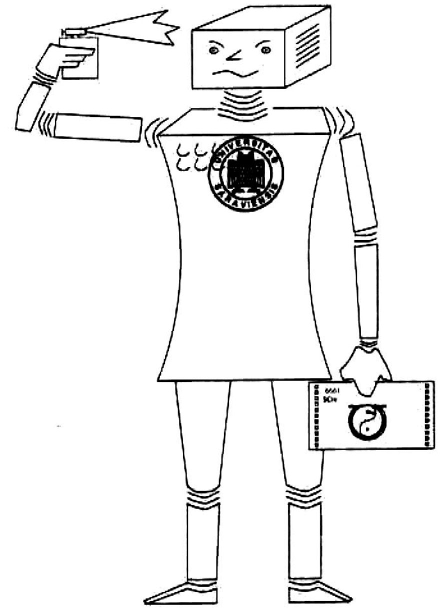
UNIVERSITÄT DES SAARLANDES
FACHBEREICH INFORMATIK
D-66041 SAARBRÜCKEN
GERMANY
WWW: http://www.ags.uni-sb.de/

SEKI Report
ISSN 1437-4447



Towards Extending Domain Representations

Andreas Meier, Erica Melis, Martin Pollet

SEKI Report SR-02-01

Towards Extending Domain Representations

Andreas Meier, Erica Melis,Martin Pollet
Universität des Saarlandes, FR Informatik
66041 Saarbrücken, Germany
email:{ameier,melis,pollet}@ags.uni-sb.de

Abstract

Based on our experience in developing and employing a rich domain representation for proof planning we compare the knowledge representation of the proof planner ΩMEGA with PDDL and with some AI planners and explore the usefulness of some extensions for AI-planning more generally. In particular, we investigate the more expressive syntax and semantics of proof planning due to the introduction of functions, full quantification, integration of constraint solving, powerful filters, and context-dependency as well as knowledge implicit in integrated systems.

1 Introduction

The emphasis of the discussions about an extension of the knowledge representation in planning has been on conditional effects, resources, as well as time representation and on representations which trade expressiveness for speed. From our experience in proof planning, we suggest other fundamental extensions of the domain representation.

The extensions that are needed for proof planning differ from other approaches, because proof planning is "simple" in the sense that it is planning in deterministic and fully observable domains. Moreover, since proof planning operators represent (complex) inference actions, there is (almost) no goal interaction because a sequence of inference actions does not destroy any object-level preconditions. Nevertheless, a comparison with the proof planning representations and mechanisms makes sense for the AI-planning community because many characteristics of proof planning domains are common to realistic world domains as well. In this sense, proof planning is "difficult".

This paper compares the knowledge representation of the proof planner ΩMEGA with PDDL (Planning Domain Definition Language) [7] and with planners that already have extended representations. Some of the expressive features in proof planning are not specific for proof planning domains and could be adopted by PDDL extensions. The paper focuses on the introduction of functions, full quantification, integration of constraint solving, powerful filters, and context-dependency as well as knowledge implicit in integrated systems.

2 Proof Planning with ΩMEGA

Proof planning [1, 10] considers mathematical theorem proving as planning problem. In proof planning, the initial state is specified by the proof assumptions, the goal is specified by the theorem to be proved.

Domain Language As opposed to PDDL and the problem representations in almost all planners, ΩMEGA uses a typed higher-order logic for the problem representation [3]. This includes universal and existential quantification as well as functions and quantification over functions and relations. The type system contains primitive types, e.g., type for individuals, numbers, and truth values, as well as functional types. Thus, in ΩMEGA the initial facts and the goal are given as higher-order formulas.

