J-TRE: An Integrated Reasoning Environment and its Application to Timeline-based Planning

Riccardo De Benedictis

CNR - National Research Council of Italy, ISTC, Rome, Italy
riccardo.debenedictis@istc.cnr.it

Abstract. The application of classical artificial intelligence techniques to realistic dynamic scenarios arises interesting challenges which require, often, the tailoring of state-of-the-art solvers to the specific domains. The introduction of a "light" and highly customizable framework allows to facilitate the way a high number of possible problems are addressed. This paper introduces the ongoing work for a novel domain-independent reasoning environment called J-TRE that takes its inspiration from Constraint Logic Programming flavoring it with Object Oriented features. The paper also addresses the customization of such an environment to a particular kind of automated planning, referred to as timeline-based, particularly suitable for complex domains. As a result, an alternative formalization of the timeline-based planning problem is proposed that allows for an interesting ability to solve both planning and scheduling problems in a uniform schema.

1 Introduction

Planning is the abstract and explicit deliberation process that chooses and organizes actions by anticipating their expected outcomes [24]. The area of Artificial Intelligence (AI) that studies this deliberation process computationally is called *automated planning*. Among the different qualities a cognitive architecture should have, automated planning constitutes a mandatory one.

This paper fits inside this context by introducing the ongoing work for a novel domain-independent reasoning environment, endowed with a special emphasis on automated planning, called Timeline Reasoning Environment for Java (J-TRE[1]). Specifically, J-TRE represents a general architecture for logic and constraint-based reasoning that brings together some key aspects of intelligent reasoning leaving freedom to specific implementations on both constraint reasoning engines and resolution heuristics. The proposed architecture is characterized by a basic core that allows the definition of classic AI problems providing the possibility for defining particular "components" devoted to the resolution of specific AI problems (in our case, timeline-based planning problems).

The paper is structured as follow. Section 2 describes the J-TRE reasoning environment with a special attention to its basic core. Section 3 describes how the basic

[1]The J-TRE environment is the tool that has been developed on purpose for performing research along my PhD route.

core and the general solving procedure have been extended to provide timeline-based planning capabilities. Section 4 contains a generic discussion about some related approaches while section 5 contains conclusions and a description of the current state of the environment.

2 The J-TRE Reasoning Environment

J-TRE is the result of last years studies on timeline-based planning [14, 15] and its application on field [4]. The reason for introducing a new architecture is to obtain a "light" and general framework for the resolution of classic AI problems, which might provide a common glue for the definition of different problems, coming from different AI fields, yet sharing constraints among them. A complex cognitive architecture, for example, might handle both temporal and spatial problems. Although there exist efficient temporal reasoners and spatial reasoners, the definition of constraints among these problems might be cumbersome and highly inefficient to be managed.

The basic core of the J-TRE environment provides an object oriented virtual environment for the definition of *objects* and *constraints* among them. The aim here is to provide to the user a minimalistic core that should be both sufficiently expressive as well as easily extensible so as to adapt as much as possible to the most variety of user requirements.

Similar to most object oriented environments, every object in the J-TRE environment is an instance of a specific *type*. J-TRE distinguishes among *primitive types* (e.g., bools, numbers, enums, strings, etc.) and user defined *complex types* (e.g., robots, trucks, locations, etc.) endowed with their *member variables* (variables associated to a specific object of either primitive or complex type), *constructors* (a special type of subroutine called to create an instance of the complex type) and *methods* (subroutines associated with an object of a complex type). Defining a navigation problem, for example, might require the definition of a *Location* complex type having two numeric member variables x and y representing the coordinates of each *Location* instance. In the following, we will address objects and their member variables using a Java style *dot* notation (i.e., given a *Location* instance l, its x-coordinate will be expressed as $l.x$).

