On the Role of Abstraction in Case-Based Reasoning

Ralph Bergmann and Wolfgang Wilke

University of Kaiserslautern,
Center for Learning Systems and Applications (LSA)
Dept. of Computer Science,
P.O.-Box 3049, D-67653 Kaiserslautern, Germany
E-Mail: {bergmann,wilke}@informatik.uni-kl.de

Abstract. This paper addresses the role of abstraction in case-based reasoning. We develop a general framework for reusing cases at several levels of abstraction, which is particularly suited for describing and analyzing existing and designing new approaches of this kind. We show that in synthetic tasks (e.g. configuration, design, and planning), abstraction can be successfully used to improve the efficiency of similarity assessment, retrieval, and adaptation. Furthermore, a case-based planning system, called PARIS, is described and analyzed in detail using this framework. An empirical study done with PARIS demonstrates significant advantages concerning retrieval and adaptation efficiency as well as flexibility of adaptation. Finally, we show how other approaches from the literature can be classified according to the developed framework.

1 Introduction

Traditionally, case-based reasoning (CBR) approaches retrieve, reuse, and retain cases in a representation at a single level of abstraction. In this predefined representation, cases, new problems, as well as general knowledge must be represented. Recently, some researchers have started to investigate the use of abstraction in CBR (e.g., [29; 16; 4; 6; 5; 7; 30; 3], cf. also [18], p. 576). However, a clear picture of how CBR can benefit from abstraction has not be drawn till now.

In AI, the use of abstraction was originally inspired by human problem solving (cf. [26; 20]) and has already been successfully used in different fields such as theorem proving (e.g. [25]), model-based diagnosis (e.g. [23]) or planning (e.g. [28; 33; 17]).[1] The basic idea that emerges from different approaches to using abstraction in CBR is to supply a CBR system with cases at different (higher) levels of abstraction. Thereby, the CBR process can be supported in the following ways:

- Abstraction can reduce the complexity of a case, i.e., it can simplify its representation, e.g. by reducing the number of features, relations, constraints, operators, etc. This simplification usually reduces the effort required for similarity assessment and/or solution adaptation.
- Cases at higher levels of abstraction can be used as a kind of prototypes (cf. also [27]), which can be used as indexes to a larger set of related, more detailed cases . Such indexes can help to improve the efficiency of the retrieval.
- Cases at higher levels of abstraction can even be used as a substitute for a set of concrete cases. Thereby, the size of the case base may be reduced significantly, which improves the efficiency of retrieval.
- Abstraction can also be used as a mean of defining the semantics of similarity. Similarity can be defined as equality on a certain level of abstraction. The lower the level of abstraction on which two cases are identical, the higher the similarity.

[1] For an overview on abstraction in AI look at [13].

– Abstraction can increase the flexibility of reuse. Adapting abstract solutions contained in cases at higher levels of abstraction can lead to abstract solutions suitable for a large spectrum of concrete problems.
– Abstraction and refinement, on their own, can be used as a method for solution adaptation. Like in hierarchical problem solving, an abstract solution (or parts of a solution) contained in a case can be refined towards a solution to the new problems that may be radically different from the original concrete solution contained in the cases.

These advantages seem to be particularly valuable in situations in which a large number of cases is available, the similarity assessment is very expensive, or flexible adaptation is required. However, abstraction is inevitably connected with a loss of information. When reasoning primarily with abstract cases, this loss of information must be compensated by other kinds of (general) knowledge which can lead to an increased effort in knowledge engineering.

In the remainder of this paper, we analyze the use of abstraction for CBR in detail. Section 2 presents a general framework for reusing cases at several levels of abstraction. This framework is particularly suited for analyzing existing and designing new approaches that bring abstraction into CBR. Section 3 describes a case-based planning system, called Paris, with respect to this framework and section 4 reports on an empirical study done with this system to validate whether the above mentioned advantages of abstraction can be recognized. Finally, section 5 discusses related work with respect to our general framework.

2 Reusing Cases at Several Levels of Abstraction

We now present a general framework for reusing cases at higher levels of abstraction covering the CBR phases [1] retrieve, reuse, and retain. This framework is particularly suited for synthetic problem solving tasks such as case-based configuration, design, or planning. Typically, these tasks are characterized through a vast space of potentially relevant solutions and a relatively low coverage of this solution space by the available cases.

2.1 What are Abstract Cases?

While cases are usually represented and reused on a single level, abstraction techniques can enable a CBR system to reason with cases at several levels of abstractions. Firstly, this requires the introduction of several distinct levels of abstraction.

Levels of abstraction Each level of abstraction allows the representation of problems, solutions, and cases as well as the representation of general knowledge that might be required in addition to the cases. Usually, levels of abstraction are ordered (totally or partially) through an abstraction-relation, i.e., one level is called *more abstract* than another level.

A more abstract level is characterized through a reduced level of detail in the representation, i.e., it usually consists of less features, relations, constraints, operators, etc. Moreover, abstract levels model the world in a less precise way, but still capture certain, important properties.

In traditional hierarchical problem solving (e.g., Abstrips [28]), abstraction levels are constructed by simply dropping certain features of the more concrete representation levels. However, it has been shown that this view of abstraction is too restrictive and representation dependent [5; 15] to make full use of the abstraction idea. In general, different levels of abstraction require different representation languages, one for each level. Abstract properties can then be expressed in completely different terms than concrete properties.