For instance, the problem LIM+ has the assumptions $\lim_{x \to a} f(x) = l_1$ and $\lim_{x \to a} g(x) = l_2$ and the goal $\lim_{x \to a}(f(x) + g(x)) = l_1 + l_2$ (the limit of the sum of two functions equals the sum of their limits) or in its expanded version:

$$\text{assumption}_1\colon \forall \varepsilon_1(0 < \varepsilon_1 \to \exists \delta_1(0 < \delta_1 \land \forall x_1(x_1 \neq a \land |x_1 - a| < \delta_1 \to |f(x_1) - l_1| < \varepsilon_1)))$$

$$\text{assumption}_2\colon \forall \varepsilon_2(0 < \varepsilon_2 \to \exists \delta_2(0 < \delta_2 \land \forall x_2(x_2 \neq a \land |x_2 - a| < \delta_2 \to |g(x_2) - l_2| < \varepsilon_2)))$$

$$\text{goal}\colon \forall \varepsilon(0 < \varepsilon \to \exists \delta(0 < \delta \land \forall x(x \neq a \land |x - a| < \delta \to |(f(x) + g(x)) - (l_1 + l_2)| < \varepsilon)))$$

Operator Language Planning operators can encode general and domain specific proof steps. For instance, operators for proof planning in the *limit domain* [8] encode steps for the estimation of inequalities as well as steps decomposing logical formulas.

An operator in ΩMEGA has the slots *parameters*, *declarations*, *premises*, *conclusions*, *application conditions*, *outline computations*, and *proof schema*. The premises and conclusions specify the object-level preconditions and the object-level effects of the operator. The conclusions should be logically inferable from the preconditions. The proof schema provides the schematic or procedural expansion functionality of HTN-operators and partially describes the involved formulas.

Premises and conclusions may be annotated with $\oplus$ and $\ominus$. The annotations are needed to indicate whether an operator is used in forward or backward search. They do not matter anymore in the final plan and its execution. $\ominus$ conclusions are goals that will be deleted (in backward planning), $\oplus$ conclusions are introduced as new assumptions into the planning state (in forward planning), $\oplus$ premises indicate new subgoals that are introduced by the operator, $\ominus$ premises are assumptions that have to be in the current planning state and are deleted by the operator, and finally blank premises are assumptions that (just) have to be in the current planning state.

Consider the operator =SUBST-B given in Figure 1. It has two preconditions L_1 and L_2, where the proof schema determines L_1 to be an equation. =SUBST-B is a backward planning operator which will close the goal L_3, and introduce the new goal L_2. The new goal will be the formula f' which results from f by substitution with the equation of L_1. This equation has to be an assumption in the planning state, i.e., an initial assumption or an inferred assumption. For instance, =SUBST-B applied to the goal $even(a+1)$ with the assumption $a = 1$ would introduce $even(1+1)$ as a new goal.

declarations introduces the variables of the operator and their types. All object-variables are typed. The type o is the type of formulas; objects with type *position* specify subterms of formulas.

parameters are specific variables that influence the behavior, when the operator is introduced into the plan. The =SUBST-B operator has the parameter *pos* which is of type *position*. The operator can be applied to different positions, e.g., for the goal $even(a+a)$ and the assumption $a = 1$ at the first or the second occurrence of a. The choice of *pos* determines which a should be replaced.

The *application conditions* are meta-level conditions that further restrict the applicability of an operator. The application conditions can be arbitrary LISP functions. The

Operator: =SUBST-B	
declarations	type-variables: α object-variables: $f{:}o$, $f'{:}o$, $t{:}\alpha$, $t'{:}\alpha$, $pos{:}position$
parameters	*pos*
appl. conds.	(1) (valid-position-p f *pos*) (2) [(term-at-position f *pos*) $= t \vee$ (term-at-position f *pos*) $= t'$]
premises	$\oplus L_2$, L_1
conclusions	$\ominus L_3$
outline computations	$f' \leftarrow$ (replace-at-position f t t' *pos*)
proof schema	L_1: $\Delta \vdash t = t'$ L_2: $\Delta \vdash f'$ (Open) L_3: $\Delta \vdash f$ (=*subst pos* L_1 L_2)

Figure 1: The =SUBST-B operator.

operator =SUBST-B has two application conditions: (1) the position *pos* has to be a valid position in the formula f and (2) the subterm in f at the position *pos* is t or t'.

The *outline computations* allow to apply arbitrary LISP functions to compute the new formulas generated by the operator. The outline computation of =SUBST-B specifies that the new formula f' is computed from f by replacing t by t' or t' by t at the position *pos* depending on whether the subterm in f at position *pos* is t or t'.

The applicability of an operator is checked as follows: First, the formulas of the conclusions and premises are unified with formulas in goals and assumptions. If this succeeds then the application conditions are evaluated. If they evaluate to true, the operator is applicable. The outline computations are performed when the operator is introduced into the plan.

