A SUMMARY OF THE PSI PROGRAM SYNTHESIS SYSTEM

Cordell Green
Computer Science Department
Stanford University
Stanford, California 94305

Abstract: This paper describes the current status of the PSI program synthesis system. It alloius program specification dialogues using natural language, traces and examples from which a high-level program model is acquired. This model is then refined into an efficient implementation of the program. PSI consists of several modules including a parser-interpreter, trace and examples inference expert, dialogue moderator, program model builder, coder and efficiency expert.

Keywords; artificial intelligence, automatic programming.

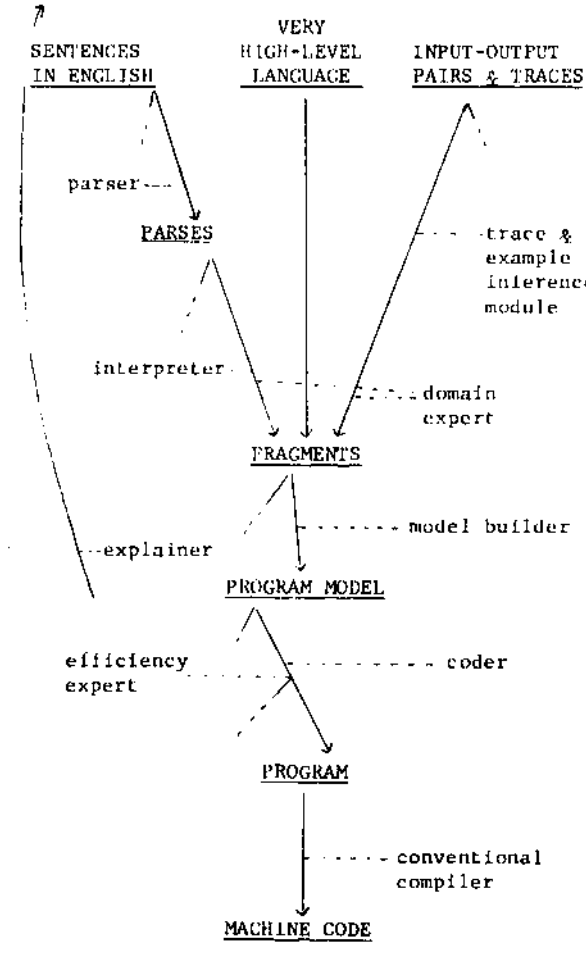
The PSI program synthesis system is a computer program that acquires high-level descriptions of programs and produces efficient implementations of these programs. Simple symbolic computation programs are specified through dialogues that include natural language, input-output pairs, and partial traces. The programs produced are in LISP or in SAIL.

PSI is organized as a collection of interacting modules or programmed experts. The overall design is a group effort with one individual having responsibility for each module as follows: parser-interpreter Jerrold Ginsparg [Ginsparg-77]; trace and example inference module Jorge V. Philips [Phillips-77]; moderator Louis I. Steinberg; domain expert Ronny van den Heuvel; model builder Brian P. McCune [McCune-77]; coder David R. Barstow [Barstow-77]; and efficiency expert Elaine Kant [Kant-77]. This paper presents a short description of PSI. For further details and discussion of related work, see the references above; for a fuller overview see the description of the design of PSI as of one year ago [Creen-76].

The major data paths and modules of the PSI system are shown in the diagram below. Multiple program specification methods are allowed in the user's dialogue with PSI, including English, input and output examples, and partial traces. A more conventional method, that of a very high-level language is a planned addition to PSI as shown in the diagram.

PSI's operation may be conveniently factored into an acquisition phase (those modules shown above the program model which acquire the model) and the synthesis phase (those modules shown below the model which produce a program from the model). Sentences are first parsed, then interpreted into fragments. The parser is a relatively general parser which limits search by incorporating considerable knowledge of English usage. The interpreter is more specific to automatic programming, using program description knowledge as well as knowledge of the last question asked and the current topic to facilitate the interpretation into fragments.

Fragments and the program model form two of the major interfaces within PSI. Both are high level program and data structure description languages. The program model includes complete, consistent, and executable (but slowly) high-level algorithm and information structures. Fragments, on the other hand, form a looser program description. Fragments occur in the order of occurrence of the dialogue, rather than in execution order, and allow less detailed, local, and only partial specification of the program. Since the fragments correspond rather closely to what the user says, they ease the burden of the interpreter as well as the trace and example inference module. The model builder must then apply knowledge of correct high level programs to convert the fragments into the model. The model builder processes fragments, checking for completeness and correctness, fills in detail, corrects minor inconsistencies, and adds cross-references. It also generalizes the program description, converting it into a form that allows the coder to look for good implementations. The completed program model may be interpreted by a special model interpreter to check that it performs as desired by the user and also to gather information needed by the efficiency expert such as statistics on set sizes and on probabilities of outcome of tests.



