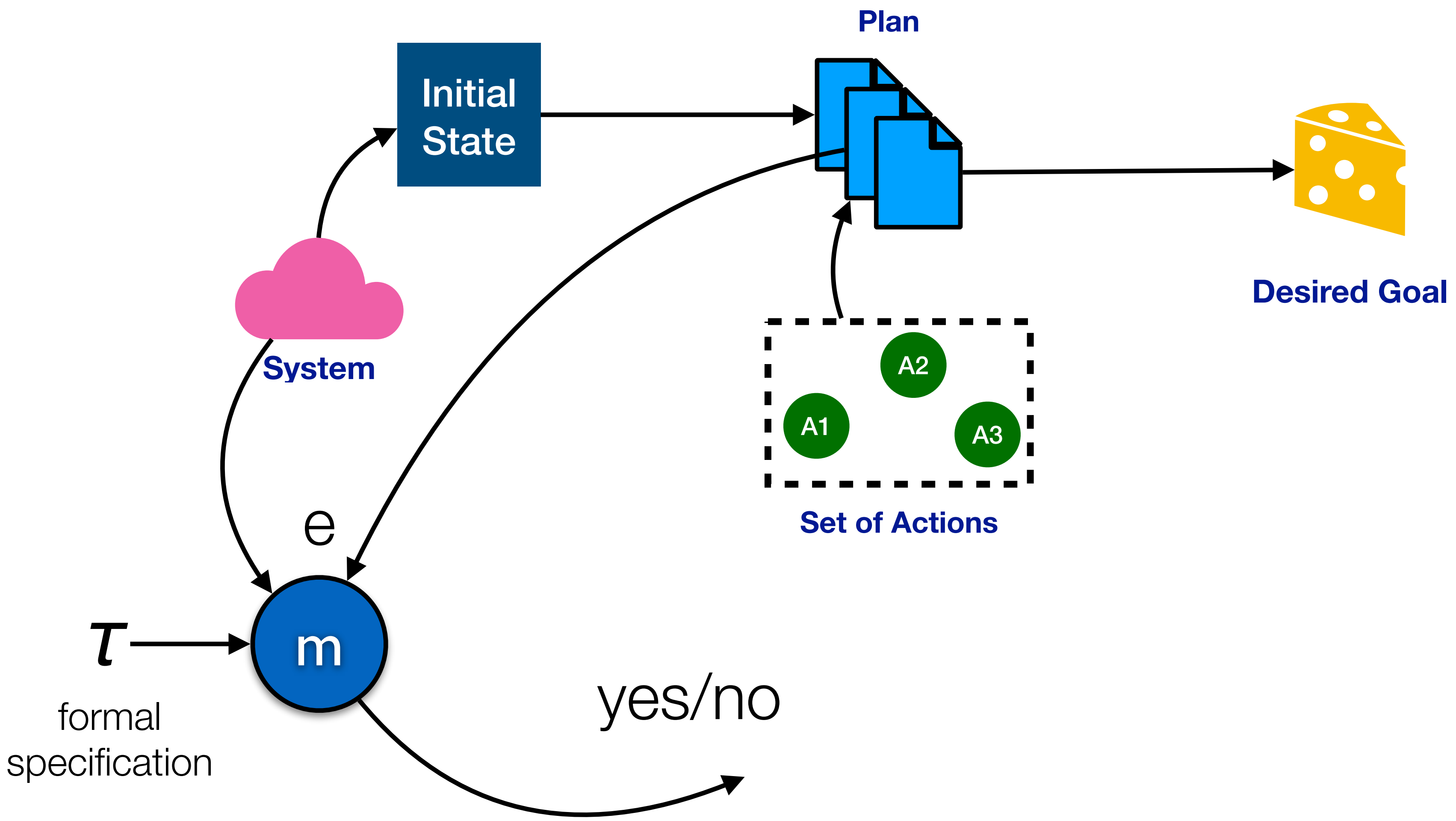
Runtime Verification is a suitable candidate for failure detection at runtime. Thus, it can be used to trigger the replanning mechanism at runtime.



RV can help detecting failures

If the model used by an automated planner is imprecise, or the system is dynamic and continuously changes, the system needs to **detect plan failure** and to consistently **trigger replanning** with an updated model.

