Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

Verification & Validation Meets Planning & Scheduling

Saddek Bensalem[1], **Klaus Havelund**[2⋆], **Andrea Orlandini**[3⋆⋆]

[1] Verimag Laboratory, Grenoble, France
[2] Jet Propulsion Laboratory, California Inst. of Technology, USA
[3] ISTC-CNR, National Research Council, Italy

Received: date / Revised version: date

Abstract. A *Planning and Scheduling* (P&S) system takes as input a domain model and a goal, and produces a plan of actions to be executed, which will achieve the goal. A P&S system typically also offers plan execution and monitoring engines. Due to the non-deterministic nature of planning problems, it is a challenge to construct correct and reliable P&S systems, including for example declarative domain models. *Verification and Validation* (V&V) techniques have been applied to address these issues. Furthermore, V&V systems have been applied to actually perform planning, and conversely, P&S systems have been applied to perform V&V of more traditional software. This article overviews some of the literature on the fruitful interaction between V&V and P&S.

1 Introduction

This article introduces a special volume of the International Journal on Software Tools for Technology Transfer, containing extended versions of selected papers presented at the 3rd ICAPS workshop on Verification & Validation (V&V) of Planning & Scheduling (P&S) Systems, abbreviated VVPS, held in Freiburg, Germany, 2011. The article provides an overview of literature on V&V of P&S systems, and more broadly on the intersection of V&V and P&S.

P&S systems are finding increased application in mission-critical systems that operate under high levels of unpredictability. Given a description of a desired goal, and a model of possible actions and their causal/temporal constraints, the planning problem consists of finding a plan, which is a sequence of actions, the execution of which is calculated to lead to the goal state under "normal" circumstances. Such technology can be used to generate plans to control a *plant* (for example a spacecraft, or a rover), driven by goals often issued by humans. Such technology is occasionally referred to as *model-based autonomy*.

One of the first applied approaches to model-based autonomy in a real-world context was the Deep-Space 1 (DS-1) experiment (1998-2001) by the NASA agency [48]. DS-1 was equipped with a "Remote Agent" (RA) software module capable of model-based goal-driven planning and scheduling, plan execution, monitoring and diagnosis, and recovery. The model-based diagnosis system of the RA was the Livingstone model [88]. This diagnosis system performed estimation of the mode of the spacecraft by updating a diagnosis model, taking into account the commands issued, and the observations perceived from the spacecraft. The RA monitoring and diagnosis system was an interesting and effective form of V&V technology, running in parallel with the executing system, and was in itself a contribution to the V&V research field.

However, broader scoped tools and methodologies for V&V [72] of P&S systems have until recently received relatively little attention, although this is changing, as documented in this article. In this regard, it is worth reminding that *verification* is the act of determining whether an artifact is correct with respect to a formalized specification, and *validation* is the act of determining whether an artifact is correct with respect to informal intentions. Another popular definition is, that verification is concerned with ensuring, that *you are building the thing right*, whereas validation is concerned with ensuring, that *you are building the right*

⋆ The work by this author was carried out at Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. The work was furthermore partially funded by NSF Grant CCF-0926190.

⋆⋆ The work by this author was partially funded by the Italian Research Ministry (MIUR) within the "Factory of the Future" Flagship Project Framework (GECKO project).

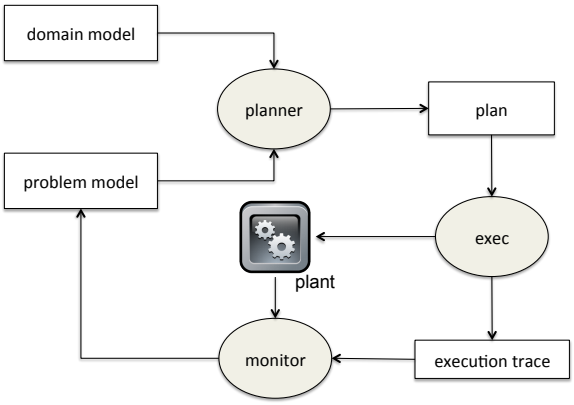

Fig. 1. Generic planning architecture

thing. The literature surveyed in this article does not in all cases conform with these definitions, and we therefore do not always conform neither, in order to be faithful to the formulations chosen by the various authors. In truth, the majority of the papers surveyed are concerned with verification rather than validation. See [80] for a description of the knowledge-acquisition process for models and heuristics of a complex autonomous system based on P&S, and [61] for a review of V&V problems and methods suitable for AI-based systems.

P&S systems have unique architectural features, that give rise to new V&V challenges. A planning system typically consists of a *planner*, see Figure 1, that is largely stable across applications. A planner takes as input a declaratively-specified *domain model*, stable for a particular application, and a *problem model* defining a given *initial state* and a *goal* to be achieved, varying within an application. From these two inputs, the planner produces, usually taking advantage of heuristic search, a *plan* of actions achieving the goal. The plan is subsequently executed by an *exec*, which controls a *plant* via actuators. The execution in turn produces an *execution trace* of actions executed, which are fed to a *monitor*, which also reads the status of the controlled plant via sensors. Based on these observations, the monitor determines whether the execution is well-behaved, and as a result provides input to generate new goals (updating the problem model) for the next planning step.

Experience has shown that most errors are in domain models, which can be inconsistent, incomplete, or inaccurate models of the target domains. There are currently few tools to support the model construction process itself, and even fewer that can be used to verify and validate the models once they are constructed [85, 76]. Another challenge to V&V of P&S systems is to demonstrate that specific heuristic strategies have reliable and predictable behaviors.

A field closely related to planning is *program synthesis*. We shall only briefly mention this area of research, without going into details. A thorough survey of the topic can be found in [16]. The general aim of program synthesis is to derive low-level programs from high-level logical declarative specifications. Planning likewise is concerned with deriving "programs" (referred to as plans) from declarative models, namely the domain models. Two kinds of synthesis are mentioned in [16]: (i) synthesis of controllers for reactive systems as well as synthesis of hardware circuits, and (ii) synthesis of data-oriented functional and imperative programs. The former category is of specific relevance to planning. A popular approach here is to synthesize finite state programs (automata) from Linear Temporal Logic (LTL) [70] specifications. Examples include [71,54,69]. Such programs are typically non-terminating, i.e., programs that accept an infinite stream of requests. In contrast, plans typically do not contain loops.

Previous work has been dedicated to the study of semantics/constraint-based work flow synthesis [36]. The basis here is SLTL (Semantic Linear Time Logic), which is interpreted over regular languages (finite words) rather than omega languages (infinite words). Other work includes [84], which presents synthesis technology to automatically compose tool chains in a goal-oriented fashion, and [83] which uses this approach to automatically generate benchmark programs with known temporal properties. The reader may find further references in [16].

The work on NASA's Remote Agent led to the creation of the VVPS workshop series in 2005[1] aiming at establishing a long-term forum focusing on the interaction between V&V and P&S. The original goal of the VVPS workshop series was to identify innovative V&V tools and methodologies, that can be applied to ensure the correctness and reliability of P&S systems. However, the workshop series has also attracted papers with slightly different bends, such as using verification systems, for example model checkers, as planners (planning as model checking). Of course, whether a model checker is used to verify a domain model by exploring its state space, or is used as *the* planner, is only a subtle difference, using similar ideas but with different objectives (verification versus planning). Finally, work has emerged that goes in the complete opposite direction, namely focusing on the use of P&S systems to solve V&V problems for traditional software systems.

