



# Decision planning knowledge representation framework: A case-study

Michal Pěchouček

*Gerstner Laboratory for Intelligent Decision Making and Control, Czech Technical University in Prague,  
Technická 2, 166 27 Prague 6, Czech Republic  
E-mail: pechouc@labe.felk.cvut.cz*

This paper discusses experiences and perspectives of utilisation of declarative knowledge structures as a convenient knowledge base medium in configuration expert systems. Although many successful systems have been developed, these are often difficult to maintain and to generalize in rapidly changing domains. In this paper we address the problem of building intelligent knowledge based systems with emphasis on their maintainability. Firstly, several industrial applications of *proof planning*, a theorem proving technique, will be described and their advantages and flaws will be discussed. This discussion is followed by the theoretical foundation of *decision planning knowledge representation framework* that, based on proof planning, facilitates separate administration of inference problem solving knowledge and the domain theory axioms. Machine learning methods for maintaining the inference knowledge to be up-to-date with permanently changing domain theory are commented and evaluated.

**Keywords:** industrial configuration, theorem proving, machine learning, multi-agent systems, expert systems

## 1. Introduction

The objective of this paper is to show how declarative knowledge structures based on *proof planning*, a theorem proving technique, may be used as a convenient knowledge base medium in the area of automated configuration. Configuration is a complex task generally involving varying measures of constraint satisfaction, optimisation, and the management of soft constraints. Although many successful systems have been developed, these are often difficult to maintain and to generalize in rapidly changing domains. In this paper we address the problem of building intelligent knowledge based systems with maintainability well to the fore in our requirements for such systems. Central to our approach is coupling proof planning structures together with machine learning methods that allows us to design and maintain a knowledge base that keeps strategic problem solving knowledge separate from the domain knowledge specifying a concrete course of decision-making. Separate administration of these two classes of knowledge and automated induction of the system's internal working knowledge base is what makes such a knowledge-based system easy to maintain and en-

hance with new pieces of knowledge and at the same time facilitates efficient consultation.

In the first part of the paper we will briefly explain the concept of proof planning. Follows a survey of application of conventional proof-plans in the areas of computer network configuration and breathing compressors production. We will analyse main advantages and show why maintainability is an important flaw of this approach. Consequently, we will introduce the concept of decision-planning graphs, a variant of proof plans, that keeps maintainable expert knowledge separate from the systems' efficiently working knowledge base. Explanation based generalisation machine-learning algorithm, that is an inseparable part of the decision planning methodology, maintains causal connection between the system working knowledge-base and the maintainable knowledge represented in the form of decision graphs. The concept of strong and weak update of the knowledge will be presented in the same section. The last section describes practical application of the decision planning methodology in the area of TV-transmitter production.

### 1.1. Knowledge classification

Knowledge that enables a knowledge based system to present really intelligent expertise can be classified within two distinct directions: (1) *knowledge source* and (2) *knowledge orientation*. There are principally two primary sources of knowledge that may be obtained. *Empirical knowledge* is mainly based on direct observation of the domain in question. This kind of knowledge is highly effective, but suffers from a lack of generality. Sometimes it is difficult to elicit empirical knowledge. *Theoretical knowledge* is based on scientific laws and principles. Contrary to empirical knowledge this type of knowledge is general but not very effective when used for automated decision making. Alongside the orientation dimension we identify three distinct classes of knowledge:

- **Object-level (domain) knowledge** – (knowing *what*, static information, data) specifies ontology of the problem in question. Object knowledge defines in terms of domain attributes properties the world we want the system to reason about.
- **Heuristic knowledge** (knowing *what if*) identifies mutual relationships within certain clusters of object knowledge. Heuristic knowledge helps to organise pieces of object-level knowledge in order to facilitate efficient and human like decision process simulation.
- **Meta-level (inference) knowledge** (knowing *how*, strategic knowledge) describes the human expert reasoning process. Meta-level knowledge reflects reasoning knowledge about knowledge at the object level. This is the core of knowledge elicitation and this is what makes an expert system capable of intelligent reasoning. This type of knowledge is multi-level so that meta-knowledge may be knowledge about meta-knowledge at a lower level.

Accordingly, when understanding and formalising the decision process, we are supposed to elicit as much knowledge about the area under investigation as possible. Local domain theory needs to be defined and distinct and case specific relations among problem entities need to be recognised. Finally, knowledge how to process these pieces of knowledge and how to draw sound conclusions from an axiomatic theory must be identified.

Throughout the article we will be investigating formal models of the meta-level inference knowledge and how it relates to heuristic and object level classes of knowledge. The subject of our investigation will be knowledge from empirical source.

### 1.2. The problem of industrial configuration

We have considered the problem of industrial configuration as an illustrative example of practical decision making in a well-circumscribed domain. Configuration, as a specific type of decision making, is defined as a problem of assembling elements of the system together in such a way that internal logical constraints are not violated [22]. Very often some kind of optimisation criteria, such as price or efficiency, is also considered. This is why we distinguish between two types of constraints: *hard constraints*, such as logical and spatial specification and *soft constraints*, such as performance or price. The extent to which soft constraints, unlike hard constraints, which must not be violated, is subject of respective optimisation (more it will be violated the less profit we get).

There is no need to emphasise that the problem of configuration is of a computationally explosive nature and applying enumeration based approaches in real life applications is hardly possible.

Configuration is understood in terms of four key elements [20]:

- **Specification language** defines the specification of the solution that needs to be satisfied. The environment nature and the specific use of the system is reflected here. A specification language may include optimisation criteria that guide the search.
- **Sub-model of parts** represents a catalogue of parts. This sub-model also describes the mutual interdependencies. Hence when a particular component is configured, there are links to all other components necessary to consider.
- **Sub-model for spatial arrangements** specifies the means for describing spatial arrangements of the components and thus defines which combinations of components are feasible.
- **Sub-model of sharing** expresses conditions under which a component can satisfy more than one set of requirements. There can be *exclusive use*, *limited sharing*, *unlimited sharing*, *serial reusability* and *measured capacity*.

A precise definition of configuration is usually highly case specific. It depends on the nature of the task and of the techniques used. The process of how these key elements defined above are manipulated when carrying out automated configuration is twofold.

This is what defines two fundamental approaches to configuration:

- **Algorithm intensive approaches**, where some of artificial intelligence techniques are applied to the process of intelligent and effective state space search (i.e., CLP – Constrain Logic Programming); or
- **Knowledge intensive approaches**, where the effort is devoted to an intelligent, very often human like, representation of problem solving knowledge (i.e., rules, frames, scripts, etc.).

This paper addresses application of the latter approach to the problem of industrial configuration through designing and implementing proof planning like methodologies for industrial decision making.

According to Najman and Stein configuration is defined as a mathematical structure with a set of objects, a set of properties for each object, a set of functionalities, a set of values for each functionality and a set of demands [14]. The process of configuration is thus viewed as a finite sequence of compositions of objects whereas the solution is a configuration object that satisfies the demand. Some authors have given a logic based description of configuration tasks [3]. The development of logical formalisms guarantees the soundness of resulting solutions. The particular constructive type theory of configuration developed by Lowe goes further in that sound configuration objects are synthesised from the specification of such an object [5]. Many authors have pointed out the maintenance problems faced when managing systems in which product information changes often [21]. Mannisto proposes a generic structure model to support different views and classifications of the same components evolving over time [9]. In our view, the problem he mentions, that the original engineers may not understand the way the products are described in the system, is a consequence of mixing different kinds of knowledge. Our approach necessitates a clean separation of object-level, heuristic, and control (strategic) knowledge, which could be separately maintained with the aid of appropriate user interfaces.

## 2. Proof Planning and industrial configuration

Proof Planning is a technique for the global control of search in theorem proving systems. The main features of proof planning were developed for the domain of theorem proving. The proving of a mathematical theorem is just an application of a set of logical rules from the theory in question. The hope constructing a proof of the desired conjecture is the main motivation of this activity.