Once objects are defined, J-TRE allows the definition of constraints among them. For example, in case the x-coordinate of a location l is partially known, the J-TRE user could assert a constraint such $l.x \leq 10$. J-TRE considers constraints as logic propositions and, as such, it allows the possibility for negating them (e.g., $\neg l.x \leq 5$), for expressing conjunctions (e.g., $l.x \leq 10 \wedge l.x \geq 5$), disjunctions (e.g., $l.x \leq 5 \vee l.x \geq 10$) and logic implications (e.g., $l.x \geq 10 \rightarrow l.y \geq 10$). Furthermore, it is possible to impose constraints on instance existential (e.g., $\exists l \in Locations : l.x \geq 10$) and universal (e.g., $\forall l \in Locations : l.x \leq 100$) quantifiers. It is worth noting that, in case the defined constraints are inconsistent among themselves (e.g., $l.x \leq 10 \wedge l.x \geq 15$) the system will return a failure.

A rather straightforward method for managing this kind of problems is to translate them into a Satisfiability Modulo Theories (SMT) problem (see, for example, [29]). There are several available SMT solvers having different performances, capabilities as well as licenses. Since J-TRE has been written in Java the only available choices are, to the best of our knowledge, the SMTInterpol [9], the MathSAT 5 [11] and the Z3 [16] solvers[2].

Although this basic core allows the definition of quite complex problems (without providing any demonstration, we can state that all NP-Complete [22] problems are covered), some of the planning problems we are interested in are excluded from the possibility of being modelled with this formalism ([13], for example, traces a threshold on planning problems that can be modelled with what we have shown till now). In order to overcome these limitations, we need something more powerful. Something that, roughly speaking, is able to "decide" the number of involved variables, together with their value. For this purpose, we have chosen to endow complex types the possibility to define *predicate schemas* over them (and, consequently, predicate instances on type instances). Intuitively, these predicate schemas are intended to define in which case their instances are logically true. An example of predicate schema that might be used for describing locations in the first quadrant of a Cartesian coordinate system is $FirstQuadrant\,(Location\; l) \Rightarrow \{l.x \geq 0 \wedge l.y \geq 0\}$. By creating a *FirstQuadrant* "query", J-TRE will create a new *FirstQuadrant* predicate instance and will assign a location to its l variable such that its coordinates belong to the first quadrant of the Cartesian coordinate system. If such location does not exists, the system will return a failure.

Similarly to standard objects, J-TRE allows the possibility to impose constraints on predicate instances as well. Moreover, it is worth to note that predicate schemas might include other predicate "queries" with a different predicate schema, possibly defined on other complex types, or even the same predicate schema (in the latter case we talk of a recursive definition of the predicate schema)[3]. In both cases the system will require some form of subgoaling during the reasoning process. Finally, when possible, two predicate instances might be "unified" by the reasoner meaning that, from now on, the two predicates will be considered as a unique object (i.e., the reasoner puts bindings among their variables).

In the following we provide some formal definition of the concept informally introduced till now.

Definition 1. *Given n complex types* $\mathscr{T} = \{\mathbb{T}_0, \dots, \mathbb{T}_n\}$*, a* domain theory $\mathscr{D} = \{c_0, \dots, c_m\}$ *is the collection of the predicate schemas defined over* $\mathscr{T}$*. A* predicate schema *is a tuple* $c = (name\,(c)\,, R\,(c))$*, where:*

- $name\,(c)$ *is the* master *(or* reference*) predicate and is an expression of the form* $n\,(x_0 \dots x_k)$*, where n is a unique predicate symbol with respect to a complex type (i.e., no two predicate schemas in a given complex type can have the same predicate symbol), and* $x_0 \dots x_k$ *are its associated variable symbols having either primitive types or complex types (in the latter case the variable will be considered as an existential quantifier).*

[2]While SMTInterpol provides a pure Java implementation, MathSAT and Z3 provide Java wrappers to their native API. We have not found other SMT solvers that provide, directly or indirectly, a Java API.

[3]Recursive definitions produce a remarkably increase in expressiveness with the predictable side effect of a decrease in performances.

– *$R(c)$ is a* requirement*, i.e. either a* slave *(or* target*) predicate, a constraint, a conjunction of requirements, a disjunction of requirements or a preferred requirement.*

In this regard, it is worth to recall that constraints can be combined in any logical combination (i.e., negations, disjunctions, etc.).

