UNIVERSITÄT DES SAARLANDES
FACHBEREICH INFORMATIK
D-66041 SAARBRÜCKEN
GERMANY

WWW: http://jswww.cs.uni-sb.de/pub/www/

SEKI Report

Island Planning and Refinement

Erica Melis

SEKI Report SR-96-10

Island Planning and Refinement

Erica Melis
Fachbereich Informatik, Universität des Saarlandes
66041 Saarbrücken, Germany

Abstract

Planning for realistic problems in a static and deterministic environment with complete information faces exponential search spaces and, more often than not, should produce plans comprehensible for the user. This article introduces new planning strategies inspired by proof planning examples in order to tackle the search-space-problem and the structured-plan-problem. Island planning and refinement as well as subproblem refinement are integrated into a general planning framework and some exemplary control knowledge suitable for proof planning is given.

1 Introduction

Refinement planning as unguided search is notoriously hard because of the combinatorial search involved. For realistic planning problems the *intractable search space* prevents many current refinement planning techniques from being successful. For a discussion see, for instance [10, 22]. Another problem we face in some domains is the need for *structured plans* that are intelligible for the user. In order to restrict the search space, planning in abstraction spaces by precondition-abstraction, for instance in [29, 23, 5], and hierarchical task network planning (HTN) [30, 8] have been suggested. The latter also serves to structure plans.

The search-space-problem applies to theorem proving and proof planning as well. As a result, rippling – a specific meta-level control – is used in CL^AM [6] for planning inductive proofs [4, 16]. One step towards solving the structured-plan-problem in proof planning is the definition of operators that provide a sequence of calculus-level proof steps[1] when executed. In this way, proof plans are abstract representations of calculus-level proofs.

Our experience in proof planning gives rise, however, to additional ideas to cope with the two problems: A *controlled mixture* of planning strategies integrated into a basic architecture for multiple planning strategies can help to reduce the search space. Establishing smaller problems, such as subproblems and abstracted problems, can help to reduce the search space. Structured proof plan construction can be supported by defining subproblems. We think that these ideas are more widely applicable in planning.

[1] e.g. of the Natural Deduction (ND) calculus

The paper is organized as follows. First, we define some basic notions by extending Kambhampati's framework [21, 19] for unifying several planning strategies that may serve to integrate our island planning and refinement. We briefly review proof planning. Then a motivating example leads to island planning and refinement as well as subproblem refinement. These strategies are introduced as refinement operations. Some exemplary control knowledge is given.

2 Planning as Refinement Search

A planning *problem* is a tripel (I, G, Ops) with the initial state description I,[2] a set G of goals, and a set Ops of operators. As usual, operators have preconditions (pre) and postconditions (post). A (ground) operator sequence $\mathcal{S}$ is a solution for a problem, if $\mathcal{S}$ can be executed from the I such that the goals from G are satisfied.

A *partial plan* π is a tuple (T, O, B, S, Aux, ST), where T is a set of steps[3] in the plan containing t_0 and t_∞, ST maps step names to operators. t_0 is mapped to `start` and t_∞ to `finish`, where `start` and `finish` are the only operators of so-called *null plans* with $t_0 \prec t_\infty$. O is a partial order over T, B is a set of binding and prohibited binding constraints on variables. We introduce S as a set of subproblems each of which has a plan that is a connected subplan of π. That is, for any $t_1, t_2 \in T_{sub}$, if $t_1 \prec t \prec t_2$, then $t \in T_{sub}$. Aux is a set of auxiliary constraints. Kambhampati [19] includes the following auxiliary constraints in order to unify state-space and plan-space refinement on one representation of partial plans:

- Interval preservation constraints specified by triples (t, p, t')
- Point truth constraints specified by pairs (p, t)
- Contiguity constraints specified by the relation $t_i * t_j$ between two steps that demands no step intervene between t_i and t_j.

u In addition, we define the auxiliary constraint

- Abstraction constraints represented by $abs(g_i, g_j)$ for goals/assumptions g_i, g_j. This constraint temporarily identifies g_i with g_j in constraints and in relations to steps. Thereby abstract steps are integrated into the partial plan temporarily. Semantically, this constrains all candidates of the partial plan to those for which g_i and g_j are identified.

Some derived notions important for state-space refinement are the *header* - the maximal sequence of steps $t_0, \ldots t_H$ with $t_0 * \ldots * t_H$, the *head step* t_H, and the *head-state* which is the state resulting from the application of the steps in the header to the initial state. Similarly, trailer, tail steps, and tail-state are defined.

