Foundations of Declarative Data Analysis Using Limit Datalog Programs

Mark Kaminski, Bernardo Cuenca Grau, Egor V. Kostylev, Boris Motik and Ian Horrocks
Department of Computer Science, University of Oxford, UK
{mark.kaminski, bernardo.cuenca.grau, egor.kostylev, boris.motik, ian.horrocks}@cs.ox.ac.uk

Abstract
Motivated by applications in declarative data analysis, we study Limit Datalog—an extension of positive Datalog with arithmetic functions over integers. This language is known to be undecidable, so we propose two fragments. In limit Datalog₂ predicates are axiomatised to keep minimal/maximal numeric values, allowing us to show that fact entailment is coNExpTime-complete in combined, and coNP-complete in data complexity. Moreover, an additional stability requirement causes the complexity to drop to ExpTime and PTime, respectively. Finally, we show that stable Datalog₂ can express many useful data analysis tasks, and so our results provide a sound foundation for the development of advanced information systems.

1 Introduction
Analysing complex datasets is currently a hot topic in information systems. The term ‘data analysis’ covers a broad range of techniques that often involve tasks such as data aggregation, property verification, or query answering. Such tasks are currently often solved imperatively (e.g., using Java or Scala) by specifying how to manipulate the data, and this is undesirable because the objective of the analysis is often obscured by evaluation concerns. It has recently been argued that data analysis should be declarative [Alvaro et al., 2010; Markl, 2014; Seo et al., 2015; Shkapsky et al., 2016]: users should describe what the desired output is, rather than how to compute it. For example, instead of computing shortest paths in a graph by a concrete algorithm, one should (i) describe what a path length is, and (ii) select only paths of minimum length. Such a specification is independent of evaluation details, allowing analysts to focus on the task at hand. An evaluation strategy can be chosen later, and general parallel and/or incremental evaluation algorithms can be reused ‘for free’.

An essential ingredient of declarative data analysis is an efficient language that can capture the relevant tasks, and Datalog is a prime candidate since it supports recursion. Apart from recursion, however, data analysis usually also requires integer arithmetic to capture quantitative aspects of data (e.g., the length of a shortest path). Research on combining the two dates back to the ‘90s [Mumick et al., 1990; Kemp and Stuckey, 1991; Beeri et al., 1991; Van Gelder, 1992; Consens and Mendelzon, 1993; Ganguly et al., 1995; Ross and Sagiv, 1997], and is currently experiencing a revival [Faber et al., 2011; Mazuran et al., 2013]. This extensive body of work, however, focuses primarily on integrating recursion and arithmetic with aggregate functions in a coherent semantic framework, where technical difficulties arise due to nonmonotonicity of aggregates. Surprisingly little is known about the computational properties of integrating recursion with arithmetic, apart from that a straightforward combination is undecidable [Dantsin et al., 2001]. Undecidability also carries over to the above formalisms and practical Datalog-based systems such as BOOM [Alvaro et al., 2010], DeALS [Shkapsky et al., 2016], Myria [Wang et al., 2015], SociaLite [Seo et al., 2015], Overlog [Loo et al., 2009], Dyna [Eisner and Filardo, 2011], and Yedalog [Chin et al., 2015].

To develop a sound foundation for Datalog-based declarative data analysis, we study Datalog₂—negation-free Datalog with integer arithmetic and comparisons. Our main contribution is a new limit Datalog₂ fragment that, like the existing data analysis languages, is powerful and flexible enough to naturally capture many important analysis tasks. However, unlike Datalog₂ and the existing languages, reasoning with limit programs is decidable, and it becomes tractable in data complexity under an additional stability restriction.

In limit Datalog₂, all intensional predicates with a numeric argument are limit predicates. Instead of keeping all numeric values for a given tuple of objects, such predicates keep only the minimal (min) or only the maximal (max) bounds of numeric values entailed for the tuple. For example, if we encode a weighted directed graph using a ternary predicate edge, then rules (1) and (2), where sp is a min limit predicate, compute the cost of a shortest path from a given source node v₀ to every other node.

\[
\text{sp}(v₀, 0) \quad (1)
\]
\[
\text{sp}(x, m) \land \text{edge}(x, y, n) \Rightarrow \text{sp}(y, m + n) \quad (2)
\]

If these rules and a dataset entail a fact sp(v, k), then the cost of a shortest path from v₀ to v is at most k; hence, sp(v, k’) holds for each k’ ≥ k since the cost of a shortest path is also at most k’. Rule (2) intuitively says that, if x is reachable from v₀ with cost at most m and (x, y) is an edge of cost n, then v’ is reachable from v₀ with cost at most m + n. This is different from Datalog₂, where there is no implicit semantic con-
connection between \( sp(v,k) \) and \( sp(v,k') \), and such semantic connections allow us to prove decidability of limit \( \text{Datalog}_Z \).

We provide a direct semantics for limit predicates based on Herbrand interpretations, but we also show that this semantics can be axiomatised in standard \( \text{Datalog}_Z \). Our formalism can thus be seen as a fragment of \( \text{Datalog}_Z \), from which it inherits well-understood properties such as monotonicity and existence of a least fixpoint model [Dantsin et al., 2001].

Our contributions are as follows. First, we introduce limit \( \text{Datalog}_Z \) programs and argue that they can naturally capture many relevant data analysis tasks. We prove that fact entailment in limit \( \text{Datalog}_Z \) is undecidable, but, after restricting the use of multiplication, it becomes \( \text{CONEXPTIME} \)- and \( \text{CONP} \)-complete in combined and data complexity, respectively. To achieve tractability in data complexity (which is very important for robust behaviour on large datasets), we additionally introduce a stability restriction and show that this does not prevent expressing the relevant analysis tasks.

The proofs of all results are given in an extended version of this paper [Kaminski et al., 2017].

2 Preliminaries

In this section, we recapitulate the well-known definitions of \( \text{Datalog} \) with integers, which we call \( \text{Datalog}_Z \).

