Information-relational semantics of the FIFTH system

Anthony Di Franco

DFKO Consulting

Abstract. This describes work in progress on the semantics of FIFTH, a declarative programming language which obtains efficiency without relying on domain-specific assumptions by means of an adaptive evaluation strategy. The adaptive evaluation strategy demands a consistent way of measuring the progress of a computation in terms of the information yielded by different control paths explored in the course of attempting to evaluate it.
The FIFTH system's semantics are situated at the intersection of ICON, PROLOG, Radul's and Sussman's information propagation networks and Codd's relational model as developed by Date *et. al.* which in turn places it somewhere between abstract execution systems for program analysis and existing logic programming tools, and promises to enrich declarative programming with the semantics of relations on answer sets, and may through the consistent use of relations achieve the combination of expressivity and efficiency characteristic of the APL family and its consistent reliance on arrays.

1 Introduction

This is a high-level overview of the key semantic features of the FIFTH system, a system for purely declarative programming with uncertain information using an adaptive evaluation strategy[4] to simultaneously fulfill two previously incompatible criteria in declarative programming: general purpose scope and efficiency. The specification of the system is a work in progress and this overview is written in anticipation of a more systematic and complete specification in the future, which will certainly involve additions, and may also involve modifications, to the traits presented here.

Before discussing what the system is, it may be useful to consider what it is not, and point out the motivations for the main differences from similar prior systems. Here and throughout the rest of this article, the presentation will assume familiarity with this prior art for the sake of focusing the presentation and complying with space limitations. The reader is referred to the references on the prior art in case an unfamiliar concept arises. Some familiarity with concepts of measure-theoretic probability will also be useful in the discussion of measures and priors and in passing.

In PROLOG,[1] the objects being manipulated are relations, however many compromises to the semantics of how they are manipulated were made to make

implementing control from within the language practical, such as cuts and negation as failure.

Meanwhile, Codd's relational model as developed by Date[2,3] provides a clean and consistent semantics of relations, but reproduces several drawbacks of common practices in contemporary computing in general:

- Relational variables have mutable semantics, despite the ability of the semantics of relations themselves to support immutable semantics and the possibility to implement immutable semantics efficiently through structure sharing;
- Types are distinct from relations, which requires superfluous structure to be elaborated;
- Boolean comparisons on values and membership predicates on relations are also developed outside of the framework of relations, which vastly complicates constraint handling; and,
- The closed-world assumption is defined into system at the global level, complicating the implementation of open-world semantics when they are desired.

In fact, both PROLOG and the relational model assume closed-world semantics.

In order to overcome these drawbacks, we unite logic programming with relational algebra by replacing logic variables with immutable relational variables. The use of relational variables proves convenient in defining the information measures used to guide the adaptive evaluation strategy, and brings side benefits in expressivity and semantic uniformity.

2 Information-relational semantics

2.1 Aggregates

Because relations permit unbounded aggregation with rich semantics, list types are not needed as they are in PROLOG; nor are set or mapping types as are found in many other languages.

2.2 Types

We begin the definition of types by recognizing a set of symbols which may be represented and manipulated by the implementation platform. At least a finite subset of the integers will be required. Typically, characters will also be available, which correspond to a finite subset of the integers, but are distinct in how they are represented.

Types are themselves represented as immutable relations with a single "value" attribute which contains all values of the type.

A sequence of values is represented by a relation with attributes "index" and "value" where the index is of integer type. In the case that values are characters, the sequence is a string, and may be rendered as such by the system's interface.

The types of compositions of relational operators are given extensionally by their headings, and intensionally by the tree of operator compositions they consist of.

Data are represented by joining with their type tables. This may serve as a basis for interned composite values and structure sharing by hash-consing or a similar technique better suited to the structure of relations as opposed to cons cells.

2.3 Interval types

Interval representations of sets are used to represent compact sets. They are maintained in a form that minimizes the number of distinct intervals by merging abutting or overlapping intervals, so that operations involving them can be carried out most efficiently. They follow the development by Date, except that an empty interval is allowed for the sake of semantic completeness and compatibility with interval arithmetic systems such as unums.[5]

2.4 Priors and measures

A relation is always present in the environment which associates a relation with its *measure,* that is, a bounded quantity representing the proportion of space it spans in the universe defined by its type. It is used for defining the information measures used by the adaptive evaluation strategy, and in implementing probabilistic semantics.

