Exploiting intra-object dependencies in parallel simulation

Francesco Quaglia [a,1] Roberto Baldoni [a,2]

[a] *Dipartimento di Informatica e Sistemistica*
Università "La Sapienza"
Via Salaria 113, 00198 Roma, Italy

Abstract

This paper introduces the notion of *weak causality* that models the intra-object parallelism in parallel discrete event simulation. In this setting, a run where events are executed at each object according to their timestamp is a correct run. The weak causality relation allows to define the largest sub-set of all runs of a simulation that are equivalent to the timestamp-based run. Finally, we describe an application of weak causality to optimistic synchronization (Time Warp) by introducing a synchronization protocol that reduces the number of rollbacks and their extent.

Key words: Causality, Optimistic Synchronization, Rollback-Recovery, Discrete Event Simulation, Time Warp

1 Introduction

A discrete event simulation consists of a model of the simulated system and the execution of a set of events occurring in a virtual time (*run*). The execution of an event usually moves the simulation model from one state to another and produces other events to be executed at their virtual times. The correctness criterion is *causality* which states that simulation results are correct if virtual time does not decrease while executing events in a run.

[1] e-mail quaglia@dis.uniroma1.it
[2] e-mail baldoni@dis.uniroma1.it

This article appeared in Information Processing Letters, vol.70, no.3, 1999.

Parallel discrete-event simulation is an attempt to speed-up the simulation process through the use of multiple processors. In some sense parallel discrete-event simulation can be viewed as generated by the following observation: "divide the simulation model into many sub-models as independent as possible of each other and then execute events at sub-models as concurrently as possible". More technically (2), the simulated system is partitioned into a number of sub-systems that are modeled by communicating objects (other papers refer such objects as *logical processes*). An object may schedule an event at virtual time t for another object by sending to it a message with timestamp t. The advantage over a sequential simulation comes from the fact that previous causality definition becomes the following weaker *local* constraint: the correctness of results is guaranteed if each object executes its events in non-decreasing timestamp order [3].

Two main synchronization protocols have been proposed to guarantee local causality. The first, namely pessimistic synchronization (1), allows an object to execute an event with timestamp t after there is the certainty that no event with timestamp less than t will be scheduled for that object in the future of the simulation execution. The second, known as optimistic synchronization (or Time Warp) (7), lets an object execute simulation events whenever they are available and uses a checkpoint-based rollback as a mean to recover from out of timestamp order executions.

However, local causality does not exploit all potential parallelism inside an object. For example, let us consider two events a and b at the same object, whose timestamps are $ts(a)$ and $ts(b)$ respectively, with $ts(a) < ts(b)$. If the out of timestamp order execution of a and b produces the same simulation result as if they were executed in timestamp order, then a and b are, in a sense, *independent* due to an intra-object parallelism. From an operational point of view this independence, which might allow out of timestamp order runs, has been used in Time Warp with *lazy cancellation* (3), in Time Warp with *lazy rollback* (9) and in the study of the *super-criticality* of synchronization protocols (4; 6; 8). For example, Time Warp with lazy cancellation assumes that rolled back events are correct unless their re-execution produces a different result (so, it allows some out of timestamp order runs). Another example is Time Warp with *lazy rollback*, which works as follows: when an event is re-executed due to a rollback, the resulting object state is compared to the state

[3] The notion of *causality* is well known in distributed systems and has been formalized by Lamport in (5) by means of the *happened-before relation*. That relation models events of a distributed computation as a partial order, then distributed algorithms based on a logical time (timestamp) have been given to produce a total ordering of events consistent with the happened-before relation. On the contrary, in the simulation context, the logical time, produced by the causality, defines an *a priori* total ordering on events and then parallel discrete event simulation extracts many sub-orders to be executed one for each object.

produced by the previous execution of that event; if those states are the same, rolled back events whose execution is dependent on the produced state are not re-executed.

However, the lack of a definition for intra-object parallelism is the cause of (possibly) unnecessary overhead in both previous protocols. Specifically, Time Warp with lazy cancellation, requires rolled back events to be re-executed (this is necessary to check their correctness), and Time Warp with lazy rollback requires state comparison. Both overheads could be removed if rolled back events are known to be independent of the event causing the rollback due to an intra-object parallelism.