Different Kinds of Cases Based on the level of abstraction, we can distinguish between two kinds of cases: *concrete cases* and *abstract cases*. A *concrete case* is a case located at the lowest available level of abstraction. An *abstract case* is a case represented at a higher level of abstraction.

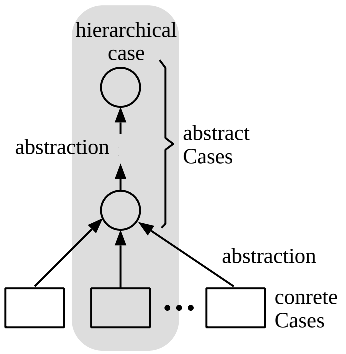

Fig. 1. Different kinds of cases

If several abstraction levels are given (e.g., a hierarchy of abstraction spaces), one concrete case can be abstracted to several abstract cases, one at each higher level of abstraction. Such an abstract case contains less detailed information than a concrete case. On the other hand several concrete cases usually correspond to a single abstract case (see Fig. 1). These concrete cases share the same abstract description; they only differ in the details.

Instead of having cases, located at a single level of abstraction, one case (called *hierarchical case*) can also contain information at several or all levels of abstractions that are available.

2.2 Acquisition of abstract cases

We can distinguish different ways in which abstract or hierarchical cases are built in order to be stored in the case base.

Abstract cases available The first, and simplest scenario is a one, in which case data is naturally available at several appropriate levels of abstraction. This can be the situation if, for example, data is modeled in an object-oriented language and stored in an object-oriented database. The abstraction present in the class hierarchy (inheritance) can then lead to different levels of abstraction and data base instances provide data for abstract or hierarchical cases.

Automatic generation of abstract cases In most situations, cases are only available in a single representation which can be considered the concrete level. Consequently abstract or hierarchical cases must be abstracted out of these concrete cases. In certain situations, such a *case abstraction* can be done automatically. This usually requires general knowledge about ways of mapping cases onto higher levels of abstraction.

Manual generation of abstract cases If abstract cases are neither available not automatically generateable, they must be abstracted manually from concrete cases. This option requires a very high effort, that – we think – cannot be justified in most applications.

2.3 Abstract Cases in Retrieval

Abstract cases located at different levels of abstraction can be used as hierarchical indexes to those concrete (or abstract) cases that contain the same kind of information but at a lower level of abstraction. An *abstraction hierarchy* can be constructed in which abstract cases at higher levels of abstraction are located above abstract cases at lower levels. The leaf nodes of this hierarchy contain concrete cases (see Fig. 2). During retrieval, this hierarchy can be traversed top-down, following only those branches in which abstract cases are sufficiently similar to the current problem.

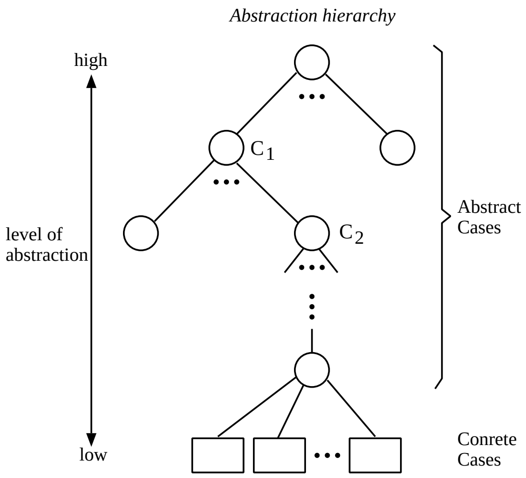

Fig. 2. Abstraction hierarchy for indexing cases

This approach to indexing, however, makes an assumption concerning the similarity assessment. It requires that a problem cannot be similar to a concrete case unless it is at least similar to this case at a higher level of abstraction. This assumption holds particularly, if similarity is defined *based* on the level of abstraction, which can be done as follows:
A problem p is more similar to the concrete case C_1 than to the concrete case C_2 if the lowest level of abstraction on which p matches C_2 is higher (more abstract) than the lowest level of abstraction on which p matches C_1.

2.4 Reuse of Abstract Cases

There are different ways of using the information provided in abstract cases for solving the current problem.

No reuse of abstract solutions Abstract cases are only used as indexes to concrete cases. For problem solving, concrete cases are used exclusively.

Abstract solutions as result The CBR system retrieves and reuses abstract cases. The abstract solutions contained in the abstract cases are not refined to more concrete levels but are directly returned as output. The interpretation of abstract solutions is up to the user.

Refinement of abstract cases The CBR system retrieves and reuses abstract cases and refines abstract solutions to the concrete level. The refined solution is then presented to the user. For her/him it is transparent, whether the solution presented by the system stems directly from a matching concrete cases or whether the solution is obtained through the refinement of an abstract case.