Semantically, a partial plan π is a set of instantiated ground operator sequences that are consistent with the constraints. This set is called the candidate set $\langle\langle \pi \rangle\rangle$ of π. Refinement planning starts with a null plan defined by the planning problem. Its candidate set is the set of all ground operator sequences.

[2] In proof planning called set of assumptions

[3] i.e., instantiated operators

Syntactically, refinement planning successively refines a plan by adding constraints until a solution can be picked from a candidate set. Semantically, a refinement operation maps a partial plan π to a set of partial plans $\{\pi_i\}$ such that $\langle\langle\pi\rangle\rangle \subset \langle\langle\pi_i\rangle\rangle$ for all ϕ_i. Kambhampati and Srivastava [21] define refinement operations for (forward and backward) state-space and plan-space refinement. They say that the control for choosing a refinement strategy has to be designed in the near future. In this paper we shall define additional refinement strategies and propose exemplary control knowledge used in proof planning.

2.1 Specifics of Proof Planning

To make a long story short, we give some essentials of proof planning only. For more details about the state of the art see [26]. For proof planning two roads join, (1) the use of tactics and (2) meta-level control. As opposed to traditional automated theorem that applies calculus-level inference rules, i.e. low level inferences, proof planning relies on tactics [14]. Tactics are procedures that produce a (not necessarily fixed) sequence of calculus-level inferences when executed. Operators, are defined as specifications of tactics by pre- and postconditions. You can imagine, e.g., an operator `ApplyEquality` that (backwardly) applies an equation $e : LHS = RHS$ to the current goal g by substituting a subterm of g that equals $\sigma(RHS)$ by $\sigma(LHS)$ for a substitution σ.[4] `ApplyEquality`(e) has the precondition $F(\sigma(LHS))$ and the postcondition $F(\sigma(RHS))$ for a meta-variable F.

Characteristics of proof planning compared to planning in other domains are

- The objects are (mathematical) objects such as numbers, lists, or trees and operators manipulate formulas describing objects and their relations and functions. In proof planning there is no goal interaction in the object-level sense because the application of a sequence of logical inference rules does not destroy object-level preconditions.
- Typically, the solutions are deep.
- Infinite branching can occur for the instantiation of existentially quantified variables, for lemma speculation, etc.[5] This makes control even more vital.
- Often, infinitely many potential bindings have to be considered. Therefore, the control of bindings and an elaborate handling of B is needed.
- The knowledge about the mathematical world is complete and certain rather than incomplete and uncertain as in many real world applications of planning. I.e., proof planning is classical planning which means planning in a static and deterministic environment with complete information.

[4] For the moment we do not care about the position of the subterm in g.

[5] One reason is that the cut rule, i.e., the backwardly applied rule

$$\frac{\Gamma, B \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A},$$

not avoidable in general. For an analysis see [24]

- Structured plans are clearly preferred by humans, see e.g. [25].
- Difference reduction techniques proved to be useful for planning inductive and equational proofs [18].

Planning strategies employed so far in the proof planners *CLAM* [6] or OMEGA [15] respectively, are forward and backward state-space refinement, HTN-like planning, and precondition abstraction. HTN-like operators can be expanded to subplans. E.g., the method induction-strategy in the proof planner *CLAM* [3] and diagonal-method in OMEGA are operators expanding to subplans. In OMEGA, a fixed precondition abstraction of operators is possible that postpones the achievement of a particular precondition to a hierarchically lower planning level. For instance, a more elaborate `ApplyEquality`(e) abstracts from the precondition that is an antecedent of an essentially equational formula e. E.g., if e is $(x \in T \rightarrow x^2 = a)$, then `ApplyEquality`($e$) applies $(x^2 = a)$ and leaves $(x \in T)$ as a precondition to be satisfied in a lower hierarchical level.

3 Island Planning and Refinement

Similar to [21], we combine several planning (refinement) strategies in a general planning algorithm. Table 1 shows the top-level control of the planner. The new strategies island planning, island refinement, and subproblem refinement are described in later subsections. The selected strategy is applied to the partial plan π to generate refinements. As in [21], we could have routines accomplishing the control, as for instance `pick-refinement`. Instead, we encode the control knowledge declaratively into control-rules and indicate their use by ♣. Both ways have their advantages.

