19. Case-based Representation and Adaptation in Design

Shen-Guan Shih

Lehrstuhl für Architektur und CAAD
ETH Zurich
CH 8093 Zurich, Switzerland

By attempting to model the raw memory of experts, case-based reasoning is distinguished from traditional expert systems, which compile experts' knowledge into rules before new problems are given. A case-based reasoning system processes new problems with the most similar prior experiences available, and adapts the prior solutions to solve new problems. Case-based representation of design knowledge utilizes the desirable features of the selected case as syntax rules to adapt the case to a new context. As a central issue of the paper, three types of adaptation aimed at topological modifications are described. The first type - case-based search - can be viewed as a localized search process. It follows the syntactical structure of the case to search for variations which provide the required functionality. Regarding the complexity of computation, it is recognized that when a context sensitive grammar is used to describe the desirable features, the search process become intractable. The second type of adaptation can be viewed as a process of self-organization, in which context-sensitive grammars play an essential role. Evaluations have to be simulated by local interaction among design primitives. The third type is called direct transduction. A case is translated directly to another structure according to its syntax by some translation functions. A direct transduction is not necessarily a composition of design operators and thus, a crosscontextual mapping is possible. As a perspective use of these adaptation methods, a CAD system which provides designers with the ability to modify the syntactical structure of a group of design elements, according to some concerned semantics, would support designers better than current CAD systems.

Introduction

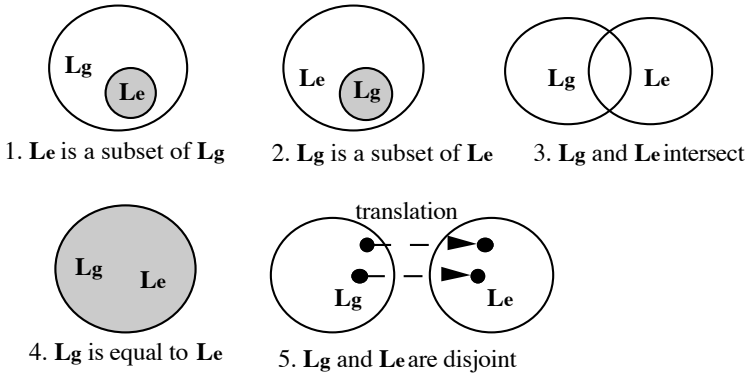
Regarding the psychological phenomenon of reminding in human thinking (Schank 1982), case-based reasoning was considered as a problem solving paradigm in which prior solutions are reused to solve new problems. For problems involving combinatorial search, the reuse of an existing near-hit solution might save a large amount of the computation in finding a solution for similar problems. For many domains where design knowledge is difficult to acquire and may not be objectively applicable, the case-based paradigm provides a model for the acquisition, organization and reuse of specific knowledge. Applying to design problems, case adaptation is chosen among several important issues in the case-based approach to be the focus of this paper. As an operational model of case adaptation, case-based representation of design knowledge is introduced as the basis for discussion.

Adaptation, explained as "a change in structure, function, or form to conform to the environment", is an act which appears repeatedly in the process which brings a design from being conceived to being mature. Current computer aided design systems provide only primitive tools for the modification of designs. To support the adaptation of designs, we clarify the motivation so as to provide designers tools for modifying the syntactical structure of a group of design elements according to some concerned semantics. The task can be further divided into three levels: to discover alternations of the given design in the same context, to modify the given design for the accommodation of new constraints, and to adapt the given design to solve a new problem. Such a system can be defined as a function which takes design descriptions as well as evaluations to be input and then generates designs which comply with the given evaluations.

Following the principle of problem solving, it is assumed that such a design adapting system consists of a generator, which defines the problem space, and an evaluator, which verifies solutions. To enable the discussion, both generating and evaluating processes are assumed to be performed by production systems. By "production systems", we mean those systems that can be defined by N. Chomsky's Phrase Structure Grammar (Chomsky 1957, 1963) or its subsets such as context-free grammar. There exists some kind of mapping between strings of symbols and any representational model of design since the manipulation of strings is the only way that computations can take place at the very primitive level. Accordingly, the generator and the evaluator can be viewed as two grammars, each of which defines a language.