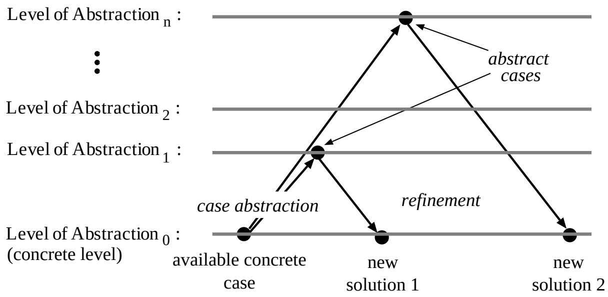

Fig. 3. Adaptation by abstraction and refinement

Please note that abstraction and refinement is already a technique for solution adaptation (see Figure 3). If available concrete cases are abstracted (e.g., automatically) to abstract cases and then getting retrieved and refined, a new solution to a new problem will be constructed. The higher the level of abstraction of the reused abstract case, the more may the newly refined solution differ from the solution contained in the original case.

We can distinguish different methods for realizing such a refinement:

Generative refinement of abstract cases This refinement is done by generative problem solving methods, e.g. hierarchical problem solving. For automatically performing this refinement task, additional general domain knowledge is usually required.

Case-based refinement of abstract cases This refinement itself is done in a case-based way, avoiding partially the need for additional general knowledge. However, case-based refinement requires cases that describe how the individual elements, the abstract solutions are built of, can be refined at a more concrete level.

2.5 Adaptation of Abstract Cases

Besides the possibility to realize adaptation by refining abstract cases, adaptation can also be done on a single level of abstraction (see Fig. 4). The spectrum of known methods for solution adaptation in CBR (for an overview see e.g., [8; 14; 34]) can also be applied to abstract cases prior the refinement. Thereby, the flexibility of reuse can be increased, i.e., a concrete case covers a larger area in the solution space.

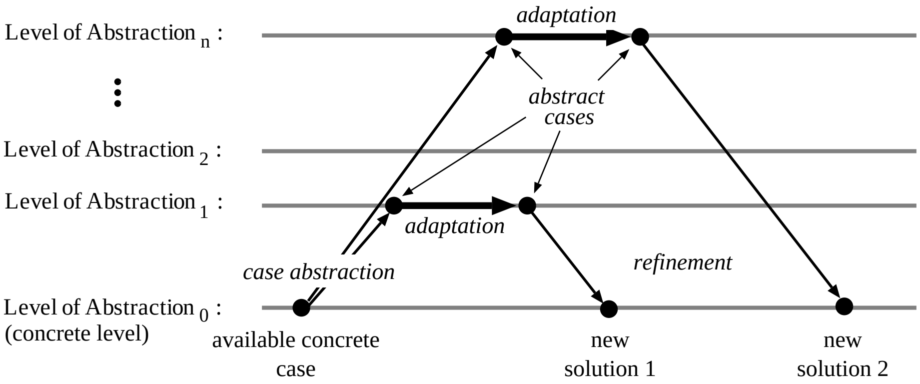

Fig. 4. Adaptation of abstract cases

2.6 Forgetting cases

The reuse of cases at several levels of abstraction also provides a frame for realizing case deletion policies [31]. Cases deletion is particularly important to avoid the utility or swamping problem [32; 21; 12] that occurs when case bases grow very large. When reusing abstract cases for indexing and reuse, case deletion can efficiently be realized through a pruning of the abstraction hierarchy. If certain concrete cases are removed from the case base, the abstract cases that remain accessible, can still cover the set of target problems previously covered by the deleted concrete case. However, this requires effective ways of refinement such as generative or case-based refinement. For selecting cases to forget, the savings due to the reduced retrieval effort must outweigh the additional effort for refining more abstract cases.

2.7 Summary of the Framework

The following Table 1 summarizes the various facets of the framework for reusing abstract cases. Existing approaches can be described and analyzed and new approaches can be designed using this framework.

kind of stored cases:	abstract	abstract & concrete		hierarchical
acquisition of abstract cases:	cases available	manual generation		automatic generation
abstract cases for indexing:	no		yes	
reuse of abstract solutions:	no	abstract result	generative refine.	case based refine.
adaptation of abstract solutions:	no		yes	
case deletion policy:	no		yes	

Table 1. Framework for reusing abstract cases

3 PARIS: Using abstraction in case-based planning

Now, we describe a concrete case-based reasoning system, called PARIS[2] [4; 5; 3] that uses abstraction for case-based planning. PARIS can be characterized according to this framework as follows:

- abstract and concrete cases are stored in the case base
- abstract cases are generated automatically from concrete cases
- abstract cases are used for indexing
- generative refinement of abstract solutions is realized
- adaptation of abstract solutions is possible
- case deletion policy is realized.

We now explain the approach in more detail.

3.1 Requirements

PARIS was designed as a generic (i.e., domain independent) case-based planning system but with a particular area of application domains in mind: process planning in mechanical engineering. From this application area, a set of CBR specific requirements have been identified:

- ability to cope with vast space of solution plans
- construction of correct solutions
- flexible reuse due to large spectrum of target problems
- processing of highly complex cases
- only concrete planning cases available (e.g. in archives of a company)

3.2 Abstract Planning Cases

We now summarize the formal definition of concrete and abstract planning cases already presented in detail in [5].

Representation of domains Following a STRIPS-oriented representation [11], a planning domain $\mathcal{D}$ defines a (possibly infinite) set of *states* $\mathcal{S}$ and a set of *operators* $\mathcal{O}$ which describe transitions from one state to a successor state. A state $s \in \mathcal{S}$ is described by a finite set of prepositions. A *problem* $\langle s_I, s_G\rangle$ in a domain is given by an *initial state* s_I and a *goal state* s_G and a plan (a solution to the problem) is a totally ordered sequence of operators $\langle o_1, \ldots, o_n\rangle$ that transform the initial state into the goal state. A case is a problem together with a solution to this problem.