The three pieces of work selected and included in this volume of the International Journal on Software Tools for Technology Transfer, are briefly described in the following.

Article [42] *"A loop acceleration technique to speed up verification of automatically-generated plans"*, by Goldman, Pelican and Musliner, presents the integration of an optimization technique (called *loop acceleration*) in the CIRCA planning system, to address the state space

[1] In 2004 Remote Agent P&S scientist Kanna Rajan suggested to V&V scientist Klaus Havelund (both at NASA Ames Research Center at the time) to organize a workshop on the topic: "V&V of P&S systems". The series was started in 2005 [2] and continued in 2009 [3] and 2011 [4].

explosion issue while performing runtime verification of reactive plans. In particular, such problem is encountered while checking CIRCA controllers, that execute quick reactions to meet environmental threats, while simultaneously monitoring long-duration processes. The paper describes the technique and its implementation, showing that it radically speeds up the verification process.

Article [26] *"Authorized workflow schemas : Deciding realizability through LTL(F) model checking"*, by Crampton, Huth, and Kuo, proposes the use of model checking of an NP-complete fragment of propositional linear temporal logic (LTL) as an alternative solution to the workflow satisfiability problem, i.e., the problem of determining whether there exists a workflow plan that realizes workflow specifications. A suitable LTL encoding is reported aiming at modeling business processes as workflows and, thus, showing the verification method effectiveness while checking authorization plans against business rules, legal requirements and authorization policies.

Article [75] *"Generating effective tests for concurrent programs via AI automated planning techniques"*, by Razavi, Farzan, and McIlraith, presents a general approach to concurrent program testing, that is based on AI automated planning techniques. A framework is proposed for finding concurrent program runs that violate a collection of specifications. The problem of finding failing runs is characterized as a sequential planning problem, with the temporally extended goal of achieving a particular violating pattern.

The remaining part of this introductory article provides a brief survey of other articles published broadly within the intersection of V&V and P&S.

The paper is structured as follows. Section 2 surveys literature on V&V of P&S systems (the originally intended theme of the workshop series). Specifically (See Figure 1) V&V of domain models in Section 2.1, plans in Section 2.2, plan executions in Section 2.3, planners in Section 2.4, execution engines in Section 2.5, and monitors in Section 2.6. Section 3 surveys literature on the use of V&V technology for performing planning and scheduling. This includes planning as model checking in Section 3.1, and logic-based approaches to planning in Section 3.2. In the other direction, Section 4 surveys literature on the use of P&S technology for verification of tradition software systems. Finally Section 5 concludes the paper.

2 V&V of P&S Systems

V&V can as already mentioned be applied to different artifacts of a P&S system, specifically the domain model, plans, plan executions, the planner, the exec, and the monitor. The following sub-sections cover selected V&V literature in these respective areas.

Most of the approaches mentioned solve specific problems. In [17], however, is described a more comprehensive approach to on-board autonomy, relying on model-based reasoning. This approach offers a uniform formal framework, including model validation, plan generation, plan validation, plan execution and monitoring, as well as fault detection identification and recovery. The approach is based on a symbolic representation of the system to control, and uses model checking techniques (specifically the NuSMV model checker) to validate the symbolic representation of the system, in essence following a *planning as model checking* approach (see later Sec. 3.1). Representing a formal model in terms of a Kripke structure allows to validate the model to guarantee, that it captures the behaviors of the system. Plans contain assumptions that can be checked during execution. The work in [17] is similar in spirit to the Remote Agent experiment, but differs by using the same formal model in all phases.

2.1 V&V *of Domain Models*

In P&S systems, the domain model plays a crucial role. A domain model formalizes what actions are possible, and their constraints (such as for example orderings: action A must always precede action B). A domain model in part reflects the complex environment in which the plant is operating. The correctness of a domain model has a direct impact on plan correctness (e.g., safety, liveness) and performance. Due to modeling errors, a plan model can, however, be inconsistent, incomplete, or simply inaccurate. Domain model languages are typically declarative, such as for example PDDL [60], and models are usually small compared to industrial sized software programs. In spite of these characteristics, it is the declarative nature of domain models, that makes it a challenge to explore all possible planning scenarios up front. For these reasons, V&V of domain models is a critical task, that has been considered by several authors, and which perhaps is the biggest V&V challenge to the P&S community.

Domain model verification aims at showing that (i) plans can, or cannot, be generated for various goals, and (ii) that generated plans satisfy given properties. This can be done using formal methods, e.g., model checking, or just using more traditional testing. Testing can only show the *presence* of errors (i.e., if no error is found there is no guarantee that none exists), whereas model checking in theory also can demonstrate the *absence* of errors (i.e., if no error is found we are guaranteed that none exists). Not surprisingly, model checking is computationally much more expensive than just testing, since the former will look at all reachable states of the domain model. Because the number of such states in general is exponential in the domain size (*state explosion*), only *moderate size* domains can typically be handled using model checking techniques for exhaustive verification. Note, that the problem is different when using model

checkers for planning (planning as model checking), since the goal there is to find a single plan (error trace), rather than to perform an exhaustive search. In a testing-based approach to domain model verification, a large number of plans are generated and then checked to verify that each of them satisfies the given properties. Testing-based domain verification rests on *plan verification*, which will be discussed in Section 2.2.

In [56] is discussed in general terms the problem of verifying and validating domain models. The paper identifies some examples of domain modeling errors and discusses how common they are, noting that domain models usually are much smaller than traditional software. One technique suggested is to cast a domain model into different representations, each focusing on different aspects of the model. This process enhances inspection by requiring the reader to go through a process of mental evaluation. The paper also discusses how *plan validation* is a process to indirectly validate domain models, comparing plan validation with unit testing.

2.1.1 V&V of Domain Models using Model Checking

In a model checking approach, a domain model is formulated as a model in the modeling language of a model checker. The model checker can then be used as a planner by formulating the planning problem as a temporal logic satisfaction problem, where the goal is transformed into a temporal logic formula representing the negation of the goal: that it cannot be reached from the initial state. The model checker will then, if the goal is reachable, produce an *error trace* leading to the goal state. The error trace represents the plan. Stated in a slightly more formal manner, although still in generic terms, a planning problem:

$$\Pi = \langle D, P(i, g) \rangle$$

consists of a *domain model* D and a *problem model* $P(i, g)$, stating a particular initial situation i and a goal g. Solving the planning problem consists of applying the *planner* to the problem to obtain a plan:

$$plan := planner(\Pi)$$

The planning problem can alternatively be reformulated as a model checking problem as follows. Let Π^{MC} be the corresponding model represented in the model checker's input language. Using LTL (Linear Temporal Logic) [70] for writing properties, and assuming for simplicity that the goal g can be carried across unmodified, the planning problem can be formulated as the following satisfaction problem:

$$\Pi^{MC} \models \neg \diamond g$$

This represents the assertion, to be proved by the model checker, that there is no execution trace in Π^{MC} from the initial state, for which it holds that eventually ($\diamond$) the goal state g is reached. If, however, there is a such, the model checker will produce an *error trace*, leading from the initial state to the goal state, effectively a *plan*.

