We present a novel approach for integrating AI systems with DBMS. The 'impedance mismatch' that has made this integration a problem is, in essence, a difference in the two systems' models of data processing. Our approach is to avoid the mismatch by forcing both AI systems and DBMS into the common model of stream processing.

By a stream here we mean an ordered sequence of data items. Stream processing is a well-known AI programming paradigm in which functional operators (which we call 'transducers') are combined to obtain arbitrary mappings from streams to streams. The stream processing paradigm can be, and has been, applied equally well as an AI programming model and as a query processing model in databases.

We argue first that, in practice, the relational model of data is actually the stream model. The pure relational model cannot capture important aspects of relational databases such as column ordering, duplicate tuples, tuple ordering, and access paths, while the stream model does so naturally.

We then describe the approach taken in the Tangram project at UCLA, which integrates Prolog with relational DBMS. Prolog is extended to a functional language called Log(F) that facilitates development of stream processing programs. The integration of this system with DBMS is simultaneously elegant, easy to use, and relatively efficient.

1. Why Connecting AI Systems with DBMS is Hard

It is currently argued that there is an impedance mismatch between AI systems and DBMS. In AI systems, a 'inference engine' naturally works on single tuples of data at a time, with a particular control strategy (such as depth-first-search with backtracking in Prolog). In DBMS, by contrast, an entire query is processed at once, by a query evaluator that selects among a variety of sophisticated algorithms. Since both systems go about work differently, it is difficult to connect the two systems in an efficient or easy-to-use way.

The key to the mismatch is that AI systems and DBMS follow different models of data processing. The formal model of the AI system (e.g., logic) evidently differs from that of the DBMS (e.g., relations), and their models of computation differ as well. In the past, solutions to the coupling problem have adopted one model of data processing or the other. Tuple-at-a-time solutions (particularly Prolog-DBMS and Expert System-DBMS interfaces) follow the AI system model. Query-at-a-time solutions that store the results of the query in the AI system workspace follow the DBMS model. It is well known that tuple-at-a-time solutions are inefficient, and query-at-a-time solutions can overwhelm the AI system with data.

We can reduce or eliminate the mismatch if we can find a common model in which both systems can be cast. There is such a model: stream processing. After summarizing what we mean by it, we describe how relational DBMS can be cast in the mold of stream processing. We then show how an integration of the two based on stream processing can be developed.

2. The Stream Processing Model

Stream processing is a popular paradigm that has traditionally been used in languages such as APL and Lisp [1]. A stream is just an ordered sequence of data items. These items can be tuples, or more generally any structure, such as a Prolog term or Lisp list. Being ordered sequences, streams can be initially thought of as lists. For example, we can think of the sequence


as a stream.

In the stream processing paradigm, compositions of functional operators are used to both define and manipulate streams. (When single-input, single-output operators are combined in sequence, we get pipelining.) One of the primary differences between stream processing and list processing is that stream processing includes the concept of lazy evaluation, a 'demand driven' mode of computation in which elements of a stream are produced by a transducer only on demand. Although programs are written as though the stream is completely available, actually the elements of the stream are produced only incrementally. Thus we integrate a model of the stream as a single object with a model of it as a sequence of objects.

3. In Practice, the Relational Model is really the Stream Model

The relational model is an elegant formal model of data. Not only is it conceptually pleasing, but its mathematical foundation and connections with both predicate logic and functional computation have facilitated many advances in the database field. Nevertheless, the stream model is a more accurate model of
databases as they are implemented today, and is increasingly becoming more accurate as a semantic model of data in some applications such as temporal data processing. Although the relational model is normally presented as 'set oriented', in practice the model is 'multiset oriented' at the least. That is, tuples are allowed to appear more than once.

Moreover, relational systems are heavily concerned with the ordering of tuples in a relation. First, this ordering is important for user interpretation of the tables. Second, the ordering of tuples has direct impact on query processing performance, including the size of indices and the kinds of algorithms that can be used for joins.

Furthermore, Codd's relational algebra can be viewed mostly as a collection of operators on streams. Query 'parse trees' are then just networks of stream operators. In some applications, such as temporal databases, streams are much more appropriate as a data model. In practice, even if not in theory, the relational model is treated as the stream model.

4. Loose Coupling of Prolog and DBMS through Stream Processing

The Tangram project at UCLA has integrated Prolog and relational DBMS through stream processing [2]. Integration with other AI systems could be used as well, provided that the AI system supports stream processing, and that the data being provided by the DBMS can reasonably fit in a stream format.

In particular, stream processing can be used for 'loose coupling' of Prolog and DBMS. By a loose coupling we mean a combination in which each system keeps its own identity, and both communicate through a well-defined interface. We see loose coupling not only as necessary because of economic, political, and other forces, but also as a desirable division of labor in many situations. One system does bulk data processing well, while the other performs arbitrary analyses on the results.

The Tangram approach for loosely coupling a DBMS with Prolog is basically to have the DBMS yield a result stream in response to a query, and have Prolog applications analyze the stream using the stream processing paradigm:

<table>
<thead>
<tr>
<th>Prolog System</th>
<th>batch query</th>
<th>Database System</th>
</tr>
</thead>
<tbody>
<tr>
<td>result stream</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Basic execution under the approach is as follows:

1. A DML request can be sent from the Prolog system to the DBMS. Naturally, many extensions suggest themselves here, such as piggybacking of requests, translation from nice syntax to DML, semantic query optimization, and so forth. These requests can (and in our impression, should) be made with full knowledge of what is available in the DBMS, such as indices or other access paths.

2. The DBMS produces the result of the DML request.

3. The Prolog system consumes the result tuples incrementally (i.e., lazily), as it needs them. Important extensions here include fetching more than 1 tuple at a time (called 'eagerness' in stream processing), performing unification or other pattern matching at the interface level, selectively retrieving only the needed fields from tuples, etc.

The diagram above shows the Prolog system and DBMS coupled directly, but in fact an interface between the two can improve performance. Such an interface can implement buffering and flow control, and sorting of results when this is not available in the DML, since stream processing often requires sorting of streams.

This general approach avoids many problems in traditional couplings between Prolog and DBMS. Couplings that offer only access method tuple-at-a-time retrieval from Prolog sacrifice the bulk query processing power of the DBMS, and make very heavy use of the Prolog/DBMS interface. High-level-query-at-a-time couplings that assert the entire query result in the Prolog workspace are slow since assert is very slow, and potentially dangerous since they can actually overflow Prolog data areas. A stream processing approach permits us to take advantage of the best performance aspects of both systems, tune the granularity of data blocks transferred from the DBMS to Prolog, and give the Prolog system access to the data without requiring it to be stored in the Prolog workspace.

References


This work done under the Tangram project, supported by DARPA contract F30601-87-C-0072.