Syntax A vocabulary consists of predicates, objects, object variables, and numeric variables. Each predicate has an integer arity \( n \), and each position \( 1 \leq i \leq n \) is of either object or numeric sort. An object term is an object or an object variable. A numeric term is a numeric variable, or of the form \( s_1 + s_2, s_1 - s_2, s_1 \times s_2 \) where \( s_1 \) and \( s_2 \) are numeric terms, and \( \times \) and \( \div \) are the standard arithmetic functions. A constant is an object or an integer. The magnitude of an integer is its absolute value. A standard atom is of the form \( B(t_1, \ldots, t_n) \), where \( B \) is a predicate of arity \( n \) and each \( t_i \) is a term whose type matches the sort of position \( i \) of \( B \). A comparison atom is of the form \( (s_1 < s_2) \) or \( (s_1 \leq s_2) \), where \( < \) and \( \leq \) are the standard comparison predicates, and \( s_1 \) and \( s_2 \) are numeric terms.

A rule \( r \) is of the form \( \bigwedge_{i=1}^m \alpha_i \land \bigwedge_{j=1}^n \beta_j \rightarrow \alpha \), where \( \alpha(i) \) are standard atoms, \( \beta(j) \) are comparison atoms, and each variable in \( r \) occurs in some \( \alpha_i \). Atom \( b(r) = \alpha \) is the head of \( r \); \( s_b(r) = \bigwedge_{i=1}^m \alpha_i \) is the standard body of \( r \); \( c_b(r) = \bigwedge_{j=1}^n \beta_j \) is the comparison body of \( r \); and \( b(r) = s_b(r) \land c_b(r) \) is the body of \( r \). A ground instance of \( r \) is obtained from \( r \) by substituting all variables by constants. A \( \text{Datalog}_Z \) program \( P \) is a finite set of rules.

Predicate \( B \) is intensional (IDB) in \( P \) if \( B \) occurs in \( \Pi \) in the head of a rule whose body is not empty; otherwise, \( B \) is extensional (EDB) in \( P \). A term, atom, rule, or program is ground if it contains no variables. A fact is a ground, function-free, standard atom. Program \( P \) is a dataset if \( h(r) \) is a fact and \( b(r) = \emptyset \) for each \( r \in P \). We often say that \( P \) contains a fact \( \alpha \) and write \( \alpha \in P \), which actually means \( \alpha \in \Pi \). We write a tuple of terms as \( t \), and we often treat conjunctions and tuples as sets and write, say, \( \alpha_i \in s_b(r), \langle t \rangle \), and \( t_i \in t \).

Semantics A (Herbrand) interpretation \( I \) is a (not necessarily finite) set of facts. Such \( I \) satisfies a ground atom \( \alpha \), written \( I \models \alpha \), if (i) \( \alpha \) is a standard atom and evaluating the arithmetic functions in \( \alpha \) produces a fact in \( I \), or (ii) \( \alpha \) is a comparison atom and evaluating the arithmetic functions and comparisons produces true. The notion of satisfaction is extended to conjunctions of ground atoms, rules, and programs as in first-order logic, where each rule is universally quantified. If \( I \models P \), then \( I \) is a model of program \( P \); and \( P \) entails a fact \( \alpha \), written \( P \models \alpha \), if \( I \models P \) holds whenever \( I \models P \).

Complexity In this paper we study the computational properties of checking \( P \models \alpha \). Combined complexity assumes that both \( P \) and \( \alpha \) are part of the input. In contrast, data complexity assumes that \( P \) is given as \( P' \cup D \) for \( P' \) a program and \( D \) a dataset, and that only \( D \) and \( \alpha \) are part of the input while \( P' \) is fixed. Unless otherwise stated, all numbers in the input are coded in binary, and the size \( ||P|| \) of \( P \) is the size of its representation. Checking \( P \models \alpha \) is undecidable even if the only arithmetic function in \( P \) is + [Dantsin et al., 2001].

Presburger arithmetic is first-order logic with constants 0 and 1, functions + and −, equality, and the comparison predicates < and ≤, interpreted over all integers \( \mathbb{Z} \). The complexity of checking sentence validity (i.e., whether the sentence is true in all models of Presburger arithmetic) is known when the number of quantifier alternations and/or the number of variables in each quantifier block are fixed [Berman, 1980; Grädel, 1988; Schöning, 1997; Haase, 2014].

3 Limit Programs

Towards introducing a decidable fragment of \( \text{Datalog}_Z \) for data analysis, we first note that the undecidability proof of (plain) \( \text{Datalog}_Z \), outlined by Dantsin et al. [2001] uses atoms with at least two numeric terms. Thus, to motivate introducing our fragment, we first prove that undecidability holds even if atoms contain at most one numeric term. The proof uses a reduction from the halting problem for deterministic Turing machines. To ensure that each standard atom in \( P \) has at most one numeric term, combinations of a time point and a tape position are encoded using a single integer.

Theorem 1. For \( P \) a \( \text{Datalog}_Z \) program and \( \alpha \) a fact, checking \( P \models \alpha \) is undecidable even if \( P \) contains no × or − and each standard atom in \( P \) has at most one numeric term.

We next introduce limit \( \text{Datalog}_Z \), where limit predicates keep bounds on numeric values. This language can be seen as either a semantic or a syntactic restriction of \( \text{Datalog}_Z \).

Definition 2. In limit \( \text{Datalog}_Z \), a predicate is either an object predicate with no numeric positions, or a numeric predicate where only the last position is numeric. A numeric predicate is either an ordinary numeric predicate or a limit predicate, and the latter is either a min or a max predicate. Atoms with object predicates are object atoms, and analogously for other types of atoms/predicates. A \( \text{Datalog}_Z \) rule \( r \) is a limit (\( \text{Datalog}_Z \)) rule if (i) \( b(r) = \emptyset \), or (ii) each atom in \( s_b(r) \) is an object, ordinary numeric, or limit atom, and \( b(r) \) is an object or a limit atom. A limit (\( \text{Datalog}_Z \)) program \( P \) is a program containing only limit rules; and \( P \) is homogeneous if it does not contain both min and max predicates.

In the rest of this paper we make three simplifying assumptions. First, numeric atoms occurring in a rule body are function-free (but comparison atoms and the head can contain arithmetic functions). Second, each numeric variable in...
a rule occurs in at most one standard body atom. Third, dis-
tinct rules in a program use different variables. The third
assumption is clearly w.l.o.g. because all variables are uni-
versally quantified so their names are immaterial. Moreover,
the first two assumptions are w.l.o.g. as well since, for each
rule, there exists a logically equivalent rule that satisfies these
assumptions. In particular, we can replace an atom such as
\( A(t, m_1 + m_2) \) with conjunction
\[
A(t, m) \land I(m_1) \land I(m_2) \land (m \leq m_1 + m_2) \land (m_1 + m_2 \leq m)
\]
where \( m \) is a fresh variable and \( I \) is a fresh predicate axiom-
tised to hold on all integers as follows:
\[
I(0) \quad I(m) \rightarrow I(m + 1) \quad I(m) \rightarrow I(m - 1)
\]
Also, we can replace atoms \( A_1(t_1, m) \land A_2(t_2, m) \) with con-
junction \( A_1(t_1, m) \land A_2(t_2, m') \land (m \leq m') \land (m' \leq m) \),
where \( m' \) is a fresh variable.