Definition 2. *A* predicate instance *is an expression of the form:*

$$n(x_0,\ldots,x_k)$$

where n is a predicate name and $x_0,\ldots,x_k$ are constants, numeric variables or object variables.

The set of objects, predicate instances and constraints among objects and/or predicate instances is called a *partial solution* and, as we shall soon, will be used to represent nodes of the J-TRE search space. More formally we have that:

Definition 3. *A* partial solution *is a tuple $\pi = (O,T,C)$, where:*

– *$O = \{o_0,\ldots,o_i\}$ is a set of objects.*
– *$T = \{t_0,\ldots,t_j\}$ is a set of predicate instances.*
– *C is a set of constraints on objects and/or on variables of predicate instances in T.*

C is required to be consistent, i.e., there exist values for all the variables that meet all the constraints. Moreover, since the real world has high degrees of uncertainty, we might be interested in maintaining different possible solutions in a single partial solution.

Finally, given a domain theory $\mathscr{D}$, a set of objects O, we can reuse the concept of requirement for defining a J-TRE *problem*, i.e. either a slave (or, in the case of a problem, a *goal*) predicate, a constraint, a conjunction of requirements, a disjunction of requirements or a preferred requirement.

Definition 4. *A* problem *is a tuple $P = (\mathscr{D},O,R)$, where:*

– *$\mathscr{D}$ is a domain theory.*
– *$O = \{o_0,\ldots,o_k\}$ is a set of objects.*
– *R is a requirement.*

From an operational point of view, J-TRE uses an adaptation of the *resolution principle* [28] extended for managing constraints in a more general scheme usually known as *constraint logic programming* (CLP) [1]. More in general, we give the name *flaw* to anything that keeps the current partial solution from being a solution. While flaws can be of different types and can arise for different reasons, what they all have in common is that a search choice is necessary to solve each of them, thus the basic operations for refining a partial solution π toward a solution are the following:

1. find the flaws of π, i.e., its subgoals.
2. select one such flaw.
3. find ways to resolve it.
4. choose a resolver for the flaw.

5. refine π according to that resolver.

Starting from an initial node corresponding to an empty solution, the search aims at a final node containing a solution that correctly achieves the required goals, i.e, when there is no flaw in π and all of its constraints are consistent then π is a solution. When possible, resolution will try to unify goals with existing predicate instances, creating a branch for all possible unifications. In case unification is not possible, resolution will create a new branch of the search space, will create a new predicate instance, will find a predicate schema suitable for the instance and will apply it.

How can such a generic framework be related with automated planning? Is it possible to extend the framework to manage other forms of reasoning? In the following, after some basic introductory information on classical and timeline-based planning, we will try to convince the reader that it is possible to give an affirmative answer to these questions.

3 Timeline-based Planning and J-TRE

Classical planning[4] approaches the planning problem as the search for a path in the graph of a state-transition system [17], thus, the search space is given directly by the state-transition system: nodes are states of the domain; arcs are state transitions or actions; a plan is a sequence of actions corresponding to a path from the initial state to a goal state.

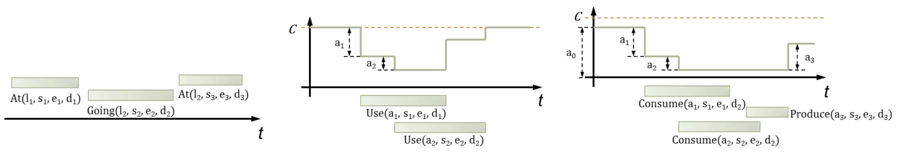
The timeline-based approach to planning represents an effective alternative to classical planning for complex domains requiring the use of both temporal reasoning and scheduling features. The timeline-based approach models the planning and scheduling problem by identifying a set of relevant *features* of the planning domain which need to be controlled to obtain a desired temporal behaviour. Timelines model entities whose properties may vary in time and which represent one or more physical (or logical) subsystems which are relevant to a given planning context. The planner/scheduler plays the role of the controller for these entities, and reasons in terms of constraints that bound their internal evolutions and the desired properties of the generated behaviours.