Whilst allowing a sufficient degree of flexibility and adaptability to prove a large variety of different kinds of theorems, the proving strategies themselves are expressed as proof plans by describing *Tactics*, *Preconditions* and *Effects*. *Preconditions* are declaratively specified pieces of information, under which *Tactics*, mainly procedurally stated pieces of code, are applicable. *Effects* describe changes or the progression of the solution if the *Tactics* are successfully applied.

The specifications of *Tactics* in terms of *Preconditions* and *Effects* are called *Methods*. The proof planning process itself is then searching for a proof of the desired conjecture by means of finding a sequence of tactics that if successfully applied will lead to the conjecture in question. If such a sequence, a proof plan, is found, tactics recorded are carried out and thus the solution itself is specified. Consequently, all the time consuming and highly branched reasoning is carried out just once along the path through the tree that was specified by planning. Describing tactics in terms of preconditions and effects can be understood as a kind of meta-level reasoning about further specified pieces of quite expensive computation.

Another virtue of proof planning with respect to its use for Knowledge Based Systems is the separation of factual knowledge and control knowledge. Factual knowledge simply represents the domain theory in terms of a set of axioms and a set of inference rules. Control knowledge describes the inferences in the process of meta-level reasoning about the problem. This fact precisely suits the needs of Knowledge Engineering.

There are three subsequent phases of computation within a proof planning oriented application:

- **planning phase** – where a meta-plan of how to prove a theorem is offered;
- **validation phase** – a plan is either confirmed or refused here;
- **execution phase** – all tedious lower level pieces of inference are being carried out here.

At the University of Edinburgh and at the University of Saarbrücken researchers developed a number of proof plans and libraries of tactics for the area of theorem proving [2,5,6]. Attempts were made to show that proof planning can be applied in other domains as a methodology for expressing strategies of the state space search. We comment below on successful use of proof plans on a computer configuration task and in the area of breathing compressors configuration [7,8,18].

### 2.1. Configuring computer networks via proof planning

The first attempt to utilise the proof planning programming methodology in a knowledge based system other than theorem proving dates back to 1993 [5].  $CL^E M$  is a prototype of the configurator that was designed to tackle the problem of configuring hardware so that customer specifications are met. A subset of Hewlett Packard HP3000 series systems component was used as a test case.

Proof planning has been taken as a pilot methodology for  $CL^E M$  implementation since the problem is of a typical synthesis nature, and there is a large amount of problem solving skills available. Researchers relied on suitable knowledge separation. The following classes of knowledge were distinguished:

- **object-level domain knowledge** – list of components, their properties and means of connection within the computer network configuration;

- **heuristic knowledge** – problem solving shortcuts that represents heuristic relations among instances of object-level knowledge (i.e., “*connecting the printer to the same channel as the disk makes the system slow*”);
- **meta-level knowledge** – strategic knowledge that formalise configuration making procedures in the area of computer networks (i.e., “*let us first decide about the processor and then specify the configuration of disk drives*”).

The motivation of the configurer was to synthesise an *optimal, legal* configuration as a result of the given initial constraints. The *legality* of the solution was viewed through *hard constraints*. The problem-solving goal was formalised in the theorem proving fashion. The technique was analogous to **constructive proof**<sup>1</sup> in the program synthesis technique [1]. The given relationship between *input* and *output* synthesise an algorithm *alg* as follows:

Regarding a field theory  $T$  find a constructive proof of theorem

$$\vdash \forall input. \exists output. spec(input, output),$$

and synthesise an algorithm *alg* from this

$$\vdash \forall input. spec(input, alg(input)).$$

A computer configuration that satisfies the specification  $spec(c)$  is to be synthesised as a by-product of the *constructive proof* of a theorem as follows:

$$\vdash \exists c. spec(c).$$

An example of specification  $spec(c)$  may be

$$spec(c) \equiv (processor(c) = specific\_proc(N)) \\ \wedge (capacity(disk(c)) \geq 500)$$

which says that we are entitled to select only processors which belong to a family of special processors and that the disk capacity must not be less than 500 memory units.

The set of tactics was implemented as lower level configuration steps. Their syntax is similar to tactics used in mathematics. An example of a tactic `configure_device/3` is shown below [6].

```
configure_device(Device, IC, C) :-
  connect_cable(Device, Cable, C),
  connect_via(Device, IC, C),
  type([Device, Cable, IC], _).
```

The argument `Device` specifies a device to be configured, the argument `IC` specifies an interface channel and resulting configuration we are to synthesise is viewed as variable `C`. There are certain preconditions and given modes specifying when this tactic

<sup>1</sup> *Constructive proof* is such a proof for which proven existence is not sufficient. The actual instantiation of free variable must be produced.

Table 1  
*configure\_device* method.

Name	<i>configure_device</i>
Input	<i>configure(Device, IC, C)</i>
Preconditions	<i>Device</i> is a device of <i>C</i> and <i>IC</i> : $\tau$ and <i>Device</i> needs slot of type $\tau$ and the number of slots available of type $\tau$ is <i>n</i>
Effects	The number of slots of type $\tau$ available is <i>n</i>
Output	Nil
Tactic	<i>configure_device(Device, IC, C)</i>

may be used. The method shown in table 1 indicates the meta-representation of this piece of problem solving knowledge.

As previously stated the system has been tested on HP 3000 systems. Series of tests were performed in order to find out the extent of maintainability and efficiency. Maintainability was successfully tested by adding a new processor or another component. Results of runtime statistics demonstrated the system produces legal and sub-optimal configurations in a reasonable amount of time. For detailed results see [6,8].

## 2.2. Proof planning and compressor configuration

Another pioneering application, which has utilised theoretical and practical experience from the area of automated theorem proving was a case-specific compressor configurator for CompAir Reavell, Siebe plc., the leading UK compressor manufacturer [18]. Researchers have tried to use proof planning for formalising certain inference knowledge in order to limit the space of possible configuration. Manipulating methods, meta-logical operators facilitate preliminary planning of the following more specific problem solving activities such as creating a construction drawing or product identification.

In the **planning stage**, using either a *user assisted planner* or an advanced planner, the system manipulates the meta-description of particular pieces of compressor configuration so that the user-defined structure can be specified. After the **validation stage** – soft constraints evaluation, the system triggers the **execution stage**, where tactics of each of selected methods are executed as pieces of inference on how to create the product identification number and the product drawing, respectively.

There have been two distinct super-methods implemented. One method describes the User Assisted Planner and the other the Advanced Planner. As the quotation process basically appears as a sequence of three subsequent activities – compressor engine specification, budget specification, accessories specification – we have designed an algorithm that will be represented in meta-terms by means of the super-method, the tactic of which

is as follows:

```
user_assisted_planner(Solution):-
    set(Engine),
    check_budget_of(Engine),
    accessories(Engine, Solution).
```

At the highest level, the ordered set of methods represents the decision process in question. When configuring the compressor, the entire quoting process can be viewed as a more or less structured ordering of the decisions to be made. Each decision is represented by a single method. A method in proof planning is a meta-logical description of what could happen, under what circumstances, which are prerequisites of a certain course of action and finally what will result from these actions. A global run of the program is nothing but (1) an appropriate instantiation of applicable methods (automated or user assisted), i.e.: decisions to be taken and (2) subsequent computation that will create the production number (drawing).

Preconditions of a method are intended to record all actions that need to be carried out before deciding whether a particular branch of a sub-tree suits the properties of a given sub-solution. The user dialogue, an unusual and non-standard action, theoretically needs to be prompted every time when making a decision on which way to head to. In case when the domain constraints permit only one possible alternative, the system does not need user assistance and thus does not prompt a question.

The object knowledge has been formalised by means of the ladder logic,<sup>2</sup> a well-known industrial representation. We decided to use this formalism mainly because of its simplicity, which is a desired feature when updating a rule-base by non-expert people. A parsing mechanism that understands an arbitrary ladder logic expression has been implemented. Consequently the knowledge engineer may use very messy expressions.