Intuitively, a limit fact \( B(a, k) \) says that the value of \( B \) for
a tuple of objects \( a \) is at least \( k \) (if \( B \) is max) or at most \( k \)
(if \( B \) is min). For example, a fact \( sp(v, k) \) in our shortest
path example from Section 1 says that node \( v \) is reachable
from \( v_0 \) via a path with cost at most \( k \). To capture this
intended meaning, we require interpretations \( I \) to be closed
for limit predicates—that is, whenever \( I \) contains a limit fact \( a \), it
also contains all facts implied by \( a \) according to the predicate
type. In our example, this captures the observation that the
existence of a path from \( v_0 \) to \( v \) of cost at most \( k \) implies the
existence of such a path of cost at most \( k' \) for each \( k' \geq k \).

**Definition 3.** An interpretation \( I \) is limit-closed if, for each
limit fact \( B(a, k) \in I \) where \( B \) is a min (resp. max) predi-
cate, \( B(a, k') \in I \) holds for each integer \( k' \) with \( k \leq k' \) (resp.
\( k' \leq k \)). An interpretation \( I \) is a model of a limit program \( \mathcal{P} \)
if \( I = \mathcal{P} \) and \( I \) is limit-closed. The notion of entailment is
modified to take into account only limit-closed models.

The semantics of limit predicates in a limit \( \text{Datalog}_L \) pro-
gram \( \mathcal{P} \) can be axiomatised explicitly by extending \( \mathcal{P} \) with
the following rules, where \( Z \) is a fresh predicate. Thus, limit
\( \text{Datalog}_L \) can be seen as a syntactic fragment of \( \text{Datalog}_Z \):
\[
\begin{align*}
\rightarrow Z(0) & \quad Z(m) \rightarrow Z(m + 1) & Z(m) \rightarrow Z(m - 1) \\
B(x, m) \land Z(n) \land (n \leq m) & \rightarrow B(x, n) & \text{for each min predicate } B \text{ in } \mathcal{P} \\
B(x, m) \land Z(n) \land (n \geq m) & \rightarrow B(x, n) & \text{for each max predicate } B \text{ in } \mathcal{P}
\end{align*}
\]

Each limit program can be reduced to a homogeneously
program; however, for the sake of generality, in our technical
results we do not require programs to be homogeneous.

**Proposition 4.** For each limit program \( \mathcal{P} \) and fact \( \alpha \), a ho-
ogeneous program \( \mathcal{P}' \) and fact \( \alpha' \) can be computed in linear
time such that \( \mathcal{P} \models \alpha \iff \text{only if } \mathcal{P}' \models \alpha'. \)

Intuitively, program \( \mathcal{P}' \) in Proposition 4 is obtained by re-
placing all min (or all max) predicates in \( \mathcal{P} \) by fresh max
(resp. min) predicates and negating their numeric arguments.

In Section 1 we have shown that limit \( \text{Datalog}_L \) can com-
pute the cost of shortest paths in a graph. We next present
further examples of data analysis tasks that our formalism
can handle. In all examples, we assume that all objects in
the input are arranged in an arbitrary linear order using facts
\( \text{first}(a_1), \text{next}(a_1, a_2), \ldots, \text{next}(a_{n - 1}, a_n) \); we use this or-
der to simulate aggregation by means of recursion.

**Example 5.** Consider a social network where agents are
connected by the ‘follows’ relation. Agent \( a_i \) introduces (tweets)
a message, and each agent \( a_i \) retweets the message if at least
\( k_{a_i} \) agents that \( a_i \) follows tweet the message, where \( k_{a_i} \)
is a positive threshold uniquely associated with \( a_i \). Our goal
is to determine which agents tweet the message eventually.
To achieve this using limit \( \text{Datalog}_L \), we encode the network
structure in a dataset \( \mathcal{D}_L \) containing facts follows \((a_i, a_j) \)
\( a_i \) follows \( a_j \), and ordinary numeric facts \( \text{th}(a_i, k_{a_i}) \) if \( a_i \)’s
threshold is \( k_{a_i} \). Program \( \mathcal{P}_L \), containing rules (3)–(8), en-
codes message propagation, where \( nt \) is a max predicate.

\[
\begin{align*}
\rightarrow tw(a_i) & \\
\text{follows}(x, y') \land \text{first}(y) & \rightarrow nt(x, y, 0) \\
\text{follows}(x, y) \land \text{first}(y) \land tw(y) & \rightarrow nt(x, y, 1) \\
nt(x, y', m) \land \text{next}(y', y) & \rightarrow nt(x, y, m) \\
nt(x, y', m) \land \text{next}(y', y) \land \\
\text{follows}(x, y) \land tw(y) & \rightarrow nt(x, y, m + 1) \\
th(x, m) \land nt(x, y, n) & \rightarrow tw(x)
\end{align*}
\]

Specifically, \( \mathcal{P}_L \cup \mathcal{D}_L \models tw(a_i) \) iff \( a_i \) tweets the message.
Intuitively, \( nt(a_i, a_j, m) \) is true if, out of agents \( \{a_1, \ldots, a_k\} \)
(according to the order), at least \( m \) agents that \( a_i \) follows
tweet the message. Rules (4) and (5) initialise \( nt \) for the first
agent in the order; \( nt \) is a max predicate, so if the first agent
tweets the message, rule (5) ‘overrides’ rule (4). Rules (6)
and (7) recurse over the order to compute \( nt \) as stated above.