The Integration of Synthesis and Analysis Knowledge

From the computational point of view, synthesis and analysis, the two classes of intelligent activities involved in design process, are performed by the generator and the evaluator respectively. The generator, which defines the vocabulary and the syntax of design, embeds synthesis knowledge. The evaluator, which verifies the semantics of design, embeds analysis knowledge. In the following discussion about the integration of synthesis and analysis knowledge, we use the classifications of constraints defined by C. Eastman (Eastman 1988). Representing analysis knowledge, constraints can be divided into three groups: universal, associated, and unassociated. Universal constraints can be embedded in the design representation in order to reduce the size of the problem space without losing its generality. The integration of universal constraints implies that the vocabulary of the generator is mapped to design elements of a higher structure. Associated constraints can be embedded in design operators, or in other words, in productions, to reduce the size of the search space. Sometimes, this increases the complexity of the grammar of the generator, and increases the difficulty of computation. Unassociated constraints which can be defined with computer programs, exist only if we restrict the grammar to being **context** free. Since the composition of primitive recursive functions is still primitive recursive and the composition of Turing computable functions is still Turing computable, and thus, the full integration of a generator and an evaluator is possible under the consideration of either solvability or computability.



Lg: the language defined by the generator.
Le: the language defined by the evaluator.

Figure 1. Possible relationships between the two languages defined by the generator and the evaluator.

The degree of integration of synthesis knowledge and analysis knowledge can be defined as the degree to which the language defined by the generator approximates to the language defined by the evaluator. The first case in figure 1 illustrates an example of partial integration. One way to increase integration is to embed more knowledge which is not really universal into the design vocabulary and inevitably, to sacrifice some generality. This could result in one of the variations shown in the second and third diagrams. Another method is to increase the expressive power of the grammar in order to accommodate more associated constraints. This reduces the size of the search space without losing generality, but it may cause the complexity of the grammar to increase. Carrying this approach to an extreme, the fourth diagram shows the full integration of the generator and the evaluator. Except for some relatively easy problems, it would become very difficult for the production system to reach a terminal state. The last diagram illustrates non-integration. In this case, the problem space does not contain any members which can be recognized by the evaluator as solutions. The generator possesses absolutely no knowledge about the evaluation which identifies the goal state. The generator and evaluator might represent two different domains such as shoe making and building evaluation. Searching within such a problem space will not lead to any solution, but there could be cross-contextual translations which provide meaningful connections between the two, seemingly unrelated domains.

Design and Dynamic Systems

Mathematical systems can be categorized as either static or dynamic. A static system is not concerned with the passage of time; whereas a dynamic system evolves over a period of time and can be formulated as the following format:fn-f(f_n-1). Along with some other conditions, such systems may behave chaotically. For a numerical function, if there is a nonlinear relationship between the input and output of each operation, the result of the recursive applications may approach one limit, or may be oscillations of different sizes, or in other cases,

be totally unpredictable. When the system is stable, it can be analyzed and predicted. When the system behaves chaotically, it can only be simulated but not analyzed.

A design generator can be viewed as a function which recursively uses its own output as the input of the next operation, and thus it can be categorized as a dynamic system. Design evaluators may be static systems in some cases. But in some design problems, they can be dynamic and can only be simulated but not analyzed. For architectural design, such problems include elevator waiting times, dynamic thermal behavior, and pedestrian circulation (Mitchell 1977).

In a generative design system, when the generator presents a chaotic behavior, it is impossible to predict what the design state is going to be after a number of operations without actually performing the operations. When the evaluator presents a chaotic behavior, it may not be possible to recognize a solution in a limited time bound. Returning to the view of generators and evaluators as production systems, whether the behavior of the system is predictable or not is a important factor which decides how effectively a heuristic search can be carried out. To discuss the behavior of production systems, it is helpful to recognize the computational complexity of the grammars which define them. Context-free grammar is an important class of grammars in the paradigm of formal languages. According to the definition of context-free grammar, all productions in the grammar must have exactly one non-terminal on its left hand side. Therefore, the applicable derivations of each non-terminal are constant. Context-sensitive grammar is a super class of context-free grammar. It allows us to have more than one terminal or non-terminal in the left hand side of productions. The applicable derivations of each non-terminal in a context-sensitive grammar are dependent on its context. Taking into account the other observations of these two classes of grammars which are addressed in the following sections, we speculate that those production systems defined with context-sensitive grammars may behave chaotically and those which are defined with context-free grammars do not.