MAJOR INFORMATION FLOW PATHS
IN THE PSI SYSTEM

Another input specification method is partial traces. A trace includes as a special case an example input-output pair. Examples are useful for inferring data structures and simple spatial transformations. Partial traces of states of internal and I/O variables allow the inductive inference of control structures. The trace and example inference module infers loose descriptions of programs in the form of fragments, rather than programs themselves. This technique allows domain support to disambiguate possible *inferences,* and also separates the issue of efficient implementation from the inference of the user's intention. General programming knowledge is distributed throughout the modules described above. Currently, domain-specific knowledge is also distributed where appropriate, but a domain expert module is being implemented. Application domain-specific knowledge (e.g., knowledge about learning programs) will be concentrated in this module, which will supply domain support by communicating with other acquisition modules through the fragment interface.

The moderator, not shown in the diagram, guides the dialogue by selecting or repressing questions for the user. It attempts to keep PSI and the user in agreement on the current topic, provides a review-preview on a topic change, helps the user that gets lost, and allows initiative to shift between PSI and the user. A new module being planned by Richard Cabriel *i%* an explainer, which will generate reasonably clear questions about and descriptions of program models as they are acquired, in order to help verify that the inferred program description is the one desired. It also will be able to explain the how and why of the acquisition and synthesis process to the interested user.

After the acquisition phase is complete, the synthesis phase begins. This phase may be viewed as a series of refinements or as a heuristic search for an efficient program that satisfies the program model. The coder has a body of program synthesis rules [Green and Barstow-75, 76] that gradually transform the program model from abstract into more detailed constructs until it is in the target language. Both algorithm and data structures are refined interdependently. The coder deals primarily with the notions of set and correspondence operations and can synthesize programs involving sequences, loops, simple input and output, linked lists, arrays, and hash tables.

The refinement tree effectively forms a planning space that proposes only legal, but possibly inefficient, programs. This tree structure is shared by the coder and the efficiency expert. In cases where the coder proposes more than one refinement or implementation, the efficiency expert reduces the search by estimating the time-space cost product of each proposed refinement. The better path is followed and there is no backup unless the estimate later proves to be very bad. An additional planned method to reduce the size of the search space is the factoriiation of the program into relatively independent parts so that all combinations of implementations are not considered. An analysis for bottlenecks can allocate synthesis effort to more critical parts of the program.

In summary, we have formulated a framework for an automatic programming system and have a start on the kinds of programming knowledge that must be embedded therein. PSI is moderately successful in that it is currently running and has synthesized many significantly different programs including simple storage and retrieval and learning programs. However, not all of the planned modules are completed yet, and it is still too early to attempt an evaluation of the overall design, its applicability, or the methods used.

Acknowledgements

The individuals responsible for the work reported here include the current PSI members - Brian P. McCune, Jorge V. Phillips, Louis I. Steinberg, David R. Barstow, Jerrold Ginsparg, Ronny van den Heuvel, and Elaine Kant, as well as former members Bruce Nelson, Avra Cohn, and Juan Ludlow.

This research was supported by the Defense Advanced Research Projects Agency at the Department of Defense under contract MDA 903-76-C-0206.

References

[Barstow.77] Barstow, David R., A Knowledge-Based System for Automatic Program Construction, to be presented at the Fifth International Joint Conference on Artificial Intelligence, Cambridge, Massachusetts, August 1977.

[Ginsparg-77] Ginsparg, Jerrold, A Parser for English and Its Application in an Automatic Programming System, Ph.D. thesis, AI Memo, Artificial Intelligence Laboratory, CS Report, Computer Science Department, Stanford University, Stanford, California, forthcoming.

[Green-76] Green, Cordell, "The Design of the PSI Program Synthesis System", *Proceedings Second International Conference on Software Engineering,* Computer Society, Institute of Electrical and Electronics Engineers, Inc., Long Beach, California, October 1976, pages 4-18.

[Green & Barstow-75] Green, Cordell, and Barstow, David, "Some Rules for the Automatic Synthesis of Programs", *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence,* Volume 1, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, September 1975, pages 232-239.

[Green & Barstow-77] Green, Cordell, and Barstow, David, "A Hypothetical Dialogue Exhibiting a Knowledge Base for a Program Understanding System", in Elcock, E. W,, and Michie, D., editors, *Machine Intelligence S: Machine Representations of Knowledge,* John Wiley and Sons, Ltd., Chichester, England, 1977.

[Kant-77] Kant, Elaine, The Selection of Efficient Implementations for a High-Level Language, to be presented at the ACM S1GART-S1GPLAN Symposium on Artificial Intelligence and Programming Languages, Rochester, New York, August 1977.

[McCune-77] McCune, Brian P., The PSI Program Model Builder: Synthesis of Very High-Level Programs, to be presented at the ACM SIGART-SIGPLAN Symposium on Artificial Intelligence and Programming Languages, Rochester, New York August 1977.

[Phillips-77] Phillips, Jorge V., Program Inference from Traces Using Multiple Knowledge Sources, to be presented at the Fifth International Joint Conference on Artificial Intelligence, Cambridge, Massachusetts, August 1977.