Planning Algorithm `PLAN`(π)/* Returns refinements of π*/
Parameters: `sol` procedure for picking solution candidates

1. **Termination Check**: If `sol`(π) returns a solution, return it and terminate. If it returns *fail*, fail. Otherwise continue.
2. **Refinement**: Pick one of the following refinement strategies ♣ and refine(π): (*Not a backtrack point*)
   - forward-state-space refinement, called progression,
   - backward-state-space refinement, called regression,
   - island-planning,
   - island-refinement,
   - subproblem-refinement.
3. **Consistency Check** (optional): If partial plan is inconsistent, prune it.
4. **Recursive Invocation**: Call `PLAN` on the refined plan.

Table 1: Algorithm outline for classical multistrategy planning

Now we give an example from proof planning. It naturally leads to island planning and refinement strategies and their combination with subproblem re-

finement. In particular, we learn about the need to integrate several planning strategies and about some control knowledge to guide the choice of planning strategies.

3.1 Example

Suppose we plan a proof of theorem 7.5.7 from Deussen's book "Halbgruppen und Automaten" [9] that states that the mapping Φ from an F-semimodul T_1 to an F-semimodul T_1 is a homomorphism,[6] where $\Phi\phi_1 = \phi_2$ holds for two homomorphisms $\phi_1 : S \mapsto T_1, \phi_2 : S \mapsto T_2$. The proof assumptions (initial state) include:
ϕ_1 is surjective; ϕ_1 is a homomorphism in the F-semimodul T_1; ϕ_2 is a homomorphism in the F-semimodul T_2. The formal goal (theorem) is

$$\forall f, x(x \in T_1 \wedge f \in F \rightarrow \Phi(f \cdot x) = f \cdot \Phi(x)). \tag{1}$$

State-space planning, progression and regression, is applied until we reach the subgoal

$$\Phi(f \cdot a) = f \cdot \Phi(a) \tag{2}$$

for a constant a, and among others the (derived[7]) assumptions in the head state

$$f \in F \wedge Y \in S \rightarrow \phi_2(f \cdot Y) = f \cdot \phi_2(Y), \tag{3}$$

$$f \in F \wedge Y \in S \rightarrow \phi_1(f \cdot Y) = f \cdot \phi_1(Y), \tag{4}$$

$$x \in S \rightarrow \Phi(\phi_1(x)) = \phi_2(x), \tag{5}$$

$$f \in F \wedge x \in S \rightarrow \phi_1(f \cdot x) = f \cdot \phi_1(x). \tag{6}$$

and

$$y \in T_1 \rightarrow \exists x(x \in S \wedge \phi_1(x) = y) \tag{7}$$

Progression is blocked and further regression would fail to find a step applicable to (2). From a more abstract point of view, the goal (2) requires to move $f\cdot$ from a position p_1 wrt. $\Phi(a)$ to the position p_2 wrt. $\Phi(a)$. The meta-reasoning suggests that presumably, a lemma has to be introduced to prove this (sub)goal.

To narrow the gap between the goal and the assumptions[8], the domain control knowledge suggests to plan at an abstract level in order to find an *island node* from which it is possible to **progressively** plan to (2). If we abstract the goal and the assumptions to *the positions of the context relative to some*

[6]Definition: Let S and T be F-semimoduls. A mapping $\phi : F \mapsto H$ is called a homomorphism from S to T) iff $\forall f, Y(f \in F \wedge Y \in S \rightarrow \phi(f \cdot Y) = f \cdot \phi(Y))$.

[7]E.g., quantifiers are removed, definition expanded.

[8]The gap corresponds to a missing lemma

skeleton,[9] we obtain the abstracted goal $@(f \cdot @) = f \cdot @$ from (2) where $@(f \cdot @)$ and $f \cdot @$ describe positions of the context $f \cdot$ relative to the skeleton $\Phi(a)$. Similarly, the abstraction of the assumption (3) is $@(f \cdot @) = f \cdot @$.[10] Applying this abstraction of (3) to the abstraction of (2) by an operator `ApplyEquality` reduces the abstract goal to $@(f \cdot @) = @(f \cdot @)$ which is an instantiation of the postcondition $x = x$ of the operator `elementary` that has the precondition *true*.