In spite of the fact that time complexity seems to be of a quite explosive combinatorial nature, the system behaves very reasonably as because of the clear separation of the independent pieces of inference represented by methods. Although object level and meta level knowledge are nicely separated and the system is maintainable and easy to enhance, the system is expected to behave slowly enough with increasing complexity of the knowledge bases. The reason is obvious. Each time when carrying out a decision the domain theory needs to be consulted. Consequently time requirements for searching through the domain theory is an important factor which determines an overall complexity of the algorithm.

This is the main issue we wanted to address: (i) how can we keep nice maintainable proof-planning like knowledge structure to guide systems decision making and (ii) at the same time the system does not need search through complicated domain theory each time in tries to make a decision. We have investigated how to construct a special, effective systems knowledge base that will make the decision making process more efficient. Two

---

<sup>2</sup> In the ladder logic we represent truthful formula (possible configuration) as a path from the root of the tree to a specific leaf node. Every single node on this true formula must be truthful. Conjunction is expressed by serial connection of the node, while for disjunction we make two parallel branches.

different knowledge-bases are used: one for the user's maintenance (well structured, maintainable) and the other is used by the system when carrying out consultation. The below explained algorithm propagates the updates between the knowledge structures. The argument why we did so is simple. Why do not we spend substantial computational resources (time) for maintaining the consistency of our knowledge base with respect to updates, provided that we will save time when carrying out consultation. We do not mind waiting some time when we want to learn the system that a new product is to be incorporated in its knowledge base, while we can get easily annoyed giving the system several seconds between questions it asks.

For the ICON system implementation, explanation of planning algorithms and results please refer to [8,18].

### 3. Decision planning knowledge representation framework

Let us describe the solution by means of a finite set of attributes and their correspondent values. Each state within the state space is represented by a partially constructed solution, i.e., a specialisation of the solution above. There is a sub-space of the state space, for which the states represent valid solutions. Let us call these goal states. The motivation of this problem solving process is to find such a goal state that does not break any hard constraints and minimises the extent to which soft constraints fail to be kept. In order to formalise the inference knowledge that would guide us through the process configuration, we consider the following state space (in terms of a state space node, state space state, goal state, etc.) [15]:

**Decision node**  $N$  (decision), a single state in the representation of the state space, describes an instantiation that enriches a partial sub-solution  $T_1$  with another property expressed in terms of an attribute and its value:

$$N \equiv T_1 \rightarrow T_2.$$

Instead of a partial solution we regard a decision, a solution refinement, as a problem-solving primitive that constitutes the state space. The decision node, a fundamental primitive of decision planning, is a meta-representation of  $T_1 \rightarrow T_2$  transformation. It either specifies in meta-terms further lower level problem solving process and/or triggers an external function, predicate, or method which carries out a required decision, for example selecting an optimal value of an attribute or prompting a user. For this simpler case we define a decision node by a couple

$$N \equiv \langle A, V_i \rangle.$$

**Decision path**  $T$  is a legal sequence of decisions taken within a defined decision space and with consideration of initial requirements. It is defined as an ordered set of attributes and their values:

$$T = \langle \{ \langle A_i, V_i \rangle \}, O \rangle,$$

where the couple  $\langle A_i, V_i \rangle$  stands for an attribute and an appropriate value set by a decision and  $O$  is the ordering of the set, where  $\forall \langle A_i, V_i \rangle: N_i \equiv \langle A_i, V \rangle, V_i \in V$ . The decision path  $T$  corresponds to a partial sub-solution.

**Decision space**  $D$  is a collection of all possible decision paths with respect to a current decision path. There exists a subset of a decision space which contains solutions – **goal decision paths**.

**Decision plan**  $P$  is then a decision space specification, describing how a certain decision space evolves throughout the course of decision making. Searching a decision plan is viewed as length-first-search algorithm, where decision nodes on a top-most level are parsed and the tactics of each of decision are collected. When an optimal arrangement of decisions (solution) on this particular level is formed the algorithm processes the collected tactics in the same manner (one level lower).

**Decision planner** is a general purpose expert system shell, based on decision planning knowledge representation methodology, that uses a case specific decision plan in order to simulate an expert-like decision making activity.

**Decision planning** is a methodology for creating decision plans describing an arbitrary synthesis-like decision making activity.

### 3.1. Decision plan

The decision plan can be either abstract or specific. While the abstract decision plan is accessible to and maintained by the knowledge engineer or the user administrating the system, the specific decision plan is a special form of knowledge base that navigates the consultation provided by the system.

*Abstract decision plan* describes conceptual structure and relationships among particular decisions. It captures only pure inference meta-knowledge with no consideration of case specific object-level knowledge. The abstract decision plan is described by an oriented graph – **decision graph** – where each node is a meta-representation of a lower graph. The decision node is defined as a quadruple  $\langle \text{Decision}, \text{Level}, \{\text{Output}_i\}, \text{Tactic}_i \rangle$ , where a Decision at a certain Level of the decision plan describes a lower level sub-plan pointed to by Tactic. In the context of the current graph the Output defines possible edges leaving the node. When a Prolog algorithm parses the decision graph it takes decision by decision alongside the graph and collects all tactics of decisions it gets through.

*Specific decision plan* is a case specific implementation of the abstract decision plan. Apart from meta-level knowledge, the specific decision plan also incorporates heuristic knowledge. The heuristic knowledge (in the form of domain theory generalisation) is needed for navigation through the decision space. Each decision should become enriched by information specifying whether this node may be taken (Precondition) and how taking this node affects the decision path (effects). The decision node in the decision graph is then defined as a quadruple  $\{\langle \text{Attribute}, \text{Level}, \{\langle \text{Precondition}_i, \text{Effect}_i, \text{Tactic}_i \rangle\}, \{\text{Output}_k\} \rangle\}$  with the third element denoting possible variations of the decision. When consulting the decision graph, Prolog takes a decision node and tries to identify a valid  $\langle \text{Precondition}_i, \text{Effect}_i, \text{Tactic}_i \rangle$  triple,

updates the decision path with `Effecti` and stores the `Tacticsi`. If it fails, another possible decision node is taken instead. `Effect` updates the decision path with the value of the desired attribute. An `Effect` could be either:

- an atom, where a couple `<Attribute: Effect>` gets appended to the current decision path (there is neither choice nor optimisation, decision is made automatically);
- or an empty list, when only a `Attribute` gets appended to the current decision path in this case (this decision does not affect directly the resulting configuration, it only keeps information how the decision path was constructed);
- or an algorithm-list couple, where `Effect`  $\equiv$  `Algorithm:List` and `Algorithm`  $\equiv$  `predicate(List, Item)` – a couple `<Attribute: Item>` gets appended to the current decision path.

**Example.** The triple below is an example of decision node in the abstract decision plan. This decision node specifies a required power of the transmitter to be configured.

```
<power, level-1, {site, access}, {engine, mains}>.
```

It simply says that the on the level `level-1` we can specify the power requirements of the solution prior to giving specifics of transmitter `engine` and `mains` specification (output nodes). Specifying power requirements consists of specifying the `site` arrangement and specification of accessories. These are links to tactics in the form of lower-level decision graphs.

The following two sextuples show instantiation of the same decision node in the specific decision graph:

```
<power, level-1, Input, {transmitter: tv ∈ Input}, {site, access},
  Input ∪ power: power ∈ {15 kW, 20 kW}, {engine, mains}>;
<power, level-1, Input, {transmitter: fm ∈ Input}, {accessories},
  Input ∪ power: power ∈ {5 kW}, {engine, mains}>.
```

In the first case we are to configure the power of a TV transmitter as given in the third element of the sextuple – `{transmitter: tv ∈ Input}` – the couple `<transmitter: tv>` will be instantiated within the parameter `Input` that specifies partially constructed decision path. The decision path, which is in the form of a set of couples `{<attribute: value>}`, is enriched by the couple `<power, 15 kW>` or `<power, 20 kW>`. This decision is made either by the system itself according to some optimisation criteria or the user is prompted with a query if both options are found equivalent. Methods that could be taken subsequently to this one could be either `site` or `accessories`. In the second case we configure an FM transmitter. The solution gets appended with the couple `<power, 5 kW>`. With an FM transmitter there is no need to configure a `site` of the solution, so the only possible successor is the `accessories` method. Both methods describe a series of tactics – `engine, mains` in a meta-representation.

