

Dynamic Datalog

David Maier

Portland State University

in collaboration with
Badrish Chandramouli,
Jonathan Goldstein
& Berkeley Dataloggers

Dynamic Datalog

1

Whence “Datalog”?

Working on a book with David S. Warren
Realized some concepts could be introduced
in simpler languages

- Resolution
 - Unification
 - Binding Frames
 - “Proplog” - propositional logic
 - “Datalog” - function-free logic
- Might not have been the first into print with it

Dynamic Datalog

2

Dynamic Datalog

Evolving EDB

- ❑ User updates
- ❑ Soft state (Decl. networking): facts expire
- ❑ Stream processing: time-tagged events

Example queries

- ❑ Network reachability, routing
- ❑ Travel-time estimates (shortest path)
- ❑ Pattern matching (augmented automata)

Dynamic Datalog

3

Network Reachability

```
reach(X) :- source(X) .  
reach(X) :- link(Y, X), reach(Y) .
```

source(a)

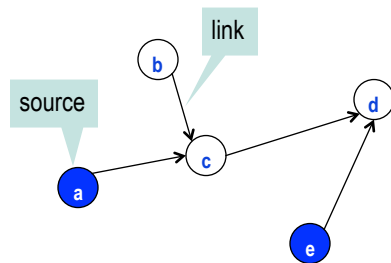
source(e)

link(a, c)

link(b, c)

link(c, d)

link(e, d)

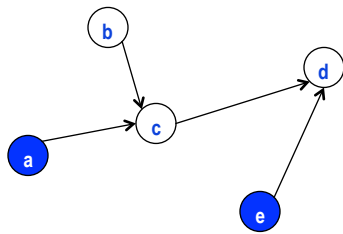


Dynamic Datalog

4

Time 1

```
reach(X) :- source(X).  
reach(X) :- link(Y, X), reach(Y).
```



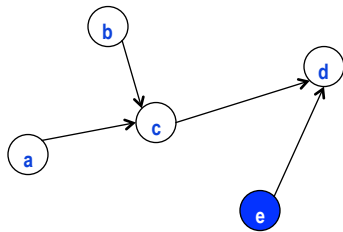
source(a)	reach(a)
source(e)	reach(e)
	reach(c)
link(a, c)	reach(d)
link(b, c)	
link(c, d)	
link(e, d)	

Dynamic Datalog

5

Time 2

```
reach(X) :- source(X).  
reach(X) :- link(Y, X), reach(Y).
```



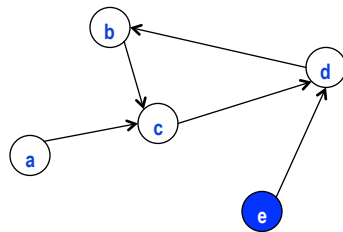
source(e)	reach(e)
	reach(d)
link(a, c)	
link(b, c)	
link(c, d)	
link(e, d)	

Dynamic Datalog

6

Time 3

```
reach(X) :- source(X).
reach(X) :- link(Y, X), reach(Y).
```



```
source(e)
link(a, c)
link(b, c)
link(c, d)
link(e, d)
link(d, b)

reach(e)
reach(d)
reach(b)
reach(c)
```

Dynamic Datalog

7

Continuous Evaluation

Can I evaluate such queries with a stream engine (push-based algebra)?

- Overlap stages of recursion
- Overlap snapshots of EDB

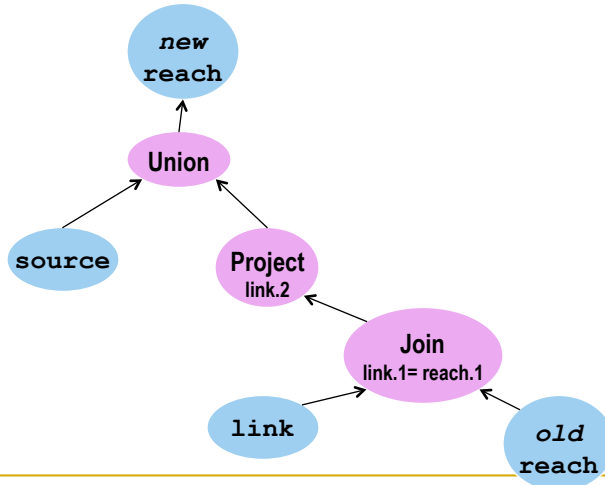
Issue: How do I know when I'm done?