In other words, by planning for an abstracted problem we find that at **some** point `ApplyEquality`(3) and `elementary` have to be applied. We name this planning *island planning*. The LHS of Figure 1 depicts the described situation. Rectangulars represent steps and ovals indicate goals/assumptions. s_a is an abstracted step. g_a denotes the abstraction of the goal g. The LHS of

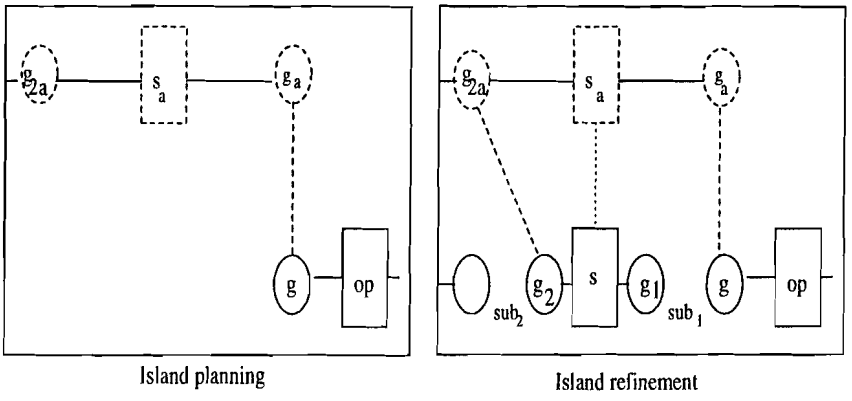

Figure 1: Island Planning and Refinement

the figure demonstrates how $abs(g, g_a)$ allows to integrate the abstract step s_a into the overall plan. The RHS of Figure 1 depicts the mapping back of the abstract level steps and goals which is called *island refinement*. This refinement produces island nodes at the ground level from nodes at the abstract level by mapping back the operator, its preconditions, and its postconditions. For instance, `ApplyEquation`$(@(f \cdot @) = f \cdot @)$ with the precondition $@(f \cdot @) = @(f \cdot @)$ and postcondition $@(f \cdot @) = f \cdot @$ is mapped back to the island step `ApplyEquation`$(f \in F \wedge Y \in S \rightarrow \phi_2(f \cdot Y) = f \cdot \phi_2(Y))$ the precondition $F_0 = \phi_2(f \cdot Y)$ and the postcondition $F_0 = f \cdot \phi_2(Y)$ for a meta-variable F_0.[11] In the RHS of Figure 1, s_a is mapped back to $mb(s_a) = s$ which has the

[9] In an equation, equivalence, or implication a *skeleton* is a term that occurs on the LHS and on the RHS – in (2), e.g. the skeleton is $\Phi(a)$. The corresponding *context* is constructed from the remaining parts of the formula – in (2) the corresponding context is $f \cdot$. The notions skeletonand context were coined by Hutter and Bundy for rippling in [16, 4].

[10] The antecendent is abstracted away.

[11] For simplicity we do not replace Y by σY. Actually, the postcondition of `ApplyEquation`$(f \in F \wedge Y \in S \rightarrow \phi_2(f \cdot Y) = f \cdot \phi_2(Y))$ is $F_0 = F_1(\phi_2(f \cdot Y))$ and the precondition $F_0 = F_1(\phi_2(f \cdot Y))$ for meta-variables F_0, F_1. The meta variable F_1 can be instantiated to the identity function because the RHS of the abstract postcondition is $f \cdot @$.

postcondition g_1 and precondition g_2.

The ground-level plan contains a gap between the goal (2) (g in Figure 1) and the postcondition (g_1 in the figure)

$$F_0 = f \cdot \phi_2(Y) \tag{8}$$

This gap constitutes a new subproblem (sub_1). Further subplan refinement tries to solve this new problem by *progression* from (8) because, as mentioned above, progression has to be chosen for planning from the goal (2): The control knowledge used in planning for the subproblem sub_1 includes the rule
"If the goal is equational (or an equivalence or implication) and if the signature of the goal and the distinguished assumption differ in their multisets S_1 and S_2 of (maximal) symbols, then try to apply equations (or an equivalence or implication) that introduce a (large) subset of S_1 or removes a (large) subset of S_2" [17].
This motivates the choice of the step `ApplyEquation`(5) that gives

$$F_0 = f \cdot \Phi(\phi_1(Y)). \tag{9}$$

because ϕ_2 is a maximal symbol defined by (5). Hence the multisets are $\{\Phi, \phi_1\}, \{\phi_2\}$ and the application of (5) serves exactly the related introduction of phi_2 and removal of Φ, ϕ_1. Note that this control-reasoning again amounts to an abstraction. We use this abstraction in the control at ground level only rather than producing an island node by island planning. This control applies because (5) is directly applicable at the ground level and no other (nested) differences between the goal and the distinguished assumption occur The subproblem planning for sub_1 can be completed by instantiating F_0 to $\Phi(f \cdot a)$ and by introducing the lemma