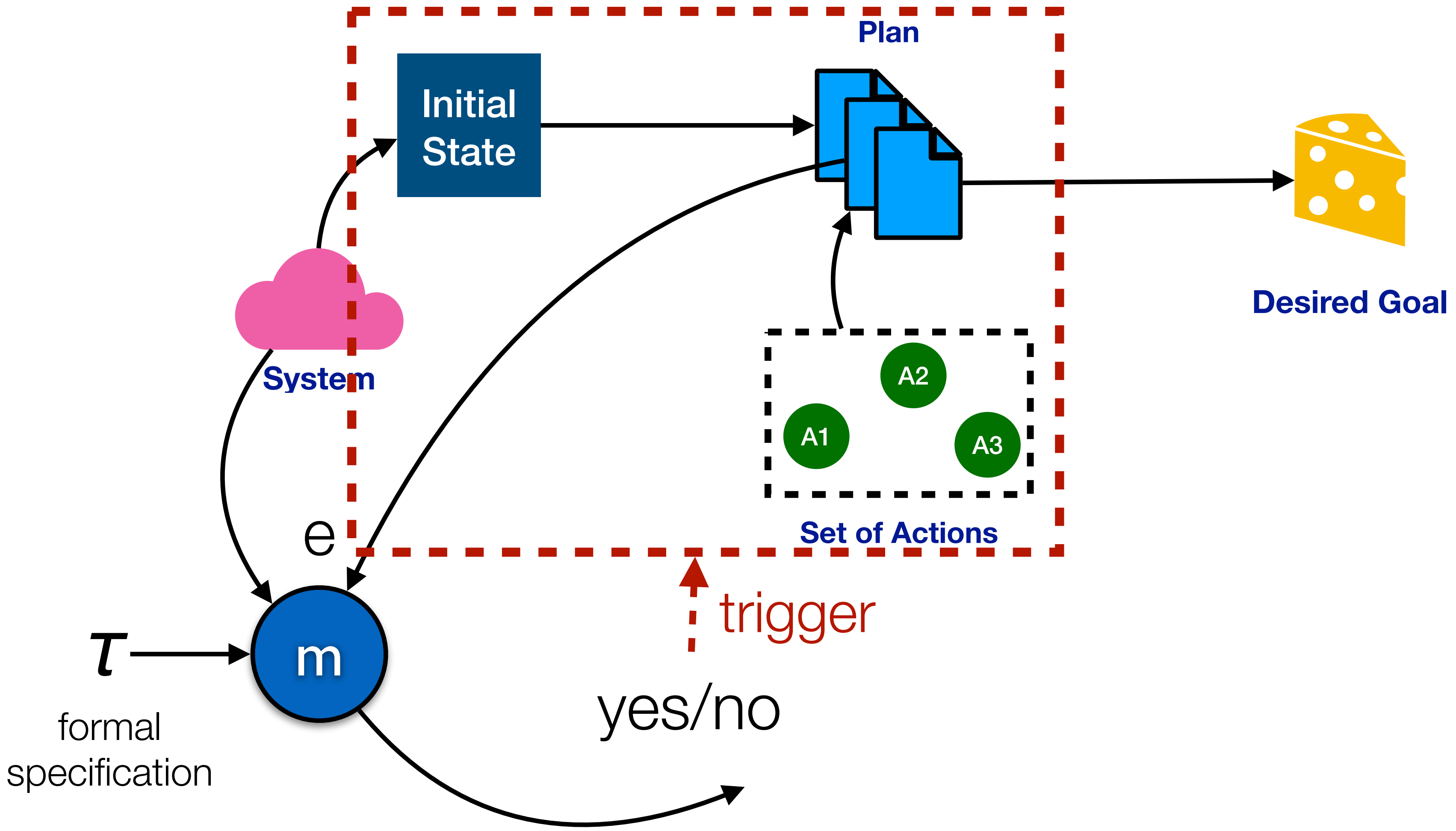
Runtime Verification is a suitable candidate for failure detection at runtime. Thus, it can be used to trigger the replanning mechanism at runtime.



RV can help detecting failures

If the model used by an automated planner is imprecise, or the system is dynamic and continuously changes, the system needs to **detect plan failure** and to consistently **trigger replanning** with an updated model.

Runtime Verification is a suitable candidate for failure detection at runtime. Thus, it can be used to trigger the replanning mechanism at runtime.



STRIPS-like Planner

A classical planner is a tuple $<C,O,I,G>$, with

- C a set of conditions (predicates),
- O a set of operations (i.e. actions) $<a,\alpha,\beta> \in O$ where a is the action and each other element specify, in order, which conditions must be true/false for the action to be executable, and which ones are made true/false by the action,
- I the set of conditions which are initially true, and
- G the set of goals to achieve, denoted as the set of conditions need to be true/false to achieve the goal.

Running example

A **rover** traverses a 2D grid map to perform the remote inspection of tanks containing radiation.

The following list of predicates is used:

• *robot-at(r,x)* and *tank-at(t,x)* indicate the position of a robot or a tank in a cell;

• *up(x,y)*, *down(x,y)*, *left(x,y)*, and *right(x,y)* indicate possible movements from cell *x* to cell *y*;

• *empty(x)* is used to mark that the cell is currently empty;

• *radiation(x)* denotes that a cell currently has high radiation;

• *inspected(t)* represents when a tank has been inspected (these are the goals of the planning problem).