**Example 6.** Limit \( \text{Datalog}_L \) can also solve the problem of
counting paths between pairs of nodes in a directed acyclic
graph. We encode the graph in the obvious way as a dataset
\( \mathcal{D}_L \) that uses object predicates node and edge. Program
\( \mathcal{P}_L \), consisting of rules (9)–(14) where \( np \) and \( np' \) are max
predicates, then counts the paths.

\[
\begin{align*}
\downarrow node(x) & \rightarrow np(x, y, z, 0) \\
node(x) \land node(y) \land \text{first}(z) & \rightarrow np'(x, y, z, 0) \\
edge(x, z) \land np(z, y, m) \land \text{first}(z) & \rightarrow np'(x, y, z, m) \\
np'(x, y, z', m) \land \text{next}(z', z) & \rightarrow np'(x, y, z, m) \\
np'(x, y, z', m) \land \text{next}(z', z) \land edge(x, z) \land np(z, y, n) & \rightarrow np'(x, y, z, m + n) \\
np'(x, y, z, m) & \rightarrow np'(x, y, z, m) \\
np'(x, y, z, m) & \rightarrow np'(x, y, z, m)
\end{align*}
\]

Specifically, \( \mathcal{P}_L \cup \mathcal{D}_L \models np(a_i, a_j, k) \) iff at least \( k \) paths
exist from node \( a_i \) to node \( a_j \). Intuitively, \( np'(a_i, a_j, a_k, m) \) is
true if \( m \) is at least the sum of the number of paths from each
\( d \in \{a_1, \ldots, a_k\} \) (according to the order) to \( a_j \) for which
there exists an edge from \( a_i \) to \( d \). Rule (9) says that each
node has one path to itself. Rule (10) initialises aggregation
by saying that, for the first node \( z \), there are zero paths from \( x \)
to \( y \), and rule (11) overrides this if there exists an edge from \( x \)
to \( z \). Finally, rule (12) propagates the sum for \( x \) to the next
\( z \) in the order, and rule (13) overrides this if there is an edge
from \( z \) to \( y \) by adding the number of paths from \( z' \) and \( z \) to \( y \).
Example 7. Assume that, in the graph from Example 6, each node $a_i$ is associated with a bandwidth $b_{a_i}$, limiting the number of paths going through $a_i$ to at most $b_{a_i}$. To count the paths compliant with the bandwidth requirements, we extend $D_{\mathcal{OP}}$ to dataset $D_{\text{bcp}}$ that additionally contains an ordinary numeric fact $bw(a_i, b_{a_i})$ for each node $a_i$, and we define $\mathcal{P}_{\text{bcp}}$ by replacing rule (14) in $\mathcal{P}_{\mathcal{OP}}$ with the following rule.

$$np'(x,y,z,m) \land bw(z,n) \land (m \leq n) \rightarrow np(x,y,m)$$

Then, $\mathcal{P}_{\text{bcp}} \cup D_{\text{bcp}} = \{np(a_i, a_j, k) \mid \text{if there exist at least k paths from node } a_i \text{ to node } a_j, \text{ where the bandwidth requirement is satisfied for all nodes on such path.}\}$

4 Fixpoint Characterisation of Entailment

Programs are often grounded to eliminate variables and thus simplify the presentation. In limit Datalog$_2$, however, numeric variables range over integers, so a grounding can be infinite. Thus, we first specialise the notion of a grounding.

Definition 8. A rule $r$ is semi-ground if every variable in $r$ is a numeric variable that occurs in $r$ in a limit body atom. A limit program $\mathcal{P}$ is semi-ground if all of its rules are semi-ground.

The semi-grounding of $\mathcal{P}$ contains, for each $r \in \mathcal{P}$, each rule obtained from $r$ by replacing each variable not occurring in $r$ in a numeric argument of a limit atom with a constant of $\mathcal{P}$.

Obviously, $\mathcal{P} \models \alpha$ if and only if $\mathcal{P'} \models \alpha$ for $\mathcal{P'}$ the semi-grounding of $\mathcal{P}$.

We next characterise entailment of limit programs by pseudo-interpretations, which compactly represent limit-closed interpretations. If a limit-closed interpretation $I$ contains $B(b,k)$ where $B$ is a min predicate, then either the limit value $\ell \leq k$ exists such that $B(b,\ell) \in I$ and $B(b,k') \notin I$ for $k' < \ell$, or $B(b,k') \in I$ holds for all $k' \leq k$, and dually for $B$ a max predicate. Thus, to characterise the value of $B$ on a tuple of objects $b$ in $I$, we just need the limit value, or information that no such value exists.

Definition 9. A pseudo-interpretation $I$ is a set of facts over integers extended with a special symbol $\infty$ such that $k = k'$ holds for all limit facts $B(b,k)$ and $B(b,k')$ in $I$.

Limit-closed interpretations correspond naturally and one-to-one to pseudo-interpretations, so we can recast the notions of satisfaction and model using pseudo-interpretations. Unlike for interpretations, the number of facts in a pseudo-model of a semi-ground limit program $\mathcal{P}$ can be bounded by $|\mathcal{P}|$.

Definition 10. A limit-closed interpretation $I$ corresponds to a pseudo-interpretation $J$ if $I$ contains exactly all object and ordinary numeric facts of $J$, and, for each limit predicate $B$, each tuple of objects $b$, and each integer $\ell$, (i) $B(b,k) \in I$ for all $k$ if and only if $B(b,\infty) \in I$, and (ii) $B(b,\ell) \in I$ and $B(b,k) \notin I$ for all $k' < \ell$ (resp. $\ell < k$) and $B$ is a min (resp. max) predicate if and only if $B(b,\ell) \in J$.

Let $J$ and $J'$ be pseudo-interpretations corresponding to interpretations $I$ and $I'$. Then, $J$ satisfies a ground atom $\alpha$, written $J \models \alpha$, if $I \models \alpha$: $J$ is a pseudo-model of a program $\mathcal{P}$, written $J \models \mathcal{P}$, if $I \models \mathcal{P}$; finally, $J \models J'$ holds if $I \models I'$.

Example 11. Let $I$ be the interpretation consisting of $A(1), A(2), B(a,k)$ for $k \leq 5$, and $B(b,k)$ for $k \in \mathbb{Z}$, where $A$ is an ordinary numeric predicate, $B$ is a max predicate, and $a$ and $b$ are objects. Then, $\{A(1), A(2), B(a,5), B(b,\infty)\}$ is the pseudo-interpretation corresponding to $I$.