$$\exists Y.\phi_1(Y) = a \tag{10}$$

by a step `LemmaSuggestion`. Then the next island refinement yields a subproblem (sub_2) with the goal g_2:

$$\Phi(f \cdot a) = \phi_2(f \cdot Y) \tag{11}$$

and the initial state containing *true*. Control knowledge to be employed here for choosing a step is the same as explicated for sub_1. This time, regression for sub_2 applies `ApplyEquation`(5) giving

$$\Phi(f \cdot a) = \Phi(\phi_1(f \cdot Y)). \tag{12}$$

The additional control knowledge
"If a goal is equational (or an equivalence or implication) and has to be reduced to *true*, then the different contexts of the LHS and RHS have to be rippled-out or rippled-in[12] by annotated rewrite rules. If only one side has a context, then

[12]rippling-in and rippling-out are notations from rippling[16]

choose rippling-in rewrites as instantiations of `ApplyEquality`"
then forces to choose the step `ApplyEquation`(6) which gives

$$\Phi(f \cdot a) = \Phi(f \cdot \phi_1(Y)). \tag{13}$$

The application of the lemma (10) results in the subgoal

$$\Phi(f \cdot a) = \Phi(f \cdot a) \tag{14}$$

which is reduced to *true* by the operator `elementary`.

3.2 Island Planning

Island planning means planning at an abstract level that is established by *problem abstraction*. The construction of an abstracted problem abstracts a goal g, the head state, and the operators of the original problem by a mapping am. Island planning as illustrated with the example is realized by the refinement strategy given in Table 2. Routines or the interpretation of control-rules for

---

Algorithm backward-island-planning(π)/* Returns refinements of π*/
Parameters: `sol` procedure for picking solution candidates
`insert` procedure for inserting an abstract plan into π.

1. **Goal selection**: Pick a goal g ♣. *Not a backtrack point.*
2. **Abstraction**: From g and the current head stateconstruct an abstracted problem P_{abs} by an abstraction mapping am ♣. Introduce the auxiliary constraint $abs(g, g_a)$ for $am(g) = g_a$.
   backtrack point for different possible abstractions
3. **Plan**(π_{abs}) for problem P_{abs}. This involves
   - Termination check: If `sol`(π_{abs}) returns a solution, return it and terminate planning(π_{abs}.
   - Refine π_{abs} using some refinement strategy,
   - Recursion on refined plan π_{abs}.
4. `insert` π_{abs} into π.

Table 2: Backward Island Planning

picking a goal and choosing an appropriate abstraction are needed and indicated by ♣.

Island planning is a refinement operation because the (abstract) steps and their constraints are introduced into the partial plan and restict the solution candidates. The LHS of Figure 1 shows how the auxiliary constraint $abs(g, g_a)$ gives rise to "connect" the ground level steps with abstract level steps.

In the example of section 3.1, the goal g is picked and abstracted to g_a. The abstracted problem P_{abs} has an initial state that includes the abstracted proof assumptions and operators that take abstract instantiations. Planning for P_{abs} inserts the operators ApplyEquation(abs(3)) and `elementary`. `insert` depends on $abs(g_i, g_j)$. In the example, ApplyEquation(am(3)) * op, `elementary` * ApplyEquation(am(3)), and `elementary` $\prec$ op are introduced into π.

Island planning as described in Table 2 is a multi-step planning that involves nested application of planning strategies in step 3. Since the abstraction belongs to the refinement procedure, the island planning has to accomplish the whole planning for the abstracted problem and this in turn may need several planning steps. Alternatively to the described procedure, the problem abstraction can be separated (which is not a plan refinement operation) and the usual refinement planning applies to the abstract plan. Then the relationship between plans at different levels of abstraction would have to be specified.

3.3 Backward Island Refinement

The abstract steps have to be refined in order to obtain a ground-level plan consisting of ground steps only. Island refinement does this job.