Furthermore, checking that a temporal property Φ is true on all plans generated from Π to reach a goal g, would correspond to formulating the following model checking problem, which states that if a trace eventually reaches the goal g then that trace also satisfies Φ:

$$\Pi^{MC} \models ((\diamond g) \Rightarrow \Phi)$$

The use of model checking for V&V of domain models was pioneered in [68], using three model checkers (Spin, SMV, Murphi). This work studied expressiveness, as well as efficiency and scalability of verification of safety and liveness properties of simple planning domains for the HSTS planner [63]. Also the work described in [81,43] explores the use of model checking with Spin to guarantee that all plans enabled by a domain model meet certain desired properties. Real-time temporal properties and temporally flexible plans are not addressed in either of these works.

Formal methods applied to timeline-based temporal planning are considered within the ANML framework, a timeline-based domain modeling language proposed at NASA Ames Research Center. In [77] the authors present a translator from ANMLite (a simplified version of ANML) to the SAL model checker. Given this mapping, the authors illustrate preliminary results to assess the efficiency of model checking in plan synthesis. The main purpose with this work, however, is to support NASA Ames in the definition of the ANML language, by offering a verification technology for analyzing ANML domain models in an exhaustive manner.

Using a more expressive temporal model to represent time constraints, the authors of [52,53] propose to map from interval-based temporal relation models (i.e., DDL models for HSTS) to timed automata models (UPPAAL). This mapping was in part introduced in order to understand the relationship between timed automata and P&S technology, and in part to explore the application of V&V techniques in timeline-based temporal planning. Analogously, [86] presents a mapping from contingent temporal constraint networks to Timed Game Automata (TGA).

2.1.2 V&V of Domain Models using Testing

In [74] is presented a methodology and tool (PDVer) for testing PDDL domain models, based on generating tests from LTL formulas. More specifically, from an LTL formula is generated a set of test cases, each consisting of a PDDL goal; and the planner itself is then used to "run the test": generate a plan or fail. LTL coverage criteria drive the test case generation. The work is based on the basic observation, that the alternative approach of translation a domain model into the language of a

model checker, and then apply model checking to explore the domain model, is not practical due to the size of the state space; and the fact that the PDDL model of the system could include features (such as durations and costs) that are hard to encode in the input language of a model checker.

In [39] is described an approach to regression testing of a plan models using a planner and a temporal property synthesizer. The scenario is one where a plan model is constantly modified, and after each modification one needs to ensure that it satisfies certain properties from a planning perspective. For a given goal (test input) the planner generates a plan, while emitting planning operations on a log. From the same input is also generated a temporal property that this planner log must satisfy. The satisfaction reflects the fact that the plan model itself is correct wrt. certain criteria. By checking logs against temporal properties, higher flexibility is achieved than if comparing logs from different regression runs against each other.

2.2 V&V *of Plans*

Plan verification consists of checking, that a generated plan satisfies certain properties. A typical approach is to generate a limited number of sample plans, which are then checked by automated test oracles. For example, this method was employed in the testing of the Remote Agent [79,78], where a few hundreds of plans were generated to validate the domain model. The effort in this case, however, was still long and expensive, since, although the automated test oracles pointed out violations, humans still had to investigate the error reports manually to identify the actual causes of the violations. This and similar efforts have indeed motivated further research on automatic tools for plan verification. Note that plan verification can be also used for automated testing of the planner as well, by showing that the planner's output is correct with respect to given properties. Checking plans is considerably easier than showing correctness of the planner itself.

Verification of temporal plans expressed in PDDL with durative actions is enabled by the VAL plan verification tool [46], that has been used during international planning competitions since 2002. However, flexible temporal plans, complex temporal constraints, and other temporal features are still to be addressed [35].

The MURPHY system [40] is proposed to analyze a plan to identify ways that uncontrolled (disturbance) actions could cause the plan to fail in execution, and produce counterexample traces that would show how failures could occur. MURPHY translates a plan into a *counter planning* problem, combining a representation of the initial plan with the definition of a set of uncontrolled actions. These uncontrolled actions may be the actions of other agents in the environment, either friendly, indifferent or hostile, or they may be events that simply occur. The result of this translation is a disjunctive planning problem, which is further processed in order to play into the strengths of existing classical planners. Using this formulation, a classical planner can find counter examples that illustrate ways a plan may go awry.

More recently, work has been performed on verifying flexible timeline-based plans, by translating them into timed game automata, which are then analyzed using model checking techniques, specifically UPPAAL-TIGA [12]. In this regard, a suitable TGA formalism has been proposed to verify flexible plans [21,22].

In [82] is described an approach for finding conditional plans with loops and branches for planning in situations with uncertainty in state properties as well as in object quantities. A state abstraction technique from static analysis of programs is used to build such plans incrementally, using generalizations of input example plans generated by classical planners. Pre-conditions of the resulting plans with loops are computed. Although this work focuses on generating conditional and looping plans, by determining the plan pre-conditions, the work effectively addresses verification of plans with program-like structure, including branches and loops.

A problem related to plan verification is the problem of determining the distance between plans generated by a planner using different planning strategies, for example in a dynamic environment where a plan has to be adapted and replaced with an alternative plan achieving the same goals. The work in [67] defines a notion of plan proximity, that is more precise than a previously suggested notion of plan stability. Plan proximity considers actions missing from the reference plan, extra actions added in the new plan, sequential ordering of the plans, and the expected outcome states of these plans. Robust plan validation during execution is considered in [34], where hybrid timed automata are deployed to handle plan validation with temporal uncertainty. The paper proposes a probing strategy, where plans around a selected original plan are generated, forming a *tube* around the original plan, to exercise robustness in the face of uncertainties concerning the timing of selected actions. The width of the tube is a parameter, which can be adjusted to present a degree of robustness testing.

In [11] is described an approach, applied in a NASA mission, to verification of *command sequences* against a set of *flight rules*, before the sequences are sent to a satellite. A command sequence is usually created manually on ground by scientists and engineers, but has the same characteristics as a plan: it is a sequence of actions (commands) to be executed on board the satellite. The flight rules are properties that command sequences must satisfy, and are in this approach formulated as monitors in the TraceContract tool, an API in the Scala programming language. The API offers classes and methods for writing linear temporal logic properties as well as data parameterized state machines. TraceContract was originally designed for analyzing program execution traces,

where an execution trace is a sequence of events that occur during execution of a program/system. However, from the tool's perspective, a command sequence is just a sequence of events (commands).

NASA operates manned spacecraft according to rigorously defined *procedures*. Procedures can be viewed as plans for crew and flight controllers. Procedure V&V is currently mostly done through human reviews. The paper [19] describes an approach to verification of procedures for human space flight (the Space Shuttle and ISS) based on model checking, specifically with the JPF (Java PathFinder) Java model checker. Procedures formulated in PRL (Procedure Representation Language) are translated into finite state machines (in Java), which, when coupled with a finite state machine representing the controlled system, can be verified.

Some procedures can be executed both automatically and manually. In some cases manual procedures are defined as backup for automated procedures. In [64] is described an approach to demonstrate that procedures defined in the two different procedure description languages SCL and PRL are equivalent. This is accomplished by translating both procedures to the common verification language Promela, and using the Spin model checker to confirm that the procedures behave identically when given identical inputs. The objective is to provide assurance for NASA engineers, that if an automatic SCL program cannot be executed, a backup manual procedure in PRL will be equivalent and safe. The approach generalizes to comparisons between other procedure representation languages.