Default measures are defined on the types and their compositions, which are used when no measure is explicitly specified, and which are called priors. The prior measure is uniform on symbols from countably finite sets, and decays exponentially with the length on sequences of measurable items, that is, the measure of each element of the sequence is scaled by 2^{-i}, where i is its index. Measures on intervals are are equal to the distance between the endpoints of intervals, defined by the differences in their values.

Measures which differ from the prior measures may be associated with data to implement fuzzy or probabilistic membership in a relation. Probabilities however require the following additional step.

Probabilities may be obtained from the measures by asserting a normalization constraint on the measures in a certain relation, that is, by relating the sum of measures in that relation to the unit. This is the way in which the system supports probabilistic computation.

2.5 Operations

New relations may be defined by analogs of the delimited control operators. The environment contains a relation identifying all the goals on the current path.

Relations support the operations of Codd's theory as developed by Date but with the following additions or modifications:

- **measure**: there exists within the system a relation that associates with any relation an information measure which indicates the extent to which the relation distinguishes itself from the universe specified by its type. The increase in the measure of the known information about a goal is used to guide the adaptive evaluation queries.
- **split**: a relation partitions itself into two relations whose measures differ by no more than the logarithm of the measure of the original relation. This facilitates distributing a computation on a relation and expressing recursive decompositions of relations for other purposes.
- **sample**: a relation produces a subset of its membership which has measure approximately equal to a given amount, and comes from a partition of the entire relation into subsets of approximately equal measure where the probability that each is returned is approximately equal. Alternatively, the partitioning occurs, and then the elements of the partition are associated with elements of a random permutation and this relation is returned. It can be implemented with recursive splitting.

The use of splitting and sampling in specifying behavior that does not require them may be an instance of encoding control in the otherwise pure logic of the specified behavior, and indeed it is expected to be used by the implementation to implement control, but it may also legitimately be used to specify semantics, such as, for example, when nondeterministic behavior is called for by a mixed strategy in a game.

2.6 Comparison operations

Generation and testing of predicates on values is treated symmetrically as in PROLOG and ICON.[8] That is, comparison of values is implemented by yielding the two relations containing the tuples satisfying and not satisfying the comparison.

Comparisons of relations are implemented by means of the equivalent relational operators which yield the portions of the compared relations satisfying the comparison.

2.7 Predicates and unification

Predicate definitions are built up with relational operations on relvars. Predicate definitions are maintained in a relation in the environment which associates predicate names with argument lists and trees of predicate calls which make up their bodies. Calls in their bodies may refer to relvars named in the argument lists, and the types of all their occurrences must match.

Unification semantics are based on relation join. Predicate calls proceed by unifying the arguments with a given relation, so unification will be illustrated here by way of the predicate call mechanism. For example, in a definition of the form

```
foo(A : AT, B : BT, C : CT) :- body
```

denoting predicate `foo`, A, B, and C denote relvars with types AT, BT, and CT. In calling `foo`, any set of relvars which may be joined to produce a subtype[1] of the type of the join of A, B, and C may be specified. If the subtype is a proper subtype (that is, if it omits some attributes in the joined argument list of `foo`, the result is another predicate with argument of type of the absent attributes. It and its argument may be associated with names and entered in the environment relation containing predicate definitions, or used in further joins. However, if the join matches all attributes, the relation becomes fully grounded and its tuple space available to direct manipulation by relational operators.

The propagation of partial information about tuples in a relation through a unification is not yet specified.

2.8 Abstraction

Abstraction is implemented via analogs of the delimited control operators.[6] A relation is maintained in the environment of each goal identifying all goals called between the goal in question and the query. The subtree below a named goal may be captured, and the unmatched arguments of any predicates in the subtree become the arguments of a predicate which instantiates a copy of the captured subtree and unifies its arguments with all the unmatched arguments in its definition.

2.9 Open-world reasoning and constraints

In Prolog, only the affirmative membership in a predicate is explicitly represented. This is convenient in part because it synergizes with Prolog's closed-world semantics, usually called "negation as failure," whereby failure to be able to prove a goal is identical to the negation of that goal.

In part because Fifth must support open-world reasoning in addition to closed-world reasoning, and in part because it must be possible to measure progress in queries involving negation, membership in a relation and membership in its complement are both represented explicitly, while it is the lack of knowledge about membership that is represented implicitly (as the default when neither of the former cases are explicitly indicated). All relational operators in the system produce information on both the membership and complement of their results. For example, the comparison $<$ yields elements that satisfy the comparison into its membership, and those which do not into the membership of its complement.