We next introduce the immediate consequence operator $T_{\mathcal{P}}$, of a limit program $\mathcal{P}$ on pseudo-interpretations. We assume for simplicity that $\mathcal{P}$ is semi-ground. To apply a rule $r \in \mathcal{P}$ to a pseudo-interpretation $J$ while correctly handling limit atoms, operator $T_{\mathcal{P}}$ converts $r$ into a linear integer constraint $C(r,J)$ that captures all ground instances of $r$ applicable to the limit-closed interpretation $I$ corresponding to $J$. If $C(r,J)$ has no solution, $r$ is not applicable to $J$. Otherwise, $h(r)$ is added to $J$ if it is not a limit atom; and if $h(r)$ is a min (max) atom $B(m)$, then the minimal (maximal) solution $\ell$ for $m$ in $C(r,J)$ is computed, and $J$ is updated such that the limit value of $B$ on $b$ is at least (at most) $\ell$—that is, the application of $r$ to $J$ keeps only the ‘best’ limit value.

Definition 12. For $\mathcal{P}$ a semi-ground limit program, $r \in \mathcal{P}$, and $J$ a pseudo-interpretation, $C(r,J)$ is the conjunction of comparison atoms containing (i) $\text{ch}(r)$; (ii) $(0 \leq 0)$ if an object or ordinary numeric atom $\alpha \in \text{sb}(r)$ exists with $\alpha \notin J$, or a limit atom $B(b,s) \in \text{sb}(r)$ exists with $B(b,\ell) \notin J$ for each $\ell$; and (iii) $(\ell \leq s)$ (resp. $(s \leq \ell)$) for each min (resp. max) atom $B(b,s) \in \text{sb}(r)$ with $B(b,\ell) \in J$ and $\ell \neq \infty$.

Rule $r$ is applicable to $J$ if $C(r,J)$ has an integer solution.

Assume $r$ is applicable to $J$. If $h(r)$ is an object or ordinary numeric atom, let $\text{hd}(h(r),J) = h(r)$. If $h(r) = B(b,s)$ is a min (resp. max) atom, the optimum value $\text{opt}(r,J)$ is the smallest (resp. largest) value of $s$ in all solutions to $C(r,J)$, or infinite if no such bound on the value of $s$ in the solutions to $C(r,J)$ exists; moreover, $\text{hd}(h(r),J) = B(b,\text{opt}(r,J))$.

Operator $T_{\mathcal{P}}(J)$ maps $J$ to the smallest w.r.t. $\sqsubseteq$ pseudo-interpretation satisfying $h(r,J)$ for each $r \in \mathcal{P}$ applicable to $J$. Finally, $T_{\mathcal{P}}^0 = \emptyset$, and $T_{\mathcal{P}}^n = T_{\mathcal{P}}(T_{\mathcal{P}}^{n−1})$ for $n > 0$.

Example 13. Let $r$ be $A(x) \land (2 \leq x) \rightarrow B(x+1)$ with $A$ and $B$ max predicates. Then, $C(r,J) = (2 \leq x)$ has no solution, and therefore rule $r$ is not applicable to the empty pseudo-interpretation. Moreover, for $J = \{A(3)\}$, conjunction $C(r,J) = (2 \leq x) \land (x \leq 3)$ has two solutions—{$x \rightarrow 2$} and {$x \rightarrow 3$}—and therefore rule $r$ is applicable to $J$. Finally, $B$ is a max predicate, and so $\text{opt}(r,J) = \max(2,1,3+1) = 4$, and $\text{hd}(h(r),J) = B(4)$. Consequently, $T_{\mathcal{P}}(J) = \{B(4)\}$.

Lemma 14. For each semi-ground limit program $\mathcal{P}$, operator $T_{\mathcal{P}}$ is monotonic w.r.t. $\sqsubseteq$. Moreover, $J \models \mathcal{P}$ if and only if $T_{\mathcal{P}}(J) \sqsubseteq J$ for each pseudo-interpretation $J$.

Monotonicity ensures existence of the closure $T_{\mathcal{P}}^\infty$ of $\mathcal{P}$—the least pseudo-interpretation such that $T_{\mathcal{P}} \sqsubseteq T_{\mathcal{P}}^\infty$ for each $n \geq 0$. The following theorem characterises entailment and provides a bound on the number of facts in the closure.

Theorem 15. For $\mathcal{P}$ a semi-ground limit program and $\alpha$ a fact, $\mathcal{P} \models \alpha$ if and only if $T_{\mathcal{P}}^\infty \models \alpha$; also, $|T_{\mathcal{P}}^\infty| \leq |\mathcal{P}|$; and $J \models \mathcal{P}$ implies $T_{\mathcal{P}}^\infty \sqsubseteq J$ for each pseudo-interpretation $J$.

The proofs for the first and the third claim of Theorem 15 use the monotonicity of $T_{\mathcal{P}}$ analogously to plain Datalog. The second claim holds since, for each $n \geq 0$, each pair of distinct facts in $T_{\mathcal{P}}^n$ must be derived by distinct rules in $\mathcal{P}$.

5 Decidability of Entailment: Limit-Linearity

We now start our investigation of the computational properties of limit Datalog$_2$. Theorem 15 bounds the cardinality of
the closure of a semi-ground program, but it does not bound the magnitude of the integers occurring in limit facts; in fact, integers can be arbitrarily large. Moreover, due to multiplication, checking rule applicability requires solving nonlinear inequalities over integers, which is undecidable.

**Theorem 16.** For \( \mathcal{P} \) a semi-ground limit program and \( \alpha \) a fact, checking \( \mathcal{P} \models \alpha \) and checking applicability of a rule of \( \mathcal{P} \) to a pseudo-interpretation are both undecidable.

The proof of Theorem 16 uses a straightforward reduction from Hilbert’s tenth problem.

Checking rule applicability is undecidable due to products of variables in inequalities; however, for linear inequalities that prohibit multiplying variables, the problem can be solved in NP, and in polynomial time if we bound the number of variables. Thus, to ensure decidability, we next restrict limit programs so that their semi-groundings contain only linear numeric terms. All our examples satisfy this restriction.

**Definition 17.** A limit rule \( r \) is limit-linear if each numeric term in \( r \) is of the form \( s_0 + \sum_{i=1}^{n} s_i \times m_i \), where (i) each \( m_i \) is a distinct numeric variable occurring in a limit body atom of \( r \), (ii) term \( s_0 \) contains no variable occurring in a limit body atom of \( r \), and (iii) each \( s_i \) with \( i \geq 1 \) is a term constructed using multiplication \( \times \), integers, and variables not occurring in limit body atoms of \( r \). A limit-linear program contains only limit-linear rules.\(^1\)

In the rest of this section, we show that entailment for limit-linear programs is decidable and provide tight complexity bounds. Our upper bounds are obtained via a reduction to the validity of Presburger formulas of a certain shape.

