Let's Learn their Language? A Case for Planning with Automata-Network Languages from Model Checking

Jörg Hoffmann and **Holger Hermanns** and **Michaela Klauck** and **Marcel Steinmetz**
Saarland Informatics Campus, Saarland University, Saarbrücken, Germany
{lastname}@cs.uni-saarland.de

Erez Karpas
Technion Israel Institute of Technology, Haifa, Israel
karpase@gmail.com

Daniele Magazzeni
King's College London, UK
daniele.magazzeni@kcl.ac.uk

Abstract

It is widely known that AI planning and model checking are closely related. Compilations have been devised between various pairs of language fragments. What has barely been voiced yet, though, is the idea to let go of one's own modeling language, and use one from the other area instead. We advocate that idea here – to use automata-network languages from model checking instead of PDDL – motivated by modeling difficulties relating to planning agents surrounded by exogenous agents in complex environments. One could, of course, address this by designing additional extended planning languages. But one can also leverage decades of work on modeling in the formal methods community, creating potential for deep synergy and integration with their techniques as a side effect. We believe there's a case to be made for the latter, as one modeling alternative in planning among others.

1 Introduction

The close relation between *AI planning (AIP)* and *model checking (MC)* is widely known. Model checkers have been used as planning back-ends (e. g. (Giunchiglia and Traverso 1999; Cimatti et al. 2003; Bogomolov et al. 2014)), compilations are known between a variety of language fragments (e. g. (Edelkamp 2003a; Abdeddaïm et al. 2007; Klauck et al. 2018)), various algorithmic ideas have been transferred (e. g. (Rintanen 2003; Edelkamp, Lluch-Lafuente, and Leue 2004; Wehrle et al. 2013)), temporal logics is frequently used in both areas (e. g. (Baier, Bacchus, and McIlraith 2009; De Giacomo and Vardi 2013)).

Despite this intense exchange across areas, it is uncommon to use a modeling language from area X to model (and solve) a problem from area Y; we discuss exceptions below. There seems to be an implicit assumption that the modeling languages of either area are best suited to model those practical problems that area is interested in solving. We believe there is good reason to challenge that assumption. We advocate the use of MC languages based on automata networks to model AIP problems, as one modeling alternative in planning. We believe that this is the rational choice for certain problem characteristics, pertaining to AIP applications involving planning agents surrounded by exogenous agents (not under control of the planner) in complex environments. Specifically, in our own work, we have considered Minecraft planning, robot-human collaboration, and drones control avoiding exogenous drones. After modeling these in AIP languages, and considering MC languages based on automata networks as an alternative, we cannot help but conclude that the latter provide a superior combination of support for (I) exogenous agents, (II) pre-defined plan structure, (III) probabilistic duration uncertainty, and (IV) data structures. Various AIP dialects address some of (I)–(IV), but often with limitations and none addresses the combination.

The technical part of the paper elaborates on these points. Before we delve into the details, let us briefly discuss the consequences of our proposal, and related prior works.

In the short term, if we model our problems in MC languages then we have to use MC tools. In the long term, however, our vision is to synergetically combine algorithmic ideas across areas. While many ideas are related, AIP has generally focused on fast methods finding solutions under run-time conditions, while MC has generally focused on complete analyses under design-time conditions. So a blueprint for combination across areas is an architecture in which planning methods serve for online decision making, while MC/verification methods serve for offline analysis and for the preparation of information to be leveraged at run-time (approximations of the safe region, etc). Note that this could transform the connection between the two areas. Apart from getting rid of the language barrier, we would elaborate and leverage the *complementarity* across areas, rather than their *commonality* as previous approaches do.

It has been previously observed in AIP that automata networks can have advantages for modeling planning problems. In the PDDL+ (Fox and Long 2006) variant of the most common planning language PDDL, the semantics is based on mapping PDDL+ to hybrid automata, motivated among other things by the potential for exchange of results. In Web service composition as planning, at the so-called process level Web services are represented as automata (Pistore, Traverso, and Bertoli 2005). Models and tools from probabilistic MC have become popular in the robotics and motion planning communities due to their expressivity (e. g. (Johnson and Kress-Gazit 2011; Lacerda, Parker, and Hawes 2015)). Work on the design of reactive model-based executives has made intensive use of automata-based semantics, in the RMPL language (Williams, Chung, and Gupta 2001) and in variants of the Burton system (e. g. (Williams and Nayak 1997; Wang and Williams 2015)).