Different levels of abstraction In PARIS, different levels of abstraction are realized by different planning domains. In the following we assume two planning domains: a *concrete domain* $\mathcal{D}_c$ and an *abstract domain* $\mathcal{D}_a$. Concrete and abstract planning domains may represent different languages with completely different states and operators. A *concrete case* $C_c = \langle\langle s_0^c, s_n^c\rangle, (o_1^c, \ldots, o_n^c)\rangle$ is a case given in the concrete planning domain, and an *abstract case* $C_a = \langle\langle s_0^a, s_m^a\rangle, (o_1^a, \ldots, o_m^a)\rangle$ is a case in the abstract planning domain.

However, not every abstract case is an abstraction of a concrete case. Additional requirements must be met to call an abstract case abstraction of a concrete case. These requirements can be expressed by the existence of two independent mappings: a *state abstraction mapping* α, and a *sequence abstraction mapping* β [2] (see Fig. 5).

[2] PARIS stands for *plan abstraction and refinement in an integrated system.*

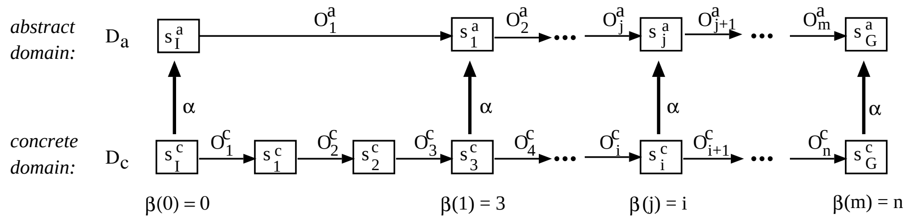

Fig. 5. Formalization of case abstraction in planning

State Abstraction A state abstraction mapping $\alpha : \mathcal{S}_c \rightarrow \mathcal{S}_a$ translates states of the concrete domain into the abstract domain. For this translation, we require additional general domain knowledge about how an abstract state description relates to a concrete state description. We assume that this kind of knowledge can be provided in terms of a domain specific *generic abstraction theory* $\mathcal{A}$. Such a generic abstraction theory (expressed by horn clauses) defines each proposition, abstract states can be composed of, in terms of propositions that can occur in the concrete states.

Sequence abstraction The solution to a problem consists of a sequence of operators and a corresponding sequence of states. To relate an abstract solution to a concrete solution, the relationship between the abstract states (or operators) and the concrete states (or operators) must be captured. Each abstract state must have a corresponding concrete state but not every concrete state must have an associated abstract state. The *sequence abstraction mapping* $\beta : \mathbb{N} \rightarrow \mathbb{N}$ selects those states of the concrete problem solution that have a related abstract state. It maps the indices $j \in \{1, \ldots, m\}$ of the abstract states s_j^a into the indices $i \in \{1, \ldots, n\}$ of the concrete states s_i^c, such that $\beta(0) = 0$, $\beta(m) = n$, and $\beta(u) < \beta(v)$ *if and only if* $u < v$. This guarantees that abstract and concrete initial and goal states correspond and that the order of states is maintained.

Case abstraction Based on the two introduced abstraction functions, our intuition of case abstraction is captured in the following definition. A case C_a is an abstraction of a case C_c if there exists a state abstraction mapping α and a sequence abstraction mapping β, such that: $s_j^a = \alpha(s_{\beta(j)}^c)$ holds for all $j \in \{0, \ldots, m\}$ (see Fig. 5). In [5] we have discussed the generality of the presented case abstraction methodology. We have formally shown that hierarchies of abstraction spaces as well as abstractions with respect to different aspects can be represented using the presented methodology. Based on the defined levels of abstraction and the generic abstraction theory, *several abstract cases* are called abstractions of a single concrete case.

3.3 Acquisition of Abstract Cases

Because cases are only available at the concrete domain in the applications domains we have in mind and manual abstraction seems to be a tremendous effort, abstract cases are generated automatically from a given concrete case. For this purpose, a particular case abstraction algorithm has been developed [5; 3] which could be proven to be correct (computes only correct abstract cases) and complete (computes all abstract cases) with respect to the above introduced model of case abstraction.

3.4 Refinement of abstract cases

In PARIS an abstract solution contained in an abstract case is refined automatically to a concrete level solution. If a new target problem is given (at the concrete level), this refinement starts with the concrete initial state from the problem statement (see Fig. 6). A search is performed to find a sequence of concrete operations which lead to a concrete state that can be abstracted with a state abstraction mapping to match the second abstract state contained in the abstract case. If the first abstract operator can be refined a new concrete state is found. This state can then be taken as a starting state to refine the next abstract operator in the same manner. If this refinement fails we can backtrack to the refinement of the previous operator and try to find an alternative refinement. If the whole refinement process reaches the final abstract operator, it must directly search for an operator sequence that leads to the concrete goal state of the new problem. If this concrete goal state has been reached, the concatenation of concrete partial solutions leads to a complete solution to the original problem.