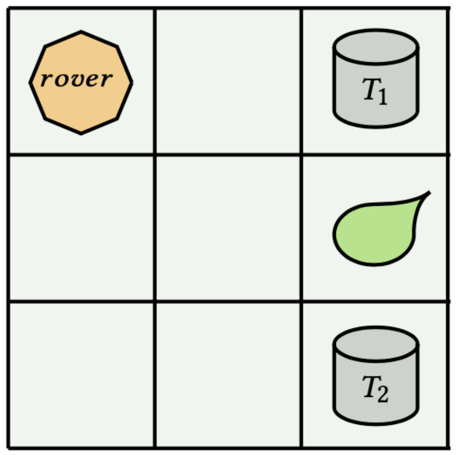


Figure 1: Initial state in a 3x3 grid; *rover* is the robot, T_1 and T_2 are the tanks to be inspected, and the green drop is radiation.

Running example (cont'd)

The domain contains actions for movement in the grid and for inspecting tanks.

Movement is separated into four actions, move **up**, **down**, **left**, and **right**.

Parameters:

The **robot** performing the action, the current **cell x**, the destination **cell y**.

Preconditions:

The **robot** is **currently at cell x**, there is a **path between x and y** (predicate up, down, left, right depending on the action), **y is empty**, and there is **no radiation in y**.

Effects:

The robot is **no longer at x** but it is **now at y**, **cell x is empty**, and **cell y is no longer empty**.

Similarly, there are four actions for inspecting a tank, **inspect-up**, **inspect-down**, **inspect-left**, and **inspect-right**.