The Problems of Using Compiled Knowledge for Design Synthesis

In the development of a generative system, the definition of design operators is very important. It implies a way of design decomposition which is beyond the level introduced by design vocabulary. When the decomposition is compatible to that which is required for the design evaluation, the search for a solution can be carried out in a effective way, and causal relationships among operations and evaluations can be established. For many design problems, a consistent decomposition of design structures which complies with the requisites of various evaluations may not exist. This is a problem of systems which introduces a pre-defined generator to embed synthesis knowledge.

In some fields of design, knowledge which is objectively applicable may not sufficiently justify all design decisions. Architecture, which is considered as one of the three types of art, has rejected all attempts to describe it with general and logical formalisms. Subjective preferences and intuitions often play important roles in the design process. In such cases, there exists knowledge which is meaningful only at a specific time, and for specific designers. To encode such knowledge and to decide when and where it should be applied is not practical. However, without such kind of knowledge, the search for a solution can not be done efficiently.

Case-based Representation of Design Knowledge

By attempting to model the raw memory of experts, case-based reasoning is distinguished from building expert systems, which compiles an expert's knowledge into rules before new problems are given. A case-based reasoning system processes new problems with the most similar prior experiences available, and adapts the prior solutions to solve new problems. This might avoid the inefficiency of solving problems from scratch every time, as well as the difficulty of deriving generalized knowledge from experts. At the same time, it also provides a way to improve its own performance by learning from new experiences.

To represent architectural design knowledge, B. Faltings proposed a model of case-based representation (Faltings et al 1991), which utilizes the desirable features of the selected case as syntactical rules to adapt the case to a new context. Given a design problem, a case which was designed for a similar context is selected from the case library. The selected case is then adapted to the new context through proper modifications which resolve the conflicts caused by differences between the original and the new context. A case is defined as a complete and precise description of a previous design solution for a specific situation. In addition to the design description, desirable features, the invariant that must not be changed during the process of modification, are to be provided in terms of verification methods. Applying this definition to our discussion about generators and evaluators, the desirable features can be described as a grammar and thus, define a language, which includes all possible designs that have the same features. Furthermore, the case serves as an initial state of the search for solutions which are supposed to have similar structures. That is to say, the selected case and the desirable features play the role of a generator in solving the new design problem. Theoretically, if the desirable features can be described in a computer program, then they can be converted into an equivalent grammar, which can produce design alternatives by applying its productions. A more detailed discussion of this can be found in the sections about case adaptation.

The use of cases as generators avoids the necessity of defining a general generator which is supposed to be applied to all given problems. If the desirable features of the selected case are essential to the perspective solutions, we can focus on each specific problem to define a good generator for the specific situation. The effectiveness of the generator will therefore rely on the adequacy of the selected case and the desirable features. Of course this does not reduce the difficulty of the problem at all. Some difficulties are simply shifted to the subprocess of case retrieval.

The dividing of the problem solving process into case selection and case adaptation provides human designers with a way to express their intentions explicitly, implicitly, and subjectively through the selection of cases. The explicit intentions are described in terms of verification methods. The implicit intentions are hidden inside the case and some of them can be detected by generalized knowledge. The fact that designers often use cases to communicate with each other when both have senses in common implies that generalized knowledge can be used to derive some of the desirable but implicit features. Designers can also select the case according to their preferences.

Case-based representation of design knowledge avoids the problem of using a fixed decomposition to find solutions for different problems. It makes use of cases to provide a

more efficient way of searching. Furthermore, designers can express their intentions and can contribute their experiences in a way which they are more familiar with - case selection.

Case Adaptation

When a case is selected for application to a different context, proper modifications of the case are required in order to recover the discrepancy caused by the differences between the new and the original contexts. The modification requires two types of operations: geometrical and topological. The required geometrical operation determines the dimension of each design element according to a set of numerical constraints which maintains the integrity as well as the required functionality of the design. In our development of a prototype system, the technique of dimensionality reduction is used to derive independent parameters which are essential to the resolution of discrepancies caused by the differences of context. Detail discussion can be found in a previous paper (Failings et al 1991). Topological operations are required when the change of dimensions alone can not successfully adapt the case. In the following figure, the U-shape case used for adaptation is shown at the left and the output of our prototype system after adaptation are shown at the middle and the right.