Planning Algorithm ΩMEGA performs state-based forward and backward planning with a goal agenda. The planning state consists of assumptions and (sub)goals. The planner searches for a solution, i.e., for a sequence of instantiated operators whose application infers the goal from the initial state. Similar to HTN planning [15], planning operators can be expanded. The recursive expansion of a plan yields a calculus-level natural deduction proof [5] that can be executed which means can be checked for logical correctness. That is, the plan execution corresponds to a soundness check which is performed at the calculus-level.

Why is proof planning difficult?

- The objects in proof planning domains are (mathematical) objects such as numbers, sets, or morphisms. In many domains, infinitely many mathematical objects exist, e.g., infinitely many real numbers.
- The objects can be represented by functional *terms* and such objects can be generated during planning, i.e., objects that are constructed from other objects.
- Objects may be constrained by domain properties different from (in)equalities.
- Actions change the planning state that consists of formulas (rather than just literals) describing objects and their relations and functions.

- Since the formulas can be higher-order, higher-order-unification with goals is required in the planning algorithm. General higher-order-unification is undecidable.
- Forward and backward planning is usually required in proof planning in order to infer new mathematical facts from already known facts and to reduce goals to subgoals.

3 Expressivity of the Language

In this section, we compare some features of ΩMEGA's rich domain and operator language with PDDL and some AI-planners and explore the usefulness of our extensions for AI-planning more generally.

3.1 Functions

PDDL excludes functions explicitly. PDDL 2.1 level 2 [7] allows restricted functors for assignments of numerical values, e.g. `(change (age ?person) (+ (age ?person) 1))`, and arithmetic relations for functors in the precondition. The Task Formalism of O-Plan [12] allows to form assignments <pattern>=<value> where 'value' may be a numerical value, a constant, or even a list of values, and 'pattern' is a functional expression, e.g. `{color car}=blue`.

The role of functions in proof planning problems is twofold: (1) functions assign values to functional expressions.[1] (2) Functions allow the construction of new terms from already introduced terms. Terms represent objects, for instance, with the function $+$ and a constant n denoting a positive natural number a term $n+n$ can be constructed. This term denotes a natural number; it is new in the sense that without further information we do not know which object of the problem domain "natural numbers" it denotes. The term itself becomes an object with properties that cannot be found in the subparts, e.g. $n+n$ has a property that is independent from its subterm n: it is an even number.

Associated with functions is a notion of well-definedness of terms. Functions have a specified number of arguments and restrictions on the types of the arguments. Thus ill-defined terms in specifications can be detected and rejected.

Although functions can be circumscribed by relations, the relational representation is rather inefficient and unintuitive. It is always possible to encode a functional expression $f(x_1,\ldots,x_n)$ by a relation R^f, where $R^f(y,x_1,\ldots,x_n)$ stands for $y = f(x_1,\ldots,x_n)$. For instance, a relational expression of the formula $|g(x) - l| < \varepsilon$ would be $R^g(y_1,x)$, $R^-(y_2,y_1,l)$, $R^{|\cdot|}(y_3,y_2)$, $R^<(y_3,\varepsilon)$ with relations $R^g, R^-, R^{|\cdot|}, R^<$ corresponding to the functions $g, -, |.|, <$. The situation becomes even more complicated, when higher-order terms have to be considered. Suppose we want to express the term $f+g$, which is the function that is the sum of the functions f and g, and apply this function to a constant c, i.e., $(f+g)(c)$. In this case, we have to encode the application of a function as a relation.

We think that in many AI-domains functional representations could be beneficial. For instance, in a scenario in which objects are composed of many subparts. Suppose a planning scenario where the processes of a service station for car repair are modeled. A car could be represented explicitly by a (nested) functional term `car(wheel-1, ..., wheel-4, motor(cylinders-1, ..., carburator-1), ...)` with functions `car` and `motor`. Planning the repair of the car could result in exchanging `cylinders-1` with `cylinders-2` in the term that represents the car. Although it is possible to express these facts by function-free relations, the functional representation allows a compact representation of the composed objects in which the subparts of an object are directly accessible. Furthermore, a functional representation is often more intuitive and can be handled by external reasoners such as Computer Algebra Systems and constraint solvers.