Most of the timeline-based planners use a more elaborate search space that is not the state-transition system anymore. Their search space has *partially specified plans* as nodes and *plan refinement operations* as arcs. Plan refinement operations are intended to further complete a partial solution, i.e., to achieve an open goal or to remove some possible inconsistency. Intuitively, these refinement operations avoid adding to the partial plan any constraint that is not strictly needed for addressing the refinement purpose (this is called the *least commitment principle*). The solving procedure starts from an initial node corresponding to an empty solution. The search aims at a final node containing a solution that correctly achieves the required goals.

Given these premises, a possible approach to the resolution of timeline-based planning problems is to provide the predicate instances introduced in the previous section with numerical variables in order to represent their starting times, their ending times

[4]Classical planning is also referred to in the literature as *STRIPS planning*, in reference to STRIPS, an early planner for restricted state-transition systems [18].

and their durations. Also, it will be required to define some specific complex types, whose instances will be called *timelines*, in order to add further "implicit" constraints on the predicate instances defined over their instances. Finally, we need to enhance the resolution procedure in order to check the consistency for every object in the current partial solution so as to make explicit the just mentioned implicit constraints.



(a) *State variable* (b) *Reusable resource* (c) *Consumable resource*

Fig. 1. Different kinds of timelines with predicate instances and resource profiles.

The minimal set of the complex types, commonly used in timeline-based planning, is the following:

- *State variables* are used to describe the "state" of a dynamical system as, for example, the position of a specific object at a given time or a simple manufacturing tool that might be operating or not. The semantics of a state variable (and thus the implicit constraints on predicate instances associated to it) is simply that, for each time instant $t \in \mathbb{T}$, the timeline can assume only one value. State variable predicate schemas are defined by the user during domain definition (temporal parameters are implicitly defined). Figure 1(a) represents an example of state variable with three predicate instances.
- *Resources* are characterized by a *resource level* $\mathscr{L} : \mathbb{T} \rightarrow \mathbb{R}$, representing the amount of available resource at any given time, and by a *resource capacity* $\mathscr{C} \in \mathbb{R}$, representing the physical limit of the available resource. We can identify several types of resources depending on how the resource level can be increased or decreased in time. A *consumable resource* is a resource whose level is increased or decreased by some activities in the system. An example of consumable resource is a reservoir which is produced when a plan activity "fills" it (i.e., a tank refueling task) as well as consumed if a plan activity "empties" it (i.e., driving a car uses gas). Consumable resources have two empty predicate schemas having a predicate $Produce\,(a, s, e, d)$ to represent a resource production of amount a from time s to time e with duration d, and a predicate $Consume\,(a, s, e, d)$ to represent a resource consumption of amount a from time s to time e with duration d. In addition, consumable resources have four member variables representing the initial and the final amount of the resource, the min and the max value for the profile. *Reusable resources* have one empty predicate schema having a predicate $Use\,(a, s, e, d)$ that represent an instantaneous production of resource of amount a at time s and an instantaneous consumption of resource of the same amount a at time e (e.g., a given number of programmers employed on a given project for a given time interval). Reusable resources have a single member variable representing the capacity of the resource. Figures 1(c) and 1(b) represent, respectively, an example of consumable resource and an example of reusable resource with some associated predicate instances.

By introducing these complex types, we require the reasoner to identify an ordering of the involved predicate instances in order to avoid object inconsistencies (e.g., different states overlapping on some state variable; resource levels $\mathscr{L}$ exceeding resource capacity $\mathscr{C}$ or going lower than zero, etc.).

The aim of the reasoner becomes now to find a legal state of the predicate instances that brings the objects into a final configuration that verifies domain theory, goals and implicit constraints. Nodes of the search space are still partially specified solutions while arcs are problem refinement operations intended to further complete a partial solution, i.e., to achieve an open goal (by applying the "consequences" of the goal as defined in the associated predicate schema), to remove a possible object inconsistency (e.g., by adding some further constraint), to chose a disjunct among a disjunction or to chose whether to satisfy a preference or not. We just need to redefine the concept of flaw so as to include the just introduced object inconsistencies.