While MC languages have been conceived as abstract models against which to check system properties, they have also been used for model-based control. To give just one example, quantitative MC supporting probabilities and arithmetics has been used to model and solve a variety of satellite control problems taking into account battery usage (Nies et al. 2018). The MC tool configuration here falsifies reachability properties corresponding to the negated satellite goal, so that effectively MC is used to solve a planning problem.

In the light of these prior works, what is new in our proposal is merely its radical nature within AIP, suggesting to use automata-based MC languages instead of PDDL dialects at the front end, where suited to the application.

Section 2 briefly introduces the planning and MC languages considered in our discussion. Section 3 gives a language comparison overview, outlining the weaknesses of AIP vs. MC languages along the lines (I)–(IV) mentioned above. Sections 4 and 5 illustrate these language differences with two use cases pertaining to drones control and robot-human collaboration. Section 6 closes the paper.

2 Background: The Languages We Consider

AI Planning

Planning languages are generally based on first-order logics, with a finite universe of objects and without function symbols. A state is an interpretation, which can be viewed as a vector of Boolean state variables. An action is described in terms of a precondition formula, and an effect specifying how the interpretation changes. Planning tasks specify the sets of objects and predicates, a set of actions with object parameters (as in "drive(x,y)"), an initial state, and a goal formula. The most common front end language is the planning domain definition language (PDDL), used in the international planning competitions (IPC). In classical planning, actions are deterministic and the initial state is fully known. Many extensions to richer models exist.

The PDDL1.0 dialect used in the 1998 and 2000 IPCs (McDermott 2000; Bacchus 2001), corresponds to classical planning with Boolean state variables only. In PDDL2.1, Fox and Long (2003) enriched this with numeric state variables and temporal planning. In PDDL2.2, Hoffmann and Edelkamp (2005) added timed initial literals to model exogenous events. In PDDL+, Fox and Long (2006) support continuous processes triggered by actions, including exogenous actions, allowing to model time-dependent effects. Tool support for this branch of PDDL is generally good, with ample coverage of PDDL1.0 notably in the FD platform (Helmert 2006), but many tools also for richer dialects (e. g. (Edelkamp 2003b; Gerevini, Saetti, and Serina 2008; Coles et al. 2012; 2013; Bryce et al. 2015)).

To support probabilistic planning, Younes et al. (2005) introduced PPDDL which extends PDDL1.0 with probabilistic action outcomes. Later on, Sanner (2010) introduced RDDL which departs from PDDL more radically, in particular adding support for probabilistic behaviors exogenous to, and concurrent with, the planning agent (such as traffic at a crossing). That support is crucial in our target use cases. Various tools support fragments of RDDL. A prominent one is Prost (Keller and Eyerich 2012) for effective satisficing planning based on Monte-Carlo tree search.

Functional STRIPS (Geffner 2000) extends STRIPS with function symbols, mapping objects to objects. This allows to refer to objects more flexibly, in particular eliminating the need to parameterize each action by all relevant objects (which often results in large grounded encodings).

In many applications it makes sense for the human modeler to impose a partially pre-defined plan structure. Support for this is provided by HTN planning (e. g. (Nau et al. 2003; Höller et al. 2014), where a hierarchy of action decompositions imposes plan structure; and by Golog (e. g. (Levesque et al. 1994; Grosskreutz and Lakemeyer 2003)), where imperative programming constructs impose plan structure.

It is finally worth mentioning MA-PDDL (Brafman and Domshlak 2008; Komenda, Stolba, and Kovacs 2016) in our context, which extends PDDL1.0 with support for multi-agent cooperative planning, where the task is to control all agents but in a decentralized manner.

Model Checking

From the large and diverse landscape of MC modeling languages, here we focus on automata networks, which naturally lend themselves to an agent-based view of the world.

Languages based on automata networks incorporate individual automata variants ranging from deterministic finite automata to ones incorporating state variables, data structures, non-determinism, time, and/or stochastic behavior. Several such automata are executed concurrently, so that states are combinations of automaton positions (and state variable values/data structure instances if present). Automata can be synchronized through shared actions (labels for transitions that have to be taken synchronously), global state variables or data structures, and communication channels transferring data between automata.

Well-known example modeling languages and tools are Promela and SPIN (Holzmann 2004); UPPAAL (Larsen, Pettersson, and Yi 1997), which focuses on timed automata models; and PRISM (Kwiatkowska, Norman, and Parker 2011), which focuses on probabilistic models.