[1] Note that an assignment is not an additional primitive of the domain language but expressible in the object language of ΩMEGA.

3.2 Meta-Variables

In AI-planning, variables may occur in the operator representation and all variables mentioned in the effects of an operator have to be included in the preconditions of an operator [16]. The initial state must not contain variables. An operator can introduce placeholders for unbound variables that have to be instantiated during the planning process.

In proof planning a placeholder mechanism exists, too. The placeholder, called *meta-variable*, can be introduced by operator applications into the proof plan and does not have to be instantiated immediately. Simplified speaking, a meta-variable can either originate from existentially quantified goals, universally quantified assumptions, or from an operator that introduces auxiliary variables.[2]

Because of the potential infinity of many mathematical domains, a delay of the instantiation of meta-variables in proof planning is indispensable rather than just more efficient than an immediate commitment. The reason is simply that a potentially infinite branching cannot be handled in (proof) planning and therefore the decision for branching has to be delayed.

The following differences between typical AI-planning and proof planning are important.

- As opposed to PDDL, in proof planning a (meta-)variable can be a placeholder for an object from a possibly infinite domain and the object can be a complex term.
- In ΩMEGA, the variables do not have to occur in what corresponds to preconditions. Rather an operator can introduce arbitrary auxiliary meta-variables, but this application of the operator is only valid (and hence the whole proof plan is only valid) if consistent instantiations for the meta-variables can be found.
- In proof planning the constraints upon the meta-variables can be more complex than simple equalities or negations of equalities. Moreover, there is no guarantee for a unique binding of these variables when planning is finished. For instance, the constraint $1 \leq d$ can be satisfied by many real numbers d including $1, 1.5, 2.2, \ldots$

Since constraints in proof planning are not just equational, constraint solvers with an elaborate theory-reasoning had to be adapted and integrated into proof planning. Since there is no guarantee for a unique instantiation, the external constraint solving includes a *search* for instantiations consistent with all constraints [11]. This is common for constraint solvers. During proof planning the constraints in meta-variables are sent to an adapted off-the-shelf external constraint solver. This external system can indicate an inconsistency which causes backtracking. Eventually, the constraint solver computes instantiations of the meta-variables that are consistent with all constraints. The realization of an interface to a constraint solver via operators is described in Section 3.4.

Currently very few AI-planner exist that can handle even simple linear constraints. Thus we think that an integration of constraint solving via meta-variables as in ΩMEGA could be beneficial. The integration of off-the-shelf external constraint solvers into the planning process allows to use their knowledge about a constraint domain. For instance, constraint solvers for equations and inequations over the natural numbers or over the reals comprise knowledge such as $x < x + 1$ that could be beneficial when dealing with resources such as fuel etc.

[2]More accurate: meta-variables originate from positively occurring existentially quantified and negatively occurring universally quantified subformulas in goals as well as from negatively occurring existentially quantified and positive occurring universally quantified subformulas in assumptions, see [4]; all other quantifications result in the introduction of new constants.

3.3 Quantification

AI-planning assumes each planning domain to have a finite set of objects. Therefore, a universally and existentially quantified formula can be replaced by a conjunction or disjunction for all elements such that quantified variables do not have to occur in planning states explicitly.

Existentially quantified effects are not included into PDDL. Why? There are different answers depending on what is taken as the semantics of existentially quantified effects. We found different interpretations of what existentially quantified effects should mean, e.g.,

- Classical planning deals with finite domains, so existential quantification would just be a shorthand for a disjunction. This would model a non-deterministic effect which classical planning does not allow and thus PDDL does not include existentially quantified effects.[3]
- Essentially, all variables in operators are implicitly existentially quantified.[4] An effect `add (p x)` with variables can either mean an implicit but eliminated ∃-quantification of `∃x(p x)` when `(p x)` unifies with a goal `(p A)`. Or the existential quantification can be used to restrict the unifiers, hence the instantiation, of variables.
- The right understanding of variables in operators is that they are universally quantified. Their apparent existential force arised because of how they are used. They seem to me to be used in a way that reflects the natural deduction inference rule "existential elimination" [14].