**Lemma 18.** For \( \mathcal{P} \) a semi-ground limit-linear program and \( \alpha \) a fact, there exists a Presburger sentence \( \varphi = \forall x \exists y . \bigvee_{i=1}^{n} \psi_i \) that is valid if and only if \( \mathcal{P} \models \alpha \). Each \( \psi_i \) is a conjunction of possibly negated atoms. Moreover, \( |x| + |y| \) and each \( \| \psi_i \| \) are bounded polynomially by \( \| \mathcal{P} \| + \| \alpha \| \). Number \( n \) is bounded polynomially by \( \| \mathcal{P} \| \) and exponentially by \( \max_{r \in \mathcal{P}} \| r \| \). Finally, the magnitude of each integer in \( \varphi \) is bounded by the maximal magnitude of an integer in \( \mathcal{P} \) and \( \alpha \).

The reduction in Lemma 18 is based on three main ideas. First, for each limit atom \( B(b, s) \) in a semi-ground program \( \mathcal{P} \), we use a Boolean variable \( \text{def}_{Bb} \) to indicate that an atom of the form \( B(b, \ell) \) exists in a pseudo-model of \( \mathcal{P} \), a Boolean variable \( \text{fin}_{Bb} \) to indicate whether the value of \( \ell \) is finite, and an integer variable \( \text{val}_{Bb} \) to capture \( \ell \) if it is finite. Second, each rule of \( \mathcal{P} \) is encoded as a universally quantified Presburger formula by replacing each standard atom with its encoding. Finally, entailment of \( \alpha \) from \( \mathcal{P} \) is encoded as a sentence stating that, in every pseudo-interpretation, either some rule in \( \mathcal{P} \) is not satisfied, or \( \alpha \) holds; this requires universal quantifiers to quantify over all models, and existential quantifiers to negate the (universally quantified) program.

Lemma 19 bounds the magnitude of integers in models of Presburger formulas from Lemma 18. These bounds follow from recent deep results on semi-linear sets and their connection to Presburger arithmetic [Chistikov and Haase, 2016].

**Lemma 19.** Let \( \varphi = \forall x \exists y . \bigvee_{i=1}^{n} \psi_i \) be a Presburger sentence where each \( \psi_i \) is a conjunction of possibly negated atoms of size at most \( k \) mentioning at most \( \ell \) variables, \( k \) is the maximal magnitude of an integer in \( \varphi \), and \( m = |x| \). Then, \( \varphi \) is valid if and only if \( \varphi \) is valid over models where each integer variable assumes a value whose magnitude is bounded by \( (2^{O(\ell \log \ell)} \cdot a^k \cdot m^k) \cdot n \cdot O(m^k) \).

Lemmas 18 and 19 provide us with bounds on the size of counter-pseudo-models for entailment.

**Theorem 20.** For \( \mathcal{P} \) a semi-ground limit-linear program, \( \mathcal{D} \) a dataset, and \( \alpha \) a fact, \( \mathcal{P} \cup \mathcal{D} \not\models \alpha \) if and only if a pseudo-model \( J \) of \( \mathcal{P} \cup \mathcal{D} \) exists where \( J \not\models \alpha \), \( |J| \leq |\mathcal{P} \cup \mathcal{D}| \), and the magnitude of each integer in \( J \) is bounded polynomially by the largest magnitude of an integer in \( \mathcal{P} \cup \mathcal{D} \), exponentially by \( \| \mathcal{P} \| \), and double-exponentially by \( \max_{r \in \mathcal{P}} \| r \| \).

By Theorem 20, the following nondeterministic algorithm decides \( \mathcal{P} \not\models \alpha \):

1. compute the semi-grounding \( \mathcal{P}' \) of \( \mathcal{P} \);
2. guess a pseudo-interpretation \( J \) that satisfies the bounds given in Theorem 20;
3. if \( \mathcal{T}_{\mathcal{P}'}(J) \subseteq J \) (so \( J \models \mathcal{P}' \)) and \( J \not\models \alpha \), return true.

Step 1 requires exponential (polynomial in data) time, and it does not increase the maximal size of a rule. Hence, Step 2 is nondeterministic exponential (polynomial in data), and Step 3 requires exponential (polynomial in data) time to solve a system of linear inequalities. Theorem 21 proves that these bounds are both correct and tight.

**Theorem 21.** For \( \mathcal{P} \) a limit-linear program and \( \alpha \) a fact, deciding \( \mathcal{P} \models \alpha \) is \text{CO\text{-}EXP\text{-}TIME}-complete in combined and \text{CONP\text{-}complete in data complexity.}

The upper bounds in Theorem 21 follow from Theorem 20, \text{CONP\text{-}hardness in data complexity is shown by a reduction from the square tiling problem, and \text{CO\text{-}EXP\text{-}TIME\text{-}hardness in combined complexity is shown by a similar reduction from the succinct version of square tiling.}

6 Tractability of Entailment: Stability

Tractability in data complexity is important on large datasets, so we next present an additional stability condition that brings the complexity of entailment down to \text{EXPTIME} in combined and \text{PTIME} in data complexity, as in plain Datalog.

6.1 Cyclic Dependencies in Limit Programs

The fixpoint of a plain Datalog program can be computed in \text{PTIME} in data complexity. However, for \( \mathcal{P} \) a limit-linear program, a naïve computation of \( \mathcal{T}_{\mathcal{P}} \) may not terminate since repeated application of \( \mathcal{T}_{\mathcal{P}} \) can produce larger and larger numbers. Thus, we need a way to identify when the numeric argument \( \ell \) of a limit fact \( A(a, \ell) \) diverges—that is, grows or decreases without a bound; moreover, to obtain a procedure tractable in data complexity, divergence should be detected after polynomially many steps. Example 22 illustrates that this can be achieved by analysing cyclic dependencies.

**Example 22.** Let \( \mathcal{P}_c \) contain facts \( A(0) \) and \( B(0) \), and rules \( A(m \rightarrow B(m) \) and \( B(m) \rightarrow A(m+1) \), where \( A \) and \( B \) are \text{max} predicates. Applying the first rule copies the value
of A into B, and applying the second rule increases the value of A; thus, both A and B diverge in T\textsubscript{\text{A}}\textsubscript{B}. The existence of a cyclic dependency between A and B, however, does not necessarily lead to divergence. Let program P\textsubscript{c} be obtained from P\textsubscript{c} by adding a max fact C(s) and replacing the first rule with A(m) ∧ C(n) ∧ (m ≤ n) → B(m). While a cyclic dependency between A and B still exists, the increase in the values of A and B is bounded by the value of C, which is independent of A or B; thus, neither A nor B diverge in T\textsubscript{\text{A}}\textsubscript{B}.