The main motivation for splitting the base of inference knowledge into the abstract decision plan and specific decision plan is as follows. Instead of maintaining a knowledge base of inference rules (in arbitrary form) we wanted to keep separate the object level data and inference knowledge. The knowledge engineer is thus required to maintain only the domain knowledge and the abstract decision plan where the inference knowledge is encoded. If a change in the object level knowledge occurs (which is very often the case) the abstract decision plan is not affected. The specific the decision plan is a knowledge base medium, which is (i) hidden from the user's point of view, (ii) facilitates case specific reasoning and (iii) is updated and maintained automatically. The process of automated decision plan updating will be explained below. For the full definition and thorough explanation of the concept see [15].

#### 4. Machine learning in decision planning

There are two principal issues we must address when designing an industrial knowledge based system. The system must be first **efficient**. Internal reasoning processes must be performed in polynomially bound time. There is no point specifying a general state space formalisation and then letting the system search through the space in exponentially explosive time. Certain heuristic and meta-level knowledge must be considered in order to make the system operate in reasonable time. On the other hand the system must be **maintainable** and easy to enhance. Object data changes often and frequent changes of the knowledge base are thus inevitable. Therefore the knowledge base must be in a certain way standard, separate and general so that further maintenance would be facilitated. Otherwise it does not make sense to implement a knowledge-based system instead of a fixed and super-efficient conventional algorithm.

We first tried to solve the problem of updating the decision plan with changing domain knowledge and then we generalised the approach by updating the *Abstract decision plan* with domain knowledge so that we came up with a *Specific decision plan*. The system then creates a ready-to-use knowledge base – the database of decisions, the specific level of the decision plan, which is used throughout a consultation. This is how we kept the inference meta-knowledge of the expert, encoded as *abstract level plan*, separate from the object-level knowledge about the domain.

##### 4.1. Decision plan extension and circumscription

Let us call the process of creating an appropriate specific decision plan *decision plan induction*. We then distinguish between *decision plan extension* and *decision plan circumscription*. When the object theory becomes more exhaustive, i.e., describes a larger scope of possible decision paths than does the decision plan, we need to enhance the decision plans, accordingly. Here we are speaking about *decision plan extension* (i.e., introducing a new product to be manufactured or a considering new component available). Under the term *decision plan circumscription* we understand the case, when the object theory update makes the decision plan to accept fewer decision paths than

it did before. We use *decision plan circumscription* in order to run consultation with respect to some predefined constrains.

The *decision plan extension* algorithm is based on a simple variation of the EBG method – Explanation Based Generalisation [13]. Rather than using a number of examples in order to form some sort of statistical generalisation, EBG enables the system to form a justified generalisation of a single positive training example provided the learning system is endowed with some explanatory power. According to Mitchell the EBG method is defined as follows:

```
for each positive example:
{
  explain(positive example) → explanation
  generalise(explanation, theory) → theory
}
```

where the first step (*explain*) constructs an explanation that proves how the positive example satisfies the given concept and the second step (*generalise*) determines a sufficient condition under which the explanation structure holds.

We have modified the algorithm so that EBG is used for generalisation when the system is presented with a negative example instead of a positive one (any example which is not accepted by the decision plan and is not explicitly stated as positive is understood as negative). The system finds an explanation that justifies why the example does not satisfy the given concept (*explain*) and then proposes such an update of the concept that the example will be satisfied by the example (*generalise*). The proposed update is in the form the list of methods with different precondition and effect that extends the former methods so that the new example can be accepted.

We have tested this algorithm on an extreme case where the system is presented with an abstract decision plan and a series of positive examples, we wanted the system to consider when configuring (see figure 1). The successful outcome of these tests (commented on in section 5.3) enables us to claim that inference knowledge, in the form of abstract decision plan and object-level knowledge may be administered separately.

*Decision plan circumscription* is used in cases when a given field theory needs to be limited. It is very often the case that consultation with the system will be triggered when considering an initial set of constraints. Configuration-oriented expert systems usually carry out the configuration with respect to the constraints, which requires the system at

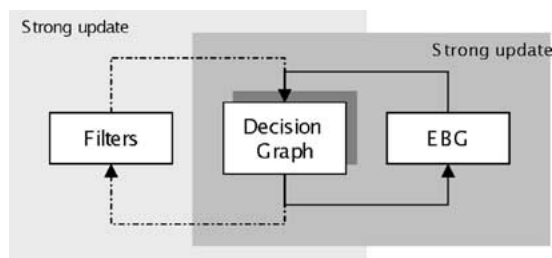


Figure 1. Decision planning knowledge evolution lifecycle.

each decision node to parse down the remainder of the decision tree in order to find all possible legal effects resulting from the decision node in question. This makes the computation time-expensive and the course of consultation much too long. Here we propose a novel approach. Instead of browsing the entire decision space let us circumscribe it first and browse only the fraction we need to.

Here we propose using *decision plan circumscription*. The process of doing so is quite straightforward. First the object theory is backed up and a copy is created. *Effects* and *Conditions* in the decision plan are set up to the value of an empty list (i.e., an abstract level of a decision plan is considered). All the data in the object-level knowledge that contradicts the initial constraints get erased and only the valid knowledge with respect to these constraints constitute object-level knowledge. According to this knowledge a new decision plan, a circumscribed one, is created. Here we have used the algorithm of *decision plan extension*. Such a decision plan is ready for use for circumscribed consultations. The initial decision plan must be downloaded after finishing the circumscribed consultation.

#### 4.2. Knowledge maintenance lifecycle

An interesting problem is how to assure appropriate evolution of system's knowledge with respect to changing data. The degree to which an expert system presents correct behaviour depends on the degree of correctness of the knowledge base. This is why the system knowledge base shall evolve in correspondence to the changes in the field theory. We have experimented with inference knowledge encoded in decision graphs and investigated how EBG updates this type of knowledge according to evolving domain theory.

##### 4.2.1. Weak and strong update

Suppose that the knowledge base  $kb$  allows the system to reason about a domain  $E$ . Domain  $E$  would be expressed as a couple of sets of objects and relations among them  $\langle \{O_i\}, \{R_j\} \rangle$ . Let the domain be extended by a new relation  $R$  such that  $E \cap R = \emptyset$ . The knowledge base –  $kb$ , domain  $E$  meta-representation, has to be updated/revise in such a way that it allows the system to reason about the domain  $E \cup R$ . Two alternative approaches can be taken in order to implement this type of functionality.

- **Strong update** is result of complete reconstruction of the entire inference knowledge base with consideration to new updated field theory (i.e., the current field theory enriched by the new relation) as a knowledge induction parameter. Application of a knowledge elicitation (machine learning) algorithm  $f$  on the full set  $E \cup R$  results in a strong update. Construction of the new knowledge base  $kb'$  (by means of an algorithm  $f$ ) is expressed as  $kb' = f(E \cup R)$ .
- **Weak update**, however, re-computes just the relevant parts of the inference knowledge base in order to ensure that the result stands for a sound and complete representation for the updated field theory  $E \cup R$ . The knowledge elicitation algorithm

takes the new relation  $R$  and current domain meta-representation  $kb$  as knowledge induction parameter. Formally  $kb' = f(kb, R)$ .

Most often it is much simpler to generate a weak update than the strong one. On the other hand, it is not very clear what efficiency will perform the system using the changed (weakly updated) knowledge base. Weak update patches the local requirements of the decision space but it does not consider its overall shape. That is why weak update is computationally less expensive, but the resulting space of configuration-like meta-level knowledge can be expected to be quite messy. Consequently, there can appear inefficient consultations in such a space often.

