1 Background

1.1 Retroactivity

The paper describes a new paradigm for the construction of data structures in which operations can be performed at any point in the data structures time line. Specifically the traditional fixed sequence of operations on a data structure is now able to be changed in a retroactive data structure, with all effects of the change propagated through to the present version. An example being erroneous data entered into a data set. It would be desirable, once the data is found to be erroneous, to correct the data in the past with all effects of the error also changed.

The naive method of retroactivity is to roll back the state of the data structure to just before desired change and then to re-execute all operations on the modified structure. The results of the paper show that indeed the naive method is sometimes the best solution. However, for some classes of data structures it is possible to improve the naive approach.

There are two types of retroactivity presented in the paper: Partial Retroactivity supports insertion and deletion of updates at past times, Full Retroactivity extends partial retroactivity to support queries at past times.

The main results of the paper give transformations that work on some classes of data structures. It also gives examples of data structures that are made retroactive and the resulting time and space complexity costs involved.

2 Open Problems

2.1 Distributed Retroactivity

Problem Statement: Extend retroactive data structures to be distributed and asynchronous.

Thoughts: Perhaps add another type of operation. The new operation would be a synchronization operation between two independent instances of the distributed data structure. Each instance necessarily has full ordering of instructions. Choosing synchronization instructions carefully could result in a full global ordering of all instructions that could affect on another. At the very least synchronizing would limit the regions of partial ordering.

Usefulness:

- Deadlock detection and prevention in an operating system, resource allocation could be decided in a retroactive data structure such that when a deadlock was detected it could be retroactively corrected eliminating the deadlock in the first place.
- Contingency Planning or Task Scheduling, retroactively being able to change assumptions about the state of the world being analyzed could allow many simultaneous planners or schedulers to work together.
- Instant messaging or chatting, is it not annoying when you say something and it has no relevance because it took to long to type. Could, use a distributed retroactivity to register events and then retroactively insert comments and text.
- Online bidding and auctions, when the network causes the delay of a result it would be ideal to retroactively correct the error.
- Distributed instruction speculation in multiprocessor machines, by using retroactivity to correct speculation errors a efficient general retroactive structure could support arbitrary time windows on arbitrary numbers of processors/nodes.
- Security, by using a retroactive process encapsulation an untrusted process can work on unknown data with the important identity or sensitive data being retroactively bound into the processing after context is secure.

2.2 Summary structure

Problem Statement: Applying the ideas from on-line data clustering improve the efficiency of a retroactive data structures by reducing the granularity of retroactive modifications.

Thoughts: Possibly it would be useful to construct a hierarchical dependence graph among the operations. This would allow the selection of sections to lump together when efficiency was suffering. Still there is a question of how best to find the proper time to perform a retroactive operation if it falls in the middle of a condensed sequence of dependent operations. Perhaps if the insertion time were not specified by a ordering or specific time, but rather the state of some data that occurred closest in the past.

Usefulness:

- Would allow even inefficient retroactive structure to have usefulness at the cost of scope of retroactivity.

2.3 Retroactive retroactivity

Problem Statement: How can retroactive operations be made retroactive efficiently?

Thoughts: Use a persistent data structure to store the retroactive structure such that previous retroactive operations can quickly be removed or added. Of course it is kinda tricky to keep things efficient as more retroactive operations are performed on many branches of the persistent core. Once again a hierarchical operation dependence graph would most likely help.

3 Resources

For the most up to date version of this document see:
netfiles.uiuc.edu/kloepper/public/open_problems_retroactivity.pdf

Papers:

1. Erik D. Demaine, John Iacono, and Stefan Langerman 2004. Retroactive data structures. In Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (New Orleans, Louisiana, January 11 - 14, 2004). Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, Philadelphia, PA, 281-290.

2. James R. Driscoll , Neil Sarnak , Daniel D. Sleator , Robert E. Tarjan, Making data structures persistent, Journal of Computer and System Sciences, v.38 n.1, p.86-124, February 1989

3. Amos Fiat , Haim Kaplan, Making data structures confluently persistent, Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, p.537-546, January 07-09, 2001, Washington, D.C., United States

4. Charu C. Aggarwal, Jiawei Han, Jianyong Wang Philip S. Yu A Framework for Clustering Evolving Data Streams,
http://www.cs.uiuc.edu/∼hanj/pdf/vldb03_clstm.pdf