In the remainder of the paper, we refer to automata-based MC languages in general when finer distinctions are not necessary. The only concrete MC language we discuss is Jani (Budde et al. 2017), see http://www.jani-spec.org/, which is ideally suited due to its broad coverage of MC models. Jani has been conceived as an overarching language to foster MC tool interoperation and comparability. It encompasses data structures, clocks, and probabilities. It defines language fragments capturing a wide range of sub-formalisms, from labeled transition systems to stochastic hybrid automata and everything in between, e. g. discrete- and continuous-time Markov chains and decision processes. Tool support for Jani is available from the Modest Toolset (Hartmanns and Hermanns 2014), Storm (Dehnert et al. 2017), and (through model translations) from PRISM among others. A large spectrum of case studies exists.

3 Language Comparison Overview

This work is motivated by, and our conclusions pertain to, AIP applications with certain problem characteristics:

(I) exogenous agents;

(II) pre-defined plan structure;

(III) probabilistic duration uncertainty;

(IV) need for non-trivial data structures.

Characteristic (I) occurs whenever the planning agent is surrounded by exogenous agents, and knowledge about those agents' behavior is available and important for the decision taking (e. g. a human co-worker). Characteristic (II) is natural for technical systems that require autonomous decisions in particular situations only, using scripted behavior otherwise (e. g. drones control in critical vs. standard situations). Characteristic (III) is important, e. g., for temporal synchronization with exogenous agents whose action durations cannot be assumed to be fixed (e. g. a human co-worker's coffee break). Characteristic (IV) occurs, for example, when acting in a large discrete space, where arrays are convenient to store location information (e. g. Minecraft). Table 1 gives an overview of our language comparison regarding (I) – (IV).

(I) Exogenous Agents

Of interest to us here specifically are agents whose underlying workflows are known or at least partially known. An example for an exogenous agent of this kind is a co-worker in robot-human collaboration, triggering exogenous events through action choice in a (partially) known workflow.

PDDL2.2 supports exogenous events as timed initial literals, that will become true at given points in time. This allows to model events triggered by exogenous agents, but it lacks the ability to model the underlying workflows. PDDL+ supports the latter more explicitly, through exogenous actions that trigger when their precondition becomes true. Neither can model non-deterministic or probabilistic behavior. In work on model-based flexible execution (Levine and Williams 2018), temporal networks have been extended with exogenously controlled decision variables, allowing to model non-deterministic but not probabilistic behavior.

RDDL provides the best AIP support for (I); indeed that is a large part of its raison d'être. The probabilistic state transition model in RDDL allows to model environment behavior. This lends itself, in particular, to exogenous agents.

MA-PDDL supports an explicit model of individual agents, but is intended for cooperative planning where all agents are under control of the planner. Somewhat similarly, the ConGolog (De Giacomo, Lespérance, and Levesque 2000) variant of Golog permits programs that are executed concurrently but that are all controlled by the planner. ConGolog furthermore adds exogenous events non-deterministically choosing environment actions.

On the MC side, automata networks are perfectly suited to model (I). Individual agents (planner-controlled or exogenous) can be modeled through one or several automata each. The underlying workflows are then explicitly specified, and they allow to incorporate non-deterministic and probabilistic behavior (where we can choose whether non-deterministic exogenous agents are cooperative or adversarial).

(II) Pre-Defined Plan Structure

It is quite natural for a technical system to use autonomous decision technology only at particular choice points, following pre-programmed standard behavior otherwise.

Jani naturally supports such specifications. In AIP, support for user-imposed plan structure has a long tradition, but remains rather an under-addressed area; e. g. it has not been part of an IPC since 2004. IPC variants of PDDL do not provide modeling support (though compilations are possible (Rintanen 2000)). The major approaches to provide such support are HTN planning and Golog as previously mentioned. In work on temporal planning networks (Kim, Williams, and Abramson 2001), a user can write a control program, imposing some structure on the plan, leaving some choices for the agent to make during execution.

(III) Probabilistic Duration Uncertainty

Action durations can be subject to uncertainty. In Section 5 we discuss the example of robot-human collaboration where action durations are uncertain for both human and robot.

PDDL 2.1 and upwards permit duration inequalities, but they assume the duration is under control of the planner. Prottle (Little, Aberdeen, and Thiébaux 2005) supports discrete probabilistic durations. Probabilistic simple temporal networks (e. g. (Vidal and Ghallab 1996; Fang, Yu, and Williams 2014; Lund et al. 2017)) are a well-investigated framework to schedule plans (not to find them). Temporal planning networks have been extended with uncertain set-bounded (interval) durations (Karpas et al. 2015), and also with probabilistic durations (Yu et al. 2017). But the latter function as constraints, bounding the risk that the real duration is outside a chosen interval.

