Gödel, Russell, Codd:

A Recursive Golden Crowd

by

C. J. Date

with apologies to Douglas Hofstadter
and his book Gödel, Escher Bach: An Eternal Golden Braid

July 17th, 2006

ABSTRACT

This paper is a response to certain criticisms that have been made of (a) *The Third Manifesto*—see reference [4]—and (b) the language **Tutorial D,** which is used in reference [4] to illustrate *Third Manifesto* ideas. The criticisms in question appear in reference [6] and can be summarized thus: *The Third Manifesto* in general, and **Tutorial D** in particular, both permit the formulation of expressions that are paradoxical, and hence undecidable.

THE PARADOX OF EPIMENIDES

Consider the following example:

- Let *P* be the predicate "There are no true instantiations of predicate *P*." Assume for now that *P* is indeed a valid predicate (I'll examine this assumption in the next section); then in fact it's not only a predicate but a proposition, because it has no parameters.

- Let *r* be the relation corresponding to *P* (i.e., the relation whose body contains all and only those tuples that represent true instantiations of *P*). Since *P* has no parameters, *r* has no attributes (i.e., it's of degree zero), and so it must be either TABLE_DEE or TABLE_DUM—where, just to remind you, TABLE_DEE is the unique relation with no

Copyright © C. J. Date 2006

attributes and just one tuple (the 0-tuple), and TABLE_DUM is the unique relation with no attributes and no tuples at all [5].

- Suppose *r* is TABLE_DEE. Then the interpretation (of the sole tuple in *r*) is that *P* is true—in which case, by definition, there are no true instantiations of *P*, and *r* shouldn't contain any tuples after all (i.e., it should be TABLE_DUM).

- Conversely, suppose *r* is TABLE_DUM. Then the interpretation (of the fact that there aren't any tuples in *r*) is that *P* is false—in which case, by definition, there must be at least one (actually, exactly one) true instantiation of *P*, and *r* should therefore contain at least one (actually, exactly one) tuple after all (i.e., it should be TABLE_DEE).

Of course, you've probably realized that this example is basically just the well-known Paradox of Epimenides ("This statement is false") in relational form. As you can surely also see, the root of the problem is the self-reference: Predicate *P* refers to itself.

————♦♦♦♦♦————

In case you're not comfortable with arguments that rely on the special relations TABLE_DEE and TABLE_DUM, let me give another example that illustrates the same general point. This example is a greatly simplified version of one originally due to David McGoveran and documented by myself in a couple of my early *DBP&D* columns [2]. Let *r* be a relation with a single attribute, *N*, of type INTEGER, and let the predicate for *r* be "The cardinality of *r* is *N*."[*] If *r* is empty, then the cardinality of *r* is zero, so the tuple *t* = TUPLE {*N* 0} should appear in *r*, so *r* shouldn't be empty after all. But then if tuple *t* = TUPLE {*N* 0} does appear in *r*, then *r* isn't empty and its cardinality clearly isn't zero, and so tuple *t* shouldn't appear in *r* after all. Again, therefore, we seem to have some kind of paradox on our hands.

DISCUSSION

For the remainder of the paper I revert to the example involving TABLE_DEE and TABLE_DUM. Now, I said in connection with that example that *P* was a predicate, and in fact a proposition—but is it? By definition, a proposition is a statement that's unequivocally either true or false. (More precisely, it's a statement that makes an assertion that's unequivocally either true or false.) But *P* is clearly neither true nor false—because if it's true it's false and vice versa—so perhaps I was wrong to say it was a proposition, as I did in the previous section.

More important, though, I don't think we need argue over whether *P* is a predicate, or more specifically a proposition; what we do need to do is decide whether our formal system—whatever system we happen to be talking about—treats it as such. If it does, we have a problem. Note very carefully, however, that:

[*] If we call this predicate *Q*, then it can equivalently be stated in the form "There are exactly *N* true instantiations of predicate *Q*"—a formulation that serves to highlight both the self-reference as such and the parallel with the previous example.

Copyright © C. J. Date 2006

- The problem isn't a problem with **Tutorial D** specifically; in fact I don't see how it could be, since I didn't appeal to **Tutorial D** at all in the example.

- Nor is it a problem with *The Third Manifesto* specifically, since I didn't appeal the *Manifesto* at all in the example, either; all I did was appeal to the well-known fact that any given predicate has a corresponding relation—namely, that relation whose body contains all and only those tuples that represent true instantiations of the predicate in question.

