

Everything That is Not Important: Negative Databases

I. Introduction

One of the tasks of the immune system is to protect the organism from disease—to detect pathogens and clear them from the body. The challenge begins with detection; how does the immune system “know” that the agent it encountered is a pathogen? How does it identify something it has never seen before? This difficulty is compounded by the fact that pathogens evolve quickly, even within the lifetime of an individual, so that things observed in the past may have significantly changed in the future.

There are several theories of how this occurs in the vertebrate immune system; predominantly, Danger theory [1] and the Self-nonsel self discrimination theory [2]. In the latter, the universe is divided in two categories: self and non-self. Self labels all the habitual components (peptide sequences) of the organism and nonself everything else. The immune cells responsible for detection are created with the mandate: bind, but do *not* bind self. The immune cells that roam the body were selected for, among other things, their inability to bind self peptides. Therefore, anything that is bound by one of these cells is nonself and possibly a pathogen.

According to the Self-nonsel self discrimination theory, from a data representation point of view, the collection of all immune cells comprises a distributed model of what self is not; if the model is good enough, then it is also a

model of self—the collection of all immune cells define what self is by individually specifying what it is not.

The work reviewed in this paper is predominantly inspired by this image and asks whether the same principle can be applied to a database—here viewed as a list of strings—and how the properties of representing a database negatively can be leveraged for its security.

Section II describes negative databases in detail, including their representation, and some of the algorithms for creating them. Section III discusses an interesting property of this representation and how it may be exploited for security purposes. Section IV explains how a negative database can be manipulated to alter its contents and Section V summarizes how to perform relational algebra operations on a negative database. Finally, Section VI points to some future areas of research.

II. Negative Databases

The data of interest, referred to as a positive database and denoted DB , is a set of binary strings of length l , i.e., $DB \subseteq \{0, 1\}^l$. For example, a list of credit-card numbers in their binary representation. A negative database, NDB , is a condensed representation of the complement of that set—all possible 16 digit numbers except the ones in DB . Compaction is achieved by defining negative databases over $\{0, 1, *\}^l$, where $*$ is known as the

“wild-card” or “don’t-care” symbol; its purpose is to represent a 0 and 1 at a given position—positions bearing a 0 or 1 are called *specified positions*. In this way, a single string with $n*$ symbols compactly stands in for 2^n binary strings (see Fig. 1).

A binary string is said to be in NDB if at least one NDB entry matches it. A match between two strings is determined by comparing the strings position by position. The positions match if their values are the same or if either one has a $*$, the strings match if all their positions match. Conversely, a binary string is not in NDB and, thus, in DB , if no NDB entry matches it.

There are several algorithms for creating a negative database (NDB) given as input DB [3]–[6], the main distinction between them is the size of the resulting NDB and the ease with which DB can be retrieved from NDB (more on this in Section III). The first algorithm developed for this purpose, proof that negative databases can be created in polynomial time, is the Prefix Algorithm [3], [7]. This algorithm creates a NDB in $O(l|DB|)$ time with $O(l|DB|)$ entries that exactly depict $\{0, 1\}^l \setminus DB$. A salient characteristic of these NDB s is that every string in $\{0, 1\}^l \setminus DB$ is matched by a single negative record and that some strings in DB can be retrieved using



Digital Object Identifier 10.1109/MCI.2008.919079

¹The symbol \setminus denotes set difference.

only a small subset of *NDB*—all *DB* can be recovered in polynomial time using all of *NDB*. The Randomized Algorithms presented in [3], [7] refine the Prefix Algorithm to rid it of these properties by reducing the number of specified bits per record and allowing a single binary string to be matched by several *NDB* entries; as a consequence, all of *NDB* must be consulted to derive a single *DB* entry (see the example in Fig. 1).

III. Security Using Negative Databases

Negative databases, by definition, store everything except the data of interest. An important difference with their positive counterparts is that individual entries, or small subsets of entries, when examined, provide limited information about the data (see Chap. 5 of Ref. [3])². This feature suggests applications for security and privacy in databases—consider the limiting case of distributing the *n* entries of a *NDB* among *n* parties, there is little an individual party can learn about the data without consulting with the rest.

There is another property, however, that follows from the 0,1,* representation of negative databases which allows a negative database to be held securely in its entirety. Retrieving *DB* from *NDB* is NP-hard, even determining whether a *NDB* negatively represents the empty set is NP-complete. Reference [7] shows a polynomial time transformation from an instance of SAT (boolean formula satisfiability) to a negative database (Fig. 2 shows an example of the mapping). The interest in these hard-to-reverse negative databases is that, despite the fact that *DB* cannot be easily deduced, they can still be used to efficiently determine the presence or absence of specific strings.