2.3 V&V *of Plan Executions*

V&V of plan executions can be categorized into *runtime verification*, verifying the executions against properties, and *runtime enforcement*, enforcing robust plan execution.

2.3.1 Verification of Plan Executions

V&V of the planner, the domain model, and even the generated plans themselves, does not guarantee robustness of actual plan *execution*. Indeed, a valid plan can be brittle at execution time due to environment conditions that cannot be modeled in advance (e.g., *disturbances*). As a last line of defense, V&V techniques can be used for plan *execution verification*, also referred to as *runtime verification*. Several P&S systems include a monitor component, which monitors plan executions, and react accordingly in case expectations are violated. The Livingstone system [88] is an example of a such, part of the Remote Agent P&S system, designed to control the NASA DS-1 spacecraft. Sub-section 2.6 is specifically devoted to V&V of such monitors.

In [7] is described an approach to automatically test the NASA K9 execution engine based on checking plan executions against temporal properties. A test case consists of a plan and an oracle expressed in temporal logic, that can be used to test, that the execution of the generated plan conforms with the intended plan semantics. As a follow-up of the experiment described in [7], the work of [37] describes a compositional approach to V&V applied to the K9 Rover executive system, using the same runtime verification techniques for checking plan executions, but in a compositional verification context. In [18] is described a study comparing different techniques to verify the K9 execution engine. The techniques include monitoring plan executions against hand-written temporal properties, reflecting the plan semantics, as well as execution trace based deadlock and data race analysis.

The K9 rover plan execution scenario is also considered in [15]. Here, a generated plan for the rover is transformed into a timed automaton. An observer is synthesized from the timed automaton to check whether the sequence of observations complies with the specification.

2.3.2 Robust Plan Execution

Robust plan execution in uncertain and dynamic environments is a critical issue for plan-based autonomous systems. Indeed, once a planner has generated a temporal plan, it is up to the exec to decide, at run-time, how and when to execute each planned activity, preserving both plan consistency and controllability. Such a capability is even more crucial when the generated plan is temporally flexible and partially specified. Such a plan captures an envelope of potential behaviors, to be instantiated during the execution, taking into account temporal/causal constraints and controllable/uncontrollable activities and events. In this regard, several authors (e.g., [62]) proposed a dynamically controllable execution approach where a flexible temporal plan is then used by an exec system that schedules activities on-line while guaranteeing constraint satisfaction. Then, given a plan, a plan controller can be defined as a scheduling function that provides suitable timings for plan actions execution. And, an exec system can be endowed with a plan controller to *guide* plan executions. In this regard, several research initiatives integrate P&S and V&V techniques aiming to enforce robust execution and monitoring.

The work in [66] presents a method to synthesize robust plan controllers for timeline-based flexible plans, solving a TGA model checking problem. In this work, flexible temporal plan evolutions are modeled as TGA automata and, a winning strategy generated after the UPPAAL-TIGA verification process is used to generate a flexible plan controller, that achieves planning goals maintaining dynamic controllability during the overall plan execution.

In [35] is described the VAL framework, coupled with a plan-execution architecture, which has been applied to on-board plan verification and repair. This can be considered as an element in robust plan execution. The ob-

servation made by the authors is, that while on-board planning technology has an important role to play, state of the art technology does not make it practical for systems with limited resources (the success in the Remote Agent experiment notwithstanding). Their goal has been to provide an on-board "planning assistant", performing adjustment and repair of plans on-board, when circumstances make it impossible to execute the plans as they were constructed on the ground by humans.

The CIRCA planning system [41] is an architecture for intelligent real-time control. It includes a real-time subsystem used to execute reactive control plans that are guaranteed to meet the domain's real-time deadlines, keeping the system safe. In this regard, CIRCA automatically creates reactive plans and uses formal verification techniques to prove that those plans will preserve system safety. In particular, the CIRCA's Controller Synthesis Module (CSM) uses timed automata to generate reactive plans as time discrete controllers, and uses a model-checking based plan verifier to check reactive plans against safety requirements.

2.4 V&V *of Planners*

V&V of a planner consists of ensuring that the planner itself works correctly. This task corresponds to the more traditional V&V task of ensuring, that a large piece of complex software works correctly. As discussed above, formal methods have mostly been applied to V&V of domain models, plans, and plan executions, since those artifacts appear somewhat more manageable than the planner software (and any of the other involved software systems, such as exec and monitor). Traditional testing is therefore still the most commonly applied V&V approach in practice to ensure correctness of planners. For example, the verification of the P&S system for the Remote Agent [65,79,48] is based on test cases to check for convergence and plan correctness. More specifically, the P&S system is verified by generating hundreds of plans for a variety of initial states and goals, and by verifying that the generated plans meet a validated set of plan correctness requirements. Plan verification can be done by using an automated plan-checker.

A similar approach has been followed at JPL for validating the EO-1 science agent [23]. A key issue in empirical testing is achieving adequate coverage with a manageable number of tests. Test selection should be guided by a coverage metric. However classical approaches used for testing traditional software systems are not suitable for planning systems because of the complex search engines and rich input/output space. Within the IDEA framework of the Remote Agent [48], model checking techniques are used to explore the space of input scenarios in order to generate tests for the planner [73].

In [47] is described a project to automate the scheduling process for NASAs Deep Space Network (DSN). The paper lays out an approach to verification and validation of the scheduling engine component of this system. The scheduling engine is responsible for interpreting users requests for communications and other services from the DSN, then generating and checking schedules that achieve those requests. The verification process described involves several elements, including regression testing, performance testing, script-based test case generation, as well as a test GUI allowing to easily experiment with the system and generate tests. Users are given access to the GUI, and are therefore part of the process of defining test cases. Various static and dynamic program analysis tools are used.

In [33] it is noted that it is easier to check that a plan is correct with respect to a model, than it is to produce a proof that the planner itself is correct. They load a plan resulting from execution of the planner into a database, and then check the database against constraints generated from the model.

2.5 V&V *of Plan Execution Engines (Exec)*

V&V of a plan execution engine (exec) consists of ensuring, that plan execution works correctly for any input plan. As is the case for V&V of the planner, this task corresponds to the more traditional V&V task of ensuring, that a piece of software works correctly. Also for the exec, traditional testing is the most commonly applied V&V approach in practice.

The work [18,7,37] already mentioned in Section 2.3.1 on verification of plan *executions*, is relevant for plan execution *engine* verification as well, for obvious reasons. In [18] is described a study comparing different techniques to verify the K9 plan execution engine, consisting of 35,000 lines of C++ code, for which a downscaled 6000 line Java version was used for part of the experiment. The techniques include monitoring plan executions against hand-written temporal properties, reflecting the plan semantics, deadlock and data race analysis, model checking, static analysis, and traditional testing.

In [7] is described an approach to automatically test the K9 execution engine, based on checking plan executions against temporal properties (using a different temporal logic than the one used in [18]). The test framework uses model checking and symbolic execution to automatically generate test cases, which are then applied. A test case consists of a plan and an oracle, expressed in temporal logic, that can be used to test that the execution of the generated plan conforms with the intended plan semantics. The plan language allows for branching based on conditions that need to be checked, and also for flexibility with respect to the starting time and ending time of an action.