In the rest of this section, we extend <, +, and − to ∞ by defining k < ∞, ∞ + k = ∞, and ∞ − k = ∞ for each integer k. We formalise cyclic dependencies as follows.

**Definition 23.** For each n-ary limit predicate B and each tuple b of n − 1 objects, let v\textsubscript{BB} be a node unique for B and b. The value propagation graph of a semi-ground limit-linear program P and a pseudo-interpretation J is the directed weighted graph G\textsubscript{P} = (V, E, μ) defined as follows.

1. For each limit fact B(b, e) ∈ J, we have v\textsubscript{BB} ∈ V.
2. For each rule r ∈ P applicable to J with the head of the form A(a, s) where v\textsubscript{BB} ∈ V and each body atom B(b, m) of r where v\textsubscript{BB} ∈ V and variable m occurs in term s, we have v\textsubscript{BB}v\textsubscript{AB} ∈ E; such s is said to produce the edge (v\textsubscript{BB}, v\textsubscript{AB}) in E.
3. For each r ∈ P and each edge e = (v\textsubscript{BB}, v\textsubscript{AB}) ∈ E produced by r, δ\textsubscript{P}e(J) = opt(r, J) − e if B and A are max, e ≠ ∞ −opt(r, J) − e if B is max, A is min, e ≠ ∞ −opt(r, J) + e if B and A are min, e ≠ ∞ −opt(r, J) if e = ∞ where ϵ is such that B(b, ϵ) ∈ J. The weight of each edge e ∈ E is then given by μ(e) = max{δ\textsubscript{P}e(J) | r ∈ P produces e}.

A positive-weight cycle in G\textsubscript{P} is a cycle for which the sum of the weights of the contributing edges is greater than 0.

Intuitively, G\textsubscript{P} describes how, for each limit predicate B and objects b such that B(b, e) ∈ J, operator T\textsubscript{P} propagates ϵ to other facts. The presence of a node v\textsubscript{BB} in V indicates that B(b, ϵ) ∈ J holds for some ϵ ∈ Z ∪ {∞}; this ϵ can be uniquely identified given v\textsubscript{BB} and J. An edge e = (v\textsubscript{BB}, v\textsubscript{AB}) ∈ E indicates that at least one rule r ∈ P is applicable to J where h(r) = A(a, s), B(b, m) ∈ sb(r), and m occurs in s; moreover, applying r to J produces a fact A(a, ϵ′) where ϵ′ satisfies ϵ + μ(e) ≤ ϵ′ if both A and B are max predicates, and analogously for the other types of A and B. In other words, edge e indicates that the application of T\textsubscript{P} to J will propagate the value of v\textsubscript{BB} to v\textsubscript{AB} while increasing it by at least μ(e). Thus, presence of a positive-weight cycle in G\textsubscript{P} indicates that repeated rule applications might increment the values of all nodes on the cycle.

### 6.2 Stable Programs

As Example 22 shows, the presence of a positive-weight cycle in G\textsubscript{P} does not imply the divergence of all atoms corresponding to the nodes in the cycle. This is because the weight of such a cycle may decrease after certain rule applications and so it is no longer positive.

This motivates the stability condition, where edge weights in G\textsubscript{P} may only grow but never decrease with rule application. Hence, once the weight of a cycle becomes positive, it will remain positive and thus guarantee the divergence of all atoms corresponding to its nodes. Intuitively, P is stable if, whenever a rule r ∈ P is applicable to some J, rule r is also applicable to each J′ with larger limit values, and applying r to such J′ further increases the value of the head.

**Definition 24.** A semi-ground limit-linear program P is stable if, for all pseudo-interpretations J and J′ with J ⊆ J′, G\textsubscript{P} = (V, E, μ) and G\textsubscript{P}′ = (V, E, μ′) and each e ∈ E, let v\textsubscript{BB}, v\textsubscript{AB} ∈ E, such that v\textsubscript{BB}v\textsubscript{AB} ∈ E and v\textsubscript{BB}v\textsubscript{AB} = e in E. We say that (J, v\textsubscript{BB}) is stable if, for some limit fact B(b, e) in J, we have v\textsubscript{BB} ∈ V. Intuitively, P is stable if, whenever a rule r ∈ P is applicable to some J, rule r is also applicable to each J′ with larger limit values, and applying r to such J′ further increases the value of the head.

**Definition 25.** A semi-ground limit-linear program P is stable if, for all pseudo-interpretations J and J′ with J ⊆ J′, G\textsubscript{P} = (V, E, μ) and G\textsubscript{P}′ = (V, E, μ′) and each e ∈ E, let v\textsubscript{BB}, v\textsubscript{AB} ∈ E, such that v\textsubscript{BB}v\textsubscript{AB} ∈ E and v\textsubscript{BB}v\textsubscript{AB} = e in E. We say that (J, v\textsubscript{BB}) is stable if, for some limit fact B(b, e) in J, we have v\textsubscript{BB} ∈ V. Intuitively, P is stable if, whenever a rule r ∈ P is applicable to some J, rule r is also applicable to each J′ with larger limit values, and applying r to such J′ further increases the value of the head.

**Lemma 26.** Let G\textsubscript{P} and each node v\textsubscript{An} on a positive-weight cycle in G\textsubscript{P}, we have A(a, ∞) ∈ T\textsubscript{P}. Algorithm 1 uses this observation to deterministically compute the fixpoint of P. The algorithm iteratively applies T\textsubscript{P}; however, after each step, it computes the corresponding value propagation graph (line 4) and, for each A(a, e) where node v\textsubscript{BB} occurs on a positive-weight cycle (line 5), it replaces e with ∞ (line 6). By Lemma 26, this is sound. Moreover, since the algorithm repeatedly applies T\textsubscript{P}, it necessarily derives each fact from T\textsubscript{P} eventually. Finally, Lemma 27 shows that the algorithm terminates in time polynomial in the number of rules in a semi-ground program. Intuitively, the proof of the lemma shows that, without introducing a new edge or a new positive weight cycle in the value propagation graph, repeated application of T\textsubscript{P} necessarily converges in O(|P|^2) steps; moreover, the number of edges in G\textsubscript{P} is at most quadratic in |P|, and so a new edge or a new positive weight cycle can be introduced at most O(|P|^2) many times.