Basing on observations of previous paragraphs, in this paper we introduce the relation of *WEak Causality* (WEC) that models the intra-object parallelism. WEC states that two events must be executed at an object according to their timestamp if their out of order execution would produce a different result. This relation is based on conflicts created by events that operate on the same data as well as on the timestamps of events. WEC allows to define a set of simulation runs that move the simulation to the same final state of a pure timestamp-based simulation run (these runs are said to be *equivalent* to a timestamp-based run). Then an optimistic synchronization protocol in which rollbacks occur only upon violations of WEC is presented. This protocol exhibits a potential reduction of the number of rollbacks and of their extent compared to previous solutions.

The remainder of the paper is organized as follows. Section 2 provides the model of an object. Section 3 introduces the WEC relation. In the same section, a theorem is proved which states that any simulation run consistent with WEC is equivalent to the timestamp-based run. A remark on the set of runs originated by WEC concludes Section 3. The application of WEC to optimistic synchronization is described in Section 4.

2 Model of an object

A parallel discrete event simulation consists of a set of objects, each object executes a sequence of events $(e_1, \ldots, e_k)$. Each object has a state Σ consisting of a set of variables $\{v_1, \ldots, v_\ell\}$ accessed by read and write operations. Each variable v_k has an initial value, namely $init(v_k)$, which is assigned by an initial fictitious write operation. The execution of an event e moves the object's state from Σ' to Σ'' (with $\Sigma' \neq \Sigma''$) and may generate new events. Each event execution is deterministic i.e., given a state Σ', the execution of e will always produce Σ'' and will always generate the same new events. When an event e is generated by object O, it is scheduled for a given virtual time (timestamp),

denoted $ts(e)$ (we assume without loss of generality that to each event in an object corresponds a distinct timestamp), then a message with the content of e and $ts(e)$ is sent by O to an object O' (not necessarily distinct from O) that will execute it[4]. Timestamps will then totally order the set of events at an object:

Definition 2.1 *Let e and e' be two events executed at an object, e precedes e', denoted $e <_t e'$, if $ts(e) < ts(e')$.*

Each event execution reads and/or writes variables forming the object's state atomically. Hence, each event can be modeled as a set of read and write operations on elements in the object's state. We define the *read set* (resp. *write set*) of an event e, denoted $R(e)$ (resp. $W(e)$), the set of the state variables read (resp. modified) while executing e. $RW(e)$ represents the set obtained by the union of $R(e)$ and $W(e)$. We assume that if a state variable v is both read and written by the event e, then the read operation takes place before the write one.

An execution of a sequence of events at an object is called *run*[5]. A run moves an object from its initial state Σ_0 to a final state. A *timestamp-based* run represents an execution in which events are processed at an object according to a non-decreasing order of their timestamps, it then represents the *correct* run with respect to the causality criterion. The final state of a timestamp-based run is denoted Σ_{ts}. A run is *equivalent* to the timestamp-based run iff it contains the same set of events of the timestamp-based run and moves the state of the object from Σ_0 to the final state Σ_{ts}. The set of runs that are equivalent to the timestamp-based run is called *correct runs set* denoted R_c.

3 The weak causality relation

In this section we define the WEak Causality (WEC) relation between events occurring at an object. By using such a relation we show that events that are not WEC related can be executed in any order as they will produce a run in the correct runs set. We present WEC through an introductory example, then we provide a formal definition. Finally, we prove a theorem stating that any run consistent with WEC is equivalent to the timestamp-based run.

[4] Note that seeds for the random number generation either are elements of the object's state or are part of the content of an event.

[5] As we are interested in modeling the intra-object parallelism, in the reminder of the paper we consider a run as an execution at an object.

Let us suppose to have a timestamp-based run at an object with six events e_1, e_2, e_3, e_4, e_5 and e_6 and that the variables v_1, v_2, v_3 and v_4 form the state of the object. Moreover to each event are associated the following read/write sets:

$$R(e_1) = \{v_3\};\ W(e_1) = \{v_1, v_2\}$$

$$R(e_2) = \{v_2\};\ W(e_2) = \{v_3\}$$

$$W(e_3) = \{v_3\}$$

$$R(e_4) = \{v_1\};\ W(e_4) = \{v_4\}$$

$$R(e_5) = \{v_2\};\ W(e_5) = \{v_4\}$$

$$R(e_6) = \{v_3, v_4\}$$

Note that in the timestamp-based run, event e_2 reads a value of the variable v_2 written by e_1 while e_3 writes v_3 previously written by e_2. Also e_4 reads v_1 that was written in the timestamp-based run by e_1 and so on. This means that e_4 could be processed just after the execution of e_1 as it does not *conflict* on state variables with e_2 and e_3. Following these arguments, we get four runs which are able to move the simulation to the same final state of the timestamp-based run:

(a) $e_1, e_2, e_4, e_3, e_5, e_6$
(b) $e_1, e_2, e_4, e_5, e_3, e_6$
(c) $e_1, e_4, e_2, e_5, e_3, e_6$
(d) $e_1, e_4, e_2, e_3, e_5, e_6$

Previous observations suggest to formalize the definition of conflict between events as follows:

Definition 3.1 *Let e and e' be two events belonging to a run r. e conflicts with e', denoted $e <_C e'$, if the following predicate $\mathcal{P}$ is true:*

$$\mathcal{P} \equiv (W(e) \cap RW(e') \neq \emptyset) \vee (R(e) \cap W(e') \neq \emptyset)$$

By using the notion of conflict between events, let us introduce the WEC relation that models the intra-object parallelism:

Definition 3.2 *Let e and e' be two events belonging to a run r. e is weakly causally related to e', denoted $e \xrightarrow{WEC} e'$, if:*

(a) $(e <_t e') \wedge (e <_C e')$*; or*
(b) there exists an event $\hat{e}$ executed at the same object such that: $(e \xrightarrow{WEC} \hat{e}) \wedge (\hat{e} \xrightarrow{WEC} e')$.

Figure 1 depicts the WEC relations of the previous example. Hence events executed at an object can be modeled as a partial order $\widehat{E} = (E, \xrightarrow{WEC})$, where E represents the set of all events executed at the object. Two events e and e' such that $\neg(e \xrightarrow{WEC} e')$ and $\neg(e' \xrightarrow{WEC} e)$ are said to be concurrent and are denoted $e||e'$. Concurrent events can be executed in any order.

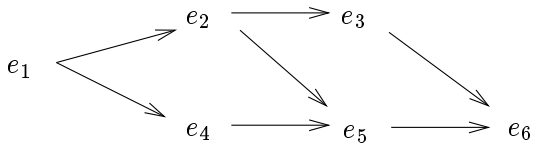

Fig. 1. The partial order on the sequence of events $(e_1, \ldots, e_6)$

3.2 Equivalence between linear extensions of $\widehat{E}$ and the timestamp-based run

We prove that any linear extension of $\widehat{E}$ represents a run equivalent to the timestamp-based one (i.e., a run in the correct runs set). Prior entering the proof, we introduce some simple notations. For each state variable $v \in R(e)$, we denote as $V_e^{ts}(v)$ the value of v read by the event e in the timestamp-based run (ts), and we denote as $V_e^{\alpha}(v)$ the value of v read by the event e in a run which is a linear extension α of $\widehat{E}$ distinct from ts.

Theorem 3.1 *Let $e \in E$ be a simulation event and α be a linear extension of $\widehat{E}$ with $\alpha \neq ts$. For any state variable $v \in R(e)$ we have $V_e^{\alpha}(v) = V_e^{ts}(v)$.*

Proof (by induction). Let us consider the sequence of events corresponding to a linear extension α with $\alpha \neq ts$. Let us denote as e_i (with $i \geq 1$) the i-th event of the sequence (i.e., the i-th event of the linear extension α). The theorem is proved by induction on i.
Base Step. $i = 1$. In this case e_i is the first event of the linear extension α. For each variable $v \in R(e_i)$ then e_i reads the initial value $init(v)$, that is $V_{e_i}^{\alpha}(v) = init(v)$. Suppose by contradiction that $V_{e_i}^{\alpha}(v) \neq V_{e_i}^{ts}(v)$, in this case we have that in ts there must exist an event $\bar{e}$ such that $\bar{e} <_t e_i$ and $\bar{e}$ writes onto v a value distinct from $init(v)$. In this case the write set of $\bar{e}$ and the read set of e_i intersect (i.e., $\bar{e} <_C e_i$), so we have by Definition 3.2 that $\bar{e} \xrightarrow{WEC} e_i$ meaning that e_i cannot be the first event of any linear extension of $\widehat{E}$. So the case assumption is contradicted and the claim follows.