As a follow-up on the K9 experiments described above, the work of [37] describes a compositional approach to V&V applied to the K9 rover plan execution engine, by deploying formal methods throughout the overall design

and development lifecycle. The approach uses the same temporal logic monitoring framework as used in [7].

Another example of checking execution traces against specifications is the work described in [10]. The approach here is to verify the operation of a spacecraft software controller (NASA's Mars Curiosity Rover [1]) by analyzing logs generated by the running software against temporal properties. The rover receives command sequences from ground, similar to plans, to be executed over a limited time period.

In [45,57] is described an application of model checking to verify the correctness of the plan execution engine for NASA's DS-1 spacecraft. An abstraction of the exec (programmed in LISP) was modeled in the Promela modeling language of the Spin model checker. In [44] is described a followup analysis (using the Java PathFinder Java model checker, based on Spin) of the same code after one of the errors, identified in [45] as a data race, actually occurred in flight causing a deadlock.

In [8] is described a dynamic data race detection analysis algorithm (analysis is performed during execution of an instrumented program) detecting inconsistencies in the way groups of variables are protected by locks in a concurrent program. Such data races involving several variables are referred to as *high-level* data races. Lack of such consistency may reflect coding errors. The high-level data race problem was inspired by the actual data race in the DS-1 spacecraft, also identified in [45] before flight using model checking, which caused the before mentioned deadlock in space, as documented in [44] after flight, also using model checking. The dynamic analysis approach is a scalable alternative to model checking for detecting this kind of error.

2.6 V&V *of Plan Execution Monitors*

A monitor analyzes the execution of a plan, and initiates recovery actions in case the expected behavior is violated. As such, a monitor is itself part of the V&V solution. However, even the monitor can be incorrectly programmed. V&V of a monitoring system consists of verifying and validating, that the monitor makes the right judgments about the correctness of the current execution, and in case of execution errors, that the right reactions are triggered.

There is not a large amount of work on V&V of P&S monitoring systems. At the best of our knowledge, the only relevant work is related to the Livingstone PathFinder (LPF) [55], a system for testing Livingstone models. Livingstone is the model-based monitoring and diagnosis system for the Remote Agent. LPF consists of a test driver, that generates a sequence consisting of either commands or injected faults, a simulator of the modeled device, and the Livingstone engine. The system checks whether the diagnosis system can detect the faults injected into the input stream.

3 V&V Systems used for P&S

In the approaches mentioned so far, various methods, including formal methods, have been used to *analyze* planning artifacts. A slightly different bend is the use of formal methods to actually *perform* planning. Such approaches reflect the observation, that both kinds of techniques (V&V and P&S) are based on search, which has created an interesting cross fertilization between formal methods and artificial intelligence. Note, that just the fact that a formal methods based tool is used for planning, does not necessarily mean that the generated plans are correct by construction. A model checker can contain errors, just as can a planner. However, some verification tools, such as some theorem provers, are based on a very small kernel, which can be estimated to be correct with a very high probability.

3.1 Planning as Model Checking

The *"planning as model checking"* approach [24,25,38] considers the planning problem as a model checking problem, using a model checker to perform planning. This approach is based on the representation of a domain model as a finite state automaton, effectively a model in the modeling language of the model checker. Planning is done by verifying whether temporal formulas are true or not wrt. the model, along the lines illustrated in Section 2.1.1. In the above mentioned work, symbolic representation and exploration techniques based on symbolic model checking, using Binary Decision Diagrams (BDDs), allow for efficient planning in non-deterministic domains. In [17] is described the later result of a comprehensive approach to on-board autonomy relying on using the NuSMV model checker to perform planning, using a symbolic representation of the system to control. Representing a formal model in terms of a Kripke structure allows to validate the model to guarantee, that it captures the behaviors of the system.

An interesting application of real-time model checking [13] has been dedicated to extend and retarget the timed automata technology towards optimal planning and scheduling. Two interesting applications have been studied, demonstrating the use of UPPAAL-CORA for the generation of optimal plans and schedules. In [14], task graph scheduling problems are modeled as networks of priced timed automata (PTA), and then solved by means of a branch-and-bound algorithm for solving cost-optimal reachability problems. In [27], planning problems are defined by means of a variant of PDDL 2.1, i.e., considering duration-dependent and continuous effects. Planning problems are then translated into linearly priced automata, and UPPAAL-CORA is exploited to generate cost-optimal traces which represent valid plans.

In [5], the authors investigate and compare constraint based temporal planning techniques and timed game

automata methods for representing and solving realistic temporal planning problems. In this direction, they propose a mapping from IxTeT planning problems to UPPAAL-TIGA game-reachability problems and present a comparison of the two planning approaches.

3.2 Logic-based Approaches to Planning

Traditionally, planning has been formalized as deduction: plans are generated by constructive proofs of so-called plan specification formulae, stating that there exists a plan leading from the initial state to a state satisfying the goal. The best-known logical formalization of planning in the deductive view is the Situation Calculus [59]. Such seminal work has influenced many works. Related to the VVPS workshop series, in [28] is described an approach, which tackles the planning problem as a theorem proving task. The paper describes a formalization of the planning problem in the Isabelle/HOL theorem proving system of Intuitionistic Linear Logic (ILL), specifying the initial state and possible actions. The theorem proving task is then to show that some goal resources can be realized from the given resources, using actions as basic inference steps. Furthermore, such a proof can be mapped mechanically into a typed functional programming language, yielding an executable plan. The plans so found are provably correct, by construction (assuming the correctness of the theorem prover).

Dually to the *planning as derivability* approach, the *planning as satisfiability* paradigm was introduced by [51], and carried on in [49,50]. According to this paradigm, a planning problem is encoded by a logical theory, modeling the rules governing the world evolution, in such a way that any model of the theory corresponds to a valid plan. Based on the above cited works is the planner SATPLAN. In SATPLAN, the target logic of the encoding is classical propositional logic. The work presented in [58, 20] conforms to the *planning as satisfiability* paradigm but, differently from [51], the logic used to encode planning problems is propositional LTL. The choice of LTL is mainly motivated by the fact that it allows a simple and natural representation of a world that changes over time. Moreover, domain dependent knowledge can be expressed in LTL, as well as domain restrictions and intermediate tasks.

It is also worth to underscore that, as a side effect, the use of logic-based approaches enables the exploitation of well known formal method tools/techniques to perform V&V of planning systems following such approaches.

4 P&S Systems used for V&V

Search heuristics, inspired by those used in planning, have been studied for verification systems, for example as described in [87]. However, one can go even further, and apply planning technology directly as a verification technology for ensuring correctness of traditional software systems. For example, the effectiveness of translating model checking inputs into PDDL has been described in several papers, including from communication protocol specification languages, [29], Petri nets [30], μ-calculus formulae [9], and graph transition systems [31].

In [6] the main idea is to formulate the system model to be analyzed, as well as the property it has to satisfy, as a planning problem. To illustrate the approach, models in NuSMV and Promela (Spin's modeling language) are translated to planning domain models and goals in PDDL. Experimental results comparing the planning approach to NuSMV and Spin show that planners can provide significant time improvements when checking safety and liveness properties *that are violated*, compared to state-of-the-art model checkers, especially on large tasks. Results are less convincing in the case where properties are *not violated*, and the whole state space therefore must be explored. In other words, for error detection (in contrast to proof of correctness) the approach appears promising.