When deciding between strong and weak update, requirements for efficiency of parsing the resulting knowledge base  $kb'$  have to be balanced with complexity requirements for the process of updating the knowledge base  $kb$ . The latter criterion should meet the demands of the dynamic properties of the application domain (such as estimated frequency of knowledge base updating and its effect on the quality of the decision space described by inference knowledge). When the revision is not required very often and strong update is not extremely time consuming then the strong update is generally recommended. When the field theory changes instantly we might be lacking resources for the strong update.

In order to illustrate this we may view the problem from purely AI point of view. Let us consider a simplified meta-knowledge decision space in a form of a decision tree, where each product configuration is a branch from a root of the tree to a certain leaf node (goal). Here, a weak update means appending a simple branch to the root of the tree. By doing so the numbers of nodes considered as well as the branching factor of the new state space increases. In this way the knowledge hidden within the original decision tree loses its significance. Using a tree with an increased branching factor or with more decision nodes results in higher memory requirements, moreover, it makes the process of state space search slower and more difficult. That is why we try to update the decision space in such a way that (1) time needed and (2) the increase of the state space size and of its branching factor are as little as possible.

Neither pure weak update, nor pure strong update guarantees optimal knowledge base maintenance. Rather than implementing a 'middling' update we seek for a compromise by playing around with frequencies of both weak and strong updates. If the weak update does not result in significant decrease of efficiency, the strong update can be done only occasionally (after several steps of weak updates). Need for strong updates and their frequency has to be determined with respect to the conditions of the application domain. In the following we will show how does the concept of weak and strong update relates to explanation-based generalisation incorporated within decision planning. For full explanation of the concept of strong/weak update see [19].

#### 4.2.2. Decision planning knowledge maintenance

Decision planning takes a constant number of steps for any consultation within the fixed task domain. This is due to the topology of the decision plan (the abstract decision plan), which remains fixed in such a case. No matter how many examples are covered,

the user has to be taken through an identical decision path, i.e., through the same number of decision nodes. Requested time does not in practice depend on the cardinality of the set of field theory examples, which represent the set of all positive examples. Checking preconditions of a decision node can become more time consuming. But a well-designed abstract decision plan minimises the extent to which the time requirements grow with increasing number of covered examples.

The nature of decision planning knowledge representation methodology is such that it does not offer any means for a strong update. System knowledge base can be either directly supplied with inference knowledge specified by user or step-by-step induced from the field theory using Explanation Based Generalisation (EBG) technique, which cares for weak updates. The knowledge maintenance lifecycle is then a sequence of weak updates. Instead of a strong update (figure 1), we have implemented algorithms for decision space filtering, aimed at reducing the number of decision nodes and the branching factor. We distinguish between following two filters:

- **Conjunctive filter** – clusters decision nodes with the same effect slot; they get unified through conjunction of corresponding preconditions;
- **Frame filter** checks relevancy of each of preconditions in order to avoid the frame problem [1]. The algorithm has to be provided with the entire possible range of values each attribute can take. Possible clusters of decision nodes having the same effect and precondition describing entire discourse are eliminated.

Time needed to learn a single example depends on the structure of the actual decision space (compared to the example to be accepted) and on the amount of the nodes to be parsed. The first objective makes an example to be learnt more difficult in the beginning of the learning process and the other objective makes it more time consuming with increasing number of accepted examples. Apart from the first 30 examples the time needed for learning is linearly proportional to the size of the decision space for our data set. When filtering the decision space the nodes have to be compared one to another (to put it simpler), thus the complexity is almost proportional to the square of number of nodes. The space of decision nodes does not grow with increasing number of positive examples proportionally. It saturates at a certain point and consequently filtering takes just certain time (seconds) but does not grow up to infinity. Experiments showed that the best practice in maintaining the knowledge base consistent with respect to frequency of updates corresponds to the rhythm in which object-level data come. As each new piece of product has got usually about 30 new variants (4–6 new attributes) we were filtering the knowledge base after each 32 arrivals of new object-level data. This mechanism provides the knowledge base with filtered data ready for consultation whereas it minimises the time needed for knowledge induction. The solid line in the figure 2 gives the size of the decision space after several tens of weak update iterations (in terms of number of decision nodes). The effect of proposed filtering can be seen on the dashed line in the figure 2.

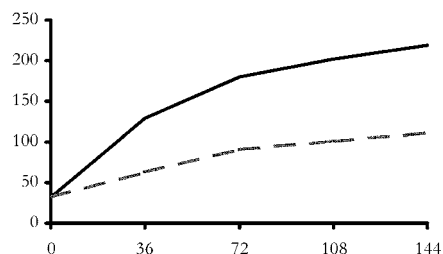


Figure 2. Positive accepted examples number of nodes in the decision space.

#### 4.3. Further perspective of machine learning in decision planning: discussion

We have illustrated that it is worthwhile to consider decision-planning decisions as dynamic, updateable entities rather than fixed structures. By introducing a novel methodology, decision planning, we argue that separating the inference meta-knowledge into an abstract, context-free level and the specific level induced from the abstract level and field theory facilitates independent maintenance of both object level data and inference knowledge. We have proposed coupling machine learning methods with meta-level programming techniques in order to compose optimal decision plans. Such a decision plan is expected to regard both (1) expert problem solving skills and (2) the object theory in question. From the philosophical point of view we have spotted three main areas to use explanation-based generalisation within decision planning [17]. We can:

- update the precondition/effect attributes through negative examples;
- update the decision plan structure through self-analysis;
- update the decision plan structure through positive examples.

In section 4.1 we describe how preconditions and effects are updated through a negative example. It is possible to start the learning process with empty precondition/effect attributes and to teach the decision plan to accept a set of positive examples

Apart from knowledge captured in the abstract level of the decision plan another kind of knowledge may be revealed within the data. It is often the case that expert problem solving skills are not refined enough or the data, the object level knowledge, do not suit the inference knowledge entirely. Precondition/effects may be learned such that an inefficient type of decision plan is created. Once in a while, either after each single example gets learned or when a certain kind of measure threshold is crossed, the system will analyse each method in terms of precondition/effects attribute. As one of the ways of doing so we suggest that the instances of a decision be reorganised anytime the conditions in the different instances do not involve the same atomic predicates, or whenever there are decisions within the path that are not included in the preconditions. Another way of refining the decision plan is to detect clusters of identical courses of decisions within the decision path. Such a cluster may be defined in meta-terms as a new decision within the decision planning structure. Here we need a positive example that causes creation of a new decision path that is to be analysed afterwards.

An algorithmic solution of topics mentioned in this summary has is not the subject of our research, however the possible usage of various ILP methods (inductive logic programming) gives a good perspective [4]. We have been trying to use ILP techniques, such as FOIL for object-level knowledge compression. Instead of using a database of about 400 Prolog tail-free predicates, the FOIL system produced 25 rules specifying legal combination of attributes and 20 rules specifying the relation between the configuration and the appropriate product parts [11].

## 5. PPA configurator implementation

The assumptions we have made about the properties of the decision planning knowledge bases, its maintenance and efficiency of inference mechanisms are supported by implementation of the proof-of-concept model. We have implemented a configurator that was supposed to quote possible variations of producible transmitters at the Tesla – TV enterprise. Tesla TV factory manufactures TV transmitters, FM transmitters and passive transmitting elements. The manufacturing process within TESLA TV classifies the factory activities as a *project-oriented production*, as each transmitter is usually unique or there is a very small number of products from a single project design. In the Gerstner Lab we have implemented ProPlanT (Production Planning Technology), a multi-agent solution for project-oriented production planning and simulation [10–12]. We determined that the course of production planning and simulation is driven by (1) the availability of various departments, units, and machines within the enterprise, which made us consider multi-agent system as an appropriate methodology for development and (2) the highly specific knowledge of the project-planning engineer in charge of product design and project planning.

PPA configurator is an encapsulated expert system that performs product quotation and configuration when reasoning above convenient meta-representation of the factory units and load and an exhaustive product range. The expert system should then play the role of a planning agent who would be responsible for configuring other agents and for setting the intentions of other agents with respect to the requirements requested.

