Proceedings, Sixth International Joint Conference on Artificial Intelligence, Tokyo, Japan, August 1979.

RESULTS IN KNOWLEDGE BASED PROGRAM SYNTHESIS

Cordell Green, Richard P. Gabriel, Elaine Kant
Beverly L. Kedzierski, Brian P. McCune, Jorge V. Phillips
Steve T. Tappel, Stephen J. Westfold

Systems Control, Inc.
1801 Page Mill Road
Palo Alto, California 94304

Abstract. This paper reviews the entire PSI program synthesis system, summarizing progress made during the past two years. PSI synthesizes efficient programs from several types of abstract specifications. The paper presents a brief summary of PSI, an example dialogue demonstrating its performance, and a discussion of its present capabilities. Explanation of the detailed operation of the system is omitted in this short paper. For an overview of prior work, see [Green-76]; for more details see [Ginsparg-78], [Steinberg-79], [Barstow-79], and [Kant-79]. [Biermann-76] surveys related work in the automatic programming field.

Keywords and phrases. Automatic programming, knowledge based programming, program synthesis, program specification, program acquisition.

1. Summary of the PSI Program Synthesis System

The PSI program synthesis system is a computer program that acquires high level descriptions of programs and produces efficient implementations of them. PSI's operation is factored into two parts: the *acquisition phase*, which acquires the high level description, and the *synthesis phase*, which produces a program from it. Simple symbolic computation programs are specified through dialogues between the user and PSI. The specification techniques available include natural language, input-output pairs, partial traces, and a high level specification language currently under development. These specifications are integrated in the program model. It is also possible to specify a problem directly in the modelling language. LISP programs are produced from the model, but experiments have shown that the system can be extended to produce code in a block structured language such as PASCAL.

The design and implementation of PSI have involved many people. The system's modules are currently the responsibility of the following individuals: parser/interpreter, Beverly Kedzierski; explainer, Richard Gabriel; trace and example inference expert and domain expert, Jorge Phillips; program model builder, Brian McCune; and efficiency expert, Elaine Kant. Additional personnel have worked on various projects within these experts. Steve Tappel wrote the rule expander for the program model builder; Stephen Westfold enhanced the examples component of the inference expert. A description of the interactions of the phases and modules of PSI can be found in [Green-77].

2. Recent Progress on and Present Capabilities of PSI

The entire PSI system can be run by a knowledgeable user. A number of tests of the entire system have been made. PSI has successfully produced LISP code that implemented the specifications given in English dialogues for four variants of CLASS, a symbolic pattern classification program.

CLASS is a simple program that includes many programming constructs necessary for more complex programs. CLASS first inputs a sample set of items (e.g., a *list of job qualifications*). Then the program repeatedly inputs a trial set (e.g., the qualifications of an applicant), tests whether the sample set (required qualifications) is a subset of the trial set (applicant qualifications), and prints "fit" or "didn't fit" accordingly. The program is exited if the word "quit" is typed instead of a trial set.

2.1 Acquisition Phase

The following dialogue was held with the PSI acquisition phase to specify CLASS.

PSI: What is the name of the program you want to write?
USER: CLASS.
PSI: Does CLASS have a special program type?
USER: Classification.
PSI: Please describe CLASS.
USER: CLASS first inputs a concept. Then it repeatedly accepts an input scene from the user, classifies it based on whether or not it fits the concept, and outputs this classification to the user.
PSI: What is the *structure of the concept*?
USER: A concept is a collection of relations. Each relation is just a string.
PSI: What is the exit test of the loop?
USER: Stop when the user types "quit" instead of a scene.
PSI: What is the *structure of the scene*?
USER: The same as the concept.
PSI: Please define what it means for a scene to fit a concept.
USER: A scene fits a concept if all the relations in the concept are in the scene.
PSI: What is the *format of the classification output*?
USER: Output "fit" if the scene fit, "didn't fit" if it didn't.
PSI: Are we finished?
USER: Yes.