In Jani, on the other hand, probabilistic duration uncertainty is a key modeling dimension, reflecting a mature subarea of the MC literature (e. g. (Baier et al. 2010)). Jani syntax allows to sample from continuous distributions and to specify clock guards on the sampled values. Exponential distributions are supported in continuous-time Markov chains/decision processes (CTMxx), more general ones in (generalized) semi-Markov models. Algorithms tackling CTMxx are well-investigated and advanced tool support is available, including also approximations to more general models.

(IV) Data Structures

Acting in a large discrete space requires to handle location information in some form. Sometimes this can be limited to waypoint graphs, but that is not always so. An example is the Minecraft game, where individual 3D positions matter. In Section 4 we discuss a use case involving drones control.

To store discrete space information, arrays are a convenient data structure. Yet PDDL encodings (e. g. of Minecraft (Roberts et al. 2017)) model positions as object tuples, yielding huge grounded representations. Functional STRIPS improves this by reducing the number of action parameters needed, and combined task and motion planning has been addressed with extensions of functional STRIPS (Ferrer-Mestres, Francès, and Geffner 2017) using state constraints to compactly express no-overlap preconditions, and using

	PDDL1.0	PDDL2.1	PDDL 2.2	PDDL+	PPDDL	RDDL	F-STRIPS	MA-PDDL	HTN (SHOP)	Golog	Jani
(I)	–	–	(✓)	(✓)	–	✓	–	(✓)	–	(✓)	✓
(II)	–	–	–	–	–	–	–	–	✓	✓	✓
(III)	–	(✓)	(✓)	(✓)	–	–	–	–	–	–	✓
(IV)	–	(✓)	(✓)	(✓)	–	(✓)	(✓)	–	✓	–	✓

Table 1: Language comparison overview. '–' no support, '(✓)' limited support, '✓' full support. See text for details.

pre-processing to store compatibility information. PDDL2.1 and RDDL support numeric fluents, i. e., predicates that map into real numbers instead of true/false. But numeric fluents cannot be used as arguments of predicates, so we cannot directly encode arrays. In the HTN variant of the SHOP system (Nau et al. 2003), action descriptions can link to arbitrary LISP code. Planning modulo theories (Gregory et al. 2012) could use arrays as a theory in principle, but that hasn't been implemented. No planning language explicitly supports arrays as part of the state model.

On the MC side, with models intended to reflect software, there is no inherent bias to logic-based encodings, and there is a variety of support for programming-like constructs. In particular, Jani admits arrays indexed with integer variables, and some tools support these effectively.

MC does also, of course, have weaknesses compared to AIP, for example regarding state uncertainty and distributed control. POMDPs are not part of the Jani language landscape. Multi-agent planning problems can be naturally modeled, but distributed planning is not an established MC problem so there is little to no tool support. Keep in mind that we are suggesting MC as a modeling alternative *for particular cases*, not as a lord of the languages to rule them all.

4 Use Case: Drone Control

Consider a scenario, like drone-fleet logistics, or the use of drones to aid disaster recovery, where we need to control a fleet of drones in an airspace containing also other moving objects, like 3rd-party drones. Drones must keep a safety distance from (fixed obstacles and) other drones.

One solution to the safety issue is to defer it to lower control layers, leveraging obstacle avoidance techniques, and maintaining a high-level road map only. Assume though that we want to take collision safety into account up front in high-level planning, based on traffic predictions. This makes sense, e. g., if the air space is typically crowded in particular places at particular times (drone fleet logistics), or if some areas leave little room for maneuvering and we are informed about exogenous drones' trajectories (disaster recovery). Note that exogenous drones, and our knowledge about their future trajectorties, are a key difference to the literature on combined task and motion planning (e. g. (Srivastava et al. 2014; Ferrer-Mestres, Francès, and Geffner 2017; Fernández-González, Williams, and Karpas 2018)).
We focus on two modeling issues:

(1) Exogenous drones' trajectories/traffic. How to model the behavior of the agents not under our control?

(2) Collision avoidance. How to ensure that our drone moves avoid collisions?

We will briefly discuss other possible challenges at the end of this section. We parameterize (1) and (2) as follows:

(1) Knowledge about exogenous drones: (a) estimates of crowded times/places; (b) assuming the current movement vectors remain fixed; (c) movement trajectories; (d) stochastic movement procedures.

(2) Movement granularity: (a) road map graph; (b) discrete 3D neighbor positions; (c) discrete 3D movement vectors; (d) continuous 3D movement vectors.