ΩMEGA's operators may have an existentially quantified effect as this is just another formula and the quantified formulas are not restricted to a finite domain interpretations.

Full quantification is one of the obvious differences between the PDDL and proof planning languages. In scenarios where large sets of objects occur or potentially infinitely many objects can be constructed, the usual techniques interpreting quantifications as conjunction or disjunction over a finite domain is inadequate. Then, techniques from proof planning that allow full quantification could be more adequate.

The semantics of quantification is described already in footnote 2. The instantiations of the meta-variables are subject of constraint solving following a least commitment strategy (see Section 3.2). Meta-variables may depend on constants. The constants represent what is circumscribed in mathematics by "arbitrary but fix" which avoids an explicit disjunctive or conjunctive interpretation over all objects of the corresponding quantifications.

Interestingly, certain extensions of the PDDL language correspond to the introduction of meta-variables in proof planning that originate from quantified goals or assumptions. In particular, the meta-variable approach in ΩMEGA's language corresponds to the Skolem terms approach in [14]. Steel suggests the extension of preconditions and effects of planning operators to full quantification by using Skolemization. Then, quantifiers are removed via the introduction of Skolem functions and Skolem constants. His example operator `hold-competition` expresses that after holding a sports competition, all the trophies have a winner. This corresponds to the situation calculus axiom:

$$\forall S: State. \forall T. trophy(T, S) \rightarrow \\ \quad \exists W. isWinnerOf(W, T, holdCompetition(S))$$

which can be skolemized to the formula

$$trophy(X, S) \rightarrow \\ \quad isWinnerOf(\$(X, S), X, holdCompetition(S))$$

with the Skolem term `$(x)`. The deletion of the state variables results in the (preconditions, effects) pair `(trophy(x), is-winner-of($(x),x))`. Here, Steel de facto extends

[3] Personal communication

[4] Personal communication

quantification in operators to existential quantification of the effects by allowing Skolem functions in the effect-part of the (preconditions, effects) pairs.

3.4 Employing External Systems

PDDL does not provide a connection to external systems. Some AI-planning systems make use of "experts" [20]. RAX-PS [6] uses experts in the development of plan fragments. Moreover, Nonlin [15] employed so-called *compute conditions* which were also used in an extended way as the interface to rich external systems in O-Plan [12].

In ΩMEGA, the application conditions and the outline computations can establish interfaces to external "expert" systems. Various "expert" systems exist for mathematical problem solving which have their specific data structures and very efficient algorithms, e.g., Computer Algebra Systems and constraint solvers. They can support the proof planning process by suggesting instances of variables or parameters, solving subproblems, or detecting inconsistencies.

Let us consider the application conditions of the operators COMPLEXESTIMATE-B and TELLCS-B. Both operators are central for planning limit problems. COMPLEXESTIMATE-B can be applied to inequality goals of the form $|b| < \varepsilon$. It contains the application condition `cas-extract(a,b)`. On polynomials a and b the function `cas-extract` calls the Computer Algebra System MAPLE [13] to compute the coefficient terms k, l for a decomposition $b = k * a + l$. If MAPLE succeeds, the condition evaluates to true and returns k, l as bindings which can be used in the outline computations to compute the new subgoals. If MAPLE fails, the condition evaluates to false and the operator is not applicable. TELLCS-B establishes the connection to $\mathcal{CoSIE}$ [11], an external constraint solver. TELLCS-B tackles inequality goals $a < b$. It contains the application conditions (`valid-cs` $a < b$) and (`test-cs` $a < b$). `valid-cs` asks $\mathcal{CoSIE}$ to check whether $a < b$ is a valid input constraint. `test-cs` calls $\mathcal{CoSIE}$ to test whether $a < b$ is consistent with its current constraint store, i.e., the constraints collected so far. Only if this is the case, the operator is applicable.

3.5 Filtering

Filtering is widely used in practical planners. The idea is to constrain the applicability of operators by additional conditions that filter out certain unintended applications of an operator. As opposed to preconditions, filters are conditions a planner wants to be true but will not make true in form of new subgoals. For instance, in a blocksworld scenario a `Move` operator that moves a block B from another block B_1 to a third block B_2 could have a filter condition that requires B_1 and B_2 to be different. This condition prohibits unintended applications of the `Move` operator but a planner will not establish $B_1 \neq B_2$ as new subgoal.