The work presented in [32] relies on the translation of concurrent C/C++ programs into PDDL domain models. The system then runs a heuristic-search based planner on such a generated PDDL model to generate a plan, read: error trace, for locating programming bugs. This counter-example error trace is then used to provide an interactive debugging aid.

5 Conclusion

This paper introduces extended versions of papers selected from the 3rd workshop on *Verification and Validation of Planning and Scheduling Systems*, The paper continues with an overview of work done in the intersection of V&V and P&S. This includes work on V&V of P&S systems for ensuring the correctness of the latter; work on using V&V systems to perform P&S, and finally the use of P&S systems to perform V&V of traditional software systems. The overview is not exhaustive by any means.

The original focus of the VVPS workshop series was the study of V&V techniques to ensure the correctness of P&S systems. The model-based nature of P&S systems should in principle make it easier to apply V&V techniques. However, as it turns out, P&S systems present features that make them hard to verify and validate, such as the non-deterministic nature of the domain models and planner heuristics. Thus, powerful tools/methods, such as model checkers originating from the formal methods/software engineering community, have been studied for this purpose.

However, it is clear that the other interactions between V&V and P&S as mentioned above are interesting, including topics such as *planning as model checking*

and *model checking as planning*. The common thread in these techniques are specification languages and search-based analysis techniques. Research in the cross section between V&V and P&S is important for both fields, since the impact seems to bi-directional.

An interesting topic for future research is the relationship between *model-based programming* and *model-based planning*. The former relies on program synthesis techniques to derive programs from models, or verification techniques to prove a program correct wrt. the model. The latter relies on generating plans (programs) on the fly from models, incorporating models, programs and fault protection within one framework. It would be desirable to formulate a unifying framework encompassing these different views.

References

1. Mars Science Laboratory (MSL) mission website. http://mars.jpl.nasa.gov/msl.
2. VVPS 2005 workshop website. http://icaps05.uni-ulm.de/workshops.html.
3. VVPS 2009 workshop website. http://www-vvps09.imag.fr.
4. VVPS 2011 workshop website. http://icaps11.icaps-conference.org/workshops/vvps.html.
5. Y. Abdedaim, E. Asarin, M. Gallien, F. Ingrand, C. Lesire, and M. Sighireanu. Planning robust temporal plans: A comparison between CBTP and TGA approaches. In *ICAPS'07. Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, pages 2–10, 2007.
6. A. Albarghouthi, J. A. Baier, and S. A. McIlraith. On the use of planning technology for verification. In *VVPS'09. Proceedings of the ICAPS Workshop on Verification & Validation of Planning & Scheduling Systems*, 2009.
7. C. Artho, H. Barringer, A. Goldberg, K. Havelund, S. Khurshid, M. Lowry, C. Pasareanu, G. Rosu, K. Sen, W. Visser, and R. Washington. Combining test-case generation and runtime verification. *Theoretical Computer Science*, 336(2–3):209–234, May 2005.
8. C. Artho, K. Havelund, and A. Biere. High-level data races. *Software Testing, Verification and Reliability*, 13(4), 2004.
9. M. Bakera, S. Edelkamp, P. Kissmann, and C. D. Renner. Solving μ-calculus parity games by symbolic planning. In D. Peled and M. Wooldridge, editors, *MoChArt*, volume 5348 of *Lecture Notes in Computer Science*, pages 15–33. Springer, 2008.
10. H. Barringer, A. Groce, K. Havelund, and M. Smith. Formal analysis of log files. *Journal of Aerospace Computing, Information, and Communication*, 7(11):365–390, 2010.
11. H. Barringer, K. Havelund, E. Kurklu, and R. Morris. Checking flight rules with TraceContract: Application of a Scala DSL for trace analysis. In *Scala Days 2011, Stanford University, California*, 2011.
12. G. Behrmann, A. Cougnard, A. David, E. Fleury, K. Larsen, and D. Lime. UPPAAL-TIGA: Time for playing games! In *Proc. of 19th International Conference on Computer Aided Verification (CAV'07)*, number 4590 in LNCS, pages 121–125. Springer, 2007.
13. G. Behrmann, A. Fehnker, T. Hune, K. Larsen, P. Pettersson, and J. Romijn. *Efficient guiding towards cost-optimality in UPPAAL*. Springer, 2001.
14. G. Behrmann, K. G. Larsen, and J. I. Rasmussen. Optimal scheduling using priced timed automata. In *VVPS'05. Proceedings of the ICAPS Workshop on Verification & Validation of Model-Based Planning & Scheduling Systems*, 2005.
15. S. Bensalem, M. Bozga, M. Krichen, and S. Tripakis. Testing conformance of real-time applications: Case of planetary rover controller. In *VVPS'05. Proceedings of the ICAPS Workshop on Verification & Validation of Model-Based Planning & Scheduling Systems*, pages 23–32, 2005.
16. R. Bodik and B. Jobstmann. Algorithmic program synthesis: Introduction. *International Journal on Software Tools for Technology Transfer, STTT*, 15(5-6):397–411, October 2013.
17. M. Bozzano, A. Cimatti, M. Roveri, and A. Tchaltsev. A comprehensive approach to on-board autonomy verification and validation. In *VVPS'09. Proceedings of the ICAPS Workshop on Verification & Validation of Planning & Scheduling Systems*, 2009.
18. G. Brat, D. Drusinsky, D. Giannakopoulou, A. Goldberg, K. Havelund, M. Lowry, C. Pasareanu, W. Visser, and R. Washington. Experimental evaluation of verification and validation tools on Martian rover software. *Formal Methods in System Design*, 25(2), 2004.
19. G. Brat, D. Gannakopoulou, M. Izygon, E. Alex, L. Wang, J. Frank, and A. Molin. Model-based verification and validation for procedure authoring. In *VVPS'09. Proceedings of the ICAPS Workshop on Verification & Validation of Planning & Scheduling Systems*, 2009.
20. S. Cerrito and M. C. Mayer. Using linear temporal logic to model and solve planning problems. In *Artificial Intelligence: Methodology, Systems, and Applications*, pages 141–152. Springer, 1998.
21. A. Cesta, A. Finzi, S. Fratini, A. Orlandini, and E. Tronci. Verifying flexible timeline-based plans. In *VVPS'09. Proceedings of the ICAPS Workshop on Verification & Validation of Planning & Scheduling Systems*, 2009.
22. A. Cesta, A. Finzi, S. Fratini, A. Orlandini, and E. Tronci. Flexible plan verification: Feasibility results. *Fundamenta Informaticae*, 107:111–137, 2011.
23. B. Cichy, S. Chien, S. Schaffer, D. Tran, G. Rabideau, and R. Sherwood. Validating the autonomous EO-1 science agent. In *VVPS'05. Proceedings of the ICAPS Workshop on Verification & Validation of Model-Based Planning & Scheduling Systems*, pages 75–85, 2005.
24. A. Cimatti, F. Giunchiglia, E. Giunchiglia, and P. Traverso. Planning via model checking: A decision procedure for AR. In S. Steel and R. Alami, editors, *ECP*, volume 1348 of *Lecture Notes in Computer Science*, pages 130–142. Springer, 1997.
25. A. Cimatti, M. Roveri, and P. Traverso. Strong planning in non-deterministic domains via model checking. In R. G. Simmons, M. M. Veloso, and S. F. Smith, editors, *AIPS*, pages 36–43. AAAI, 1998.