RVPLAN - Overview

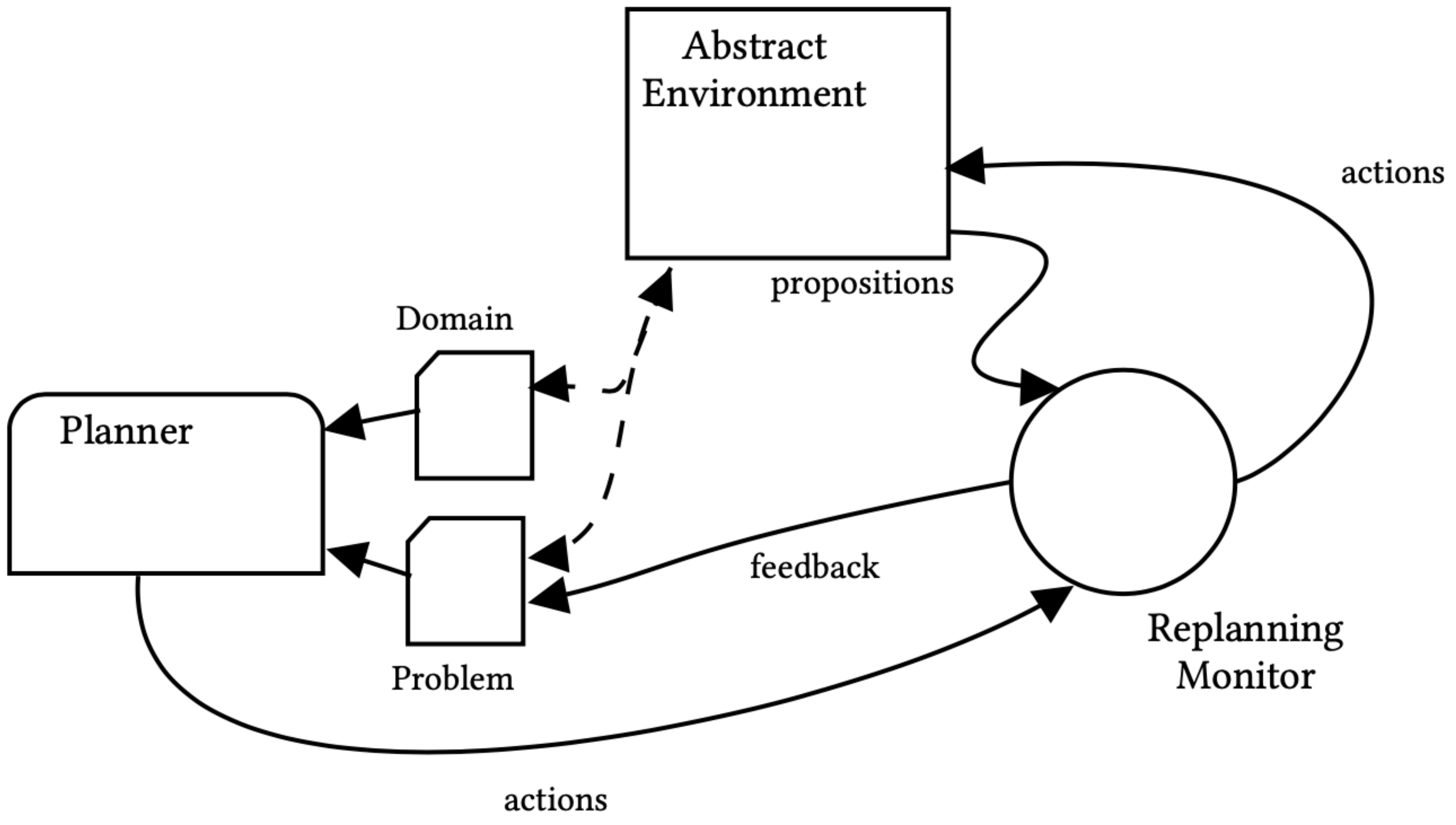
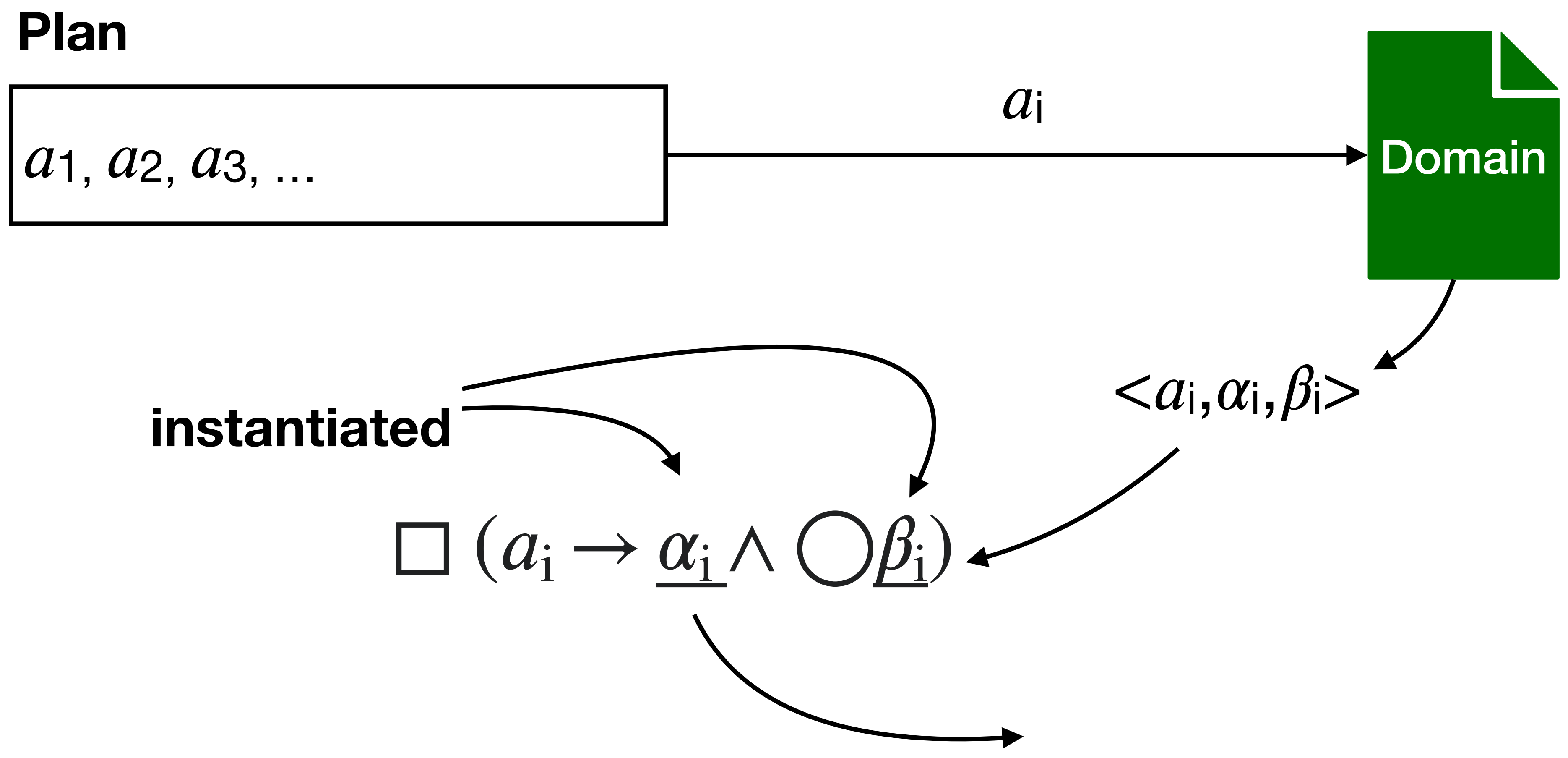


Figure 2: RVPLAN overview. Dashed lines represent relation, solid lines represent communication.