Regarding (1), in PDDL2.2 we can write a temporal model with timed initial literals. This necessitates to precompute all relevant points in time and space, and list these explicitly in the initial state. The structure underlying this data is lost completely, and the model may be huge. The only scenario where this approach is plausible is (1a), i. e., if we want to represent a small selection of critical time/space situations. A special case PDDL+ can handle is that of (1c) fully known piecewise linear trajectories for exogenous drones, in a continuous setting (2d), through exogenous events triggering processes, and "over all" conditions enforcing minimum distances. But that requires a separate process for every drone-trajectory segment, and "over all" conditions ranging over all objects. In RDDL we can model (1a)–(1d) through probabilistic state-transition functions.

Regarding space & movement granularity (2), the only clearly viable scenario is (2a). Scenarios (2b) and (2c) require either a "blocked" predicate for all possible (x,y,z) positions, or collision checks implementing a disjunction over all other objects. The latter is awkward, and results in unwieldy precondition formulas as the number of objects grows. The former severely limits scalability as the x, y, and z dimensions must be encoded via objects resulting in large ground encodings. Scenario (2c) additionally requires to check whether *any position on the move trajectory* is blocked. Expressing this as a precondition formula (or an artificial check-move phase in the planning model) is possible when distinguishing "all relevant cases" (naïvely: all possible movement vectors). But this is cumbersome to say the least. In scenario (2d), the only way to avoid collisions are preconditions ranging over all other objects. The sole exceptions to these difficulties are HTN planning with SHOP using LISP code, and Golog through its program structure.

Summing up, in PDDL dialects the only easy case are crowded times/places (1a) on a road map (2a), supported by timed initial literals. RDDL solves (1) but does not help with (2), SHOP and Golog solve (2) but do not address (1).

Consider now MC languages based on automata networks, in particular Jani. Figure 1 illustrates an example.

The behavior (1) of exogenous drones can be explicitly modeled through automata representing individual such drones in (1b)–(1d), or through an automaton representing a traffic summary in (1a). We can additionally choose whether to encode continuous time through timed automata, or discrete time through discrete clocks and synchronization.

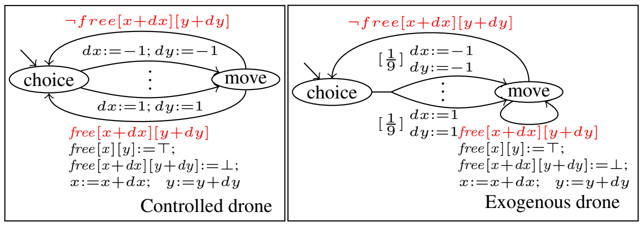

Figure 1: Illustration of automata-network encoding of type (2b) with exogenous drone of type (1d).

Collision avoidance (2) in discrete space (2b)–(2c) can be encoded in Jani based on a "free[x,y,z]" array of booleans, thus replacing an awkward logic-based encoding with the simple data structure actually needed here. Scenario (2c) (discrete movement vectors) requires a move-check phase, as in planning languages. But that phase can be implemented explicitly as a simple algorithm forming part of the agent's automaton, again replacing an awkward logic-based encoding with a much simpler explicit form. Similarly, in scenario (2d) parallel automata give us ample means for arithmetic modeling of exogenous drones, and a simple algorithm can be added for arithmetic intersection checks.

So MC languages and Jani have advantages (I) and (IV) from Section 3. With a partially predefined drone workflow we would also get (II), and with probabilistic duration uncertainty (e. g. about exogenous drones' movements) we would get (III). Potential weaknesses of MC lie in state uncertainty and distributed control as discussed in Section 3.

5 Use Case: Robot-Human Collaboration

We now consider a scenario where humans and robots are collaborating in some form of production. Specifically, say that the robots should behave like well-informed human aides would, i. e., autonomously and flexibly adapt to support what their human co-workers are currently trying to do. Say that the human workers follow fixed workflows, but that these are non-deterministic (ordering of working steps, alternate production routes, online job arrival, coffee breaks), and there is temporal uncertainty as different things will take different time depending on circumstance. The robots need to flexibly adapt to the workflow execution. Arguably, this setting is typical of many situations where well-informed human aides make sense, for example in small-batch production or in work at construction sites. Related settings have been considered, for example, in work on model-based flexible execution (e. g. (Levine and Williams 2018)).

A concrete use case we have previously explored is a riveting process in aircraft assembly (Rekik et al. 2019). Depending on the type of aircraft, there can be millions of rivets in one aircraft. Many riveting steps are manual. This requires two operators, one to perform the actual riveting, the other to perform a counter-holder role. The counter-holder positions are often hard to reach and work in, so the idea is to use a robot for that part of the work. The robot planner should flexibly adapt to the human's workflow decisions, pertaining to the ordering of riveting points and to breaks. In particular, the planner should exploit human breaks to inspect previous riveting points, while taking into account the time needed to come back when the human returns. Similar forms of collaboration and flexibility requirements arise, e. g., in work on a car chassis if the robot should assist in certain steps and/or hand over the required tools at the right points in time.