- Nor is it a problem with the relational model specifically, for essentially the same reason.

- Nor is it a problem that arises from the fact that *The Third Manifesto* requires **Tutorial D** (indirectly) to be computationally complete. I mention this point because the *Manifesto* has been criticized on precisely these grounds: the grounds, that is, that computational completeness "creates a language with logical expressions ... that are provably not decidable" [6]. Note carefully that I'm not saying this criticism is incorrect; I'm merely pointing out that the lack of decidability in the specific example under discussion, regarding whether relation *r* contains a tuple or not, doesn't seem to have anything to do with the fact that **Tutorial D** is computationally complete.

Rather, the problem, if problem there is, seems to be with *logic*. That is, either logic admits *P* as a predicate or it doesn't. If it does, there's a problem with logic. If it doesn't, then the problem with logic goes away, and hence the problems (if any) with the relational model, *The Third Manifesto,* and **Tutorial D** all go away too, a fortiori.

Note: As far as I understand it, Russell's theory of types had to do with the idea that *P* and statements like it might need to be rejected as legal predicates. Even if so, however, it doesn't invalidate my point, which is (to repeat) that if there's a problem in this area, then it's intrinsic—it isn't the fault of either *The Third Manifesto* or **Tutorial D** as such.

REMARKS ON TUTORIAL D

Now let me concentrate on **Tutorial D** specifically for a moment. It might be thought that the following is a formulation of the Paradox of Epimenides in **Tutorial D** terms (and if it is, it might therefore be thought that there is indeed a problem with **Tutorial D** specifically):

```
VAR R { } KEY { } ;

CONSTRAINT EPIMENIDES COUNT ( R ) = 0 ;
```

More specifically, it might be thought that constraint EPIMENIDES is a formal expression of predicate *P* ("There are no true instantiations of predicate *P*," or equivalently "The number of true instantiations of predicate *P* is zero"). But it isn't. The reason it isn't is that The Closed World Assumption doesn't apply to constraints. The Closed World Assumption says (speaking rather loosely) that relvar *R* should contain all and only the tuples at a given time that satisfy the predicate for *R* at that time. But **Tutorial D** isn't aware, and can't be aware, of relvar predicates; all it can be aware of is relvar constraints. And there's no requirement—nor can there be a requirement, in general—that relvar *R* contain all and only the tuples at a given time that satisfy

the constraints that apply to *R* at that time. So the fact that (in the example) relvar *R* is always empty does not of itself lead to any paradox.

I switch now to another tack. Here's another attempt to formulate the Paradox of Epimenides in **Tutorial D** terms:

```
VAR R { } KEY { } ;

CONSTRAINT EPIMENIDES
   IF COUNT ( R ) = 1 THEN COUNT ( R ) = 0 AND
   IF COUNT ( R ) = 0 THEN COUNT ( R ) = 1 ;
```

Constraint EPIMENIDES here is logically equivalent to the following:

> If there exists a tuple in relvar *R*, then relvar *R* must be empty, and if relvar *R* is empty, then there must exist a tuple in relvar *R*.

In other words, the constraint is a contradiction. (Please note that I'm using the term *contradiction* here in its formal logical sense, to mean a predicate whose every possible invocation is guaranteed to yield FALSE regardless of what arguments are substituted for its parameters.) But if a constraint is a contradiction, then there is—or at least should be—no way to add it to the system in the first place! To be more precise, if some user attempts to define some new constraint for some database, the first thing the system must do is check that the database in question currently satisfies it. If that check fails, the constraint must obviously be rejected; and if that constraint is a contradiction, there's no way the database can currently satisfy it.

Of course, I'm assuming here for the sake of the argument that the system is indeed able to detect the fact that the database fails to satisfy some proposed constraint. What happens if this assumption is invalid is discussed, implicitly, in a companion paper [1]. For completeness, however, I give here a brief sketch of what should happen in a properly designed system when some user attempts to define some new database constraint *DC:*

1. The system evaluates *DC* against the current state of the database.

2. If the result of that evaluation is TRUE, the system accepts *DC* as a legitimate constraint and enforces it from this point forward (until such time as it's dropped again).

3. If the result of that evaluation is FALSE, the system rejects *DC* as not being legitimate at this time.

4. If the evaluation fails to terminate after some prescribed period of time, a time-out occurs and the system rejects *DC*—not because it knows it's not legitimate, but because it's too complex for the system to handle.