Instantiated Action Monitoring

The standard formalism to specify formal properties in RV is propositional **Linear-time Temporal Logic (LTL)**.

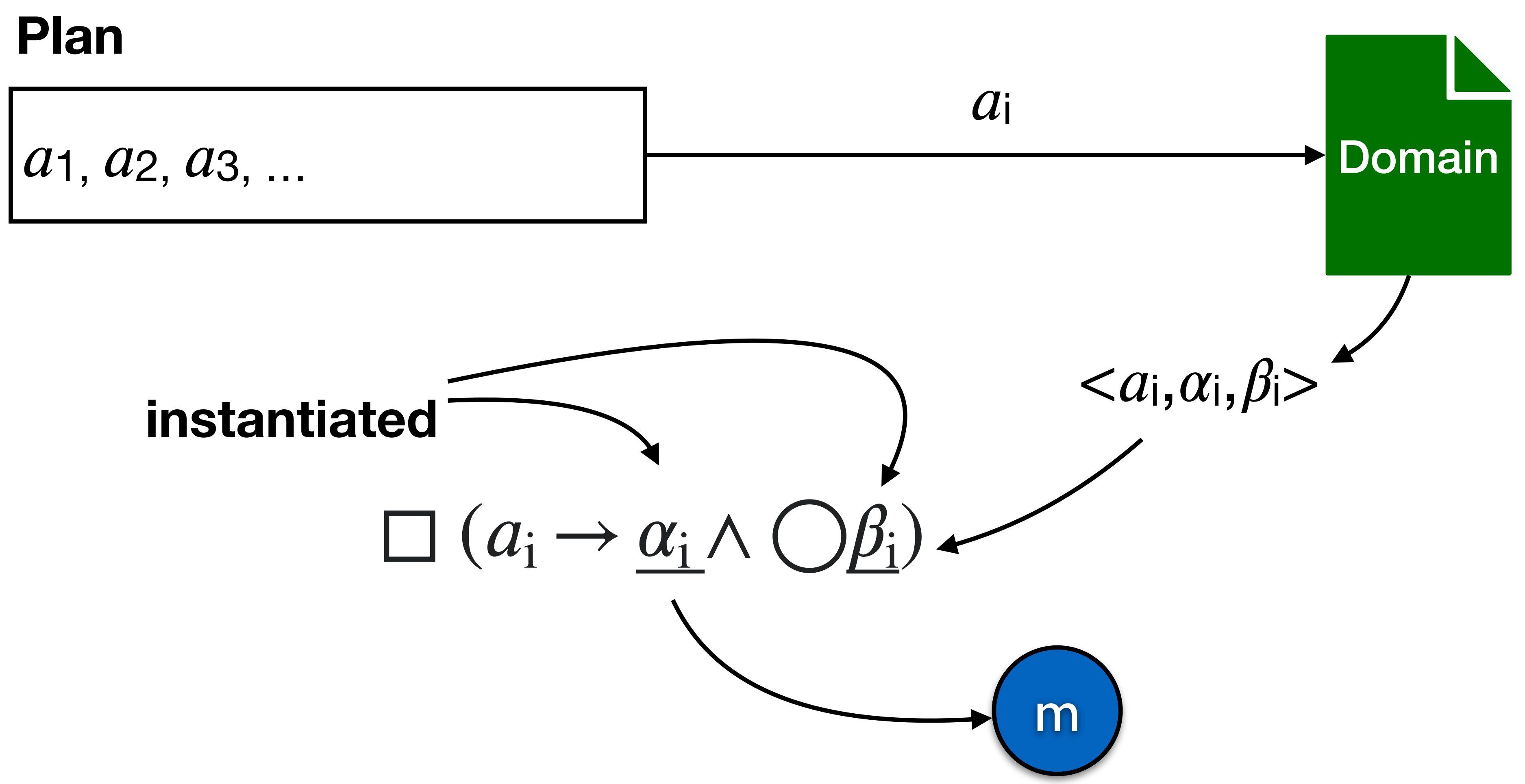
Since a **plan** is a **sequence of instantiated actions**, we can extract the propositions representing the **pre- and post-conditions** that have to be met.


Plan
$a_1, a_2, a_3, \ldots$
a_i
Domain
$<a_i, \alpha_i, \beta_i>$
instantiated
$\square\,(a_i \rightarrow \underline{\alpha_i} \wedge \bigcirc \underline{\beta_i})$

Instantiated Action Monitoring

The standard formalism to specify formal properties in RV is propositional **Linear-time Temporal Logic (LTL)**.