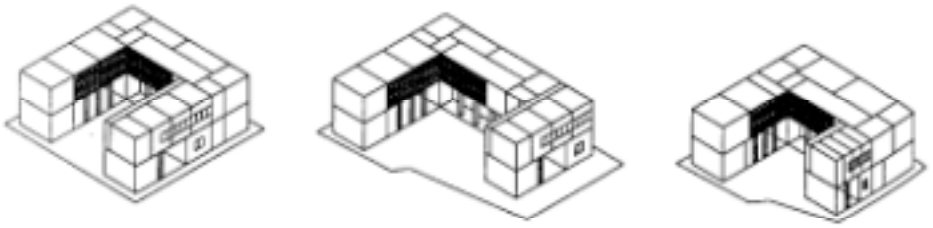


Figure 2. left: the case used for adaptation
middle: the result of topological and geometrical operations
right: the result of only geometrical operation

In this case, the U-shapedness is defined as a desirable feature. The building at the right hand side is a result of pure dimensional modification which resolves the discrepancy caused by the change of the site. The result is not satisfactory because some spaces are reduced to unusable sizes. The system ignores some constraints in order to resolve the over-constrainted problem. The building in the middle of the figure shows the result of both geometrical and topological operations. The adaptation changes the locations of some spaces but maintains the U-shapedness and most functional relationships.

In the following sections we focus on the discussion of topological operations for adaptation. Three types of adaptation under different levels of computational complexity will be discussed. The three types are case-based search, self-organization, and direct transduction.

Context Free Grammar and Case-Based Search

The first type - case-based search - is basically a localized search for the intersection of the two languages defined by the generator and the evaluator. Suppose that a case has been selected, which is represented by C, and its desirable features can be verified with a grammar E_1. Therefore, C is a member of $L(E_1)$, the language defined by E_1. There is another grammar E_2, which defines the goal. The design solution D has to be verified by E_2 and thus D is a member of $L(E_2)$. The purpose of adaptation is to modify the case C such that it becomes a member of $L(E_2)$ without losing the desirable features defined by E_1. It is supposed that the case is selected properly such that E_1 specifies syntactical structures which are essential to the resolution of E_2.

The grammar E_1 can be used as a generator if its instances can be derived efficiently. This would require that the derivations do not lead to too many dead-ends or loops. An ideal case occurs when the derivations of the grammar can be described as a tree with all of its leaves in terminal states. In such a case, any derivation sequence will eventually lead to an instance of the language. The selected case serves as a focal point from which the system starts to search in nearby branches for a design alternative which is also a member of $L(E_2)$. In this circumstance, the derivation sequence which leads to the generation of C is required to enable backtracking.

When the grammar of the generator is context free, the search space can be described as a tree if some priorities of operations are introduced. Ambiguity in the grammar can be avoided by introducing priorities or can be handled by including all possibilities. We can parse the selected case to derive the path which generates the case. By backtracking the path, we are able to search the neighboring branches and test for solutions. Since the relationships between design components are fixed after they are established in a context free grammar, branch and bound search can be effective for some evaluations and thus, reduce the search space to a more tractable size. The terminal nodes being realized can be used to determine the lowest bounds of the search since they will not be removed afterwards.

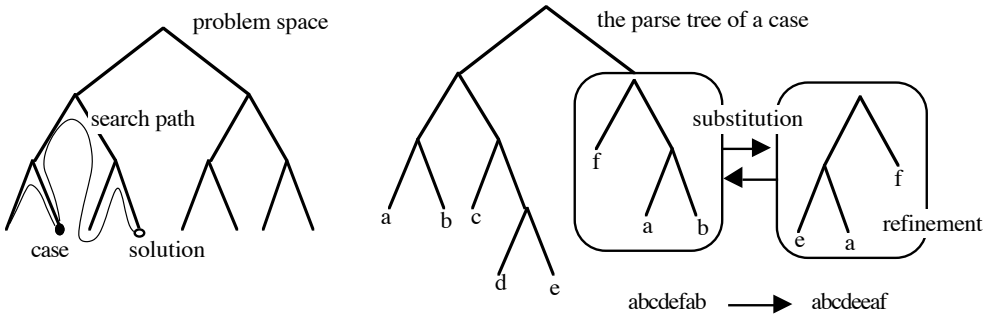


Figure 3. a. case-based search in a hierarchical problem space.
b. the substitution and partial refinement of a parse tree.

