Zilles: Before the conference began, I asked three people to give their perception of the conference as a whole. This is in contrast to the session oriented summaries that follow some of the topical session.

Let me begin by giving a couple of my own observations. One thing that struck me very strongly, was the role of mathematics in what we are all doing. It became very clear that every notation should have some concrete basis, that is, be defined using a well-understood framework, e.g., predicate calculus, algebraics or Scottery. Some reasons are: it helps you debug and understand your system, it helps communicate your framework to others, and, as expressed by Reiter and Paolini, you can use a formalism for comparing frameworks. One frequently made comment was that the formalism doesn't have to be the user interface; it may only be a formal tool for understanding a high-level representation. There may still be considerable effort in tailoring the user-interface to the modelling formalism.

My second observation is that in the three areas -- AI, DB and PL -- researchers speak in different terminologies and think in different ways. One reasons for this is the difference in scale of the applications. Programming languages were developed in the context of large numbers (billions) of calculations. Data bases are concerned with lots (millions) of records. And, in the AI area many (thousands of) facts are dealt with.

My main point is that the scale of these applications are orders of magnitude apart, and this influences the way we think about our problems. It influences what mechanisms we put into our modelling system. An example is exception handling in PLs, error recovery in DBs, and in AI, mechanisms for recovering from failures in heuristic search. These represent, in order, increasingly more complex mechanisms.

Another example of the differences in scale is the extent to which information is stored explicitly rather than being derived (implicitly) from more primitive information. In PL, very little implicit calculation is carried out - there is a principle of no surprises. In PL people often complain about such things as garbage collection, which is a form of implicit computation. In the DB area, it is acceptable to have "looser bindings", because the flexibility to change in the future is more important than the ability to run efficiently now. So database systems allow such things as runtime query optimization. In the AI community, there is a general reliance on (possibly complex) inference mechanisms to derive facts that are not stored explicitly.

If we have trouble communicating with each other, what then is the hope for cooperation in the future? Well, it was clear on the first day of the workshop that there was almost as much dissention within an area as there was across the areas. But it was also clear that a good idea from one area was immediately picked up and used and integrated into other people's models.

The remainder of the session consists of personal statements on the workshop by Ira Goldstein, Ted Codd, and Mary Shaw.

AN ARTIFICIAL INTELLIGENCE VIEW

Goldstein: One thing that struck me was the lack of a general statement about the theme of AI work. Long ago the theme was expressed as the Turing Test. The notion was that, given a description of a software problem or a data base design problem of the kind a person gets, you would like to build a system or a design that is acceptable by human standards. I do not plan to pull out of a shoe box a program that does DB design or automatic programming at human levels, but by recognizing that theme, you can understand some of the motivation for the kind of AI work that has been mentioned at this conference.

Some of the programmer's knowledge has found its way over the years into compilers. Programmers used to do some of the things that compilers do today, and the design of progressively better compilers is, in that sense, at the intersection of AI and PLs. Similarly, the design of smarter query optimizers sits at the intersection of DB and AI. AI has had a lot of trouble with the problem of applying the kind of general knowledge that is employed in design and programming. This is enormously harder than one might think. The "joke" in AI is that it is harder to simulate a 6-year old than to simulate a geologist. Some years ago people would not have thought that.

Here are some of the technical problems people are working on in AI.

Defaults and constraints are important because people want to bring their knowledge to bear on a given situation. People also want to look at a situation from different points of view. Networks were originally introduced because the human memory was believed to have an associative quality, and there is a sense in which algorithms on networks appear to have that quality. I am not arguing for these; I am merely remarking on them being the kind of technical solutions that AI searches for.