### 5.1. Configuration knowledge representation

The PPA – *project planning agent* should simulate the technology and project design department [16]. It plays manifold role, but its key duty is to set intentions and local goals and motivations of the community of agents planning the production process. The PPA agent represents the gateway of the project planning process, as it transforms, vaguely specified, requirements from the customer to the internal language of the enterprise. The rigorous project layout, bill of materials and a set of distributed responsibilities is the result of the PPA problem solving process. The PPA planning agent has been implemented in LPA Win-Prolog.

We have clustered knowledge with respect to section 1.1 in (i) object-level knowledge, (ii) inference knowledge and (iii) heuristic knowledge. The object-level knowl-

edge has been encoded in the form of a conventional Prolog database. The object-level knowledge gives specification of a component list of various producible transmitters or their parts.

**Example.** Here is a real example of a single clause the database consists of. It says that a transmitter engine with the attributes collected in the list as a first argument of the engine/2 predicate requires the components listed in the second argument of the predicate.

```
engine([manufacturer:[tesla],power:[2],serie:['IV G'],
      channell:['21MHz-40MHz'], transformer:[no],
      backup_exciter:[no]],
      [093390067000, 043693050000, 043693050000, 043693050000,
      043693049000, 043693054000, 043693054000, 043693054000,
      043693054000, 043693054000, 043693054000, 043693054000,
      043693053000, 043693054000, 043693054000, 043693054000,
      043693054000, 043693054000, 043693054000, 043693054000,
      043693053000, 033847005000, 023412026000, 023412028000,
      023412040000, 023412041000, 023412042000, 023412033000,
      093415013000, 093415060000, 033890003000, 033890003000,
      023492020000, 033888119000, 053060022000, 093414058000,
      093414043000, 831181500101, 043693050000]
      ).
```

The heuristic knowledge bridges the object-level knowledge and meta-level knowledge through various problem-solving shortcuts. As a heuristics we understand the relations between different pieces of domain knowledge.

**Example.** Here is a real example of a heuristic knowledge. The clause gives information that with the power more than 10 kW we must have a backup exciter. The second heuristic says that with Thomson-made aerial we must have either 00987165 or 00987265 backup exciter.

```
product(X,backup_exciter:[yes]):-
  product(X,power:[P]), P>10.
product(X,backup_exciter:[00987265,00987165]):-
  product(A,aerial:[A]),member(A,thompson).
```

The inference knowledge has been kept in the form of decision planning methods, all clustered in decision graphs. For the abstract decision graph see figure 3.

**Example.** The couple of decisions depicted below formalise the decision graph depicted in figure 3.

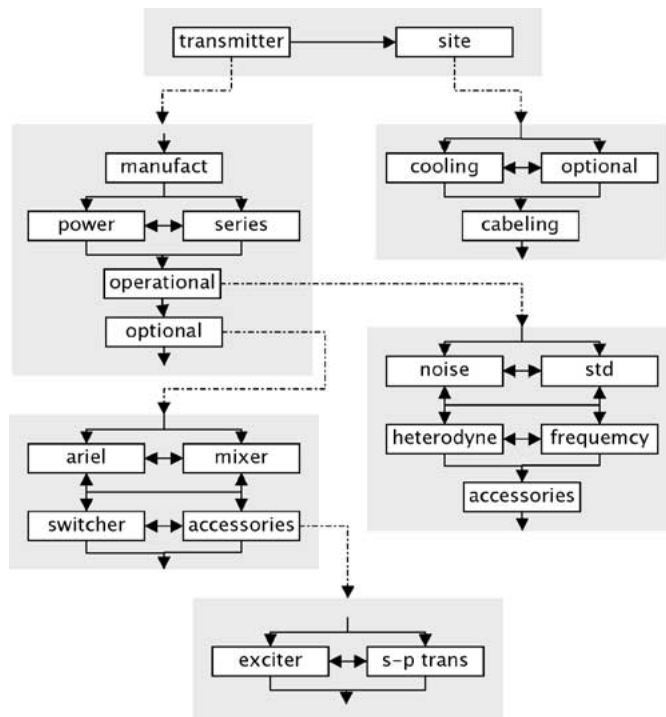


Figure 3. Abstract decision graph.

```

decision(node:exciter,
  level:accessories,
  input:Configuration,
  conditions:[],
  effects:[],
  output:[transformer,last],
  tactics:[]).
decision(node:transformer,
  level:accessories,
  input:Configuration,
  conditions:[],
  effects:[],
  output:[transformer,last],
  tactics:[])

```

Heuristics based on object-level types of knowledge have been incorporated at the specific level of the decision plan. Heuristic and object level knowledge, actually fills in condition and effects slots in the `decision/6` clause. The specific decision plan is

constructed automatically. For the condition and effect instantiation of the two above shown decision nodes see below:

```
decision(node:exciter,
         level:accessories,
         input:Configuration,
         conditions:[serie:['IV G']],
         effects:[crystal,synth, offset],
         output:[transformer,last],
         tactics:[]).
decision(node:exciter,
         level:accessories,
         input:Configuration,
         conditions:[serie:['III F']],
         effects:[gener, ocx, external],
         output:[transformer,last],
         tactics:[]).
decision(node:exciter,
         level:accessories,
         input:Configuration,
         conditions:[],
         effects:[],
         output:[transformer,last],
         tactics:[]).
decision(node:transformer,
         level:accessories,
         input:Configuration,
         conditions:['IV G'],
         effects:[],
         output:[yes,no],
         tactics:[]).
decision(node:transformer,
         level:accessories,
         input:Configuration,
         conditions:[],
         effects:[],
         output:[transformer, last],
         tactics:[]).
```

## 5.2. Implementing configuration consultation

The process of consultation is driven by the convenient representation of the decision space. The algorithm searches each single layer in depth (decision space within one layer are not really large). Once the algorithm finds a decision node that is a meta-

representation of another decision graph, it does not go in depth, while it continuously searches the respective layer and collecting the other graph to be searched afterwards. Once it finds the appropriate decision path on the given layer, it searches through the graphs it found on its way. For the higher-level predicate of the consultation implementation see below:

```
consultation([], A, A) :- !.
consultation(Levels, ConfigIn, Result) :-
    consult_plan(Levels, ConfigIn, ConfigOut, [], Tactics),
    consultation(Tactics, ConfigOut, Result).

consult_plan([], A, A, B, B).
consult_plan([Level|Rest], Config, Result, TacticsIn, Tactics) :-
    consult_level(Level, [first], Config, ConfigOut, TacticsIn,
                  TacOut),
    consult_plan(Rest, ConfigOut, Result, TacOut, Tactics).
```

This algorithm allows very efficient decision-making as it combines virtues of depth first and width first search. When searching on the level  $l_1$  single local decision  $d_1$  with a tactic pointing to the level  $l_2$  does not take the algorithm down to this level. It very often happens that following decision  $d_2$  on the level  $l_1$  causes backtracking the decision  $d_1$ . With the conventional depth-first-search algorithm we would have to redo all the decisions on the levels below the decision  $d_1$ . Structuring the decision making process in the decision graphs will let the knowledge control the process of search for configuration in natural and effective way.

The algorithm of knowledge maintenance implements the process of decision planning extension (see section 4.1). The main algorithm searches throughout to database and transform one by one every negative decision path such that (1) the object theory gets updated and (2) the decision plan gets extended with such respect.

```
accept :- newdb(Decision_path, Object_data),
          explain_failure(Decision_path, Failure),
          update_plan(Failure),
          retract(newdb(Decision_path, Object_data)),
          assert(db(Decision_path, Object_data)),
          accept.
```

The `failure` list identifies all decision nodes within the new decision path that have an influence on the decision plan updating. The list is parsed node by node, such that each of nodes causes an update with respect to either *precondition refinement*, or *effect refinement*, or a *new decision node* gets created. For more details on PPA agent implementation refer to [15] and for information about the ProPlanT multi-agent system see [10–12].