Since a **plan** is a **sequence of instantiated actions**, we can extract the propositions representing the **pre- and post-conditions** that have to be met.


Plan
$a_1, a_2, a_3, \ldots$
a_i
Domain
$<a_i, \alpha_i, \beta_i>$
instantiated
$\square\,(a_i \rightarrow \underline{\alpha_i} \wedge \bigcirc \underline{\beta_i})$
m

Instantiated Action Monitoring applied to Example

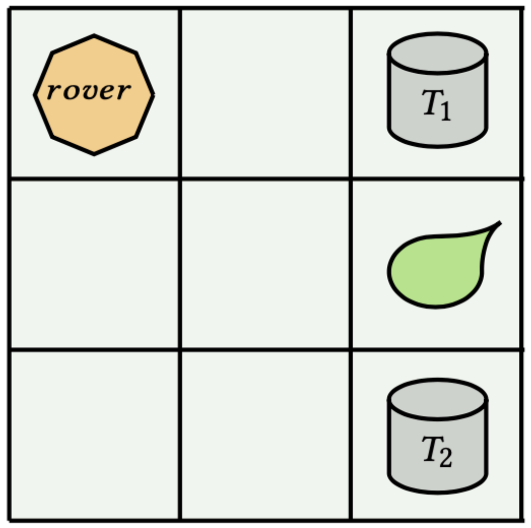


Figure 1: Initial state in a 3x3 grid; *rover* is the robot, T_1 and T_2 are the tanks to be inspected, and the green drop is radiation.

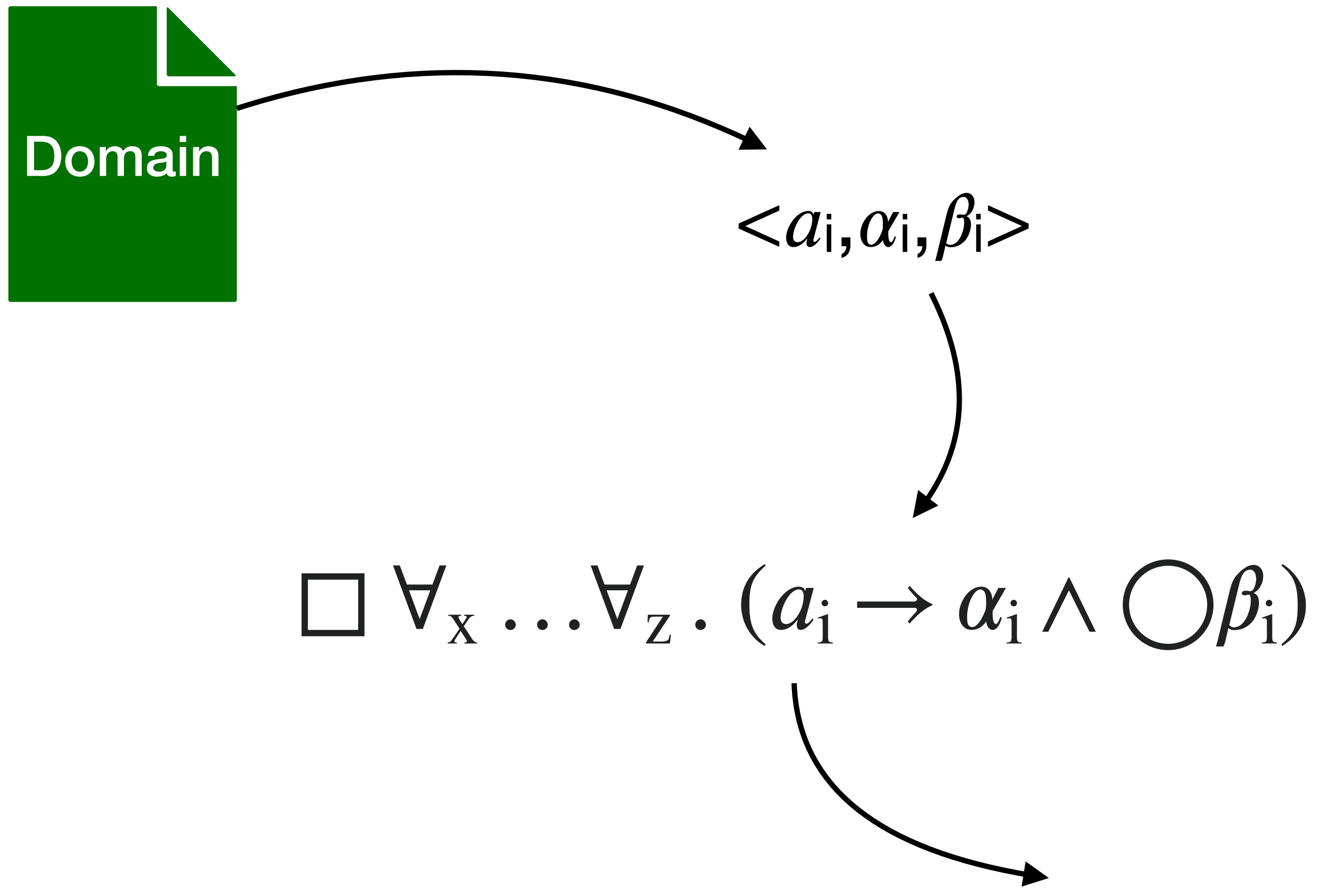
$\varphi = \square(right(rover, cell\_0\text{-}0, cell\_1\text{-}0) \Rightarrow (robot\text{-}at(rover,cell\_0\text{-}0) \wedge right(cell\_0\text{-}0,cell\_1\text{-}0) \wedge empty(cell\_1\text{-}0) \wedge \neg radiation(cell\_1\text{-}0) \wedge \bigcirc(robot\text{-}at(rover,cell\_1\text{-}0) \wedge empty(cell\_0\text{-}0) \wedge \neg robot\text{-}at(rover,cell\_0\text{-}0) \wedge \neg empty(cell\_1\text{-}0))))$.