The modeling challenges we are interested in here are:

(1) Human co-worker workflows. How to model these and their non-determinism?

(2) Human workflow step duration uncertainty. How to inform the robot about the time distributions?

(3) Robot workflow structure. How to model predefined parts of the robot's workflow?

The problem parameters we discuss are:

(1) Human co-worker workflow non-determinism: (a) non-deterministic transitions; (b) probabilistic transitions with fixed probabilities; (c) probabilities depending on time (e. g. likelihood of taking a break).

(2) Human workflow step duration uncertainty: (a) exponential (memoryless) distributions; (b) general ones.

(3) Robot workflow structure: (a) robot action preconditions; (b) workflow with individual choice points.

The support of planning languages for (1) is limited. As human co-workers are exogenous agents not under control of the planner, the difficulties with (1) relate closely to those discussed for exogenous drones in Section 4. PDDL does not support non-deterministic exogenous events. A possibility might be PPDDL, using additional actions with probabilistic outcomes to model workflow non-determinism. RDDL naturally supports (1a) and (1b), but does not support (1c) other than by encoding aspects of "time" as a discrete state variable affecting the probabilities in question.

As discussed in Section 3, support for (2) in planning is basically non-existent. Regarding (3), (3a) can easily be dealt with by action preconditions in any planning language. But no language other than Golog supports the specification of a partial plan as a workflow with choice points.

Summing up, RDDL tackles (1) in discrete-time settings, and Golog tackles (3), but neither helps with the respective other problem and (2) remains basically unsupported.

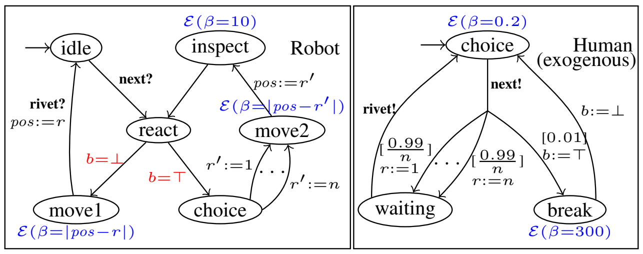

Figure 2: Illustration of automata-network encoding for the riveting application, with robot workflow of type (3b), and human workflow of type (1b) and (2a). $\mathcal{E}(\beta{=}x)$ denotes the exponential distribution with mean x.

The situation for MC languages, in particular Jani, is much more positive. Figure 2 illustrates an example. Continuous time and duration uncertainty (2) are a key modeling dimension in Jani. Mature tool support exists for exponential distributions (2a), and approximations to more general models (2b) exist as well. Scenarios (3a) and (3b) can be naturally and explicitly encoded into fixed parts of the robot workflow, with specific choice points to be resolved by the planner. Similarly, (1a) and (1b) are naturally and explicitly supported by parallel automata not under control of the planning agent, capturing the humans' workflows (online job arrival can be modeled by additional parallel automata). Scenario (1c) can be captured through parallel automata with uncertain durations, controlling conditions under which the humans' workflows can transition to different states.

In short, MC languages here have advantages regarding (I) exogenous agents, (II) pre-defined plan structure, and (III) probabilistic duration uncertainty. (IV) data structures may be helpful too, for 3D navigation. Potential weaknesses again lie in state uncertainty and distributed control.

6 Conclusion

The quest for suitable modeling languages is complex. So far that quest has been pursued largely independently in AI planning and model checking, with both communities devoted to their own language landscape. Yet, while what is "suitable" depends on the application, some scenarios traditionally viewed as applications of area X do not clearly favor that area's languages. Prior work provides evidence for this claim through various successful uses of compilations across areas, and through the use of MC tools for model-based control. Here we suggest to go one step further, using the other area's surface languages where they are more suitable. This can go both ways in principle, though our central proposition here is to use Jani, rather than adding yet more PDDL dialects covering the difficulties (I)–(IV) we described.

To be absolutely clear: we do not mean to imply that current planning languages are useless. Our proposition is to consider MC languages as yet another modeling alternative for planning. We hope that this will inspire other researchers from both communities, spawning a new brand of cross-area research and integration. For example, a first step may be to run an ICKEPS model-and-solve competition at ICAPS, without restrictions on the modeling language used.

7 Acknowledgments

This work was funded by DFG grant 389792660 as part of TRR 248 (see https://perspicuous-computing.science, and was partially supported by the ERC Advanced Investigators Grant 695614 (POWVER). We thank J. Benton and Jeremy Frank for discussions. The idea to run a no-restrictions ICAPS model-and-solve competition is Jeremy's.