Induction Step. We suppose the result is true for all the events e_i with $i > 1$ and show that is holds for the event e_{i+1}. Suppose by contradiction that there exists a variable $v \in R(e_{i+1})$ such that $V^{\alpha}_{e_{i+1}}(v) \neq V^{ts}_{e_{i+1}}(v)$. We have three cases:
(a) there does not exist any event $\bar{e} \in E$ such that $\bar{e}$ is distinct from e_{i+1} and $v \in W(\bar{e})$ (i.e., no event distinct from e_{i+1} writes v). In this case the event e_{i+1} reads from v the initial value $init(v)$ either in α or in ts. Thus the assumption is contradicted and the claim follows;
(b) all the events $\bar{e} \in E$ distinct from e_{i+1} and writing v are such that $e_{i+1} <_t \bar{e}$ (i.e., all the events which are distinct from e_{i+1} and write the variable v have timestamp larger than $ts(e_{i+1})$). In this case, for all such events $\bar{e}$ then $e_{i+1} \stackrel{WEC}{\longrightarrow} \bar{e}$ meaning that in the linear extension α cannot exist any event e_j with $j < i+1$ (i.e., e_j precedes e_{i+1} in α) which writes v. As for case (a), the event e_{i+1} reads from v the initial value $init(v)$ either in α or in ts. Thus the assumption is contradicted and the claim follows;
(c) there exists at least one event $\bar{e} \in E$ such that $v \in W(\bar{e})$ and $\bar{e} <_t e_{i+1}$ (i.e., there exists at least one event $\bar{e}$ which writes v and has timestamp less than $ts(e_{i+1})$). In this case $\bar{e} \stackrel{WEC}{\longrightarrow} e_{i+1}$ meaning that $\bar{e}$ must necessarily precede e_{i+1} in the linear extension α (i.e., $\bar{e} = e_j$ with $j < i+1$). Let e_m be the event in E with the highest timestamp and satisfying the condition of case (c). Then e_m (with $m < i+1$) is the latest event which writes v and precedes e_{i+1} both in ts and in α. We have supposed by contradiction that $V^{\alpha}_{e_{i+1}}(v) \neq V^{ts}_{e_{i+1}}(v)$. In this case during the execution of α, e_m writes on v a value distinct from the one that would have be written in ts. As the execution of e_m is deterministic, then the only way for this to occur is that e_m reads in α values different from the ones that would have been read in ts. This contradicts the induction step assumption that all the i events (including e_m) that precede e_{i+1} in α actually read the same values that would have been read in ts. Therefore the claim follows.

Corollary 3.2 *Any linear extension α of $\widehat{E}$ is equivalent to ts.*

Proof. By Theorem 3.1 we have that in any linear extension α of $\widehat{E}$ all the events in E read the same values that would have been read in the timestamp-based run. As the execution of each event is deterministic, then each event will produce the same updates and generates the same new events of ts (those updates on state variables will lead to the same final state of ts, namely Σ_{ts}), therefore, by definition, the linear extension α is equivalent to ts.

3.3 The set of correct runs originated by WEC

Let us now show that the set of runs R_c^{WEC} consisting of the linear extensions of $\widehat{E} = (E, \stackrel{WEC}{\longrightarrow})$ is the largest set containing correct runs with respect to the

model introduced in Section 2. To this purpose, we first show that any other relation WEC', weaker than WEC, defined on events allows runs that are not equivalent to the timestamp-based one (so any relation weaker that WEC may generate runs which are not correct). Then we show that any relation WEC' which is not weaker than WEC cannot originate a set of correct runs $R_c^{WEC'}$ which is larger than R_c^{WEC}.

To obtain a relation WEC' weaker than WEC we relax the notion of conflict between events. Hence, let us assume a conflict be determined by a predicate $\mathcal{P}'$ such that $\mathcal{P}' \Rightarrow \mathcal{P}$. By means of a case analysis, we show that for each predicate $\mathcal{P}'$ satisfying previous relation, there exists a run allowed by WEC' which is not equivalent to the timestamp-based run. We obtain the following cases:

$\mathcal{P}' \equiv FALSE$
In this case WEC' allows any out of timestamp order execution. Let consider a run with two events, say e_1, e_2 such that $e_1 <_t e_2$, and let v be the only variable forming the object's state. Events are such that: $W(e_1) = \{v\}$, $W(e_2) = \{v\}$. Trivially, if events are executed out of timestamp order, then the final state of the object is different from Σ_{ts}.