Parameterised Action Monitoring

A problem related to the previous approach is that, given a new plan, we might need to recreate the monitor. An **alternative** is to consider **parameterised** actions.

Since domain actions contain variables, we cannot translate them to pure LTL.

A suitable candidate is FO-LTL (First-Order Linear-time Temporal Logic), an extension of LTL that supports variables.



$$<a_i, \alpha_i, \beta_i>$$

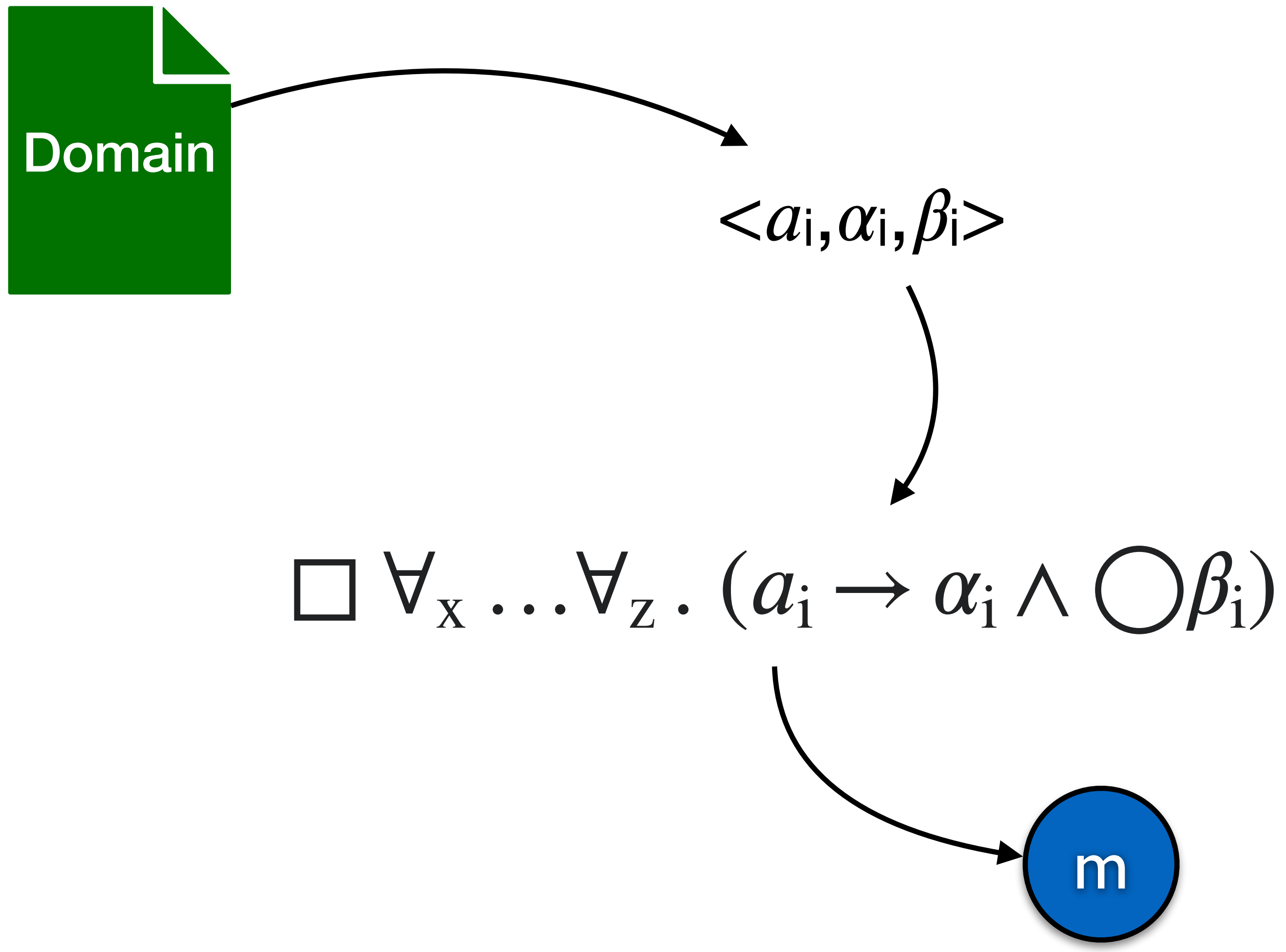
$$\square \, \forall_x \ldots \forall_z \,.\, (a_i \rightarrow \alpha_i \wedge \bigcirc \beta_i)$$