26. J. Crampton, M. Huth, and J. H.-P. Kuo. Authorized workflow schemas : Deciding realizability through LTL(F) model checking. *International Journal on Software Tools for Technology Transfer, STTT*, in this issue, 2014.
27. H. Dierks. Finding optimal plans for domains with restricted continuous effects with UPPAAL-CORA. In *VVPS'05. Proceedings of the ICAPS Workshop on Verification & Validation of Model-Based Planning & Scheduling Systems*, 2005.
28. L. Dixon, A. Smaill, and A. Bundy. Verified planning by deductive synthesis in intuitionistic linear logic. In *VVPS'09. Proceedings of the ICAPS Workshop on Verification & Validation of Planning & Scheduling Systems*, 2009.
29. S. Edelkamp. Promela planning. In T. Ball and S. K. Rajamani, editors, *SPIN*, volume 2648 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 2003.
30. S. Edelkamp and S. Jabbar. Action planning for directed model checking of Petri nets. *Electr. Notes Theor. Comput. Sci.*, 149(2):3–18, 2006.
31. S. Edelkamp, S. Jabbar, and A. Lluch-Lafuente. Cost-algebraic heuristic search. In M. M. Veloso and S. Kambhampati, editors, *AAAI*, pages 1362–1367. AAAI Press / The MIT Press, 2005.
32. S. Edelkamp, M. Kellershoff, and D. Sulewski. Program model checking via action planning. In R. van der Meyden and J.-G. Smaus, editors, *MoChArt*, volume 6572 of *Lecture Notes in Computer Science*, pages 32–51. Springer, 2010.
33. M. Feather and B. Smith. Automatic generation of test oracles - from pilot studies to application. *Automated Software Engineering*, 8(1):31–61, 2001.
34. M. Fox, R. Howey, and D. Long. Exploration of the robustness of plans. In *VVPS'05. Proceedings of the ICAPS Workshop on Verification & Validation of Model-Based Planning & Scheduling Systems*, pages 67–74, 2005.
35. M. Fox, D. Long, L. Baldwin, G. Wilson, M. Woods, D. Jameux, and R. Aylett. On-board timeline validation and repair: A feasibility study. In *IWPSS'06. Proceedings of 5^{th} International Workshop on Planning and Scheduling for Space*, 2006.
36. B. Freitag, T. Margaria, and B. Steffen. A pragmatic approach to software synthesis. In J. M. Wing and R. L. Wexelblat, editors, *Workshop on Interface Definition Languages, Portland, Oregon, USA*, pages 46–58. ACM Press, 1994.
37. D. Giannakopoulou, C. S. Pasareanu, M. Lowry, and R. Washington. Lifecycle verification of the NASA Ames K9 rover executive. In *VVPS'05. Proceedings of the ICAPS Workshop on Verification & Validation of Model-Based Planning & Scheduling Systems*, pages 75–85, 2005.
38. F. Giunchiglia and P. Traverso. Planning as model checking. In S. Biundo and M. Fox, editors, *ECP*, volume 1809 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 1999.
39. A. Goldberg, K. Havelund, and C. McGann. Runtime verification for autonomous spacecraft software. In *Proceedings of IEEE Aerospace Conference*. IEEE Computer Society, 2005.
40. R. P. Goldman, U. Kuter, and A. Schneider. Using classical planners for plan verification and counterexample generation. In *Proceedings of AAAI Workshop on Problem Solving Using Classical Planning*, 2012.
41. R. P. Goldman, D. J. Musliner, , and M. J. Pelican. Exploiting implicit representations in timed automaton verification for controller synthesis. In *HSCC'02. Proceedings of the Fifth Int. Workshop on Hybrid Systems: Computation and Control*, 2002.
42. R. P. Goldman, M. J. Pelican, and D. J. Musliner. A loop acceleration technique to speed up verification of automatically-generated plans. *International Journal on Software Tools for Technology Transfer, STTT*, in this issue, 2014.
43. K. Havelund, A. Groce, G. Holzmann, R. Joshi, and M. Smith. Automated testing of planning models. In *Proceedings of the Fifth International Workshop on Model Checking and Artificial Intelligence*, pages 90–105, 2008.
44. K. Havelund, M. Lowry, S. Park, C. Pecheur, J. Penix, W. Visser, and J. L. White. Formal analysis of the Remote Agent - before and after flight. In *The Fifth NASA Langley Formal Methods Workshop, Virginia.*, 2001.
45. K. Havelund, M. Lowry, and J. Penix. Formal analysis of a spacecraft controller using SPIN. *IEEE Transactions on Software Engineering*, 27(8):749–765, 2001. An earlier version occurred in the Proc. 4th SPIN workshop, 1998.
46. R. Howey and D. Long. VAL's progress: The automatic validation tool for PDDL2.1 used in the international planning competition. In *Proceedings of the ICAPS Workshop on The Competition: Impact, Organization, Evaluation, Benchmarks*, pages 28–37, Trento, Italy, June 2003.
47. M. D. Johnston and D. Tran. Verification and validation of a deep space network scheduling application. In *VVPS'09. Proceedings of the ICAPS Workshop on Verification & Validation of Planning & Scheduling Systems*, 2009.
48. A. Jonsson, P. Morris, N. Muscettola, K. Rajan, and B. Smith. Planning in interplanetary space: Theory and practice. In *AIPS'00. Proceedings of the Fifth Int. Conf. on Artificial Intelligence Planning and Scheduling*, pages 177–186, 2000.
49. H. Kautz and B. Selman. BLACKBOX: A new approach to the application of theorem proving to problem solving. In *AIPS'98 Workshop on Planning as Combinatorial Search*, pages 58–60, 1998.
50. H. Kautz and B. Selman. Unifying sat-based and graph-based planning. In *Proc. of the International Joint Conference on Artificial Intelligence. (IJCAI-99)*, volume 99, pages 318–325, 1999.
51. H. A. Kautz, B. Selman, et al. Planning as satisfiability. In *Proc. of the International European Conference on Artificial Intelligence. (ECAI-92)*, volume 92, pages 359–363, 1992.
52. L. Khatib, N. Muscettola, and K. Havelund. Verification of plan models using UPPAAL. In *First International Workshop on Formal Approaches to Agent-Based Systems, NASA's Goddard Space center, Maryland*, volume 1871 of *Lecture Notes in Artificial Intelligence*. Springer, 2000.
53. L. Khatib, N. Muscettola, and K. Havelund. Mapping temporal planning constraints into timed automata. In *TIME'01. The Eigth Int. Symposium on Temporal Representation and Reasoning*, pages 21–27, 2001.