Algorithm backward-island-refinement(π)/* Returns refinements of π*/
Parameters: `introduce-subproblem` procedure

1. **Step selection**: Pick abstract step s_a ♣ with $abs(g, post(s_a))$ and remove the constraint $abs(g, post(s_a))$ for some g.
   Not a backtrack point
2. **Island instantiation**: Replace s_a in π by a mapped-back ♣ step $mb(s_a)$. For any t this involves
   - replacing $s_a * t$ by $mb(s_a) \prec t$ and $t * s_a$ by $t \prec mb(s_a)$
     $s_a \prec t$ by $mb(s_a) \prec t$ and $t \prec s_a$ by $t \prec mb(s_a)$
   - introducing auxiliary constraints: for each precondition pre_k of s_a introduce $abs(pre_k(mb(s_a)), pre_k)$.
     Backtrack point; all possible mb have to be considered.
3. **Subproblem construction**: `introduce-subproblem` introduces into S a problem P_{sub} with $G = g$ and $post(mb(s_a)) \in I$.

Table 3: Backward Island Refinement

The choice of the abstract step s_a is subject to control knowledge. The `introduce-subproblem` routine depends on the domain characteristics. In proof planning, the constructed subproblem has the initial state $head\_state \cup post(mb(s_a))$ (because no element of the head state is destroyed by any operator application), the goal g, and $Ops_{sub} = Ops$ for the original set of operators Ops.

In general, different mappings mb might be possible. In order to reduce backtracking, we allow for meta-variables rather than fully instantiating goals where terms or formulae cannot completely be specified. The instantiation of these meta-variables is successively restricted by a constraint solver processing the bindings and prohibited bindings in B.

For an illustration of island refinement see the RHS of Figure 1, where the step s_a is picked. The dashed lines indicate removed and introduced abstraction constraints, respectively and the dotted line indicates mapping back of the step s_a. The ground step s is introduced into the plan π, $s_a * op$ is replaced by $s \prec op$. $abs(g, g_a)$ is removed and for $pre(s) = \{g_2\}$, the auxiliary constraint $abs(g_2, g_{2a})$ is introduced. The procedure `introduce-subproblem` returns the problem sub_1 and its null plan.

3.4 Subproblem Refinement

Subproblem refinement takes subproblems, resulting, e.g., from island refinement or given by the user, plans for these subproblems, and inserts new constraints into the partial plan π. The insertion involves at least replacing t_0 and

---

Algorithm subproblem-refinement(π)/* Returns refinements of π*/.
Parameters: `sol` procedure, `insert-subproblem` procedure

1. **Pick-subproblem**: A subproblem P_{sub} is chosen ♣.
   Not a backtrack point
2. **Planning** by refining π_{sub}. This involves
   - Checking termination: if `sol`(π_{sub}) finds a solution of P_{sub}, return it, and then terminate.
   - Refining π_{sub} by some planning strategy,
   - Recursively call planning for π_{sub}.
3. `insert-subproblem` π_{sub} into π.

Table 4: Subproblem Refinement

t_∞ in π_{sub} by steps of π. `sol` is the routine for picking solution candidates from the candidate set of the partial subplan. In case `sol` does not return a solution, the algorithm fails for the current branch or is continued, respectively.

As island planning, subproblem refinement is a multistep planning with nested planning strategies because it involves inserting the completed subplan into π. This multistep planning is all right for proof planning. In order to avoid multiple steps in subproblem planning for other domains, an alternative subproblem refinement algorithm involves

1. Pick subproblem P_{sub} with current plan π_{sub}.
2. Refine π_{sub}.
3. Propagate auxiliary, binding constraints, and order constraints of π_{sub} to π by a `propagate-constraints` routine.

In our example, the plan of the subproblem sub_1 is refined by progression that inserts `ApplyEquation` and `LemmaSuggestion`. The subplan is inserted 'between' the steps s and op. Similarly, regression erfines the plan of the subproblem sub_2. A constraint solver for the constraints of B eagerly computes instantiations of meta-variables F_0, F_1.

4 Control

Control knowledge consists of (domain dependent) heuristics concerning decisions at choice points. They can be encoded into **compiled procedures** or as **declarative control-rules**. Depending on the strictness of a heuristic, different types of control-rules can be designed: `choose`, `don't-choose`, and `prefer` rules [27].

