Dedalus[*] : Datalog in Space and Time

Peter Alvaro, Neil Conway, Tyson Condie, Joseph M. Hellerstein, David Meiers

ABSTRACT
foo

1. INTRODUCTION

1.1 Datalog

1.1.1 The basics

1.1.2 Negation and stratification

1.1.3 Local stratification

1.1.4 Choice

1.2 Motivation: Distributed Systems

Datalog is static. What does Datalog even *mean* extended in time?

Overlog pretended there was no time; we used the "chain of fixpoints" model and treated the database as overwritable storage.

2. DEDALUS

By reifying time as data, we are able to reason about time in our logic. some useful things fall out of this right away: persistence is now programatic rather than a separate type, ditto key constraints. event creation vs. effect ambiguities are resolved.

Perhaps more importantly, the infinite sequence of abstract time gives us a way to reason about ordering, which is particularly difficult in a set-oriented language like Datalog. The ordering over any program inputs (e.g. message queues) can be represented as a mapping between the ordering domain of the input and the time relation.

[*]Dedalus is intended as a precursor language for **Bloom**, a high-level language for programming distributed systems that will replace Overlog in the **BOOM** project. As such, it is derived from the character Stephen Dedalus in James Joyce's *Ulysses*, whose dense and precise chapters precede those of the novel's hero, Leopold Bloom. The character Dedalus, in turn, was partly derived from Daedalus, the greatest of the Greek engineers and father of Icarus. Unlike Overlog, which flew too close to the sun, Dedalus remains firmly grounded.

ACM PODS '10 Indianapolis, IN, USA

2.1 Syntax

2.1.1 Events

```
likes(peter, swimming)@123;
```

2.1.2 Persistence

```
likes(Person, Activity)@N+1 :-
  likes(Person, Activity)@N,
  notin del_likes(Person, Activity)@N;
```

or does this look better?

$likes(Person, Activity)@N + 1 \leftarrow$
$likes(Person, Activity)@N,$
$\neg\ del\_likes(Person, Activity)@N;$

2.1.3 State Change

2.2 Semantics

2.2.1 Chain of fixpoints interpretation

2.2.2 Instantaneous time interpretation

2.3 Examples

3. EQUIVALENCES

3.1 Dedalus is Datalog if we project out time

3.1.1 Theorem 1

every local Dedalus program P with only deductive rules is equivalent to a Datalog program P'.

3.1.2 Theorem 2

every local Dedalus program P with deductive and inductive rules and trace T is equivalent to a Datalog program P' with an EDB T'.

3.1.3 Theorem 3

some theorem about distributed programs :)

3.2 Dedalus programs are stratifiable if the equivalent Datalog program is stratifiable

3.2.1 Theorem 4

3.3 Space is simply time

Dedalus programs can model many classes of distributed systems. Take the (distributed) program

$ping(@A, B)@N + r(A, B) \leftarrow$
$init(A, B)@N;$

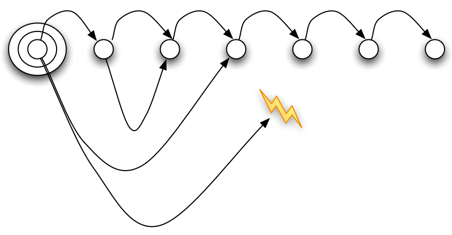

Figure 1: Time moves forward in three ways: across strata, to the next fixpoint, and to some future fixpoint.

$pong(@B, A)@N + r(A, B) \leftarrow$
$ping(@A, B)$

We may regard *r()* as a function over the attributes occurring in the body of the rule. The implementation or *r()* is provided by the model. For example:

3.3.1 Synchronous Systems

r(_) = 1 for all _. Computation proceeds in rounds.

3.3.2 Asynchronous Systems

The return value of r may be any arbitrary integer, positive or negative, including a NULL integer indicating an infinite value.

3.3.3 Lamport Clocks

As a middle ground, we might wish to enforce a constraint that $r(\_) > 0$. Doing so would entail implementing a *Lamport Clock*.

3.4 Dedalus Programs can be efficiently implemented

:)

3.4.1 Local stratification on time

3.4.2 Time is infinite but punctuated

and indeed, so are any input streams. we may dequeue as many events as we like from a given stream by creating a mapping between the elements in the queue and the time relation. in the simplest case this mapping would be from the ordering domain of the queue to the time domain, but we can establish more complicated data-dependent mappings by including other attibutes (e.g. we could implement QoS by including the address in the mapping and dequeueing a different number of items per host per time unit).

3.5 Dedalus Programs are storage-optimal

:)

3.5.1 We need only store the event tuple with max(Timestamp) for each projection of the other columns to query the present time

3.5.2 perhaps we can admit queries over the past that are bounded and pre-stated, and do GC

4. CORRECTNESS PROPERTIES

4.1 Safety

we need merely to show the base case, in which an invariant is established (in fact, this should *always* be a valid initial state, or the safety property isn't guaranteed to always hold), the existence of an inductive rule that ensures that every subsequent step maintains the invariant, and the absence of any rule that can break the induction by deducing a del_* rule.

4.2 Liveness

We cannot state a liveness property as an assertion, because doing so would involve quantifying over time, and time is an infinite sequence. Instead, we must prove liveness properties, expressed as temporal logic formulae, given the rules of the program and any given safety properties as axioms.

In many cases it will be possible to show that attempts to achieve liveness properties will occur infinitely often, and that these attempts have a nonzero probability of success. We'd like to do better than this and provide realistic guarantees...

4.2.1 Deadlock

4.2.2 Livelock

4.3 Examples

5. FUTURE WORK