Contradictions are noted but do not constitute an error or otherwise influence the inference procedure in any special way; for example, a query for $P\&!P$ can be evaluated, thus it is up to the user how to treat contradictions and a strategy for doing so is not an inherent feature of the system. Likewise, closed-world behavior

[1] In this context, a subtype of a relation type is another relation type with a set of attributes which is a subset of the given relation type.

can be reinstated case-by-case or globally by asserting the law of the excluded middle for some relation or all relations.

This simplifies constraint handling because contradictions are identified and associated with the offending data in the same way that other queries are answered.

2.10 Incremental integration of information

Information may come incrementally from fact relations or from the environment. The implication for the evaluation strategy is that information must be able to travel both up the tree from callee to caller as in PROLOG, and from sibling to sibling and down the tree as well, in a manner reminiscent of Radul's and Sussman's information propagation networks.[10] Again resembling their strategy, propagation of information is governed by a monotonic process of information accumulation where goals propagate the implications of information received via one child to the other and to the call site, or from the call site to children, only if the received information caused its own state of knowledge about its membership to change.

2.11 Keys

Since constraint enforcement is dealt with in the formulation of a query rather than in the maintenance of the state of the data in the system, the only use for the concept of a key is in the implementation of the indexing of stored data.

This may be revised to permit or require the use of explicit keys to allow to ensure intensional referential integrity in joins, instead of basing joins on extensional type compatibility. This may be necessary to disambiguate equivalently-typed parts of arguments to predicates defined by the abstraction mechanism.

3 Implementation strategies

3.1 Manifold approximation for relations

The space spanned by attributes of relations with distance functions may be approximated by using manifold approximation techniques. This can effect a reduction in the dimensionality of the relation to that inherent in its membership and thus avoid the combinatorial explosion in dimensionality which occurs when relation membership is represented naively in terms of the universes of the types of the attributes.[7]

3.2 Holographic reduced representations of relation membership under operator composition

Holographic reduced representations[9] are a technique which can associate fixed-length codes with values associated with attributes under the composition of relational operators. They may be used to construct approximate key-like identifiers

for tuples in relations which reflect their compositional structure and which can be used in conjunction with an associative memory to efficiently identify and retrieve information, potentially improving the efficiency of structure sharing in implementing the immutable relation semantics here.

4 Future work

It remains to define a semantics of forgetting information, which ideally would be accomplished without unintended loss of information. A possibility is to use the detection of redundancy from the information propagation semantics to discard information which cannot significantly change the answer set of a query.

This will improve handling unbounded streams of data from the environment, such as is encountered in robotics applications.

Specifying the interaction of the system with its external environment remains for future work as well, which will involve the use of planning operators which may depend on external information to model the effects of actions on the environment.

5 Acknowledgements

The author would like to thank the Statebox project[2] for generously funding this research.

References

1. Ivan Bratko. *PROLOG Programming for Artificial Intelligence*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1990.
2. Hugh Darwen and C. J. Date. The third manifesto. *SIGMOD Record*, 24(1):39–49, 1995.
3. C. J. Date, Hugh Darwen, and Nikos A. Lorentzos. *Temporal data and the relational model*. Elsevier, 2002.
4. Anthony Di Franco. Information-gain computation. *CoRR*, abs/1707.01550, 2017.
5. Laslo Hunhold. The unum number format: Mathematical foundations, implementation and comparison to IEEE 754 floating-point numbers. *CoRR*, abs/1701.00722, 2017.
6. Oleg Kiselyov. Delimited control in ocaml, abstractly and concretely, 2012.
7. Yunqian Ma and Yun Fu. *Manifold Learning Theory and Applications*. CRC Press, 2012.
8. Todd Proebsting Microsoft, Todd A. Proebsting, and Gregg M. Townsend. A new implementation of the icon language. *Software: Practice and Experience*, 30:925–72, 1999.
9. Tony Plate. Holographic reduced representations. *IEEE Transactions on Neural Networks*, 6:623–641, 1991.
10. Alexey Andreyevich Radul and Gerald Jay Sussman. A flexible and expressive substrate for computation, 2009.

[2] `https://statebox.org`