For applying the state-space refinement strategies other kinds of control knowledge can be used, as practiced, e.g., in Prodigy [27]. This knowledge supports the decisions:

- Choose-goal
- Choose-bindings
- Choose-operator
- Apply-operator

Corresponding to the decisions to be made in the planning described in the previous sections, the additional classes of control knowledge need to be considered:

- Pick-refinement
- Choose-abstraction
- Pick-subproblem
- Pick-goal to be abstracted
- Pick-abstract-step to work on

Finally, in order to give an idea of what the control knowledge may look like in proof planning, we propose some exemplary control-rules formulated in natural language here. In particular, we extracted some rules from the proof planning example in section 3.1:

1. Pick-refinement control knowledge
   - If there is an abstract step in the plan, then `prefer` island-refinement.
   - If lemma speculation needed, then `prefer` island-planning and mark current goal with `f`.
   - If the goal in P_{sub} is marked with `f`, `choose` progression in the planning for π_{sub}.
2. Choose-abstraction control knowledge.
   - If the current goal is an equational formula and the LHS and RHS differ in the occurring function symbols, then `prefer` the abstraction to maximal_symbol_multisets[13] described in [17].
   - If the current goal is an equational formula and the LHS and RHS differ in the (position of a) context only, then `prefer` an abstraction to the position of the context relative to the skeleton.
   - If the current goal is an equational formula and the LHS and RHS differ in the number of function occurrences, then `prefer` the *# function-occurrence* abstraction [7].

[13] used in the subproblem refinements above

Clearly, the meta control-rules has to be formalized in a meta-level language that can express properties of the planning state, the planning history, the partial constraint solution, and measures of the progress of global control that might be explicitly encoded into annotations of goals and of assumptions.

5 Conclusion and Related Work

From an AI planning point of view, we introduced new planning strategies that support a reduction of the search space and that help to structure plans. From a proof planning point of view, we introduced a formal framework to make planning strategies explicit and to integrate them into a systematic, unified planning framework rather than hiding them in code. This framework also helps to explicate, where which kind of control knowledge is needed. That is, we bring together ideas from different origins, namely planning, proof planning, and abstraction.

We have learned from proof planning examples the need for defining and integrating different planning strategies. Therefore, we propose several planning strategies that can be invoked in a planning framework, among them, island planning and island refinement. These strategies cope with abstracted problems or subproblems. Establishing smaller problems, such as subproblems and abstracted problems, can help to reduce the search space. Furthermore, the construction of well-structured proof plans can be supported by defining subproblems and integrating their solutions into the overall plan in an isolatedway only.

Allowing for different planning strategies adds a choice point to the general planning algorithm. Hence we need control knowledge on when to choose which planning strategy in addition to the control knowledge used within planning strategies. For proof planning we have presented exemplary control knowledge.

Work on combining different strategies within one planning framework has been published by Kambhampati et al. [19, 20, 21] and for Prodigy in [11]. There is a variety of ideas about abstraction. Problem abstraction to guide problem solving has, for instance, been addressed in [23, 13, 31]. More specifically in theorem proving a classical paper is Plaisted's [28]. In ABSFOL [12] the mapping back is supported for user-provided abstractions. Hutter and Autexier proposed concrete abstractions for equational theorem proving such as maximal_ used in our example above, path_to_common_skelton_parts, rewriting_abstraction in [18, 1, 17] for particular problems in theorem proving. In planning, declarative control-rules for choosing goals, operators, and bindings are used in Prodigy [27] and such a control is also described in [2].

References

[1] S. Autexier. Heuristiken zum Beweisen von Gleichungen. Master's thesis, FB Informatik, Universität des Saarlandes, 1996.

[2] A. Barrett, K. Golden, J.S. Penberthy, and D. Weld. *USPOP User's Manual, Version 2.0.* Dept.of Computer Science and Engineering, University of Washington, 1993. Technical Report 93-09-06.

[3] A. Bundy. The use of explicit plans to guide inductive proofs. In E. Lusk and R. Overbeek, editors, *Proc. 9th International Conference on Automated Deduction (CADE)*, volume 310 of *Lecture Notes in Computer Science*, pages 111–120, Argonne, 1988. Springer.

[4] A. Bundy, Stevens A, F. Van Harmelen, A. Ireland, and A. Smaill. A heuristic for guiding inductive proofs. *Artificial Intelligence*, 63:185–253, 1993.

[5] A. Bundy, F. Giunchiglia, R. Sebastiani, and T. Walsh. Computing abstraction hierarchies by numerial simulation. In *Proceedings of the 13th National Conference on AI*, pages 523–529. AAAI, 1996.

[6] A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7:303–324, 1991.

[7] J. Cleve and D. Hutter. A methodology for equational reasoning. In J.F. Nunamaker and R.H. Sprague, editors, *Hawaii International Conference on System Sciences.* IEEE Computer Society Press, 1994.