54. O. Kupferman and M. Y. Vardi. Safraless decision procedures. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05), Pittsburgh, PA*, pages 531–542, October 2005.
55. T. Lindsey and C. Pecheur. Simulation-based verification of autonomous controllers with Livingstone PathFinder. In *Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04), Barcelona, Spain*, volume 2988 of *Lecture Notes in Computer Science*, 2004.
56. D. Long, M. Fox, and R. Howey. Planning domains and plans: validation, verification and analysis. In *VVPS'09. Proceedings of the ICAPS Workshop on Verification & Validation of Planning & Scheduling Systems*, 2009.
57. M. R. Lowry, K. Havelund, and J. Penix. Verification and validation of AI systems that control deep-space spacecraft. In *Foundations of Intelligent Systems, 10th International Symposium, ISMIS'97, Charlotte, North Carolina, USA, October 15-18, 1997, Proceedings*, volume 1325 of *Lecture Notes in Computer Science*, pages 35–47. Springer, 1997.
58. M. C. Mayer, C. Limongelli, A. Orlandini, and V. Poggioni. Linear temporal logic as an executable semantics for planning languages. *Journal of Logic, Language and Information*, 16(1):63–89, 2007.
59. J. McCarthy and P. Hayes. *Some philosophical problems from the standpoint of artificial intelligence.* Stanford University, 1968.
60. D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the Planning Domain Definition Language. Technical Report CVC TR98003/DCS TR1165, New Haven, CT: Yale Center for Computational Vision and Control, 1998.
61. T. Menzies and C. Pecheur. Verification and validation and artificial intelligence. *Advances in Computers*, 65:5–45, 2005.
62. P. H. Morris and N. Muscettola. Temporal Dynamic Controllability Revisited. In *Proc. of the 20th National Conference on Artificial Intelligence (AAAI-05)*, 2005.
63. N. Muscettola. HSTS: Integrating planning and scheduling. In Zweben, M. and Fox, M.S., editor, *Intelligent Scheduling.* Morgan Kauffmann, 1994.
64. D. J. Musliner, M. J. S. Pelican, and P. J. Schlette. Verifying equivalence of procedures in different languages: preliminary results. In *VVPS'09. Proceedings of the ICAPS Workshop on Verification & Validation of Planning & Scheduling Systems*, 2009.
65. P. P. Nayak, D. E. Bernard, G. Dorais, E. B. Gamble, B. Kanefsky, J. Kurien, W. Millar, N. Muscettola, K. Rajan, N. Rouquette, B. D. Smith, and W. Taylor. Validating the DS1 Remote Agent experiment. In *iSAIRAS'99. Proceeedings Fifth Int. Symposium on Artificial Intelligence, Robotics and Automation in Space*, 1999.
66. A. Orlandini, A. Finzi, A. Cesta, and S. Fratini. TGA-based controllers for flexible plan execution. In *KI 2011: Advances in Artificial Intelligence, 34th Annual German Conference on AI.*, volume 7006 of *Lecture Notes in Computer Science*, pages 233–245. Springer, 2011.
67. P. Patron and A. Birch. Plan proximity: an enhanced metric for plan stability. In *VVPS'09. Proceedings of the ICAPS Workshop on Verification & Validation of Planning & Scheduling Systems*, 2009.
68. J. Penix, C. Pecheur, and K. Havelund. Using model checking to validate AI planner domain models. In *Proceedings of the 23rd Annual Software Engineering Workshop*, 1998.
69. N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive(1) designs. In *7th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'06)*, volume 3855 of *Lecture Notes in Computer Science*, pages 364–380. Springer, 2006.
70. A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE Computer Society, 1977.
71. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Symposium on Principles of Programming Languages (POPL'89)*, pages 179–190, 1989.
72. A. Preece. Evaluating verification and validation methods in knowledge engineering. In R. Roy, editor, *Micro-Level Knowledge Management*, pages 123–145. Morgan-Kaufman, 2001.
73. M. D. R-Moreno, G. Brat, N. Muscettola, and D. Rijsman. Validation of a multi-agent architecture for planning and execution. In *DX'07. Proceedings of 18th International Workshop on Principles of Diagnosis*, 2007.
74. F. Raimondi, C. Pecheur, and G. Brat. PDVer, a tool to verify PDDL planning domains. In *VVPS'09. Proceedings of the ICAPS Workshop on Verification & Validation of Planning & Scheduling Systems*, 2009.
75. N. Razavi, A. Farzan, and S. A. McIlraith. Generating effective tests for concurrent programs via AI automated planning techniques. *International Journal on Software Tools for Technology Transfer, STTT*, in this issue, 2014.
76. M. Shah, L. Chrpa, F. Jimoh, D. Kitchin, T. McCluskey, S. Parkinson, and M. Vallati. Knowledge engineering tools in planning: State-of-the-art and future challenges. In *Proc. of the ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS 2013)*, 2013.
77. R. I. Siminiceanu, R. W. Butler, and C. A. Munoz. Experimental evaluation of a planning language suitable for formal verification. In *Proceedings of the Fifth International Workshop on Model Checking and Artificial Intelligence*, pages 18–34, 2008.
78. B. Smith, M. Feather, and N. Muscettola. Challenges and methods in testing the Remote Agent planner. In *AIPS'00. Proceedings of the Fifth Int. Conf. on Artificial Intelligence Planning and Scheduling*, pages 254–263, 2000.
79. B. Smith, W. Millar, J. Dunphy, Y.-W. Tung, P. Nayak, E. Gamble, and M. Clark. Validation and verification of the Remote Agent for spacecraft autonomy. In *Proceedings of IEEE Aerospace Conference*, 1999.
80. B. Smith, K. Rajan, and N. Muscettola. Knowledge acquisition for the onboard planner of an autonomous spacecraft. In *EKAW'97. 10th European Workshop on Knowledge Acquisition, Modeling and Management*, volume 1319 of *Lecture Notes in Computer Science*, pages 253–268, 1997.
81. M. H. Smith, G. J. Holzmann, G. C. Cucullu, and B. D. Smith. Model checking autonomous planners: Even the best laid plans must be verified. In *Proceedings of IEEE Aerospace Conference*, pages 1 – 11. IEEE Computer Society, 2005.
82. S. Srivastava, N. Immerman, and S. Zilberstein. Finding plans with branches, loops and preconditions. In

VVPS'09. Proceedings of the ICAPS Workshop on Verification & Validation of Planning & Scheduling Systems, 2009.

83. B. Steffen, M. Isberner, S. Naujokat, T. Margaria, and M. Geske. Property-driven benchmark generation. In E. Bartocci and C. Ramakrishnan, editors, *Model Checking Software*, volume 7976 of *Lecture Notes in Computer Science*, pages 341–357. Springer Berlin Heidelberg, 2013.
84. B. Steffen, T. Margaria, and V. Braun. The electronic tool integration platform: concepts and design. *International Journal on Software Tools for Technology Transfer, STTT*, 1(1-2):9–30, 1997.
85. T. Vaquero, J. Silva, and J. Beck. A brief review of tools and methods for knowledge engineering for planning & scheduling. In *Proc. of the ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS 2011)*, 2011.
86. T. Vidal. A unified dynamic approach for dealing with temporal uncertainty and conditional planning. In *AIPS'00. Proceedings of the Fifth Int. Conf. on Artificial Intelligence Planning and Scheduling*, 2000.
87. M. Wehrle and M. Helmert. The causal graph revisited for directed model checking. In J. Palsberg and Z. Su, editors, *SAS*, volume 5673 of *Lecture Notes in Computer Science*, pages 86–101. Springer, 2009.
88. B. C. Williams and P. P. Nayak. A model-based approach to reactive self-configuring systems. *AAAI/IAAI*, 2:971–978, 1996.