Existentially Quantified Values for Queries and Updates of Facts in Transaction Logic Programs

Paul Fodor
Department of Computer Science
Stony Brook University
Stony Brook, NY 11794-4400, U.S.A
pfodor@cs.sunysb.edu

Abstract

In several applications of logic programming and Transaction Logic, such as, planning, trust management and independent Semantic Web Services, an action might produce incomplete facts and leave existential values in an incrementally generated data structure. The same action or other producer or consumer actions might read, modify or communicate through these facts, making this technique a powerful communication technique. In this poster, we present a definite semantics for these existentially quantified values that occur only in facts, queries and updates of facts. Although this simple semantics applies only to facts and not to clauses, it is relevant to many applications, including artificial intelligence planning, workflow modeling and verification, and updates of facts in the Semantic Web.

Sequential Transaction Logic and Existential Values in Queries and Updates of Facts

This poster reports on the study of existential quantified values in updates of facts for the Transaction Logic programs with a definite semantics. We applied this solution for updates with incomplete data structures in facts for planning with actions with partial effects.

Sequential Horn Transaction Logic (Bonner and Kifer, 1994, Bonner and Kifer, 1995) is an extension of classical logic programming with state changes, adding to the syntax two fundamental ideas: serial conjunction and elementary transitions. The serial conjunction operator ⊗ specifies the order in which predicates have to be executed. The elementary transitions (i.e., *ins.* and *del.*) specify basic updates of the current state of the database, executed by a strict oracle, i.e., *ins.t* fails if *t* is already in the current database state, and *del.t* fails if *t* fails in the current database state. For instance, two planning actions where conditions and effects are represented as queries and updates of facts are exemplified below. The *pickupfrom*/2 predicate specifies that a block *X* can be picked up from a block *Y* if we have an empty hand, *X is* clear and *X* is on the block *Y*. The results of this action are: the hand is not empty, it holds the block *X*, and *X* is not on *Y*.

$$pickupfrom(X,Y) \leftarrow block(X) \otimes block(Y) \otimes handempty \otimes clear(X) \otimes on(X,Y) \otimes del.handempty \otimes del.on(X,Y) \otimes ins.holding(X).$$

$$puton(X,Y) \leftarrow block(X) \otimes block(Y) \otimes X \backslash= Y \otimes holding(X) \otimes clear(Y) \otimes del.holding(X) \otimes del.clear(Y) \otimes ins.handempty \otimes ins.on(X,Y).$$

The Sequential Horn Transaction Logic's semantics is based on a few fundamental ideas: the transaction execution paths, database states, and executional entailment. Basically, a query (e.g., "*pickupfrom*(*a*,*b*)") can execute on a sequence of states (i.e., a path), if there is a resolution for this query that takes the system from an initial state to a final state of the database. For instance, for an initial database {*handempty*, *on*(*a*,*b*), *on*(*b*,*c*), *on*(*c*,*table*), *clear*(*a*)} and the query *pickupfrom*(*a*,*b*), the system takes us to a final returning state: {*holding*(*a*), *on*(*b*,*c*), *on*(*c*,*table*)}.

The problem of existential quantifier in updates has not been studied yet, due to the novelty of the field of database updatable logics and the difficulty to extend the solutions used in classical rule systems for transaction logics. In classical rule systems, existential quantified variables in the rule heads are usually replaced with Skolem functions (Chang, C. and Lee, R. 1997), function symbols invented by the programmer. For instance, the rule that "every person has a parent" can be written as:

$$parent(X, parent(X)) \leftarrow person(X).$$

In the case of updates, this simple solution is insufficient to model the common intuition. For instance, inserting both the facts: *ins.parent*("*Joseph*",*parent*("*Joseph*")), followed by *ins.parent*("*Joseph*","*Paul*") would create two different facts in the database, where the equality of *parent*("*Joseph*") and *Paul* is unknown. Subsequent insertion of a new fact *parent*("*Joseph*","*Anne*") or deletion of the fact *parent*("*Joseph*","*Paul*") would generate database states that might contradict the common intuition. The Skolem function solution introduces undecidability. For instance, by adding a second rule to specify that each parent is also a person (see (2)), a query *parent*(*X*,*Y*) executed in an initial empty database would enter in an infinite loop, generating an infinite hierarchy of parents.

$parent(X, parent(X)) \leftarrow person(X).$ (1)
$person(Y) \leftarrow parent(\_, Y).$ (2)

We define a definite semantics for existential values in queries and updates of facts by generating unique identifiers to represent *eX*-istential values in facts (see below). If we want to insert a fact $P(X,Y)$ in the database, then we check if all arguments are not variables. For un-instantiated arguments, we generate special object identifiers with a special semantics.