References

Abdeddaïm, Y.; Asarin, E.; Gallien, M.; Ingrand, F.; Lesire, C.; and Sighireanu, M. 2007. Planning robust temporal plans: A comparison between CBTP and TGA approaches. In *Proc. ICAPS*.

Bacchus, F. 2001. The AIPS'00 planning competition. *AI Magazine*.

Baier, J. A.; Bacchus, F.; and McIlraith, S. A. 2009. A heuristic search approach to planning with temporally extended preferences. *AI*.

Baier, C.; Haverkort, B. R.; Hermanns, H.; and Katoen, J. 2010. Performance evaluation and model checking join forces. *Communications of the ACM*.

Bogomolov, S.; Magazzeni, D.; Podelski, A.; and Wehrle, M. 2014. Planning as model checking in hybrid domains. In *Proc. AAAI*.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *Proc. ICAPS*.

Bryce, D.; Gao, S.; Musliner, D. J.; and Goldman, R. P. 2015. Smt-based nonlinear PDDL+ planning. In *Proc. AAAI*.

Budde, C. E.; Dehnert, C.; Hahn, E. M.; Hartmanns, A.; Junges, S.; and Turrini, A. 2017. JANI: quantitative model and tool interaction. In *Proc. TACAS*.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *AI*.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2012. COLIN: planning with continuous linear numeric change. *JAIR*.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2013. A hybrid LP-RPG heuristic for modelling numeric resource flows in planning. *JAIR*.

De Giacomo, G., and Vardi, M. Y. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *Proc. IJCAI*.

De Giacomo, G.; Lespérance, Y.; and Levesque, H. J. 2000. Congolog, a concurrent programming language based on the situation calculus. *AI*.

Dehnert, C.; Junges, S.; Katoen, J.; and Volk, M. 2017. A storm is coming: A modern probabilistic model checker. In *Proc. CAV*.

Edelkamp, S.; Lluch-Lafuente, A.; and Leue, S. 2004. Directed explicit-state model checking in the validation of communication protocols. *STTT*.

Edelkamp, S. 2003a. Promela planning. In *Proc. SPIN*.

Edelkamp, S. 2003b. Taming numbers and durations in the model checking integrated planning system. *JAIR*.

Fang, C.; Yu, P.; and Williams, B. C. 2014. Chance-constrained probabilistic simple temporal problems. In *Proc. AAAI*.

Fernández-González, E.; Williams, B. C.; and Karpas, E. 2018. Scottyactivity: Mixed discrete-continuous planning with convex optimization. *JAIR*.

Ferrer-Mestres, J.; Francès, G.; and Geffner, H. 2017. Combined task and motion planning as classical AI planning. *CoRR* abs/1706.06927.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *JAIR*.

Fox, M., and Long, D. 2006. Modelling mixed discrete-continuous domains for planning. *JAIR*.

Geffner, H. 2000. Functional strips: A more flexible language for planning and problem solving. In *Logic-Based AI*.

Gerevini, A.; Saetti, A.; and Serina, I. 2008. An approach to efficient planning with numerical fluents and multi-criteria plan quality. *AI*.

Giunchiglia, F., and Traverso, P. 1999. Planning as model checking. In *Proc. ECP*.

Gregory, P.; Long, D.; Fox, M.; and Beck, J. C. 2012. Planning modulo theories: Extending the planning paradigm. In *Proc. ICAPS*.

Grosskreutz, H., and Lakemeyer, G. 2003. ccgolog – A logical language dealing with continuous change. *Logic Journal of the IGPL*.

Hartmanns, A., and Hermanns, H. 2014. The Modest Toolset: An integrated environment for quantitative modelling and verification. In *Proc. TACAS*.

Helmert, M. 2006. The Fast Downward planning system. *JAIR*.

Hoffmann, J., and Edelkamp, S. 2005. The deterministic part of ipc-4: An overview. *JAIR*.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2014. Language classification of hierarchical planning problems. In *Proc. ECAI*.

Holzmann, G. 2004. *The Spin Model Checker - Primer and Reference Manual*. Addison-Wesley.

Johnson, B., and Kress-Gazit, H. 2011. Probabilistic analysis of correctness of high-level robot behavior with sensor error. In *Robotics: Science and Systems*.

Karpas, E.; Levine, S. J.; Yu, P.; and Williams, B. C. 2015. Robust execution of plans for human-robot teams. In *Proc. ICAPS*.