The diagram in figure 4a shows a search path from a case to a solution. Each node of the tree represents a design state. Each edge represents an operation performed on a design state. When causal relationships between operations and evaluations can be established, it is possible to identify the part of the design which causes discrepancies between the selected case and the new problem. The trouble maker can then be replaced and refined. The diagram in Figure 4b shows the decomposition of a case as a parse tree. The adaptation of the case appears as a process of substitution and refinement of a part of the design.

Although context-free grammars have relatively limited expressive power, their tractability may overcome this drawback. Context-free grammars may not be able to describe complicated features, but a larger language can be defined that encompasses them and a filter used to filter out the unwanted. If we make an analogy between programming and designing, a programming language is like the syntax of a class of designs, and a program is like a design. The grammar checks the correctness of the syntax, but the semantics is handled mainly by the programmer and has to be tested by its input and output. The unsolvable halting problem implies that no grammar can fully guarantee the correctness of a program. In design, we should not expect the grammar to be able to guarantee the quality of the design either, therefore designs have to be tested by required evaluations. Considering that the syntax of most programming languages are defined with a subset of context-free grammar, it might be more appropriate to have also a context-free grammar for design in which the interpretation of design description can be more tractable.

Making up for its poor expressive power, the tractability of context-free grammar provides the possibility of performing analytical operations on design descriptions. Since a generator is constructed according to desirable features, a context free grammar allows one to decompose the selected case, or the derived design, to various levels in a hierarchy for analytical purposes. Such an advantage is used in the design of some compilers, where some of the semantics of programs such as type checking and variable declaration can be checked efficiently within compilation. Optimization or reuse of programs is also possible.

It has been shown that desirable features can be derived by learning from examples. Tree automata (Thatcher 1967, 1973) can serve to derive common features of syntax tree of context-free grammars. In the work of C.A. Mackenzie (Mackenzie 1989), methods of grammatical inference were used to recognize Palladian-like villa plans based on relational grammar by inferring from 10 examples. Thereby, the desirable features of cases could be derived from multiple selections, and the knowledge which is specific to cases could be acquired automatically.

Case-based search can be more efficient than search from scratch if there are solutions in the nearby branches of the selected case. The tractability of the search relies on the complexity of the grammar. When the grammar is context-free, the decomposition of a design description can be derived for back tracking, desirable features can be inferred from multiple selection of cases, and heuristic search strategies can be more reliable.

Context Sensitive Grammar and Self-Organization

When the generator is defined with a context sensitive grammar, the problem space will be more complex, though it can more precisely approximate the solution set. There is no general way to parse a selected case in order to find the derivation sequence which generates

the case, because such languages can not be recognized in a limited time bound. If casebased search is to be used as a strategy for case adaptation under context-sensitive grammars, a full description of the case needs to include the derivations which generated the case itself, to enable a search for close instances. The search would be inefficient because of the existence of ambiguity, loops and dead ends. The branch-and-bound algorithm is not applicable because the relationships between design elements might change after they are established. The decomposition of a design would then be ambiguous. An example is shown in the following figure.

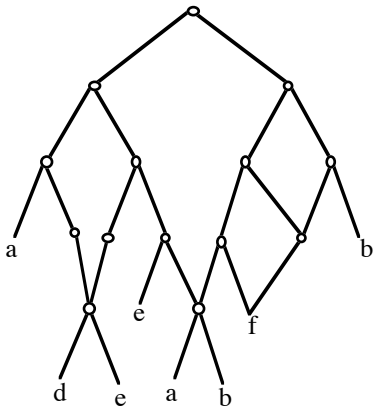


Figure 4. Possible derivations which generate the string "adeeabfb" with a context-sensitive grammar.

Based on the above observations about context free and context sensitive grammars, it is suspected that a generator defined with a context-free grammar has a stable behavior and is thus more or less predictable. A generator defined with a context sensitive grammar may behave chaotically and is unpredictable. For a specific generator and evaluator, it may be possible to define a specific strategy for resolving the intractability of a specific context-sensitive grammar. With a case-based reasoning approach, the generator is not defined ad hoc, and thus, the search process is left to be intractable. In such cases, a case-based search will not be feasible for case adaptation.