Imagine a goods company that wishes to keep a list of risky credit-card numbers: credit-cards that have been

<i>DB</i>	Prefix <i>NDB</i>	Randomized <i>NDB</i>
0011	10**	10**
0101	000*	*00*
1100	011*	011*
1111	0010	0*10
	0100	0*00
	1101	1*01
	1110	**10
		*110
		*010

FIGURE 1 Column 1 gives an example *DB*, column 2 shows an *NDB* created by the prefix algorithm, and column 3 gives an example *NDB* generated by a randomized algorithm.

involved in suspicious activities. A hard-to-reverse negative database can be created such that the ability to retrieve individual numbers is hindered, preventing a malicious party from stealing any numbers, and the capability to validate specific numbers is preserved.

Section IV describes a set of operations that can be performed on a negative databases that lends a great amount of flexibility to this security scheme (far beyond what hashed lists provide). Some applications of hard-to-reverse negative databases, such as digital credentials [6], are actively being investigated.

Even though it was shown that negative databases are hard to reverse in general, many instances are easy; in fact, the prefix algorithm mentioned in [7] always outputs easily reversible *NDBs*. The challenge is to develop algorithms that generate hard-to-reverse negative databases on average. The work in [5] accomplished this by adapting an algorithm used to create hard satisfiability formulas. The resulting negative databases, however, represent only positive sets of size one or zero, and a collection of them must be used to represent larger databases (some applications of this arrangement are discussed in the same paper). In [8] hard-to-reverse negative databases are introduced that rely on cryptographic primitives, rather than on the NP-hardness of reversing them, for their security. These databases differ from the ones discussed in the rest of this review in that they are not string

based, and are, thus, not amenable to the relational algebra described in Section V; they do, nevertheless, share the functionality of concealing the contents of the database while allowing simple membership queries.

IV. Operators

In the previous section we saw how a negative database is created given *DB*, but what if *DB* changes over time and its changes are to be reflected in *NDB*? In [4] three basic operations are presented: Insert, Delete, and Morph. Insert takes as input a negative database

NDB and string *x* in $\{0, 1\}^l$ and outputs a *NDB'* that matches every string matched by *NDB* and every string matched by *x*. Delete takes the same input as Insert but outputs a *NDB'* that matches every string matched by *NDB* minus those matched by *x*. In effect, if a record is to be included in *DB* it must be removed from *NDB* using the Delete operation; similarly, inserting a string into *NDB* removes it from *DB*. These operations are interesting because they function without reversing *NDB*. This is significant in light of the fact that some *NDBs* cannot be reversed efficiently.

Consider the credit-card watchlist discussed before. Suppose that the department that holds the negative database discovers that some number in the list is not delinquent. The Insert operation allows that party to remove the entry without finding out anything about any other record in the database. Likewise, if a new number comes along that is regarded as suspicious, it can be safely included in the watchlist via the Delete operation.

The Morph operation takes as input a *NDB* and outputs a *NDB'* that matches exactly the same set of binary strings; however, *NDB* and *NDB'* are different—some *NDB* records are not in *NDB'* and vice-versa. This illustrates the ability of the negative database scheme to have different representations for the same data. A property that makes it difficult to determine if two negative databases are equivalent (match exactly

²I refer to specially crafted *NDBs*, not the ones produced by the Prefix algorithm.

the same set of binary strings) without reversing them. Uses of the Morph operation include removing superfluous entries from the negative database and obscuring the history of operations performed on it. A watchlist application can benefit from continually changing the structure of its negative database and preventing onlookers from knowing whether its contents have been modified.

V. Negative Algebra

Adding and removing entries from a database are special cases of uniting and subtracting data sets. Union and Set-Difference are two of the five operations described by Codd in [9], [10] as forming the basis for a database query language. Reference [11] explores how these five operations: Union, Select, Cross-product, Project and Set-Difference can be performed using negative databases. When negative databases are used as surrogates for the data, they are manipulated so that the resulting *NDB*, when reversed, agrees with the result of performing the corresponding operation on their positive counterparts. For instance, the Negative-Union operation takes as input *NDB*₁ and *NDB*₂ and out-

Boolean Formula	<i>NDB</i>
$(x_1 \text{ or } \bar{x}_3) \text{ and}$	0*1**
$(\bar{x}_2 \text{ or } \bar{x}_5) \text{ and}$	*1**1
$(x_1 \text{ or } \bar{x}_3 \text{ or } x_5) \text{ and}$	0*1*0
$(x_2 \text{ or } \bar{x}_3 \text{ or } \bar{x}_4) \text{ and}$	*011*
$(\bar{x}_1 \text{ or } x_2 \text{ or } \bar{x}_4 \text{ or } x_5)$	10*10