The PDDL syntax allows to specify so-called *filter preconditions*. The syntax of these filters is restricted to the language of PDDL preconditions. That is, only filters formalized at the object-level can be realized; to formalize filters at a meta-level is not possible. Sipe/Sipe-2 [18, 19] have the same kind of filters as PDDL. Sipe's operator language distinguishes between preconditions and goals, where the preconditions correspond to the filter preconditions in PDDL and the goals correspond to achievable preconditions in PDDL. Again the language of the preconditions is restricted to the object language. The check of meta-level conditions is not provided. Prodigy's [2] operator language has non-achievable preconditions. These preconditions can include arbitrary LISP functions. O-Plan [12] allows to filter matchings. It is even possible to call arbitrary LISP functions with a filter ?{**has** *functionname arguments result*}. This filter matches with every object *obj* for which the LISP expression *(functionname obj arguments)* evaluates to *result*.

In ΩMEGA the *application conditions* are meta-level conditions, i.e., they can restrict the formulas involved in the inference action beyond matching or unification with goals and assumptions in the planning state. Consider the =SUBST-B operator in Figure 1. Its application conditions relate to meta-properties of the formula f (Is *pos* a valid position in f? Is the subterm of f at position *pos* t or t' ?) that cannot be specified within the object-level language.

Syntactically, calls of arbitrary LISP functions are allowed in the application conditions. Thus the application conditions can even call tests performed by procedures or stand-alone systems (see Section 3.4 for examples).

3.6 Context Dependency

In order to have as abstract as possible operators and to reduce the number of operators in a domain description, several syntactical extensions have been used to make operators *context dependent* such that one general operator can represent a set of operators.

For this purpose, PDDL includes conditional effects. For instance, in the blocksworld, the effects of moving a block B may depend where B is moved. If B is moved on top of another block B', then an effect of the action is that B' is not clear anymore. But if B is moved to the table, then the table remains clear. Without conditional effects two operators were needed to express this difference. Context dependency is also important for practical planners (see, e.g., [17]). For instance, Sipe/Sipe-2 [18, 19] use so-called ***deductive operators*** to represent context dependency. Different deductive operators provide a way to distinguish side effects and to realize conditional effects.

Among others, the parameters in ΩMEGA's operators make the operators dependent on the context. More detailed, application conditions and outline computations allow to call LISP functions which can be applied to parameters. The result of these function calls depends on the parameter instantiations.

A simple example for the context dependency in ΩMEGA is given in the operator =SUBST-B in Figure 1. The instantiation of *pos* determines which subterm of the goal formula is substituted by the operator. The application conditions use the two LISP functions `valid-position-p` and `term-at-position` to test properties of the formula f depending on the parameter *pos* and the equation subterms t and t'. The outline computations return the new goal f' by the LISP function `replace-at-position` from f, t, t', and *pos*. A more complicated example is the operator INDUCTION-B that reduces a universally quantified goal $\forall x P(x)$ to the base case goal, $P(zero)$, and the step case goal, $P(c) \Rightarrow P(s(c))$, of an induction. INDUCTION-B has the parameters *zero* and *s*, where the instantiation of *zero* specifies the base element of the induction and the instantiation of *s* specifies the successor-function used in the induction. Note that parameters are not bound during the matching of an operator with the planning state. Rather, the "right" choice of parameter instantiations, e.g., the choice of the suitable base case and the successor-function for an application of INDUCTION-B, is subject to the control of the proof planner.

In the proof planning domains context dependency is a very important feature since often the set of corresponding non-parameterized operators can be potentially infinite. For instance, without context dependency infinitely many =SUBST-B operators for all possible subterm positions and infinitely many possible assumptions would have to replace the parameterized =SUBST-B operator. Thus ΩMEGA's context dependency goes beyond the approaches used in other systems and in PDDL. In particular, as opposed to PDDL and most AI-planners which give the effects of an operator explicitly the effects of an ΩMEGA operator can be given implicitly by LISP functions. These functions allow to compute the effects of an operator completely by procedures.