The second type of case adaptation comes from the concept of self-organization. It re-quires that evaluations be approximated by knowledge about local interactions among primitive elements and that they be fully integrated into the generator. Such systems can be viewed as systems of self-organization and can be adaptive to their environments. The growth of an embryo, ants' construction of an ant hill, or the development of an urban area, are all natural phenomena of self-organization. Each case can be viewed as a result of a self-organizing activity. If it is possible to replay the process with a new context initialized, a system might demonstrate an adaptive behavior.

The difficulty is the approximation of global evaluations with local interactions. It is more proper to simulate only localized adaptive behavior. Context-sensitive grammar plays an essential role in this type of adaptation. The grammar defines the derivation of each design element according to the immediate context of the element. For computational

purposes, cellular automata (Von Neumann 1966) and interactive Lindenmayer grammar (Lindenmayer 1968) are potential models for the simulation of self-organizations. Achievements have been shown in this way in various fields such as biological simulations, fluid dynamic simulations, and urban development. J. Gero's Prototype (Gero 1987) can be used as a model for the simulation of localized adaptation in design domains. Prototypes, as abstractions of design elements, define possible derivations of the elements with domain knowledge. If a prototype is linked to the design context, domain knowledge can be used to match contextual situations to find if any action can take place to further refine the prototype. The refinement may then trigger chain reactions of other prototypes.

Direct Transduction

The third type of adaptation is direct transduction. Through a proper translation function, a case can be translated directly into the language defined by the evaluator without search or simulation. The transduction is not necessarily a composition of design operators and thus, may exceed the range of the problem space. A meaningful transduction has to embed the semantics, which are usually verified by the evaluator.

Syntax directed translation is one simple example of such transductions. Syntax directed translations can be derived from the permutations of non-terminals and substitutions or alterations of terminals of each production in the grammar. In such transductions, design elements can be substituted by other elements, and the syntax structure of the design may be altered as well. An example is **illustrated** as below:

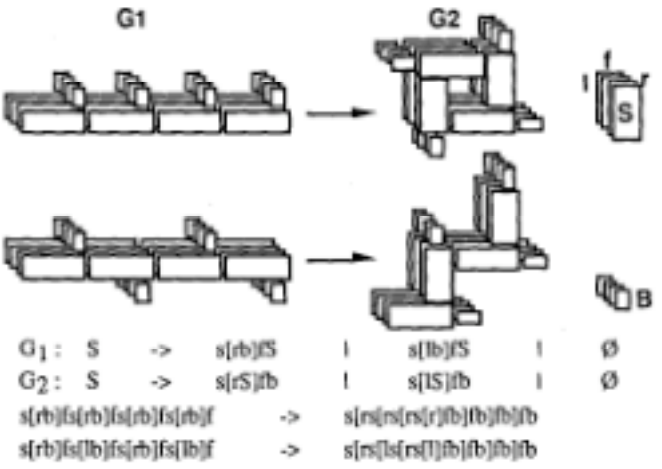


Figure 5. An example of syntax directed translation.

Here we use a notation which originated in logo turtle and was used in Lindenmayer grammar to map a string of symbols to graphics (Lindenmayer 1968). In grammars G_1 and G_2, the letters s and b represent the design elements shown in the figure. These elements are placed by a imaginary turtle according to its current position and orientation, the symbols r, l, and f represent the transformations required to move the turtle to the right wing of s, left wing of s, and front of s respectively, the symbols [and] pushes and pops the state of the turtle to and from a stack.

This type of adaptation enables cross-contextual mapping between designs. It means that a design solution might be found by applying cross-contextual mapping to another design for a different design domain. Sycara and Navinchandra indicated in their study (Sycara and Navinchandra 1991) that cross-contextual reminding is one of the two processes of using cases creatively. After such a reminding has been done, cross-contextual mapping would be a necessary process to finish this creative use of case.

Conclusion

By introducing cases, the search for a solution to a design problem is divided into case selection and case adaptation. Concerning the selected case, desirable features, which are defined explicitly by the designer or detected by generalized knowledge, define the space of possible solutions. In the first type of adaptation discussed in this paper, the case serves as a starting point for the search of design solutions. To search for similar design alternatives, the derivations which generate the case have to be derived by parsing when the grammar is context free, or be given when the grammar is context sensitive. The search for a new solution will be more tractable when the generator is defined with a context-free grammar. When the generator is defined with context-sensitive grammar, the behavior of the system can be chaotic and unpredictable. In such cases, a goal directed search or heuristic search strategy will not be reliable. Consequently, case-based search can not be carried out effectively.