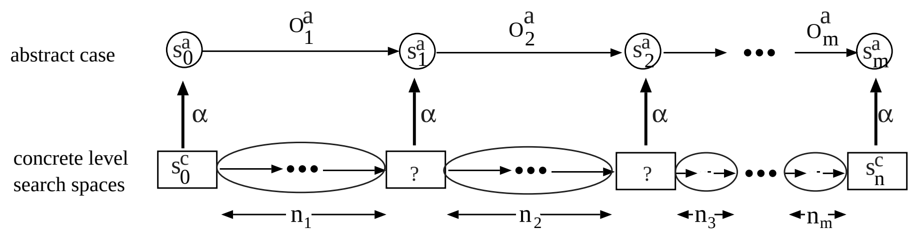

Fig. 6. Refinement of abstract cases

Abstract solutions decompose the original problem into a set of much smaller subproblems. These subproblems are solved by a search-based problem solver. The problem decomposition leads to a significant reduction of the overall search that must be performed to solve the problem [19]. With pure search the worst-case time complexity for finding the required solution is $O(b^n)$, where n is the length of the solution and b is the average branching factor[3]. If the problem is decomposed by an abstract solution into m subproblems, each of which require a solution of length $n_1, \dots, n_m$, respectively, with $n_1 + n_2 + \cdots + n_m = n$, the worst-case time complexity for finding the complete solution is $O(b^{n_1} + b^{n_2} + \cdots + b^{n_m})$ which is $O(b^{max(n_1,n_2,\dots,n_m)})$. Please note that the refinement effort increases with the level of abstraction the abstract case is located on, because more abstract solutions contain a smaller number of abstract operators than the detailed solutions do.

3.5 Adaptation of Abstract Cases

PARIS performs solution adaptation also at a single level of abstraction. For that purpose, an abstract or concrete case is *generalized* into a generalized case (similar to a schema or script). Such a generalized case does not only describe a single problem and a single solution but a *class of problems* $\langle s_I(x_I), s_G(x_G) \rangle$ together with a *class of solutions*

[3] The branching factor is the average number of successor states that can be reached through the application of available operators.

$\langle o_1(x_1), \ldots, o_n(x_n)\rangle$. Such classes are realized by introducing the *variables* x_j into the initial and goal state as well as into the plan. Additionally, a generalized case contains a *set of constraints* $C(x_I, x_G, x_1, \ldots, x_n)$ that restricts the instantiation of these variables.

PARIS includes an algorithm for automatically generalizing concrete or abstract cases into schemas (see [3]) by applying explanation-based generalization [22; 9]. This algorithm guarantees the correctness of the computed generalized case, i.e., every instantiation σ of the occuring variables such that all constraints $C(x_I, x_G, x_1, \ldots, x_n)\sigma$ are fulfilled leads to a correct case, i.e., the plan $\langle o_1(x_1), \ldots, o_n(x_n)\rangle\sigma$ solves the problem $\langle s_I(x_I), s_G(x_G)\rangle\sigma$

Adaptation with generalized cases is done by finding an instantiation σ of the variables such that $\langle s_I(x_I), s_G(x_G)\rangle\sigma$ matches the target problem to be solved and such that the constraints are fulfilled. The solution to the target problem results from applying σ to the solution class of the generalized case.

In PARIS, matching (similarity assessment) and adaptation is done by a constraint satisfaction problem solver (see [3] for details). The effort for solving this constraint satisfaction task can be very high: in the worst case it is exponential in the number of constraints and the size of the problem class. Typically, the representations at a higher level of abstraction are less complex than representations at lower levels. Consequently, generalized cases at higher levels of abstraction contain less constraints and the problem class is composed of a small number of prepositions. Therefore, adaptation of abstract cases requires less effort than adaptation of concrete cases.

3.6 Retrieval with Abstract Cases

In PARIS abstract and concrete cases are stored in the case base which is organized by an abstraction hierarchy. The idea for constructing this hierarchy is based on the following condition: *a case C_1 is located above C_2 (cf. Fig. 2) if for* **every problem** *p holds that if C_2 is adaptable for p (at some level of abstraction) then C_1 is adaptable for p as well.* If this condition can be fulfilled, retrieval will be improved, because if C_1 is *not adaptable* for solving p, then none of the cases in the sub-tree below C_1 can be adaptable and must consequently not be accessed.

Unfortunately, this condition is undecidable in general [3]. However, an abstraction hierarchy can be constructed such that the above condition holds at least for the problems already known (the case base) instead of holding for all possible problems. This approximation approaches the original condition as more and more cases arise. In PARIS such an abstraction hierarchy is built and updated incrementally.

3.7 Case Deletion Policy

In PARIS a utility problem can occur, if the representation (e.g., the concrete domain) is very complex such that matching and adaptation of generalized cases through the constraint propagation becomes very costly. In this case, refining an abstract case at a higher level of abstraction can involve less effort than adapting a case at a lower level. To cope with this problem, a case deletion policy is realized which works by pruning sub-trees of the abstraction hierarchy [35]. A sub-tree is pruned if matching and adapting abstract or concrete cases contained in this sub-tree requires more effort than refining a more abstract available case. These efforts are estimated through measuring the run-time for matching and adaptation and the run-time for solution refinement based on the problems already contained in the case base.

4 Experimental Evaluation

We now present the results of an experimental study on the benefits of using abstraction in CBR. This study was done using the fully implemented Paris system in the domain of manufacturing planning for rotary symmetric workpieces on a lathe (see [5] for details of the domain). For the experiments, 100 concrete cases were generated randomly. From these concrete cases 28 abstract cases at four levels of abstraction could be generated.