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DDC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

From the above dialogue the parser/interpreter [Ginsparg-78] produces a program net, a common representational base used by all modules of the acquisition phase. The parser/interpreter understands over seventy programming concepts and has a vocabulary of more than 175 words. Its programming concepts include data and control structures, primitive operations, and more complicated algorithmic ideas (e.g., user-program interchanges, set construction, quantification). The system can sometimes determine the meaning of unknown words (e.g., what concept they represent) from the context in which they appear. The dialogues that the system has understood include those specifying many variants of CLASS, several variants of NEWS (a news story retrieval program), TF (a learning program that uses CLASS as a subroutine), and graph reachability.

The dialogue moderator [Steinberg-79] chooses which of the questions posed by the parser/interpreter to ask next. A number of simulated dialogues have been gathered, with a member of the PSI group playing the role of PSI and people not part of the group as users. The question choosing algorithm of the dialogue moderator has been improved by comparing its behavior with the data from these dialogues. The moderator also has mechanisms (not yet interfaced to the rest of PSI) to answer the question, "Where are we?", and most of the mechanism needed to handle a request to change topic. The moderator has handled dialogues for NEWS and variants of CLASS.

The questions that are asked of the user are quite readable and coherent. Questions use the same terms as the user did in previous sentences of the dialogue. For example, rather than asking for the definition of "A0018", PSI asks what it means for "a scene to fit a concept". The question generation system has been used in the dialogues for CLASS, NEWS, RECIPE (a recipe retrieval program), and TF. It produces about twenty substantially different sentence types. The question generator is being expanded into a more general explainer that will explain PSI's understanding of the program specification given by the user.

PSI allows programs to be specified by the use of traces and examples. The trace component of the inference expert [Phillips-77] handles simple loop and data structure inference such as that needed for the CLASS and TF dialogues. The examples component determines from an example input-output pair for a certain data object a suitable program transformation that could have carried the object from its initial to its final state.

An initial version of a domain expert for information retrieval has been implemented, using the program net as an interface with the rest of the system. The domain expert has been used in the generation of a variant of NEWS.

Preliminary designs are complete for an additional program specification technique, a formal system with the flavor of a very high level programming language. The language allows manipulation of abstract algebraic structures such as mappings and sets. The semantic support available through the domain expert will allow the use of domain specific jargon in this language. The language will allow the user to specify quickly and precisely program descriptions that have already been well thought out.

The program model builder [McCune-77] uses the program net produced by the other acquisition modules to construct a complete and consistent model of the program. There are about 350 rules in the model builder's knowledge base. Rules incorporate knowledge of mappings and primitive operations for accessing them, of procedures and procedure invocations, and of type coercion. The model builder also resolves type-token ambiguities and transforms expressions to canonical forms. It has built a number of program models that are variations of CLASS as part of the PSI system. Separately the model builder has successfully constructed a model for RECIPE.

The rule expander for model building rules makes writing such rules easier. Rule preconditions are written in a concise declarative language. Then the rule expander translates the declarative form into the required fetch and test operations, taking into account any ordering constraints that the preconditions may have and avoiding retesting preconditions unnecessarily.

A program has been written that prints concise, readable versions of program models. The internal representation of the model is designed for programming efficiency and is hard for people to understand. Listings in the concise notation are thus valuable for debugging. The model is printed in a very high level language, using a syntax similar to PASCAL. Any or all of the parts of a model may be printed, and cross-reference tables are available to index the concise listing and the original model.

The program model interpreter executes models interpretively as an alternative to coding them and running the target program. It correctly interprets all program models available. The interpreter parses arbitrary input data, including those which are of any type occurring in a tree of legal alternatives.