With respect to reasoning strategies, we have seen discussions of inheritance and analogy by Jaime Carbonell. I have seen programming by generalization from examples, we have seen discussions of heuristic search and theorem-proving, both of which are methods of finding conclusions given problem statements. More specifically, we have seen discussions of John Sowa's conceptual graphs and Gary Hendrix's partitioned semantic networks. In Chuck Rich's work, he discusses programming by inspection, that is to say, looking at a problem statement, recognizing it as being composed of instances of problems that have been solved before, pulling out a cliche from a set of familiar schemata, and adapting it to the problem at hand. One might say that is a good way for a person to program, but the AI motivation is, somewhere down the road, to capture enough of the process that a machine can do some or all of the work. The work by Balzer and Feather on the JIST language is concerned with the development of a correct implementation from a specification, but this is viewed in the context of automatic programming.

My recent work has, for the last year and a half, not really been in AI, but rather in the programming lan-

guage design, extending SMALLTALK. The purpose is to make certain structures such as defaults and constraints more explicit so that we will have a better target for using AI techniques. If I have a lower level language, then the AI task becomes progressively harder. This is my last point: the extent to which implicit domain constructs are made explicit makes the AI task easier.

Hayes: I agree with nearly all you have said, Ira, except that you appear to give the impression that AI was identical to automatic programming, which is a rather unfortunate impression.

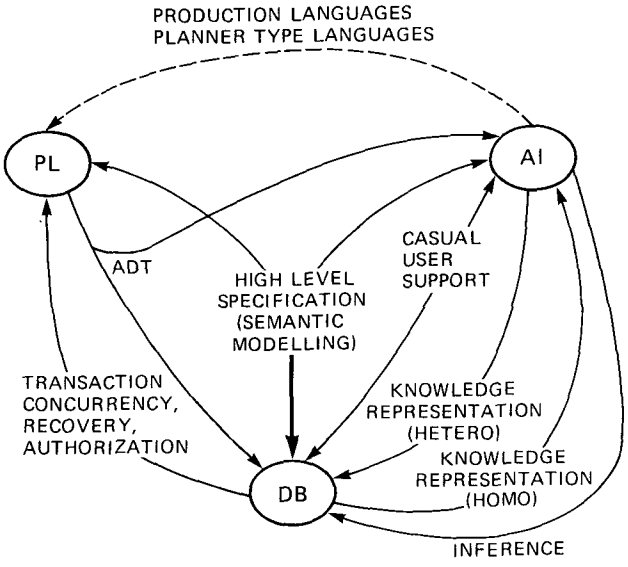
Goldstein: No. I meant automatic programming as an example of the AI problem.

Deutsch: A similar comment. There are two contexts in which the term AI system has come up at this workshop. One is AI systems as being similar to database systems or programming systems. The other is AI systems being used to represent or talk about or construct systems. The first is the one we have generally been talking about and the second is the one you chose to focus on.

Goldstein: Yes. I felt the second view was missing from the workshop and that its absence obscures some of the motivation for the AI work.

A DATABASE VIEW

Codd: I feel this workshop has been the best such event that I have attended in 31 years, in regard to stimulation and communication and all of those good things. I have become more aware of ideas of common interest, but I am one of those who needs time to digest before trying to give a sermon or put a real interpretation on what has gone on here. I am going to make some rather miscellaneous remarks, but they will be based on this little diagram.



Codd-1

When I was invited to this workshop, I was already thinking that it is very timely that PLs and DB sublanguages come together in terms of compatible support for common concepts. This does not mean, however, that I believe in unification of these fields into one programming language that will in some way address all of the possible problems that could appear in the future. I believe definitely in a specialized language, but I think that more work needs to be done on frameworks for coupling host languages to specialized sublanguages, and I am pleased to see that so much work is going on, say on the use of abstract data types to support database concepts and requirements.

An example of the difficulty that will be encountered in this very vague goal of a framework that will enable you to couple languages is the concurrency area. It seems to me that it is very difficult to think of primitives in any host language that in some way will support unanticipated concurrency requirements in some new field. Nevertheless, I feel that something needs to be done in that area. In particular, in programming languages, I think that more needs to be done about isolation concepts in concurrency, the kind of isolation constructs that have been developed in the database world.