FIGURE 2 Mapping SAT to *NDB*: A Boolean formula written in conjunctive normal form is transformed into a negative database by translating each clause into a negative record. Each record has a position for every variable in the formula (position 1 to variable x_1 , etc.) and its value is 1 if the variable appears negated in the clause, 0 if its un-negated, and * otherwise. In this example the assignment $\{x_1 = \text{FALSE}, x_2 = \text{TRUE}, x_3 = \text{TRUE}, x_4 = \text{TRUE}, x_5 = \text{FALSE}\}$ does not satisfy the formula and the corresponding string 01110 is matched by *NDB*, while the assignment $\{x_1 = \text{TRUE}, x_2 = \text{FALSE}, x_3 = \text{TRUE}, x_4 = \text{FALSE}, x_5 = \text{FALSE}\}$ satisfies it and the string 10100 is not in *NDB*.

<i>DB</i> ₁	<i>DB</i> ₂	<i>DB</i> ₁ ∩ <i>DB</i> ₂
111111	101101	111111
010111	111111
....
<i>NDB</i> ₁	<i>NDB</i> ₂	<i>NDB</i> ₁ ∩̄ <i>NDB</i> ₂
1**01*	00***1	1**01*
1**10*	0**11*	1**10*
....	00***1
....	0**11*
....

FIGURE 3 Intersection of two databases *DB*₁ and *DB*₂ and the equivalent operation (Negative-Intersection) on their corresponding negative databases. Only a subset of each database is shown.

<i>DB</i> ₁	<i>DB</i> ₂	<i>DB</i> ₃ = <i>DB</i> ₁ ⋈ <i>DB</i> ₂
111111	010111-00010	010111-00010
010111	110011-00100	111111-00101
....	111111-00101
....
<i>NDB</i> ₁	<i>NDB</i> ₂	<i>NDB</i> ₃ = <i>NDB</i> ₁ ⋈̄ <i>NDB</i> ₂
1**01*	1***0*-0**1*	1**01*-*****
1**10*	*11**1-1*1**	1**10*-*****
....	00*1**-*1*11	1***0*-0**1*
....	01*0**-*011*	*11**1-1*1**
....	00*1**-*1*11
....	01*0**-*011*
....

FIGURE 4 The Join of *DB*₁ and *DB*₂ and the equivalent operation on their negative databases (only a subset of each database is shown). Notice how the Negative-Join of *NDB*₁ and *NDB*₂ matches 110011-00100, which is not in the Join of *DB*₁ and *DB*₂. The fields credit-card number and balance are shown separated by a dash for exposition purposes only.

puts a negative database *NDB*₃ that matches exactly the same strings as a negative database created from the union of *DB*₁ and *DB*₂, i.e., the reverse of *NDB*₃ is *DB*₁ ∪ *DB*₂.

Consider the credit-card number watchlist alluded to in the previous sections. Suppose it is desired to screen only those credit-cards that have also been flagged by a competitor company. Appending the two negative databases together results in a negative database of the intersection of the underlying positive lists. Figure 3 shows an example of this operation. For presentation purposes, all examples in the remainder of the paper have credit-card numbers of only six bits in length.

Now suppose the watchlist, *DB*₁, must be restricted to only the numbers held by another department and each credit-card augmented with its balance (*DB*₂ has the tuples <credit-card number, balance>). The Negative-Join of *NDB*₁ and *NDB*₂ accomplishes this (see Fig. 4). The algorithm is presented below. It is a simplification of the Negative-Join algorithm introduced in [11]; here every position in *NDB*₁ is used as part of the Join condition and all such positions are contiguous in *NDB*₂.

Negative-Join (⋈̄)

Input: *NDB*₁, *NDB*₂ of strings of length n and m respectively

Output: A negative database *NDB*₃ of $n + m$ -length strings that matches every string except the Join of *DB*₁ with *DB*₂

1. Initialize *NDB*₃ to the empty set
2. For each string $x \in \text{NDB}_1$:
 - (a) Create a string z with x as its prefix and m^* as its suffix
 - (b) Append z to *NDB*₃
3. For each string $x \in \text{NDB}_2$:
 - (a) Create a string z with n^* as prefix and x as suffix
 - (b) Append z to *NDB*₃

Suppose further that we wish to select all the entries with a balance greater than or equal to 3,000 (the numbers in the database are in thousands and the binary number for 3 is 11) from the negative database of credit-card numbers and balance tuples. The following paraphrases the general algorithm given in [11] for this operation. Figure 5 shows an example.