**Lemma 27.** When applied to a semi-ground stable program P, Algorithm 1 terminates after at most 8|P|^6 iterations of the loop in lines 2–8.

Lemmas 26 and 27 imply the following theorem.

**Theorem 28.** For a semi-ground stable program, D a dataset, and α a fact, Algorithm 1 decides P ∪ D |= α in time polynomial in |P ∪ D| and exponential in max\textsubscript{r∈P} |r|.
Algorithm 1 Entailment for Semi-Ground Stable Programs

**Input:** semi-ground stable program \( P \), fact \( \alpha \)

**Output:** true if \( P \models \alpha \)

1: \( J' := \emptyset \)
2: repeat
3: \( J := J' \)
4: \( G_P' := (V, E, \mu) \)
5: for each \( r \in A \in V \) in a positive-weight cycle in \( G_P' \) do
6: \( \text{replace } A(a, \ell) \text{ in } J \text{ with } A(a, \infty) \)
7: \( J' := T_P(J) \)
8: until \( J = J' \)
9: return true if \( J \models \alpha \) and false otherwise

Algorithm 1 combined with a semi-grounding preprocessing step provides an exponential time decision procedure for stable, limit-linear programs. This upper bound is tight since entailment in plain Datalog is already \( \text{ExpTime} \) hard in combined and \( \text{PTIME} \)-hard in data complexity.

**Theorem 29.** For \( P \) a stable program and \( \alpha \) a fact, checking \( P \models \alpha \) is \( \text{ExpTime} \)-complete in combined and \( \text{PTIME} \)-complete in data complexity.

### 6.3 Type-Consistent Programs

Unfortunately, the class of stable programs is not recognisable, which can again be shown by a reduction from Hilbert’s tenth problem.

**Proposition 30.** Checking stability of a limit-linear program \( P \) is undecidable.

We next provide a sufficient condition for stability that captures programs such as those in Examples 5–7. Intuitively, Definition 31 syntactically prevents certain harmful interactions. In the second rule of program \( P' \) from Example 22, numeric variable \( m \) occurs in a max atom and on the left-hand side of a comparison atom \((m \leq n)\); thus, if the rule is applicable for some value of \( m \), it is not necessarily applicable for each \( m' \geq m \), which breaks stability.

**Definition 31.** A semi-ground limit-linear rule \( r \) is type-consistent if
- each numeric term \( t \) in \( r \) is of the form \( k_0 + \sum_{i=1}^{n} k_i \times m_i \), where \( k_0 \) is an integer and each \( k_i \), \( 1 \leq i \leq n \), is a nonzero integer, called the coefficient of variable \( m_i \) in \( t \);
- if \( h(r) = A(a, s) \) is a limit atom, then each variable occurring in \( s \) with a positive (resp. negative) coefficient also occurs in a (unique) limit body atom or \( r \) that is of the same (resp. different) type (i.e., min vs. max) as \( h(r) \); and
- for each comparison \((s_1 < s_2)\) or \((s_1 \leq s_2)\) in \( r \), each variable occurring in \( s_1 \) with a positive (resp. negative) coefficient also occurs in a (unique) min (resp. max) body atom, and each variable occurring in \( s_2 \) with a positive (resp. negative) coefficient also occurs in a (unique) max (resp. min) body atom of \( r \).

A semi-ground limit-linear program is type-consistent if all of its rules are type-consistent. Moreover, a limit-linear program \( P \) is type-consistent if the program obtained by first semi-grounding \( P \) and then simplifying all numeric terms as much as possible is type-consistent.

The first condition of Definition 31 ensures that each variable occurring in a numeric term contributes to the value of the term. For example, it disallows terms such as \( 0 \cdot x \) and \( x - x \), since a rule with such a term in the head may violate the second condition. Moreover, the second condition of Definition 31 ensures that, if the value of a numeric variable \( x \) occurring in the head ‘increases’ w.r.t. the type of the body atom introducing \( x \) (i.e., \( x \) increases if it occurs in a max body atom and decreases otherwise), then so does the value of the numeric term in the head; this is essential for the first condition of stability (cf. Definition 24). Finally, the third condition of Definition 31 ensures that comparisons cannot be invalidated by ‘increasing’ the values of the variables involved, which is required for both conditions of stability.

Type consistency is a purely syntactic condition that can be checked by looking at one rule and one atom at a time. Hence, checking type consistency is feasible in \( \text{LOGSPACE} \).

**Proposition 32.** Each type-consistent limit-linear program is stable.

**Proposition 33.** Checking whether a limit-linear program is type-consistent can be accomplished in \( \text{LOGSPACE} \).

### 7 Conclusion and Future Work

We have introduced several decidable/tractable fragments of Datalog with integer arithmetic, thus obtaining a sound theoretical foundation for declarative data analysis. We see many challenges for future work. First, our formalism should be extended with aggregate functions. While certain forms of aggregation can be simulated by iterating over the object domain, as in our examples in Section 3, such a solution may be too cumbersome for practical use, and it relies on the existence of a linear order over the object domain, which is a strong theoretical assumption. Explicit support for aggregation would allow us to formulate tasks such as the ones in Section 3 more intuitively and without relying on the ordering assumption. Second, it is unclear whether integer constraint solving is strictly needed in Step 7 of Algorithm 1: it may be possible to exploit stability of \( P \) to compute \( T_P(J) \) more efficiently. Third, we shall implement our algorithm and apply it to practical data analysis problems. Fourth, it would be interesting to establish connections between our results and existing work on data-aware artefact systems [Damaggio et al., 2012; Koutsos and Vianu, 2017], which faces similar undecidability issues in a different formal setting.

**Acknowledgments**

We thank Christoph Haase for explaining to us his results on Presburger arithmetic and semi-linear sets, which were key to some of our technical results. Our work has also benefited from discussions with Michael Benedikt. This research was supported by the Royal Society and the EPSRC projects DBOnto, MaSI, and ED³.

**References**

[Alvaro et al., 2010] Peter Alvaro, Tyson Condie, Neil Conway, Khaled Elmeleegy, Joseph M. Hellerstein, and Rus-