2.2 Synthesis Phase

The program model is refined into target language code during the synthesis phase. Dividing PSI into two separate phases allows program optimization to take different runtime environments into account. The program can be specified once and a program model built. Then different target language programs can be produced for different size estimates, probabilities, or cost functions. The programs will have the same input-output behavior, but the code will be optimized differently based on the data structure sizes or other such parameters.

For example, recall that CLASS reads a sample set of items, then repeatedly inputs a trial set and tests whether the sample set is a subset of the trial set. Since the universe of the sets is not known, a fast subset test using a bit map is not possible. So the subset test is implemented as an enumeration through the elements of the sample set, testing each element for membership in the trial set. When the trial set is small, a simple list (the same as the input format) is a good choice of representation for the sets. When the trial set is large, however, representation as a hash table may prove more efficient because the membership test is much faster. The efficiency expert checks whether such savings outweigh the cost of the representation conversion.

The knowledge base of the coder [Barstow-79] has about 450 rules. These rules have been used to code a variety of programs, including graph reachability and prime number finding. The sets and mappings used in these programs can be represented as lists, arrays, Boolean mappings, or property lists. Several versions of CLASS, RECIPE, NEWS, TF, and insertion and selection sorts have been coded. Rules about reusing the space in arrays have been written and used to synthesize in-place selection and insertion sorts.

The efficiency expert [Kant-79] has been used with the coder to write RECIPE, NEWS, insertion and selection sorts, and several variants of CLASS. In all cases different implementations are selected when different data structure sizes (for example) are assumed. More than one representation for the same data structure can be used in a program. Efficiency rules suggest the circumstances under which various representations are plausible or implausible, which greatly reduces the search space from the original space of all legal programs. Space-time cost estimates are used to compare different implementations and to identify the decisions that may have the greatest impact on the global program cost; the decision making resources are allocated accordingly.

3. Acknowledgment

This paper describes research being done at Systems Control, Inc. The research is supported in part by the Defense Advanced Research Projects Agency under DARPA Order 3687, Contract N00014-79-C-0127, which is monitored by the Office of Naval Research. The views and conclusions contained in this paper are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of SCI, DARPA, ONR, or the US Government.

4. References

[Barstow-79] David R. Barstow, *Knowledge Based Program Construction*, Elsevier North-Holland, Inc., 1979.

[Biermann-76] Alan W. Biermann, "Approaches to Automatic Programming", in M. Rubinoff and M. Yovits, editors, *Advances in Computers*, Volume 15, Academic Press, Inc., 1976, pages 1-63.

[Ginsparg-78] Jerrold M. Ginsparg, *Natural Language Processing in an Automatic Programming Domain*, Ph.D. thesis, STAN-CS-78-671, Computer Science Department, Stanford University, June 1978.

[Green-76] Cordell Green, "The Design of the PSI Program Synthesis System", *Proceedings Second International Conference on Software Engineering*, October 1976, pages 4-18.

[Green-77] Cordell Green, "A Summary of the PSI Program Synthesis System", *Proceedings of the Fifth International Joint Conference on Artificial Intelligence—1977*, August 1977, pages 380-381.

[Kant-79] Elaine Kant, *Efficiency Considerations in Program Synthesis: A Knowledge Based Approach*, Ph.D. thesis, Computer Science Department, Stanford University, 1979 (in progress).

[McCune-77] Brian P. McCune, "The PSI Program Model Builder: Synthesis of Very High Level Programs", Proceedings of the Symposium on Artificial Intelligence and Programming Languages, *SIGART Newsletter*, Number 64, August 1977, pages 130-139.

[Phillips-77] Jorge V. Phillips, "Program Inference from Traces Using Multiple Knowledge Sources", *Proceedings of the Fifth International Joint Conference on Artificial Intelligence—1977*, August 1977, page 812.

[Steinberg-79] Louis I. Steinberg, *A Dialogue Moderator for Program Specification Dialogues in the PSI System*, Ph.D. thesis, Computer Science Department, Stanford University, 1979 (in progress).