4.1 Improving Efficiency of Similarity Assessment and Adaptation

The purpose of the first experiment was to evaluate how the effort for similarity assessment and adaptation decreases with higher levels of abstraction. For this purpose, we measured the run-time for matching and adapting concrete and abstract generalized cases with the constraint propagation mechanism. A time limit of 200 seconds was imposed for the constraint propagation procedure. If this limit was exceeded, the procedure was terminated and the matching failed.

Table 2. Comparison of matching and adapting concrete and abstract cases

kind of cases	number of constraints	run-time in sec.	percentage of failures
concrete cases	76.3	95.97	42 %
abstract cases	21.5	1.11	0 %

Table 2 shows for abstract and concrete cases, the average number of constraints to be considered during constraint propagation, the average run-time for matching and adapting, and the percentage of failures due to exceeding of the time limit. As expected, these results show a strong decrease in the run-time for abstract cases which is due to the reduced complexity of abstract cases compared to concrete cases.

4.2 Improving Retrieval by Abstract Cases

The purpose of the second experiment is to evaluate the speedup in retrieval time when using abstract cases (organized in an abstraction hierarchy) for indexing. We built up a case-base with 100 concrete and 28 abstract cases. The abstract cases were used as indexes only. We measured the time for retrieving (including matching and adapting) a concrete case with the abstraction hierarchy and compared it to the time for a sequential retrieval of concrete cases. Again, a time limit of 200 seconds was imposed for retrieval.

Table 3. Retrieval with abstract cases

retrieval method	average retrieval time	percentage of failures
sequential retrieval	185	76 %
retrieval with abstract cases	127	58 %

Table 3 shows the average results for retrieving 100 different cases. We can see a good improvement in the retrieval time as well as a reduction in the number of failures due to exceeding of the time limit.

4.3 Problem Solving Performance

The purpose of this experiment is to evaluate the overall problem solving performance and competence for reusing abstract cases vs. reusing concrete cases. From the 100 available cases, we have randomly chosen 10 training sets of 5 cases and 10 training sets of 10 cases. These training sets are selected independently from each other. For each training set, a related testing set is determined by choosing those of the 100 cases which are not used for training. We trained PARIS with each of the training sets separately and measured the time for problem solving on the related testing sets. Again, a time-bound of 200 CPU seconds was used for each problem. If the problem could not be solved within this time limit, the problem solver was aborted and the problem remained unsolved. The number of unsolved problems was also evaluated.

Table 4. Reuse of abstract cases vs. reuse of concrete cases

Reuse method	size of training set	average problem solving time	percentage of unsolved problems
reuse abstract cases	5	56	16%
	10	49	13 %
reuse concrete cases	5	157	71 %
	10	154	68 %

Table 4 shows the average problem solving time and the average percentage of solved problems for the training sets of the two different sizes and the different kinds of reuse. These average numbers are computed from the 10 training and testing sets for each size. We can see a strong improvement through reusing abstract case. Additionally, these results were analyzed with the maximally conservative sign test as proposed in [10]. It turned out that in all 20 (10+10) experiments, the improvement was significant ($p < 0.05$).

4.4 Flexibility of Reuse

The purpose of this experiment was to evaluate the flexibility of the reuse. For each of the 100 cases, we evaluated how many of the problems in the remaining 99 cases could be solved through reuse of this case within the time limit of 200 seconds. We compared the flexibility of reusing concrete and abstract cases separately.

Figure 7 shows the results plotted for each case. On the abscissa, the 100 cases are ordered according the complexity. Case No. 1 is the simplest case with a plan composed of 4 operators and Case No. 100 is the most complex case containing 18 operators. The ordinate shows the number of problems (of the remaining 99 cases) for which this case can be reused. We can see a strong advantage when reusing abstract cases. The conservative sign test shows the significance of this result ($p < 0.001$).

4.5 Case Deletion Policy

Finally we evaluated the impact of the case deletion policy. For that purpose, we trained the system with all available cases and used the same cases for testing it again. In one run, the case deletion policy was active, in the other run it was disabled. Table 5 shows the average problem solving time as well as the number of problems solved within the 200 second time limit. We can identify a significant improvement (rank test, $p < 0.05$) caused by the case deletion policy.

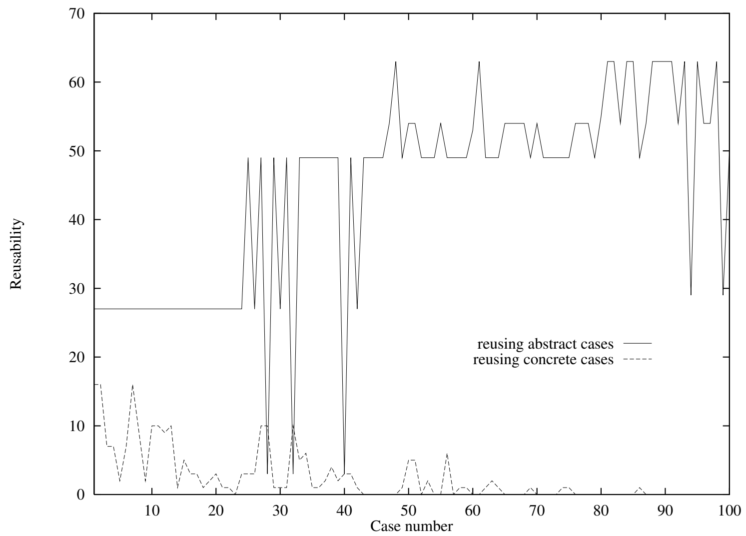

Fig. 7. Flexibility of reuse

Table 5. Case deletion policy

Case deletion policy	average problem solving time	percentage of unsolved problems
disabled	69	28 %
enabled	36	6 %