Definition 5. *A* flaw *in a partial solution $\pi = (O, T, C)$ is either: (i) an open goal, (ii) an object inconsistency, (iii) a disjunction or (iv) a preference.*

As expected, the refinement operations avoid adding to the partial solutions constraints that are not strictly needed following preserving the least commitment principle. Starting from an initial node corresponding to an empty solution, the search aims at a final node containing a solution that correctly achieves the required goals. This algorithm (specified as a recursive non-deterministic schema in Figure 2) represents a slight adaptation of the classical Partial Order Planning algorithm (see, for example, [30]).

```
procedure SOLVE($\pi$)
    $flaws \leftarrow$ OpenGoals $(\pi) \cup$ Inconsistencies $(\pi) \cup$ Disjs $(\pi) \cup$ Prefs $(\pi)$
    if $flaws = \emptyset$ then return $\pi$
    select any flaw $\phi \in flaws$
    $resolvers \leftarrow$ Resolve $(\phi, \pi)$
    if $resolvers = \emptyset$ then return $\perp$
    non-deterministically choose a resolver $\rho \in resolvers$
    $\pi' \leftarrow$ Refine $(\rho, \pi)$
    return Solve $(\pi')$
```

Fig. 2. The J-TRE solving procedure.

4 General Discussion

There is plenty of related literature which achieve different results from those presented in this paper. The concept of predicate schema has its roots in the research on *automatic theorem proving*. Exception made for mathematical constraints, predicate schemas are very similar to standard Logic Programming (LP) clauses. Although standard Prolog implementations allow the imposition of mathematical constraints, imposing such constraints in a general manner would possibly lead to run-time errors since arithmetic tests can be made only if the arguments are instantiated. Constraint Logic Programming overcomes this limitation by extending LP so as to allow linear constraints over real variables CLP(R) (with namesake implementation [25]) and among variables ranging over finite domains CLP(FD) (like ECL^{i}PS^e [1], SWI-Prolog [31] and many others).

Many results have been achieved also with regard to the timeline-based planning. Similar to what has been described in this paper, [10] proposes a theoretical formalism (and the relative implementation) for the timeline-based planning problem. [12] exploits SMT technology to address a similar problem, also taking into account temporal uncertainty. Some theoretical work on timeline-based planning like [20] was mostly dedicated to explain details of EUROPA [26], or to identify connections with classical planning a-la PDDL [19]. The work on ASPEN [8], IXTET [23] and TRF [21, 5] have tried to clarify some key underlying principles but mostly succeeded in underscoring the role of time and resource reasoning [7, 27].

The object oriented environment, provided by J-TRE, allows an easy customization for specific problems, such those required for timeline-based planning, yet maintaining a shared glue, made by a uniform resolution procedure and a unified constraint propagation infrastructure, among the different problems. This flexibility could be highly appreciated when using the tool in real contexts.

The search control part has always remained significantly under explored. The current realm is that although these planners capture elements that are very relevant for applications, their theories are often quite challenging from a computational point of view and their performance are rather weak compared with those of state of the art classical planners. Indeed, timeline-based planners are mostly based on the notion of partial order planning [30] and have almost neglected advantages in classical planning triggered from the use of GRAPHPLAN and/or modern heuristic search [2, 3]. Furthermore, these architectures rely on a clear distinction between temporal reasoning and other forms of constraint reasoning and there is no sign of attempts to change.

Constraint disjunctions, provided by the underlying SMT solver, should be exploited as much as possible. When available, for example, the J-TRE solver creates a disjunction of all possible unifications for a goal. Temporal mutual exclusion for state variable predicate instances, as well as overproductions and overconsumptions of resources, could be handled, without resorting to the concept of flaw, by imposing temporal disjunctions. The reason for preferring disjunctions to flaws is that SMT solvers have their own internal search space that is highly optimized thanks to state-of-the-art techniques such as backjumping and nogood learning. It is not yet clear if (and how) backjumping and nogood learning techniques could be applied to the J-TRE search procedure.