Copyright © C. J. Date 2006

Since it's essentially rather simple, the Paradox of Epimenides as such doesn't illustrate the point, but in general the idea of "predicates referencing predicates" corresponds in relational terms to relations having relation-valued attributes (RVAs). In particular, a directly self-referencing predicate—i.e., a predicate *P* that includes a direct reference to *P* itself—would correspond to a relation of some type *T* that has an attribute of that same type *T*. That relation type *T* is then said to be a recursively defined type. Reference [4] has the following to say on such matters:*

<quote>

> It's an open question as to whether any relation type can be defined, directly or indirectly, in terms of itself. More precisely, let RELATION{*H*} be a relation type, and let *S*(1), *S*(2), ... be a sequence of sets defined as follows:
>
> *S*(1) = { *t* : *t* is the type of some attribute in {*H*} }
>
> *S*(*i*) = { *t* : *t* is the type of some component of some possrep
> for some scalar type, or the type of some
> attribute of some relation type, in S(*i*-1) }
> (*i* > 1)
>
> If there exists some *n* (*n* > 0) such that RELATION{*H*} is a member of *S*(*n*), then that type RELATION{*H*} is recursively defined. (This definition requires a slight extension if type inheritance is supported, but this detail needn't concern us here.) Thus, the open question is whether such recursively defined types should be permitted. We don't feel obliged to legislate on this question so far as our model is concerned; for the purposes of the present book, however, we follow *The Principle of Cautious Design* [3] and assume (where it makes any difference) that such types aren't permitted.

</quote>

It seems to me that the arguments of the present paper make it desirable to strengthen the foregoing position. Specifically, I now think that recursively defined relation types should be explicitly prohibited. Points arising from this position:

- Please understand that I'm not saying that relation-valued attributes (RVAs) should be outlawed entirely. I mention this point because *The Third Manifesto* has been criticized by many people for supporting RVAs, on the basis that they take us into the realms of second (or higher) order logic. This latter claim is presumably true, but nobody has yet

* Observe that the extract quoted covers recursive types that are defined indirectly as well as directly defined ones. (I've modified the text slightly, but I haven't changed the meaning in any important respect. You can ignore the reference to "possreps" if you don't know what they are.)

Copyright © C. J. Date 2006

demonstrated a specific problem that's caused by that fact (by that fact alone, that is)—at least, nobody has demonstrated such a problem to us, the authors of reference [4].

- The extract quoted above from reference [4] says, to repeat, that for the purposes of that book we assume that recursively defined types aren't permitted. However, **Tutorial D** as described in that same book does permit such types to be defined (possibly indirectly). Here's a simple example:

```
VAR RX BASE RELATION { A1 INTEGER, A2 SAME_TYPE_AS ( RX ) };
```

A more complex example:

```
VAR RX BASE RELATION { A1 INTEGER, A2 SAME_TYPE_AS ( RY ) };

VAR RY BASE RELATION { A3 INTEGER, A4 SAME_TYPE_AS ( RX ) };
```

So perhaps **Tutorial D** as described in reference [4] does suffer from a lack of decidability. I presume, however, that the implementation could be designed to reject any attempt to make use of this "feature" by rejecting (preferably at compile time) any attempt to define, either directly or indirectly, a type *T* in terms of itself—much as the system should reject any attempt to define a constraint that can't be shown to evaluate to TRUE at the time it's defined.

REFERENCES

1. C. J. Date: "And Now for Something Completely Computational," *www.thethirdmanifesto.com* (July 2006).

2. C. J. Date: "How We Missed the Relational Boat" and "Answers to Puzzle Corner Problems (Installments 13-17)," in C. J. Date, *Relational Database Writings 1991-1994*. Reading, Mass.: Addison-Wesley (1995).

3. C. J. Date: "The Principle of Cautious Design," in C. J. Date (with Hugh Darwen): *Relational Database Writings 1989-1991*. Reading, Mass.: Addison-Wesley (1992).

4. C. J. Date and Hugh Darwen: *Databases, Types, and the Relational Model: The Third Manifesto* (3rd edition). Reading, Mass.: Addison-Wesley (2006).

5. Hugh Darwen: "The Nullologist in Relationland; *or*, Nothing Really Matters," in C. J. Date (with Hugh Darwen): *Relational Database Writings 1989-1991*. Reading, Mass.: Addison-Wesley (1992).

6. Anon.: Private correspondence with Hugh Darwen (December 2005 - January 2006).

***** End *** End *** End *****

Copyright © C. J. Date 2006