More precisely, how do I know how done I am?

Dynamic Datalog

8

Can Implement Stage as Query

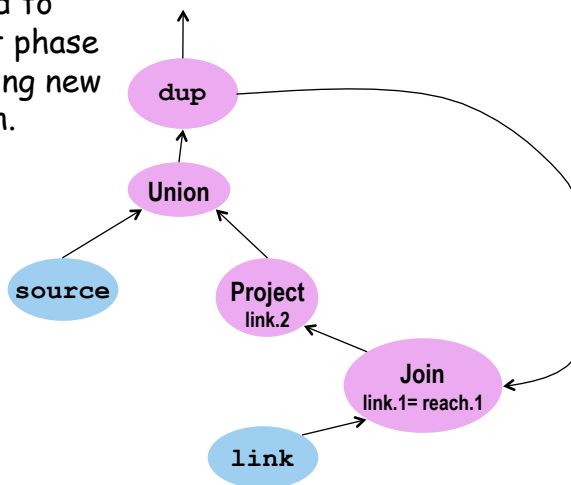


Dynamic Datalog

9

Free-Running Evaluation

Don't really need to finish current phase to start pushing new reach through.



Dynamic Datalog

10

Overlapping Snapshots

Do a timestamped version of the data.

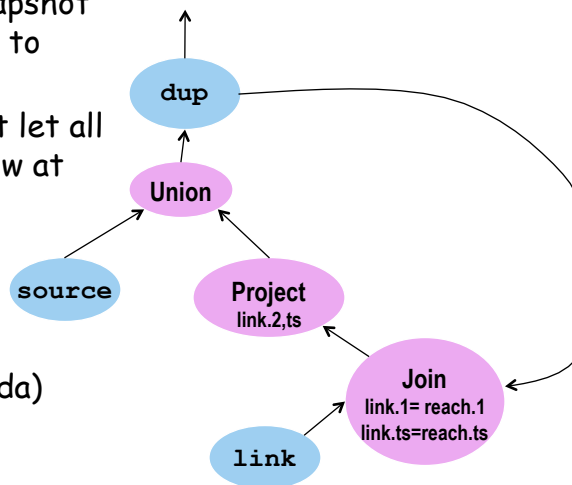
```

source(a, 1)           link(e, d, 1)
                       link(e, d, 2)
source(e, 1)          link(e, d, 3)
source(e, 2)
source(e, 3)          link(d, b, 3)
                       ...
    
```

Modify Join to Compare Times

Could do one snapshot at a time, run to completion.

But why not just let all the tuples flow at once?



This works! (kinda)

Digression 1: Re-derivation

Deriving the same facts at many time steps
Use time-interval semantics

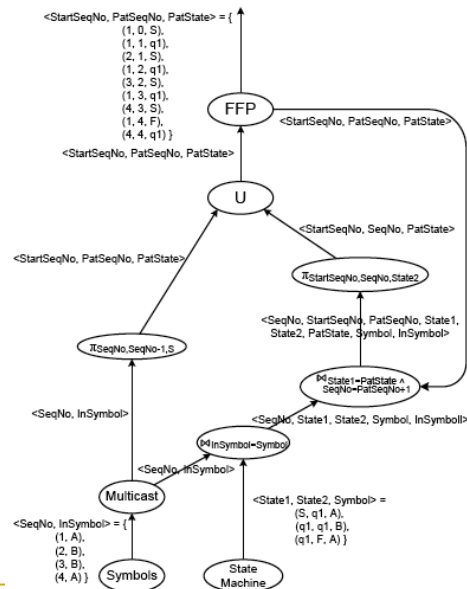
reach(e, [1, 3]), link(e, d, [1, 3]) → reach(d, [1, 3])
reach(a, [1, 1]), link(a, b, [1, 3]) → reach(b, [1, 1])

For free: CEDR (StreamInsight) had this semantics already for its operators

NFA

Pattern matching with a finite automaton over a window of events

Cool bit: State machine can be a streaming input!