5 Conclusions and current status

This paper presents the ongoing work on J-TRE, a novel domain-independent reasoning environment, and its particular customization to a timeline-based planning domain. We have proposed a constraint-logic based formalism for the J-TRE reasoner and we have seen that it can be easily extended to the timeline-based case.

Despite the practical use of timeline-based planning is often more intuitive than classical planning, these kind of planners have to cope with performance issues due to the complexity that derives from their expressiveness. The use of Satisfiability Modulo Theories (SMT) for performing constraint propagation and for managing disjunctions has proved to be effective for solving these kinds of problems. We are currently working on the development of benchmark problems for the proposed formalism as well as translators from other planning languages such as PDDL2.1 [19] and later versions. Despite the reasoner has not been extensively tested yet, [6] contains some promising initial results. Finally, unlike [12] we do not (yet) account uncertainty in our domains.

Thanks to its domain causality and its constraint reasoning capabilities, J-TRE has been successfully used as the back-end reasoner for the PANDORA European project to create, reproduce and adapt a crisis scenario simulation suitable for the psychophysiological state of the involved users (see [4] for further details). Since the complexity of the possible interactions of an intelligent system with a real environment might be overly complex to be encoded in predicate schemas, inside the PANDORA project we have supported J-TRE with machine learning techniques so as to achieve an high level interpretation of psychophysiological features and user classification. A tighter integration of these technologies might be further investigated in order to reach a more complex cognitive architecture that might be able to perform complex tasks in a dynamic environment.

Acknowledgements I am thankful to the Planning and Scheduling Technology Laboratory (PST) and especially its coordinator Amedeo Cesta. My studies are partially supported by EU under the GiraffPlus Project (contract #288173. FP7 - ICT - Challenge 5) and by MIUR/CNR under Flagship Initiative FdF-SP1-T2.1 - GECKO Project.

References

1. Apt, K.R., Wallace, M.G.: Constraint Logic Programming Using ECL^{i}PS^e. Cambridge University Press, New York, NY, USA (2007)
2. Blum, A., Furst, M.L.: Fast Planning Through Planning Graph Analysis. In: IJCAI. pp. 1636–1642. Morgan Kaufmann (1995)
3. Bonet, B., Geffner, H.: Planning as Heuristic Search. Artificial Intelligence 129(12), 5–33 (2001)
4. Cesta, A., Cortellessa, G., Benedictis, R.D.: Training for Crisis Decision Making - An Approach Based on Plan Adaptation. Knowledge-Based Systems 58, 98–112 (2014)
5. Cesta, A., Cortellessa, G., Fratini, S., Oddi, A.: Developing an End-to-End Planning Application from a Timeline Representation Framework. In: IAAI-09. Proceedings of the 21st Innovative Applications of Artificial Intelligence Conference, Pasadena, CA, USA (2009)
6. Cesta, A., De Benedictis, R., Orlandini, A., Umbrico, A., Bernardi, G.: Integrating Planning and Scheduling Capabilities in a Space Robotics Domain. In: ASTRA 2013 - 12th Symposium on Advanced Space Technologies in Robotics and Automation. vol. 12. ESA (2013)
7. Cesta, A., Oddi, A.: Gaining Efficiency and Flexibility in the Simple Temporal Problem. In: Chittaro, L., Goodwin, S., Hamilton, H., Montanari, A. (eds.) Proceedings of the Third International Workshop on Temporal Representation and Reasoning (TIME-96). IEEE Computer Society Press: Los Alamitos, CA (1996)
8. Chien, S., Tran, D., Rabideau, G., Schaffer, S., Mandl, D., Frye, S.: Timeline-Based Space Operations Scheduling with External Constraints. In: ICAPS-10. Proc. of the 20th Int. Conf. on Automated Planning and Scheduling (2010)
9. Christ, J., Hoenicke, J., Nutz, A.: Smtinterpol: An interpolating SMT solver. In: Model Checking Software - 19th International Workshop, SPIN 2012, Oxford, UK, July 23-24, 2012. Proceedings. pp. 248–254 (2012)