$insert(P(X,Y)) \leftarrow ground(X) \otimes ground(Y) \otimes ins.P(X,Y).$

$ground(X) \leftarrow nonvar(X).$
$ground(X) \leftarrow var(X) \otimes newoid(X).$

The common human intuition follows a definite semantics, i.e., if a fact with existentially quantified values is substituted by a definite fact, then the first fact is not considered anymore. For instance, if only the following two facts are in the database: *parent*("*Joseph*",*parent*("*Joseph*")) (with the Skolem function *parent*("*Joseph*")) and *parent*("*Joseph*","*Paul*"), then the answer to an aggregate query: *count*{*X* | *parent*("*Joseph*",*X*)} should return 1.

These *eX*-istential values extend the usual domain where we interpret constants (with a unique-name-assumption-as-failure property: $x \neq y$ unless we can prove $x = y$) with a domain for interpreting the new *eX* symbols under the assumption that an existential might be equal to some constant, but we do not know that unless it follows from some other information. For the *eX* symbols we leave the possibility that some *eX* might be equal with some constants even if we cannot prove that.

The Transaction Logic's semantics is a modal-like semantics, where each state represents a database, and each elementary update causes a transition from one state to another. At this point, we extend this model theory, with the fact that: in each state, the facts with *eX* values that are subsumable by real (ground) facts can be eliminated. For instance, the fact *parent*("*Joseph*",*eX*) can be eliminated if there is also *parent*("*Joseph*","*Paul*") in the current database. The truth of transactional queries is still determined on paths structures, i.e, sequences of states. This semantics would return an answer if and only if the answer is definite in all possible models for the program and the execution path. In the rest of the poster, we will exemplify this semantics with a set of examples.

Example 1. If the current state of the database is: {*parent*("*Joseph*",*eX*), *parent*("*Joseph*","*Paul*")}, then *count*{*X* | *parent*("*Joseph*",*X*)} = 1 because the fact *parent*("*Joseph*",*eX*) is subsumed by the fact *parent*("*Joseph*","*Paul*").

Example 2. If the current state of the database is: {*parent*("*Joseph*",*eX*) }, then
count{*X* | *parent*("*Joseph*",*X*)} = 1 because we know that there is at least one fact *parent*("*Joseph*",*eX*) in the database.

Example 3. The sequence of operations:
ins.parent("*Joseph*",*eX*) ⊗ *ins.parent*("*Joseph*","*Paul*")} ⊗ *del.parent*("*Joseph*","*Paul*") ⊗ *parent*(?*X*,?*Y*)
would fail because it would take the database through the path of states:
{*parent*("*Joseph*",*eX*)} → {*parent*("*Joseph*","*Paul*")} → { }
and the query *parent*(?*X*,?*Y*) would fail in the empty state.

Moreover, un-instantiated deletes (such as *del.parent*(?*X*,?*Y*)) will be translated into definite transactions by re-writing with queries (for instance: *parent*(?*X*,?*Y*) ⊗ *del.parent*(?*X*,?*Y*)).

Our definite semantics of Skolem constants for existentially quantified values that occur in updates of facts, do not inherit the undecidability result of Skolem functions. An execution with the tabled resolution on states (http://www.cs.sunysb.edu/~pfodor/webpageTabledTR/) would terminate because any fact *P*(*eX*,*Rest*) subsumes any other fact *P*(*eZ*,*Rest*), so we can have only a finite number of states.

Conclusions and Future Work

In this poster we presented a definite semantics for existentially quantified values that occur in queries and updates of facts. This semantics is beneficial for applications of transaction logic, including: artificial intelligence planning, workflow modeling and verification, and updates of RDF facts.

References

Bonner, A.J. and Kifer, M. 1994, *An Overview of Transaction Logic*, in Theoretical Computer Science, 133(2), 205-265.

Bonner, A. J., and Kifer, M. 1995, *Transaction Logic Programming (or, a logic of Procedural and Declarative Knowledge)*, Technical report, University of Toronto.

Chang, C. and Lee, R. 1997, *Symbolic Logic and Mechanical Theorem Proving*. New York: Academic Press.

Davulcu, H., Kifer, M., Ramakrishnan, C. R., and Ramakrishnan, I. V. 1998, *Logic Based Modeling and Analysis of Workflows*, Principles of Database Systems.