I do not yet see any significant impact of AI languages, production languages, PLANNER-type languages, etc. on the programming language world. This may be my blindness, but don't see any impact, and I would have expected some drastic impact.

In the DB world, we have concentrated very hard on how to handle very regular data. We have been pushed into handling heterogeneous data. This is because the metadata we need in addition to the data is of a heterogeneous kind. So I think we need to make use of knowledge representation ideas from AI, just as perhaps they very well might take advantage of the ideas from the DB area.

One reason the DB world is so dynamic and forceful is not from its ideas but from the money. I think that the money will require support for handling more heterogeneous data. I think we will come to a state of affairs where the products handling the homogeneous data will be reasonably satisfactory. As we move into the more unstructured parts of companies and enterprises, so we will have to record a lot more of these heterogeneous types of knowledge. Here, I would like to say is a tremendous opportunity for joint venture by AI and DB people together.

I want to take up Pat Hayes's remark about the importance of investigating control of inference. I think we would have a lot of logical deductive power in our database systems already if we knew a little more about it and in particular could reduce the cost of it.

I want to make a remark about casual users. Both in the DB and AI worlds there are subcultures concerned with intelligent interfaces for use by the casual user. I feel there should be more support not only for translation of natural language, but more generally interactive communication such as the system asking clarification questions of the user. I think that is a great field of opportunity.

A PROGRAMMING LANGUAGE VIEW

Shaw: I would like to make three sorts of statements: (1) to observe some distinctions between the three areas, (2) to make some remarks that could be interpreted as advice from PLs to the other two areas, and (3) to remark on what needs to be addressed in PLs.

I see the following distinctions between the areas. First, data sharing and the dominant importance of data over program receives more attention in AI and particularly in DB than in PL. Second, as Steve Zilles remarked, implicit computations are voiced as a difference in AI and DB. As PLs tend toward higher-level languages, more attention will be paid to this. Thirdly,

formal techniques have received much more attention in PLs than in DB or AI. I sense that this is changing. Finally, there is a difference in texture that has been nibbling at me all week. PL refers to a means to an end, that is, we need programming languages to write programs, whereas DB and AI are ends in themselves.

Deutsch: I would say that AI is also a means to an end.

Hayes: I would strongly disagree [with Deutsch].

Shaw: We should concentrate on our goals when talking to each other and avoid getting hung up on vocabulary. Terms such as specification, constraint, type, description, etc., have been used in many different ways. For example, I have heard uses of "type" ranging from a "partition of a space of data values" to "a type is an attribute of a variable", and practically everything in between. I think all these words are still soft terms to us. I would like not to get hung up on the difference between, say, types and constraints, but rather to focus on what they are being used for.

There is some tendency in PL, and perhaps in DB, to invent a new language when a new problem comes along. In Alphard, we once had about 4 or 5 specialized sublanguages. One day we came to our senses and got rid of them. Coping with all the variations on the language can be very confusing. I think language extension is a better approach, and that we should provide in a language means for defining new operations and new constructs. Finally, I would like to urge that formal methods of various kinds receive more careful attention.

Here are the issues I am taking away as being least well attended to in PLs. They are not necessarily the most important, rather the most neglected. For example, synchronization is not on my list because it is receiving a lot of attention. Implicit computation is important, and coming soon, I think. Inconsistent or partially correct systems are an issue. In PLs, verification has been an all or nothing thing. We need to do some work on reliable systems that aren't necessarily completely correct. I have never known a completely correct operating system, for example, but they have been pretty reliable. There has been some attention to data sharing. I think that needs more attention. Also, strictly hierarchical type systems, as is strongly the case in abstract data types, need to be relaxed.

Codd: In the data base world there is a certain amount of interest in handling incomplete information, and one aspect of it is trying to deal with the null value, which means "missing information". Do you think that that should be on your list. Ways that PLs can deal with incomplete information. It is a little disturbing to try to graft on a data sublanguage that does handle the null value to a host language that does not. Of course, this remark goes right on down to the hardware.

Shaw: It goes up to design, too.