4 Conclusions

We presented the domain representation of proof planning domains and discussed some domain-independent features that go beyond the PDDL standard. These features include: functions and full quantification in the problem and operator language, powerful filters restricting the choice of instances of operators, context dependent parameterized operators, meta-variables and their constraints, and knowledge implicitly represented in integrated external systems.

The described extensions are not domain-specific for mathematical domains. Therefore we think that they could be useful extensions for other AI-planners as well. For instance, we think that in many AI-domains functional representations could be beneficial. Moreover, an integration of constraint solving like that in ΩMEGA could be adopted where existing external service systems communicate with the planner and solve constraints. In scenarios where large sets of objects occur or potentially infinitely many objects can be constructed, the usual techniques interpreting quantifications as conjunction or disjunction over a finite domain is inadequate. Then, techniques that allow to deal with full quantification via meta-variables as in proof planning could be more adequate.

In addition to its expressive domain and operator language ΩMEGA employs also control rules and strategies to formalize domain knowledge. Control rules encode heuristic knowledge about when and how mathematical inferences (operators) should be applied. Furthermore, strategies reflect the knowledge about different proof techniques for a class of problems [9].

References

[1] A. Bundy, 'The use of explicit plans to guide inductive proofs', in *Proc. of CADE-9*, LNCS 310, pp. 111–120. Springer, (1988).

[2] J.G. Carbonell, Jim Blythe, Oren Etzoni, Yolanda Gil, R. Joseph, D. Kahn, C. KNoblock, S. Minton, A. Perez, S. Reilly, M. Veloso, and X. Wang, *Prodigy 4.0: The Manual and Tutorial*, cmu-cs-92-150 edn., 1992.

[3] A. Church, 'A Formulation of the Simple Theory of Types', *JSL*, **5**, 56–68, (1940).

[4] M. Fitting, *First-Order Logic and Automated Theorem Proving*, Graduate Texts in Computer Science, Springer, second edn., 1996.

[5] G. Gentzen, 'Untersuchungen über das Logische Schließen I und II', *Mathematische Zeitschrift*, **39**, 176–210, 405–431, (1935).

[6] Ari K. Jonsson, Paul H. Morris, Nicola Muscettola, Kanna Rajan, and Ben Smith, 'Planning in interplanetary space: Theory and practice', in *Proc. of AIPS 2000*, (2000).

[7] D. Long M. Fox, 'PDDL2.1: An extension to PDDL for expressing temporal planning domains'. 2001.

[8] E. Melis, 'The "limit" domain', in *Proc. of AIPS 1998*, pp. 199–206, (1998).

[9] E. Melis and A. Meier, 'Proof planning with multiple strategies', in *Proc. of the First International Conference on Computational Logic*, LNAI 1861, pp. 644–659. Springer, (2000).

[10] E. Melis and J. Siekmann, 'Knowledge-based proof planning', *Artificial Intelligence*, (1999).

[11] E. Melis, J. Zimmer, and T. Müller, 'Integrating constraint solving into proof planning', in *Proc. of FroCoS'2000*, LNAI 1794, pp. 32–46. Springer, (2000).

[12] O-Plan-Team, *O-Plan Task Formalism (TF) Manual*, AI Applications Institute, University of Edinburgh, 1995.

[13] Darren Redfern, *The Maple Handbook: Maple V Release 5*, Springer, 1998.

[14] S. Steel, 'Full quantification in partial order plans by using skolem constants', in *Proc. of 19th Workshop, UK Planning and Scheduling Special Interest Group*, pp. 215–228, (2000).

[15] A. Tate, 'Generating project networks', in *Proc. of IJCAI-77*, pp. 888–893. Morgan Kaufmann, (1977).

[16] D.S. Weld, 'An introduction to least committment planning', *AI magazine*, **15**(4), 27–61, (1994).

[17] David E. Wilkins, 'Domain-independent planning: Representation and plan generation', *Artificial Intelligence*, **22**, 269 - 301, (1984).

[18] David E. Wilkins, *Practical Planning*, Morgan Kaufmann, San Mateo, 1988.

[19] David E. Wilkins, 'Using the sipe-2 planning systems (a manual for sipe-2, version 6.1)', Technical report, SRI, (2000).

[20] David E. Wilkins and Marie desJardins, 'A call for knowledge-based planning', *Artificial Intelligence*, **22**, (2001).