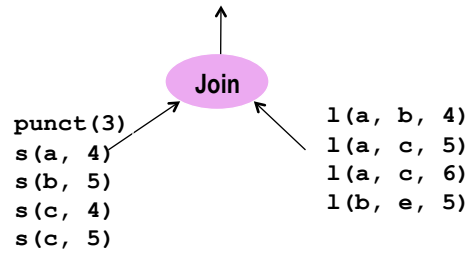


Punctuation: Signaling Progress

Don't want to require streams be ordered on time

Insert punctuations to mark lower bound on advancement of time

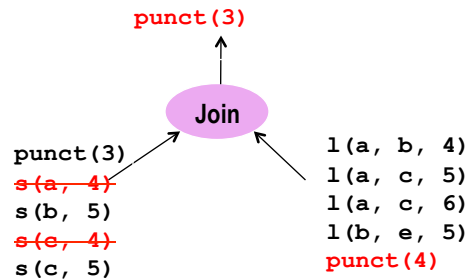
punct (t) : have received all events up to time t



Purge and Propagate

Use punctuation to purge state

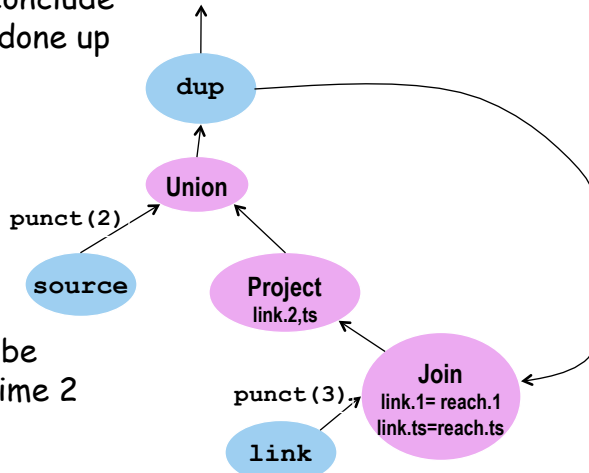
Propagate progress to downstream operators