$\mathcal{P}' \equiv W(e) \cap RW(e') \neq \emptyset$
Let us consider a run with two events, e_1 and e_2 such that $e_1 <_t e_2$, and let v_1 and v_2 be variables forming the object's state. Suppose events are such that: $R(e_1) = \{v_1\}$, $W(e_1) = \{v_2\}$, $W(e_2) = \{v_1\}$. Under $\xrightarrow{WEC'}$, $e_1||e_2$, hence both runs (e_1, e_2) and (e_2, e_1) are allowed. However, the run (e_2, e_1) moves the state of the object to a value different from Σ_{ts} since e_1 reads the state variable v_1 updated by e_2 and writes v_2.

$\mathcal{P}' \equiv R(e) \cap W(e') \neq \emptyset$
Let consider a run with two events, e_1 and e_2 such that $e_1 <_t e_2$, and let v_1 and v_2 be variables forming the object's state. Suppose events are such that: $W(e_1) = \{v_1, v_2\}$, $W(e_2) = \{v_1\}$. Under $\xrightarrow{WEC'}$, we have $e_1||e_2$ hence both runs (e_1, e_2) and (e_2, e_1) are allowed. However, the run (e_2, e_1) moves the state of the object to a value different from Σ_{ts} since e_1 re-updates v_1 previously written by e_2.

Let us now investigate on the size of the set of runs $R_c^{WEC'}$ allowed by a relation WEC' which is not weaker than WEC. This relation is originated by a predicate $\mathcal{P}'$ such that $\mathcal{P}' \not\Rightarrow \mathcal{P}$ (recall that in this case WEC' allows only correct runs). The following case analysis shows that $R_c^{WEC'} \subseteq R_c^{WEC}$:

$\mathcal{P}' \Leftrightarrow \mathcal{P}$
In this case the two conflict predicates are equivalent, so we have that for any pair of events $< e, e' >$ belonging to E, if $e||e'$ under WEC' then $e||e'$ under

WEC, the converse is also true. This implies that the set of linear extensions of $\widehat{E} = (E, \overset{WEC'}{\longrightarrow})$ is identical to the set of linear extensions of $\widehat{E} = (E, \overset{WEC}{\longrightarrow})$.

$\mathcal{P}' \Leftarrow \mathcal{P}$
In this case we have that for any pair of events $< e, e' >$ belonging to a set of events E, if $e||e$ under WEC' then $e||e'$ under WEC, but the converse is not necessarily true. Therefore the set of linear extensions of $\widehat{E} = (E, \overset{WEC}{\longrightarrow})$ may contain at least one element which is not included in the set of linear extensions of $\widehat{E} = (E, \overset{WEC'}{\longrightarrow})$.

4 An application of weak causality to optimistic synchronization

In optimistic synchronization (Time Warp) objects execute events whenever they are available under the optimistic assumption that the execution order does not violate causality. Each time a causality violation is detected (i.e., an event is being executed whose timestamp is lower than that of some previously executed events), then the object is rolled back to its state immediately prior the violation and the execution resumes (7). In this section we show how WEC can be used to implement a particular kind of Time Warp synchronization; such a synchronization will be referred to as WEC-based Time Warp. The advantage of the latter over the original protocol lies on the possibility to not roll back the object state each time an out of timestamp order execution is detected. In particular, all the out of timestamp order runs that respect the partial order defined by WEC are allowed by the protocol.

As formalized in Section 3, events at an object are modeled by the partial order $\widehat{E} = (E, \overset{WEC}{\longrightarrow})$ and they can be executed according to any linear extension of $\widehat{E}$. Upon the execution of an event e, we can identify the set $E_{E_x}(e)$ (with $E_x(e) \subseteq E$) as the set of executed events [6] . Furthermore we denote as $E_{RL}(e)$ the subset of $E_{E_x}(e)$ consisting of all the events e' such that $e <_t e'$. We consider $E_{RL}(e)$ totally ordered by timestamp.