Parameterised Action Monitoring

A problem related to the previous approach is that, given a new plan, we might need to recreate the monitor. An **alternative** is to consider **parameterised** actions.

Since domain actions contain variables, we cannot translate them to pure LTL.

A suitable candidate is FO-LTL (First-Order Linear-time Temporal Logic), an extension of LTL that supports variables.


Domain
$<a_i, \alpha_i, \beta_i>$
$\Box \forall_x \ldots \forall_z . (a_i \rightarrow \alpha_i \wedge \bigcirc \beta_i)$
m

Parameterised Action Monitoring applied to Example

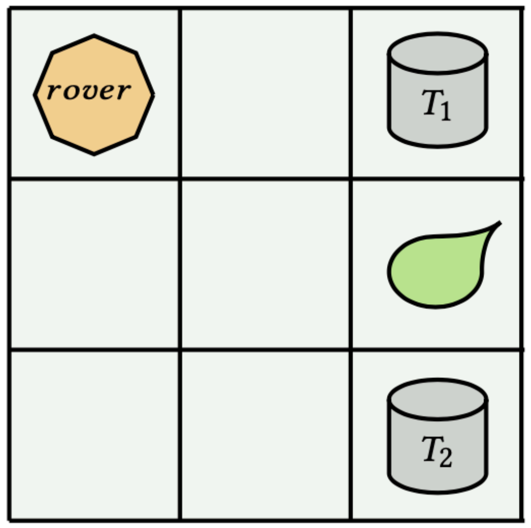


Figure 1: Initial state in a 3x3 grid; *rover* is the robot, T_1 and T_2 are the tanks to be inspected, and the green drop is radiation.

$$\varphi^{FO} = \Box(\forall_{r \in \{rover\}}.\forall_{x,y \in \{0,1,2\}}.(right(r,x,y) \Rightarrow (robot\text{-}at(r,x) \land right(x,y) \land empty(y) \land \neg radiation(y) \land \bigcirc(robot\text{-}at(r,y) \land empty(x) \land \neg robot\text{-}at(r,x) \land \neg empty(y)))))).$$

Replanning through monitoring

A replanning monitor is a transducer which given an action and a set of propositions, returns a pair of actions to perform on the abstract environment and the planner, respectively.

It is defined as follows:

$$RM_\varphi^G(a, Prop) = \langle a, nop \rangle \qquad \text{if } M_\varphi(\{a\} \cup Prop) \neq \bot$$

$$RM_\varphi^G(a, Prop) = \langle nop, replan(Prop, G \setminus Prop) \rangle \qquad \text{if } M_\varphi(\{a\} \cup Prop) = \bot$$

Where $replan(Prop, G \setminus Prop)$ denotes the action of replanning, while nop means no operation has to be performed.

Conclusions and Future Work

Replanning and plan repair techniques are **required** when using automated planning in dynamic and **unpredictable environments**.

RVPLAN offers two different approaches for generating the replanning monitors:

• based on instantiated actions in the plan that the planner has sent as output;

• or based on the actions in the domain specification.

The former is best when the logic cannot cope with parameters, such as LTL; the latter is more appropriate when using parameterised logics that support variables.

Future work:

We want to **compare** the computational cost of our approaches for automatically synthesising replanning monitors. Note that performance will be tied to the logic chosen to represent the monitor; thus, it will be necessary to test several different logics in both approaches to try to obtain interesting results.

Finally, it would be interesting to try to extend our replanning monitors to perform **plan repair**, not only updating the problem specification, but also **reconfiguring** action specifications.