4.6 Conclusion from experiments

These experiments clearly demonstrate the benefits we hoped to gain from introducing abstraction into case-based reasoning (cf. section 1). However, these experiments are performed in a very specific scenario (planning task, domain: process planning for rotary symmetric workpieces, particular representation). Whether these results can be generalized for different tasks and domains still has to be proven. However, the results obtained by Branting and Aha [7] – also for a planning task – strongly support our results.

5 Related Work

We now discuss related work with respect to the general framework introduced in section 2. We consider the following approaches that reuse cases at several levels of abstraction:

– Déjà Vu [29; 30]: design of control software,
– PRIAR [16]: domain-independent action planning,
– MoCas [24; 4]: model-based case adaptation for diagnosis of technical systems,
– COVER and CLOSEST [7]: algorithms for hierarchical A* search.

Table 6 shows the result of classifying these approaches according to the framework.

kind of stored cases:	*abstract*	*abstract & concrete*	*hierarchical*	
		PARIS DEJA-VU COVER CLOSEST	PRIAR MOCAS	
acquisition of abstract cases	*cases available*	*manual generation*	*automatic generation*	
	PRIAR COVER DEJA-VU CLOSEST		PARIS MOCAS	
abstract cases for indexing:	*no*	*yes*		
	MOCAS PRIAR	PARIS COVER CLOSEST DEJA-VU		
reuse of abstract solutions:	*no*	*abstract result*	*generative refine.*	*case based refine.*
		MOCAS	PARIS PRIAR COVER	DEJA-VU CLOSEST
adaptation of abstract solutions:	*no*	*yes*		
	COVER	PARIS CLOSEST PRIAR MOCAS DEJA-VU		
case deletion policy:	*no*	*yes*		
	COVER CLOSEST PRIAR MOCAS DEJA-VU	PARIS		

Table 6. Comparison of other approaches using the framework

Kind of stored cases We can see that there is no approach that is limited to the reuse of abstract cases. All approaches that reuse cases at several levels of abstraction always include the concrete level. The reason for this is that the methods of reusing abstract cases can also be directly applied to concrete cases. However, the experiments with Paris have shown that it is often not useful to reason with concrete cases, if abstract cases are available. It turned out that the case deletion policy, which has been shown to improve performance, tends to delete almost all concrete cases, if the case-base is growing.

Acquisition of abstract cases Almost all current approaches assume that cases are available at several levels of abstraction. Priar, COVER, and CLOSEST are based on a hierarchical problem solver. Consequently, cases at all levels of abstraction are available when problems become solved. Paris and MoCas are the only system which supports the automatic generation of abstract cases out of concrete ones.

Abstract cases for indexing Not all approaches make use of abstract cases as a means for indexing. Some approaches does not address the retrieval problem in deep (e.g. MoCas and PRIAR).

Reuse of abstract cases MoCas seems to be the only system that directly returns abstract solutions to the user. The reason for this is that in the diagnostic domain, abstract solutions correspond to a complex component. Knowing that a complex component is defect may be as useful as knowing that a particular part of this component (concrete solution) causes the failure. All approaches that refine cases by a generic method are built on the integration of a from-scratch problem solver. Paris uses a depth-first iterative-deepening search, CLOSEST uses A^* search, and Priar uses a hierarchical task network planner.

Adaptation of abstract solutions Almost all systems make use of the advantage that adapting an abstract case is simpler than adapting a concrete case. COVER explicitly renounces adaptation for experimental purposes.

Case deletion policy Till now, PARIS is the only system that makes use of a case deletion policy. This policy has shown to significantly improve the performance of the system (cf. section 4.5). However, Branting and Aha [7] describe a variant of the CLOSEST algorithm (called CLOSEST-THRESHOLD) that stops the CBR process if reuse is more expensive than from-scratch problem solving. This algorithm does not delete any cases.

6 Conclusion

The presented framework for reusing cases at several levels of abstraction shows in its facets different possibilities of how abstraction can be incorporated into CBR. Unfortunately, there are only a few systems available that make full use of the abstraction idea. Most of them are in the area of planning. Future research should try to use this framework for developing similar approaches for different tasks and domains (e.g. configuration) to validate the positive results gained so far.

Acknowledgements
The authors want to thank Klaus-Dieter Althoff and Hector Munioz for helpful remarks on earlier versions of this paper. This research was partially funded by the Commission of the European Communities (ESPRIT contract P6322, the INRECA project). The partners of INRECA are AcknoSoft (prime contractor, France), tecInno (Germany), Irish Medical Systems (Ireland) and the University of Kaiserslautern (Germany) and partially funded by the "Stiftung Rheinland-Pfalz fuer Innovation".