### 5.3. Testing

We have tested the PPA application according to two main dimensions – (i) complexity of the knowledge base and accuracy of consultation and (ii) maintainability of the knowledge base. The subject of testing was the exhaustive product range of Tesla TV production capabilities. We have tested 11 specific product types, which together comprise more than 327 transformers. The attribute space varies and depends on the product type. The transmitter core usually contains about 5 decision attributes, whereas the solution body is based on 15 decision attributes. The field theory is encoded within multidimensional charts – which have been transformed into Prolog database.

The specific decision plan contains decision nodes encoded in Prolog. We consider a decision node to be a collection of `decision/6` predicates with the first argument satisfied with a unique atom. The decision plan contains 21 decision nodes, out of which there are 14 terminal decision nodes, i.e., decision nodes with no further tactic. There are 35 decision nodes in the database, as we have to include 14 decision nodes that define the initial and final point of the decision graph at a certain level.

*5.3.1. Accuracy and complexity of consultation.* The configurator produces the same quotation as engineers produce, whereas it sometimes reveals unknown possible production variations of a transmitter. It is worth noting that the application is considerably faster when consulting data sets that are larger than data sets used by ICON application (see section 2.2). The PPA does not need to consult the base of object-level knowledge every time it makes a decision. The truth is that if there were little or no causality among object-level data, which is not very often the case, the consultation complexity would converge to the case of ICON as there would be as many methods as object-level facts.

We have compared the decision space and time requirements for consultation if driven by (i) specific decision graph and (ii) abstract decision graph and domain theory, in order to illustrate efficiency improvements. The graph in figure 4 illustrates the time

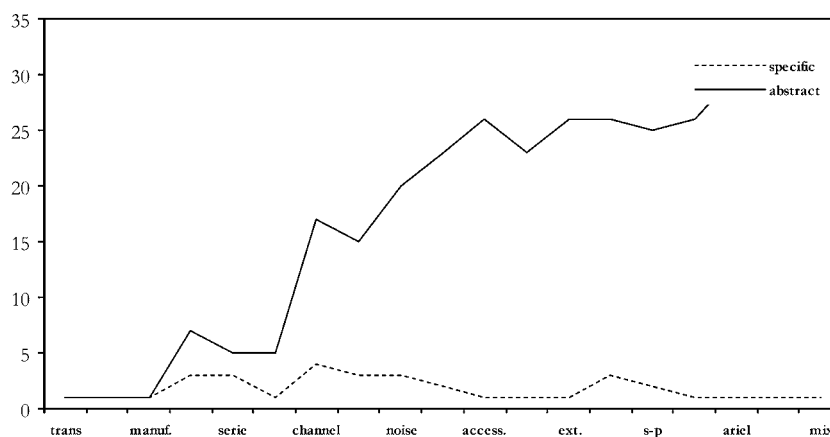


Figure 4. Graph illustrating time requirements for consultation if driven by (i) specific decision graph and (ii) abstract decision graph and domain theory.

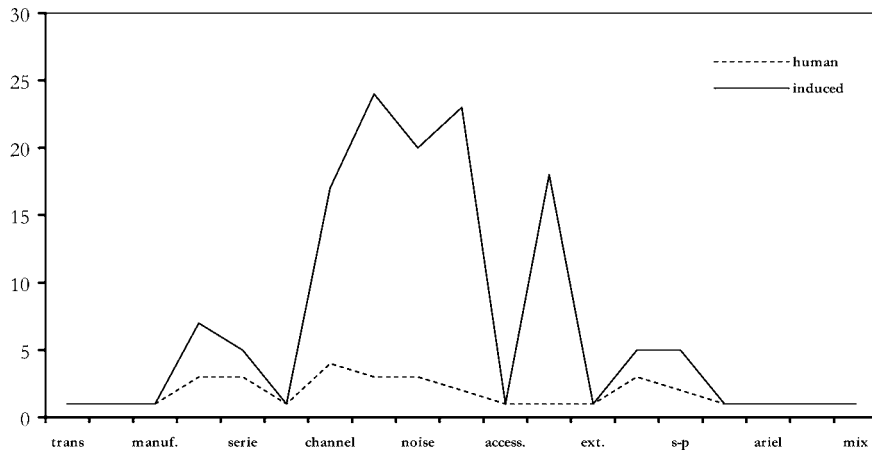


Figure 5. Graph illustrating number of predicates (i) induced or (ii) encoded by human for each decision node.

(in 0.1 seconds) needed for making a decision in either of cases. The time needed is strictly proportional to how many predicate needs to be consulted in order to come up with an appropriate question and set of possible options.

*5.3.2. Accuracy of heuristics induction (maintainability).* The decision plan induction mechanism induced a specific level of the decision plan with all required and many other decision/6 predicates in comparison to the human-made plan (51 human made and 151 induced). Automated induction did not utilise possible disjunctive coupling of precondition within a method that did not change consultation, however. A more important flaw is rooted in the frame problem. As the induction is carried out sequentially, the system generalises the precondition for one effect incrementally, which may end up in disjunction of the precondition that is always true. The process of consultation is the same here, but the complexity is more explosive. Differences in a number of predicates (induced or human encoded) for each decision node are depicted in the graph in figure 5.

This tool is being used within the ProPlanT system for testing multi-agent technology for project-oriented production planning. Successful integration has been also carried out within the PVS'98 multi-agent system implemented within the framework of EUREKA PVS'98 project. Apart from this, people at Tesla TV use PPA as a product configurator for on-line consultation on TV transmitters. If the customer is off-site the sales people should inquire by mail whether this variation is legal and how much this option would cost. They consult the Prolog database of predicates in a decision planning fashion, which is ported onto a portable instead. Testing results are in details described in [15].

#### 5.4. Discussions: business benefits and flaws

We set a knowledge representation framework and implemented configuration-like expert systems that help quotation experts with configuring a product, and producing a

specific unique component list. Coupling with the production planning multi-agent system (ProPlanT) provides the user with the most accurate cost and deadline estimate. The ProPlanT helps to organise and administer the process of production planning and thus solves a number of bottleneck issues. Moreover the proper administration of production planning related information flows, material flows and flows of workforce offers a suitable platform for optimisation of the process of manufacturing. A further virtue of the multi-agent methodology is that the system provides standard mechanisms for integration of pre-existing pieces of software (i.e., database management systems, diagnostic expert systems, scheduling applications based on advanced computing strategies such as neural networks and genetic algorithms, etc.).

The knowledge representation medium used within an encapsulated configuration expert system is novel for a number reasons described within the article. From the user's perspective it offers several nice features. Unlike classical rule-based expert systems it structures problem solving knowledge so that the domain world the system is expected to represent (product range in the case of manufacturing) can be kept enhanced without a loss of consistency and a need to re-specify strategic knowledge. This claim is supported by the PPA expert system implementation and testing (see sections 5.1, 5.3). The PPA system separately maintains domain knowledge in a simple database format (we have used Prolog facts). General, context-free problem solving knowledge, which is a subject of knowledge acquisition, are stored in the form of the abstract decision graph. An efficient consultation knowledge base is automatically created and maintained by means of weak updates – implemented by Explanation Based Generalisation.

Owing to the graph-like representation of the consultation knowledge base, the system facilitates various feasible tracks of decision-making. When using a classical decision tree, the order of questioning presented is more or less fixed. The proposed knowledge representation allows the user to ask for another question if he or she is not ready to answer the question given. Graph parsing algorithms may provide the user with all possible questions he or she may start with. This is very important. As our experience indicates engineers using expert systems would start by specifying what they already know and letting the systems figure out the details rather than answering system's questions.

The system thus behaves in a more or less case-based reasoning (CBR) way, whereas the knowledge engineer acquiring and formalising expert knowledge may use techniques related to classical decision trees and avoid all known difficulties from the area of CBR.

People have often objected that the use of Prolog within such a robust and efficient application may cause problems. As the system uses an artificially induced, optimal and efficient knowledge base, it behaves reasonably even for a large volume of data. On the other hand it is fair to admit that solving the problem of knowledge base induction with other than Prolog based mechanism would provide more efficiency and flexibility.