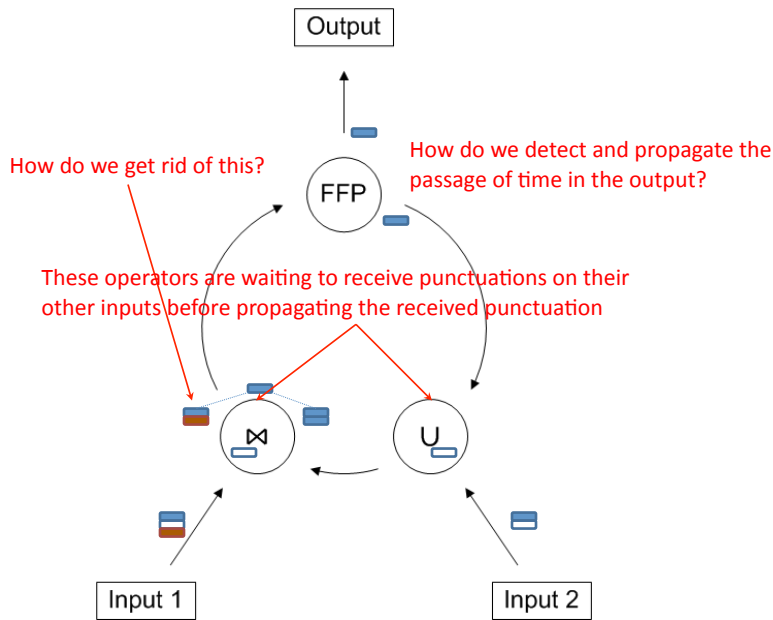
Punctuating Cyclic Queries

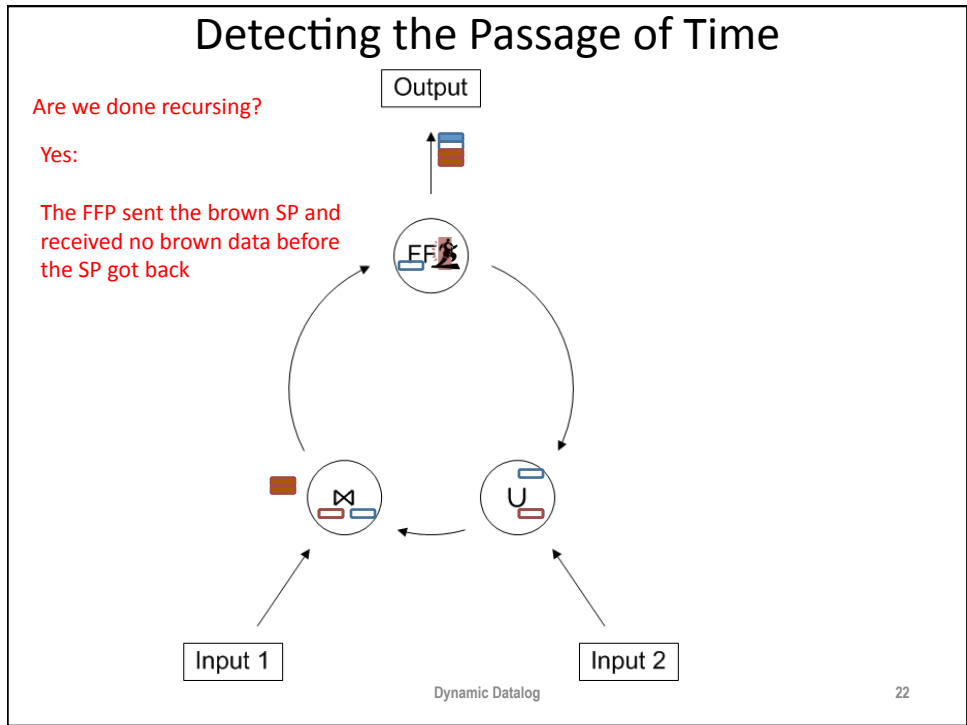
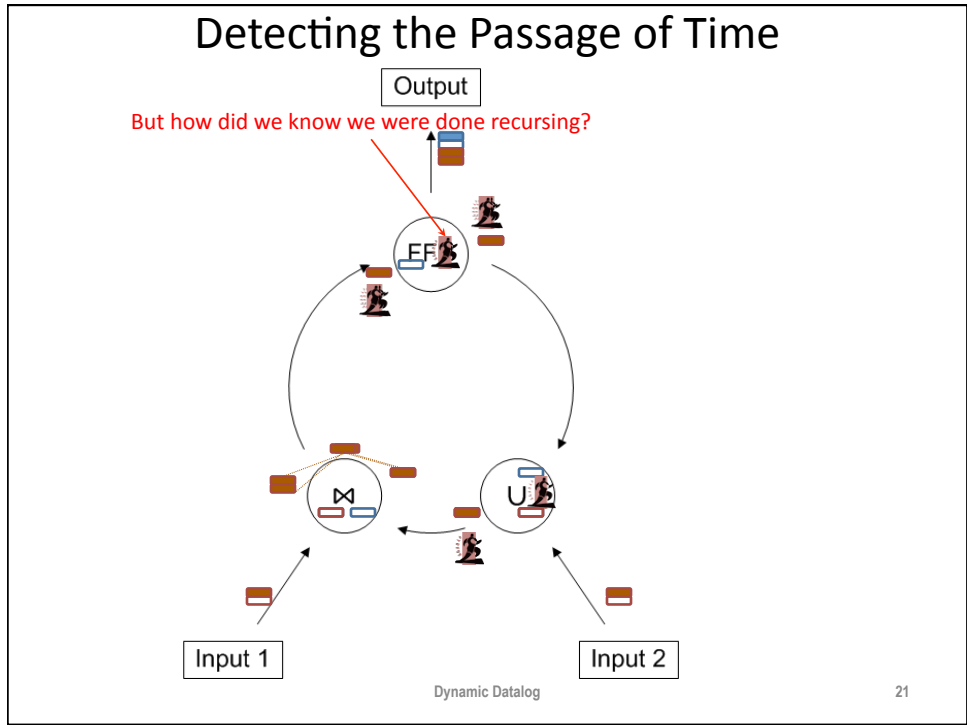
Would like to conclude that we are done up to time 2.

But might still be phases for time 2 running.