[8] K.W. Currie and A. Tate. O-plan: The open planning architecture. *Artificial Intelligence*, 52(1), 1991.

[9] P. Deussen. *Halbgruppen und Automaten*, volume 99 of *Heidelberger Taschenbücher.* Springer, 1971.

[10] M. Drummond. On precondition achievement and the computational economics of automatic planning. In *Current Trends in AI Planning*, pages 6–13. IOS Press, 1994.

[11] E. Fink and M. Veloso. Formalizing the Prodigy planning algorithm. In M. Ghallab and A. Milani, editors, *New Directions in Planning*, pages 261–272. IOS Press, Amsterdam, Oxford, 1996. Extended version as technical report CMU-CS-94-123, 1994.

[12] F. Giunchiglia and A. Villafiorita. ABSFOL: a Proof Checker with Abstraction. In , editor, *Proceedings of the 13th International Conference on Automated Deduction (CADE-13)*, pages –, 1996. To appear in proceedings of the the 13th Conference on Automated Deduction (CADE-13). Also IRST-Technical Report 9602-20, IRST, Italy and DIST-Technical Report 96-0036, DIST, University of Genova, Italy.

[13] F. Giunchiglia and T. Walsh. A theory of abstraction. *Artificial Intelligence*, 57:323–390, 1992.

[14] M. Gordon, R. Milner, and C.P. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation.* Lecture Notes in Computer Science 78. Springer, Berlin, 1979.

[15] X. Huang, M. Kerber, M. Kohlhase, and J. Richts. Methods - the basic units for planning and verifying proofs. In *Proceedings of Jahrestagung für Künstliche Intelligenz KI-94*, Saarbrücken, 1994. Springer.

[16] D. Hutter. Guiding inductive proofs. In M.E. Stickel, editor, *Proc. of 10th International Conference on Automated Deduction (CADE)*, volume Lecture Notes in Artificial Intelligence 449. Springer, 1990.

[17] D. Hutter. Equalising terms by difference reduction techniques. RP –, Univ. of Edinburgh, Department of AI, Edinburgh, 1996.

[18] D. Hutter. Using rippling for equational reasoning. In *KI-96: Advances in Artificial Intelligence. 20th Annual German Conference on Artificial Intelligence*, volume 1137 of *LNAI*, pages 121–133. Springer, 1996.

[19] S. Kambhampati. A comparative analysis of partial-order planning and task reduction planning. *ACM SIGART Bulletin, Special Section on Evaluating Plans, Planners, and Planning Agents*, 6(1):16–25, 1995.

[20] S. Kambhampati, C. Knoblock, and Q. Yang. Planning as refinement search: A unified framework for evaluating design tradeoffs in partial-order planning. *Artificial Intelligence, special issue on Planning and Scheduling*, 76:*, 1995.

[21] S. Kambhampati and B. Srivastava. Universal classical planner: An algorithm for unifying state-space and plan-space planning. In M. Ghallab and A. Milani, editors, *New Directions in AI Planning*, pages 61–78. IOS Press, Amsterdam, Oxford, 1996.

[22] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the AAAI-96*, pages 1194–1201. Morgan Kaufmann, 1996.

[23] C. A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68:243–302, 1994.

[24] G. Kreisel. Mathematical logic. In T. Saaty, editor, *Lectures on Modern Mathematics*, volume 3, pages 95–195. J. Wiley & Sons, 1965.

[25] U. Leron. Structuring mathematical proofs. *The American Mathematical Monthly*, 90:174–185, 1983.

[26] E. Melis and A. Bundy. Planning and proof planning. In S. Biundo, editor, *ECAI-96 Workshop on Cross-Fertilization in Planning*, pages 37–40, Budapest, 1996.

[27] S. Minton, C. Knoblock, D. Koukka, Y. Gil, R. Joseph, and J. Carbonell. *PRODIGY 2.0: The Manual and Tutorial.* School of Computer Science, Carnegie Mellon University, Pittsburgh, 1989. CMU-CS-89-146.

[28] D. Plaisted. Theorem proving with abstraction. *Artificial Intelligence*, 16:47–108, 1981.

[29] E.D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[30] A. Tate. Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 888–893. Morgan Kaufmann, 1977.

[31] R. Washington and B. Hayes-Roth. Incremental abstraction planning for limited-time situations. In M. Ghallab and A. Milani, editors, *New Directions in Planning*, pages 91–102. IOS Press, Amsterdam, Oxford, 1996.