Although we have tested the system on the site of our industrial partner we still feel a need for further real life testing. Such a intense testing may reveal other weaknesses and identify additional user requirements that will help to transform a working prototype

into a marketable product. The problem of identifying an intelligent way of transforming data from the conventional systems into local knowledge bases of system agents and the problem of connecting standard knowledge acquisition techniques and tools with the configuration expert systems is deemed a challenge for further research and development in this area.

## 6. Conclusions

In this paper we have shown the practical application of proof planning programming methodology outside the theorem proving areas. Three novel applications have been developed in the area of industrial configuration using the proof planning knowledge structures. Building upon the success of  $CL^E M$ , HP computer configurator and ICON, compressor configurator we have developed a general methodology for the decision process formalisation – decision planning. Decision planning has been successfully used for product configuration within a multi-agent system aimed at project oriented production planning and simulation in Tesla TV, an enterprise manufacturing TV transmitters.

According to Bundy [2] there are the following specific features that are typical for meta-level reasoning in terms of proof planning guiding a search through solution space:

- **Efficiency:** if proof plans are well designed, the space complexity is considerably reduced.
- **Generality:** a given proof plan may be applied to a large number of tasks, especially where hierarchical plans are concerned.
- **Maintainability:** the separation of object knowledge and meta knowledge eases the process of maintainability.
- **Explanatory power:** meta-level explanation can be more comprehensible and used in comparison with long paths of low-level object theory inferences.

We have shown in the paper that the decision plans share the some of these features. As explained in the study and supported by the experiments in section 5.3 the specific decision plans allow efficient consultation as they contained layered and well structured combination of domain, heuristic and control knowledge. Decision planning satisfies the requirement for generality. The abstract decision plan, which is the knowledge medium accessible to the user, can be applied to a number of different domain knowledge instantiations as the EBG mechanism will induce different specific decision plans for different set of domain knowledge. Maintainability has been arguably the most important issue of this study. This property of the decision plans has been achieved by separate maintenance of inference knowledge, heuristics and domain knowledge. More importantly, user update different set of knowledge (abstract decision plan, and domain knowledge), while the system uses for consultation different, more efficient knowledge base. If a new product updates the domain database, the configuration knowledge base is

automatically updated. The EBG mechanism works as causal connection between these two instances of knowledge medium.

Decision planning differs from the previous proof planning base application in a number of features. It offers a general graph-like formalisation, which we found more suitable especially for user guided configuration. It allows structuring of inference knowledge in an arbitrary number of reasoning levels. We see an interesting potential of this methodology in the domain of distributed artificial intelligence. A decision does not have to correspond to a lower level reasoning process only; it can also be a meta-representation of other agent capabilities and loads in a multi-agent environment. The planning agent thus manipulates only the meta-description of other collaborating agents, and if planning succeeds it contracts collaborators by executing the tactics. Similarly the domain knowledge may contain the information about the participating agents. As a status of some agent changes (losing or acquiring a new capability) the domain knowledge has to be updated. The EBG mechanism will make sure that this update will propagate in the configurator's decision plans and it may be incorporated in further decision making.

### Acknowledgements

This paper could not have been written without the significant support of my colleagues Olga Štěpánková, Vladimír Mařík and Tomáš Hazdra. I am also grateful to Rannan Banerji and Christophe Roche for a number of comments as much as Helen Lowe and Alan Bundy for introducing me to the world of proof planning. The research in the Czech Republic was funded through EUREKA PVS'98 project and research in the United Kingdom was facilitated thanks to the EXCALIBUR SCHOLARSHIP SCHEME.

### References

- [1] J.L. Bates and R.L. Constable, Proofs as programs, *ACM Transactions on Programming Languages and Systems* 7(1) (1985) 113–136.
- [2] A. Bundy, A science of reasoning, in: *Computational Logic: Essays in Honour of Alan Robinson*, eds. J.-L. Lassez and G. Plotkin (MIT Press, Cambridge, MA, 1990).
- [3] R.A Klein, Logic-based description of configuration: The constructive problem solving approach, in: *Proceedings of the AAAI 1996 Fall Symposium on Configuration* (1996).
- [4] N. Lavrač, I. Weber, D. Zupanič, D. Kazakov, O. Štěpánková and S. Džeroski, ILPNET repositories on WWW: inductive logic programming systems, datasets and bibliography, *AI Communications* 9 (1996) 157–206.
- [5] H. Lowe, The application of proof plans to computer configuration problems, Ph.D. Thesis, Department of Artificial Intelligence, University of Edinburgh (1993), unpublished.
- [6] H. Lowe, Proof planning: A methodology for developing systems incorporating design issues, *Artificial Intelligence for Engineering Design and Manufacturing* 8(4) (1994) 307–317.
- [7] H. Lowe, M. Pěchouček and A. Bundy, Proof planning and configuration, in: *Proceedings of the Ninth Exhibition and Symposium on Industrial Applications of Prolog*, Tokyo (October 1996).

- [8] H. Lowe, M. Pěchouček and A. Bundy, Proof planning for maintainable configuration systems, *Artificial Intelligence for Engineering Design and Manufacturing, Special Issue on Configuration* September (1998) 345–356.
- [9] T. Mannisto, H. Peltonen and R. Sulonen, View to product configuration knowledge: modelling and evolution, in: *Proceedings of the AAAI 1996 Fall Symposium on Configuration* (1996).
- [10] V. Mařík, M. Pěchouček, T. Hazdra and O. Štěpánková, ProPlanT – multi-agent system for production planning, in: *Proceedings of the 15th European Meeting on Cybernetics and Systems Research*, Vienna (1998) pp. 725–730.
- [11] V. Mařík, M. Pěchouček, J. Lažanský and C. Roche, PVS'98 agents: structures, models and production planning application, *International Journal of Robotics and Autonomous Systems – Special Issue on Multi-Agent Systems Applications* 27 (1999) 29–43.
- [12] V. Mařík, M. Pěchouček and C. Roche, PVS'98 agent models and their application in production planning, in: *Intelligent Systems for Manufacturing: Multi-Agent Systems and Virtual Organisations*, eds. L.M. Camarinha Matos, H. Afsarmanesh and V. Marik (Kluwer Academic, Dordrecht, 1988) pp. 13–22.
- [13] T.M. Mitchell, R.M. Keller and S.T. Kedar-Cabelli, Explanation based generalisation: A unifying view, *Machine Learning* 1(1) (1986) 47–80.
- [14] O. Najmann and B. Stein, A theoretical framework for configuration, in: *Proceedings of the 5th IEAAIE* (1992).
- [15] M. Pěchouček, Advanced planning techniques for project oriented production, Ph.D. Dissertation, Czech Technical University, Prague (1998), unpublished.
- [16] M. Pěchouček, Proof planning: methodology for product configuration, in: *Proceedings of European Conference – Product Data Technology Days*, Watford, UK (1998) pp. 287–294.
- [17] M. Pěchouček, R. Bunerji and O. Štěpánková, Improving proof plans through learning from examples, Technical Memo of Gerstner Laboratory – GLM-04/97, Prague (1997).
- [18] M. Pěchouček, H. Lowe and A. Bundy, Proof planning and industrial configuration, in: *Proceedings of the Fifth Practical Application of Prolog*, London (April 1997).
- [19] M. Pěchouček, O. Štěpánková and P. Mikšovský, Maintenance of discovered knowledge, in: *Proceedings of the Third European Conference on Principles and Practice of Knowledge Discovery in Databases*, Prague (August 1999).
- [20] M. Stefik, *Introduction to Knowledge Systems* (Morgan Kaufman, San Mateo, CA, 1995).
- [21] L. Steels and J. McDermott, *The Knowledge Level in Expert Systems* (Academic Press, Boston, MA, 1993).
- [22] D.S.W. Tansley and C.C. Hayball, *Knowledge-Based Systems Analysis and Design* (Prentice-Hall, Englewood Cliffs, NJ, 1993).