Simulation of self-organization is the second type of case adaptation, in which context sensitive grammar plays an essential role. The coding of global evaluations into local interactions can be very difficult to realize. Therefore, it is more adequate to simulate localized adaptation with this approach. It can be combined with the other types of adaptation. Case-based search and direct transduction can be used to modify topological relationships of design elements; whereas self-organization is used to conform each element to its immediate context.

Direct transduction is a kind of short cut when the mapping of the two languages defined by the generator and the evaluator is known or can be derived by inference from examples. When cross-contextual reminding and mapping are applied properly, this may be a key to creativity because it can exceed the range of the problem space. The three types of adaptation can be distinguished by the embedding of semantics defined by the domain knowledge. In case-based search, semantics are examined by the evaluator. In self-organization, semantics of design are fully integrated into the generator and become a part of the syntax. In direct transduction, semantics are embedded in the translation function.

It is expected that the study of case-based representation and adaptation may lead to a change in computer aided design systems so that CAD systems become a tool in making modifications on complete solutions instead of composing pieces. More explicitly, case adaptation can be applied to the evolution of design from one version of a solution to an-

other, to the accommodation of new design constraints, or in the reuse of a prior design for a new design problem. Besides, case selection can become a process which enables designers to express their intentions by choosing cases, without going through the whole process of knowledge acquisition and knowledge engineering.

Acknowledgement

This paper describes ongoing research being carried out as part of a project in the Swiss National Research Program in Artificial Intelligence (NFP 23).

The work is a result of collaborative research with the Laboratoire d'Intelligence Artificielle, and the Institut de Construction Metallique (ICOM) of the EPFL Lausanne. Discussions with the collaborators of this project have made the development of the ideas possible. Special thanks to my advisor Prof. Gerhard Schmitt, to Prof. Boi Faltings, and to my colleagues Chen-Cheng Chen and Sharon Refvem for their advice.

References

Chomsky, N. 1957. *Syntactic Structures*. Mouton, The Hague.

Chomsky, N. 1963. 'Formal properties of grammars." In D. Luce, E. Bush, and E. Galanter (eds.). *Handbook of Mathematical Psychology,* no. 2: 323-418. New York: John Wiley.

Eastman, C. 1988. 'Automatic Composition in Design." In 1988 *NSF Grantee Workshop on Design Theory and Methodology, pp*. 158-172.

Faltings, B., K. F. Hua, G. Schmitt, and S. G. Shih. 1991. "Case-based Representation of Architectural Design Knowledge." In DARPA *Case-Based Reasoning Workshop* 1991.

Gero, J. S. 1987. *Prototypes: A New Schema for Knowledge-Based Design*. Architectural Computing Unit, Department of Architectural Science, University of Sydney.

Lindenmayer, A. 1968. "Mathematical Models for Cellular Interactions in Development." Parts I and II. *Journal of Theoretical Biology* 18: 280-315.

Mackenzie, C.A. 1989. 'Inferring relational design grammars." *Environment and Planning B: Planning and Design*. 1989, vol. 16: 253-287.

Mitchell, W. J. 1977. "The computer's role in design." chapter 2 in: *Computer-Aided Architectural Design,* New York: Van Nostrand Reinhold Company.

Schank, R. C. 1982. "Reminding and Memory." Chapter 2 in: *Dynamic Memory - A Theory of Reminding and Learning in Computers and People*. Cambridge: Cambridge University Press.

Sycara, K. P., and D. Navinchandra. 1991. "Influences: A Thematic Abstraction for Creative Use of Multiple Cases." In DARPA *Case-Based Reasoning Workshop* 1991.

Thatcher, J.W. 1967, "Characterizing Derivation Trees of a Context-Free Grammar through a Generalization of Finite-Automata Theory." *Journal of Computer and System Sciences 1:* 317-322.

Thatcher, J. W. 1973. "Tree Automata: An Informal Survey." In A.V. Aho (ed.). *Currents in the Theory of Computing, pp*. 143-172. New Jersey: Prentice-Hall.

Von Neumann, J. 1966. *Theory of Self-Reproducing Automata* (edited and completed by Arthur Burks), Univ. of Illinois Press.