Each time an object schedules for execution an event e such that $E_{RL}(e) = \emptyset$, it executes e as in classical Time Warp synchronization. On the contrary, each time an event e is scheduled such that $E_{RL}(e) \neq \emptyset$, the object behaves as follows: it traverses $E_{RL}(e)$, starting from the minimum, until it finds an event e_r, if any, such that $e <_C e_r$. If e_r does not exists then e is executed. Otherwise, all events in $E_{RL}(e)$ with timestamp larger than $ts(e_r)$ are undone (i.e., they correspond to events not executed yet), and the state of the object is rolled back to its value immediately prior e_r's execution. Then e is executed.

[6] Note that events executed and then rolled back are not part of $E_{E_x}(e)$.

In both cases, we obtain a run (with e as last executed event) which is a prefix of a linear extension of $\widehat{E}$.

It is clear that the rollback extent (number of events undone due to rollback) is, on the average, smaller than the one of the original Time Warp protocol. In the best case it is zero (and no state restoration is executed at all), in the worst case it is equal to the cardinality of $E_{RL}(e)$.

4.1 Operational issues

From an operational point of view, the condition $e <_C e_r$ cannot always be tracked before executing e as sometimes $R(e)$ and $W(e)$ are determined during e's execution. In order to overcome this problem, we can define the read and write sets of an event e before e's execution, denoted $RB(e)$ and $WB(e)$ respectively, as the *maximum* set of state variables that e can *potentially* read and write during its execution (i.e., $RB(e) \supseteq R(e)$ and $WB(e) \supseteq W(e)$). Hence, the rollback point can be determined replacing $RB(e)$ and $WB(e)$ in the definition of conflict between events. The usage of RB and WB overestimates the number of weak causality dependencies between events but leads to a safe solution. Obviously, WEC-based Time Warp can lead to performance improvement whenever the encoding of Read/Write sets of events and their comparison is less space-time expensive[7]. Finally we remark that, even if there exists an event e_r in $E_{RL}(e)$ such that $e <_C e_r$, it might be possible that after the re-execution of the rolled back events we get the same result of the previous execution. This could happen, for example, if e does not really updates the object's state. In such a case, lazy cancellation and lazy rollback can be embedded in the previous WEC-based rollback scheme as they are an orthogonal approach to avoid unnecessary undoing of events.

5 Summary

In this paper we have provided a formal model for the parallelism of events in simulation objects. This parallelism allows to enlarge the set of correct runs of a simulation program. i.e., runs moving the object to the same state of a timestamp-based run. To model this parallelism, we have introduced a weak causality relation which is based on events' conflict as well as on events' timestamps. From an operational point of view, we have shown how weak

[7] This may happen, for example, either when events can be classified into types and each type corresponds to a given Read/Write set on state variables, or when Read/Write sets have small cardinality.

causality can be well suited to optimistic (Time Warp) synchronization to avoid unnecessary rollbacks in an orthogonal way with respect to both lazy cancellation and lazy rollback.

References

[1] K.M. Chandy, J. Misra, Distributed Simulation: a Case Study in Design and Verification of Distributed Programs, IEEE Transactions on Software Engineering, SE5 (5) (1979) 440-452.

[2] R.M. Fujimoto, Parallel Discrete Event Simulation, Communications of ACM, 33 (10) (1990) 30-53.

[3] A. Gafni, Space Management and Cancellation Mechanisms for Time Warp, Tech. Rep. TR-85-341, University of Southern California, Los Angeles (Ca,USA).

[4] M. Gunter, Understanding Supercritical Speedup, Proceedings of 8th Workshop on Parallel and Distributed Simulation, 1994, pp.81-87.

[5] L. Lamport, Time, Clock and the Ordering of Events in a Distributed System, Communications of the ACM, 21 (7) (1978) 558-565.

[6] D. Jefferson, P. Reiher, Supercritical Speedup, in: Proc. 24th Annual Simulation Symposium, pp.159-168.

[7] D. Jefferson, Virtual Time, ACM Trans. on Programming Languages and Systems, 7 (3) (1985) 404-425.

[8] S. Srinivasan, P.F. Reynolds, Super-Criticality Revisited, in: Proc. 9th Workshop on Parallel and Distributed Simulation, 1995, pp.130-136.

[9] D. West, Optimizing Time Warp: Lazy Rollback and Lazy Reevaluation, Master's Thesis, University of Calgary, January 1988.