References

1. A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59, 1994.
2. R. Bergmann. Learning plan abstractions. In H.J. Ohlbach, editor, *GWAI-92 16th German Workshop on Artificial Intelligence*, volume 671 of *Springer Lecture Notes on AI*, pages 187–198, 1992.
3. R. Bergmann. *Effizientes Problemlösen durch flexible Wiederverwendung von Fällen auf verschiedenen Abstraktionsebenen.* PhD thesis, University of Kaiserslautern, 1996.
4. R. Bergmann, G. Pews, and W. Wilke. Explanation-based similarity: A unifying approach for integrating domain knowledge into case-based reasoning. In S. Wess, K.-D. Althoff, and M.M. Richter, editors, *Topics in Case-Based Reasoning*, volume 837 of *Lecture Notes on Artificial Intelligence*, pages 182–196. Springer, 1994.
5. R. Bergmann and W. Wilke. Building and refining abstract planning cases by change of representation language. *Journal of Artificial Intelligence Research*, 3:53–118, 1995.
6. K. Börner. Toward formalizations in case-based reasoning for synthesis. In David L. Aha, editor, *Proceedings AAAI-94 Case-Based Reasoning Workshop*, 1994.
7. K. Branting and D. Aha. Stratified case-based reasoning: Reusing hierarchical problem solving episodes. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 384–390, 1995.
8. P. Cunningham, D. Finn, and S. Slattery. Knowledge engineering requirements in derivational analogy. volume 1 of *LNAI*, pages 234–245. Springer Verlag, 1994.
9. G. DeJong and R. Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145–176, 1986.

10. O. Etzioni and R. Etzioni. Statistical methods for analyzing speedup learning. *Machine Learning*, 14:333–347, 1994.
11. R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.
12. A. G. Francis and A. Ram. The utility problem in case-based reasoning. In *Proceedings AAAI-93 Case-Based Reasoning Workshop*, 1993.
13. F. Giunchiglia and T. Walsh. A theory of abstraction. *Artificial Intelligence*, 57:323–389, 1992.
14. K. Hanney, M. Keane, B. Smyth, and P. Cunningham. Systems, tasks and adaptation knowledge: Revealing some revealing dependencies. In M. Veloso and A. Aamodt, editors, *Case-based Reasoning Research and Development*, volume 1010 of *Lecture Notes in AI*, pages 461–470, 1995.
15. R. C. Holte, T. Mkadmi, R. M. Zimmer, and A. J. MacDonald. Speeding up problem solving by abstraction: A graph-oriented approach. Technical report, University of Ottawa, Ontario, Canada, 1995.
16. S. Kambhampati and J. Hendler. A validation-structure-based theory of plan modifications. *Artificial Intelligence*, 1992.
17. C. A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68:243–302, 1994.
18. J. L. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, 1993.
19. R. E. Korf. Planning as search: A quantitative approach. *Artifical Intelligence*, 33:65–88, 1987.
20. M. Minsky. Steps toward artificial intelligence. In E. Feigenbaum, editor, *Computers and Thought*. McGraw-Hill, New York, NY, 1963.
21. S. Minton. Quantitativ results concerning the utility of explanation-based learning. *Artifical Intelligence*, 42:363–391, 1990.
22. T. M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1(1):47–80, 1986.
23. I. Mozetic. Abstraction in model-based diagnosis. In *AAAI Workshop on Automatic Generation of Approximations and Abstractions*, pages 64–75, Boston, MA, 1990.
24. Gerd Pews and Stefan Wess. Combining case-based and model-based approaches for diagnostic applications in technical domains. In *Proceedings EWCBR93*, volume 2, pages 325 – 328, 1993.
25. D. Plaisted. Theorem proving with abstraction. *Artifical Intelligence*, 16:47–108, 1981.
26. G. Polya. *How to solve it*. Princeton University Press, Princeton, N.J., 1945.
27. B. W. Porter, R. Bareiss, and R. C. Holte. Concept Learning and Heuristic Classification in Weak-Theory Domains. *Artificial Intelligence*, 45, 1990.
28. E.D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.
29. B. Smyth and P. Cunningham. Deja vu: A hierarchical case-based reasoning system for software design. In *ECAI-92*, pages 587–589, 1992.
30. B. Smyth and M. Keane. Retrieving adaptable cases. In S. Wess, K.-D. Althoff, and M. M. Richter, editors, *Topics in Case-Based Reasoning*, pages 209–220. Springer, 1994.
31. B. Smyth and M. Keane. Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems. In Chris S. Mellish, editor, *Proceedings of the International Conference on Artificial Intelligence*, pages 377–383. Morgan Kaufmann Publishers, 1995.
32. M. Tambe and A. Newell. Some chunks are expensive. In *Proceedings of the 5th International Conference on Machine Learning*, pages 451–458, 1988.
33. J. Tenenberg. *Abstraction in Planning*. PhD thesis, Computer Science Department, University of Rochester, New York, 1988.
34. A. Voss. Exploiting previous solutions - made easy. ftp://ftp.gmd.de//GMD/ai-research/Publications/Fabel/Prev-sol-voss.ps.gz, 1995.
35. W. Wilke. Entwurf, Implementierung und experimentelle Bewertung von Auswahlverfahren für abstrakte Pläne im fallbasierten Planungssystem PARIS. Master's thesis, Universität Kaiserslautern, Germany, 1994.