10. Cialdea Mayer, M., Umbrico, A., Orlandini, A.: A Formal Account of Planning with Flexible Timelines. In: 21st International Symposium on Temporal Representation and Reasoning (TIME 2014) (2014)
11. Cimatti, A., Griggio, A., Schaafsma, B., Sebastiani, R.: The MathSAT5 SMT Solver. In: Piterman, N., Smolka, S. (eds.) Proceedings of TACAS. LNCS, vol. 7795. Springer (2013)
12. Cimatti, A., Micheli, A., Roveri, M.: Timelines with Temporal Uncertainty. In: AAAI (2013)
13. Cushing, W., Kambhampati, S., Mausam, Weld, D.S.: When is temporal planning really temporal? In: Proceedings of the 20th International Joint Conference on Artifical Intelligence. pp. 1852–1859. IJCAI'07, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2007)
14. De Benedictis, R., Cesta, A.: New Reasoning for Timeline based Planning - An Introduction to J-TRE and its Features. In: ICAART 2012 - 4th International Conference on Agents and Artificial Intelligence. pp. 144–153. SciTePress (2012)
15. De Benedictis, R., Cesta, A.: Timeline Planning in the J-TRE Environment. In: Felipe, J., Fred, A. (eds.) Agents and Artificial Intelligence. ICAART 2012 Revised Selected Papers, vol. 358, p. 218233. Springer Berlin Heidelberg (2013)
16. De Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. In: Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. TACAS'08/ETAPS'08, Springer-Verlag, Berlin, Heidelberg (2008)
17. Dean, T.L., Wellman, M.P.: Planning and Control. Morgan Kaufmann Publishers Inc. (1991)
18. Fikes, R., Nilsson, N.J.: STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. In: IJCAI. pp. 608–620 (1971)
19. Fox, M., Long, D.: PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. Journal of Artificial Intelligence Research 20, 61–124 (2003)
20. Frank, J., Jónsson, A.K.: Constraint-Based Attribute and Interval Planning. Constraints 8(4), 339–364 (2003)
21. Fratini, S., Pecora, F., Cesta, A.: Unifying Planning and Scheduling as Timelines in a Component-Based Perspective. Archives of Control Sciences 18(2), 231–271 (2008)
22. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA (1990)
23. Ghallab, M., Laruelle, H.: Representation and Control in IxTeT, a Temporal Planner. In: AIPS-94. Proceedings of the 2nd Int. Conf. on AI Planning and Scheduling. pp. 61–67 (1994)
24. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Morgan Kaufmann Publishers Inc. (2004)
25. Jaffar, J., Michaylov, S., Stuckey, P.J., Yap, R.H.C.: The CLP(R) Language and System. ACM Transactions on Programming Languages and Systems 14(3), 339–395 (May 1992), http://doi.acm.org/10.1145/129393.129398
26. Jonsson, A., Morris, P., Muscettola, N., Rajan, K., Smith, B.: Planning in Interplanetary Space: Theory and Practice. In: AIPS-00. Proceedings of the Fifth Int. Conf. on AI Planning and Scheduling (2000)
27. Laborie, P.: Algorithms for propagating resource constraints in AI planning and scheduling: existing approaches and new results. Artificial Intelligence 143, 151–188 (February 2003)
28. Robinson, J.A.: A Machine-Oriented Logic Based on the Resolution Principle. Journal of the Association for Computing Machinery 12(1), 23–41 (1965)
29. Sebastiani, R.: Lazy Satisability Modulo Theories. JSAT 3, 141–224 (2007)
30. Weld, D.S.: An Introduction to Least Commitment Planning. AI Magazine 15(4), 27–61 (1994)
31. Wielemaker, J., Schrijvers, T., Triska, M., Lager, T.: SWI-Prolog. Theory and Practice of Logic Programming 12(1-2), 67–96 (2012)