Detecting the Passage of Time



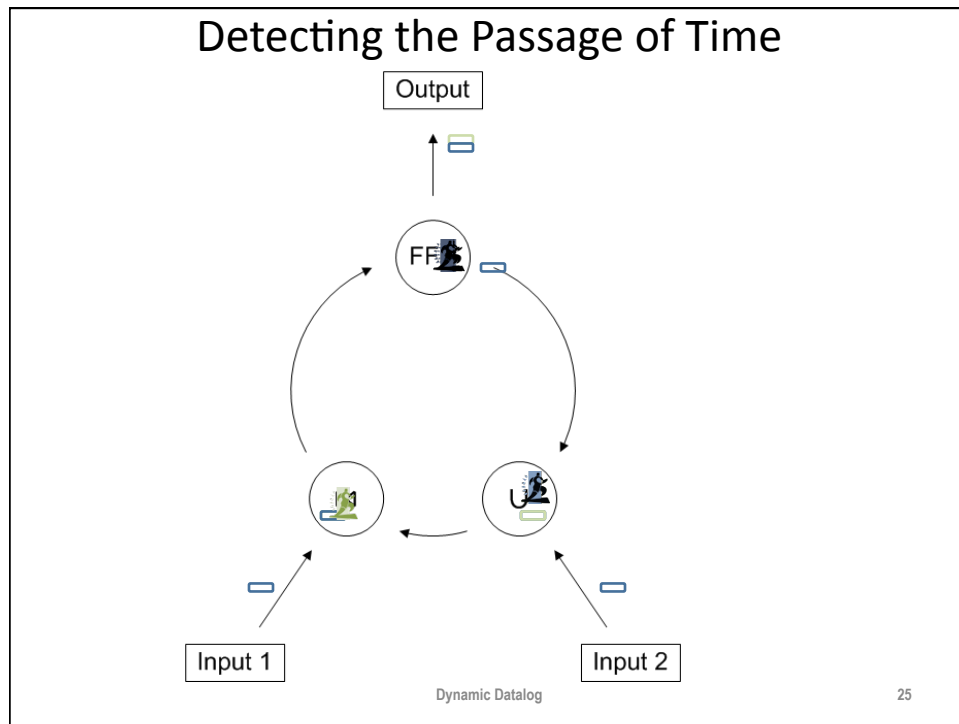


Detecting the passage of time

- How did we know to send the brown SP?
 - Proposal #1 : High watermark
 - FFP maintains a high watermark for the latest data received.
 - After the current SP is converted to a punctuation, a new SP with the latest high watermark is started.
 - What if we receive punctuations from the input but no data?
 - High watermark will never produce punctuations in the output

Proposal #2: Probing

- Always have an active SP. Start it off with the maximum punctuation time
- Allow the SP to demote its time based on received guarantees in the input
- If the SP comes back with the last propagated time, send it out with the maximum time.



Probing

- Passage of time in the input always propagates into passage of time in the output

BUT

- Expensive during periods of low activity
- Guarantees wasteful 100% CPU utilization all the time

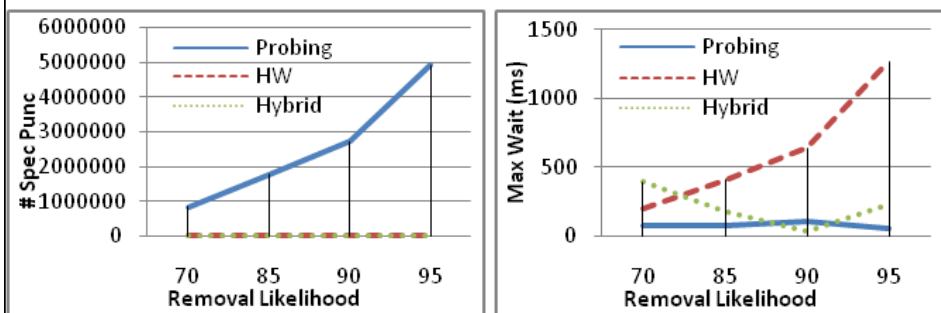
Proposal #3: Hybrid

- Like probing, allow the demotion of SPs to earlier times. Start them at max time.
- Don't have an SP at all times:
 - External progress (EP(t)) events are sent through the plan to FFP when a punctuation at time t is received
 - Start an SP only if an EP(t) is received with $t >$ latest output punctuation

Dynamic Datalog

27

Effect of Lulls in the Data



Lulls vs. # SPs

Lulls vs. responsiveness

Dynamic Datalog

28

Wrap Up

Need to manage a changing EDB

- Approach 1: materialize and incrementally maintain IDB
- Approach 2: Put life-spans on tuples; use a cyclic query plan with speculative punctuation

Approach 1 is likely more general

Approach 2 works with dense, disordered data; "fuzzy" EDB

Digression 2: Retraction

S'pose life-span of a tuple can change

`link(a, c, [1,9]) → link(a, c, [1,6])`

Causes problems with duplicate elimination

Can avoid dup-elim if queries are **strongly convergent**

"Finite proofs" vs. "finite results"

`reach(X, [X]) :- source(X).`

`reach(X, [Y|P]) :- link(Y, X),
reach(Y, P), not_in(X, P).`

Event-bounded queries are strongly convergent