Keller, T., and Eyerich, P. 2012. PROST: Probabilistic planning based on UCT. In *Proc. ICAPS*.

Kim, P.; Williams, B. C.; and Abramson, M. 2001. Executing reactive, model-based programs through graph-based temporal planning. In *Proc. IJCAI*.

Klauck, M.; Steinmetz, M.; Hoffmann, J.; and Hermanns, H. 2018. Compiling probabilistic model checking into probabilistic planning. In *Proc. ICAPS*.

Komenda, A.; Stolba, M.; and Kovacs, D. L. 2016. The international competition of distributed and multiagent planners (codmap). *AI Magazine*.

Kwiatkowska, M. Z.; Norman, G.; and Parker, D. 2011. Prism 4.0: Verification of probabilistic real-time systems. In *Proc. CAV*.

Lacerda, B.; Parker, D.; and Hawes, N. 2015. Optimal policy generation for partially satisfiable co-safe LTL specifications. In *Proc. IJCAI*.

Larsen, K. G.; Pettersson, P.; and Yi, W. 1997. UPPAAL in a Nutshell. *STTT*.

Levesque, H. J.; Reiter, R.; Lespérance, Y.; and Scherl, R. B. 1994. GOLOG: a logic programming language for dynamic domains. *J. Logic Programming*.

Levine, S. J., and Williams, B. C. 2018. Watching and acting together: Concurrent plan recognition and adaptation for human-robot teams. *JAIR*.

Little, I.; Aberdeen, D.; and Thiébaux, S. 2005. Prottle: A probabilistic temporal planner. In *Proc. AAAI*.

Lund, K.; Dietrich, S.; Chow, S.; and Boerkoel, J. 2017. Robust execution of probabilistic temporal plans. In *Proc. AAAI*.

McDermott, D. 2000. The 1998 AI planning systems competition. *AI Magazine*.

Nau, D. S.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *JAIR*.

Nies, G.; Stenger, M.; Krl, J.; Hermanns, H.; Bisgaard, M.; Gerhardt, D.; Haverkort, B.; Jongerden, M.; Larsen, K. G.; and Wognsen, E. R. 2018. Mastering operational limitations of LEO satellites the GomX-3 approach. *Acta Astronautica*.

Pistore, M.; Traverso, P.; and Bertoli, P. 2005. Automated composition of web services by planning in asynchronous domains. In *Proc. ICAPS*.

Rekik, K.; Hoffmann, J.; Müller, R.; Steinmetz, M.; and Vette-Steinkamp, M. 2019. Planning for human-robot collaboration using markov decision processes. In *Proc. Robotix Academy Conference for Industrial Robotics*.

Rintanen, J. 2000. Incorporation of temporal logic control into plan operators. In *Proc. ECAI*.

Rintanen, J. 2003. Symmetry reduction for SAT representations of transition systems. In *Proc ICAPS*.

Roberts, M.; Piotrowski, W.; Bevan, P.; Aha, D.; Fox, M.; Long, D.; and Magazzeni, D. 2017. Automated planning with goal reasoning in minecraft. In *Proc. ICAPS Workshop on Integrated Execution of Planning and Acting*.

Sanner, S. 2010. Relational dynamic influence diagram language (rddl): Language description. Available at http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf.

Srivastava, S.; Fang, E.; Riano, L.; Chitnis, R.; Russell, S. J.; and Abbeel, P. 2014. Combined task and motion planning through an extensible planner-independent interface layer. In *Proc. ICRA*, 639–646.

Vidal, T., and Ghallab, M. 1996. Dealing with uncertain durations in temporal constraint networks dedicated to planning. In *Proc. ECAI*.

Wang, D., and Williams, B. C. 2015. tBurton: A divide and conquer temporal planner. In *Proc. AAAI*.

Wehrle, M.; Helmert, M.; Alkhazraji, Y.; and Mattmüller, R. 2013. The relative pruning power of strong stubborn sets and expansion core. In *Proc. ICAPS*.

Williams, B. C., and Nayak, P. P. 1997. A reactive planner for a model-based executive. In *Proc. IJCAI*.

Williams, B. C.; Chung, S.; and Gupta, V. 2001. Mode estimation of model-based programs: Monitoring systems with complex behavior. In *Proc. IJCAI*.

Younes, H. L. S.; Littman, M. L.; Weissman, D.; and Asmuth, J. 2005. The first probabilistic track of the international planning competition. *JAIR*.

Yu, P.; Williams, B. C.; Fang, C.; Cui, J.; and Haslum, P. 2017. Resolving over-constrained temporal problems with uncertainty through conflict-directed relaxation. *JAIR*.