Negative-Greater-than-Equal ($\bar{\geq}$)

Input: A negative database NDB of l -length strings, a selection field, and a value ν . ν is assumed to have the same number of bits as the selection field, with the most significant bits zero filled accordingly

Output: A negative database NDB' that matches the same binary strings as NDB except those whose values in the selection field are less than ν

1. For each bit i in ν with a value of 1:
 - (a) Create a string x of length l with the corresponding position in the selection field set to 0, all positions to its left (more significant positions) set to 0 where ν is 0, and all other positions set to *
 - (b) Append x to NDB

A consequence of working with the negative image of a set is that the complexity of some operations is reversed. For example, while the Cross-product or Join of two databases is, in general, quadratic, the negative Cross-product and Negative-Join are linear; and while the union of two positive databases is linear, the Negative-Union is quadratic. The complexity, however, of Set-difference and Project is far greater when negative databases are used. It was shown in [11] that if the negative databases are hard-to-reverse the said operations are NP-complete—if this were not the case, projecting NDB onto any bit position would quickly reveal if DB is empty and violate its security.

VI. Negative Databases in the Future

The introduction of negative databases for security is fairly recent, and there is

$\sigma_{Balance \geq 00011} (DB_3)$	$\bar{\sigma}_{Balance \geq 00101} (NDB_3)$
111111-00101	1**01*-****
....	1**10*-****
	1***0*-0**1*
	*11**1-1*1**
	00*1**-*1*11
	01*0**-*011*
	*****-0000*
	*****-000*0

FIGURE 5 Selection of records with a balance of more than or equal to 3,000 ($\nu = 00011$) from DB_3 (see Fig. 4), and the equivalent operation (Negative-Greater-than-Equal) performed on NDB_3 . NDB_3 was modified by adding two entries which are shown in bold. Notice that NDB_3 matches any binary string with a balance of 0, 1 or 2 (binary 00000, 00001, 00010).

still a significant amount of work ahead. For example, algorithms for efficiently creating hard-to-reverse, string based negative databases that represent thousands or millions of positive entries will be needed for future applications. Also, among the features that make this scheme attractive is the wealth of operations that can be applied to the data without knowing what the data is. The operators of Section IV are examples of this, but future applications will require expanding this toolbox. Moreover, the algorithms for manipulating negative databases do not have efficiency in mind. More efficient versions are warranted. Other operations such as Morph suggest that the history of operations on a negative database can be erased; this is a very interesting and useful feature that needs further exploration.

Most of the work on negative databases relating to security is based on hard-to-reverse negative databases; however, there are other properties of storing the negative image of a data set that can be of consequence for securing data. In particular, the manner in which information about the positive data set is distributed among the negative entries—examining a small subset of NDB provides limited insight as to the identity of any DB entry; all NDB must be held to

recover DB ; and all NDB must be parsed to be certain a given string is in DB . This feature hints to a natural way to share a secret and requires further study. Further study is also needed to unearth other properties of this scheme and the applications for which it might be uniquely suited (see [12] for a very different use of the same concept).

References

- [1] P. Matzinger, "The danger model: A renewed sense of self," *Science*, vol. 296, no. 5566, pp. 301–305, 2002.
- [2] C.A. Janeway, P. Travers, and M. Walport, *Immunobiology*, Garland Publishing, 1999.
- [3] F. Esponda, *Negative Representations of Information*, Ph.D. thesis, University of New Mexico, 2005.
- [4] F. Esponda, E.S. Ackley, S. Forrest, and P. Helman, "On-line negative databases (with experimental results)," *International Journal of Unconventional Computing*, vol. 1, no. 3, pp. 201–220, 2005.
- [5] F. Esponda, E.S. Ackley, P. Helman, H. Jia, and S. Forrest, "Protecting data privacy through hard-to-reverse negative databases," *International Journal of Information Security*, vol. 6, no. 6, pp. 403–416, Oct. 2007.
- [6] M. de Mare and R. Wright Secure, "Set membership using 3sat," in *Proceedings of the Eighth International Conference on Information and Communication Security (ICICS '06)*, 2006.
- [7] F. Esponda, S. Forrest, and P. Helman, "Enhancing privacy through negative representations of data," *Technical report*, University of New Mexico, 2004.
- [8] G. Danezis, G. Diaz, S. Faust, E. Kasper, C. Troncoso C., and B. Preneel, "Efficient negative databases from cryptographic hash functions," in *Information Security Conference, Springer LNCS*, Ed., vol. 4779, pp. 423–436, 2007.
- [9] E.F. Codd, "A relational model for large shared data banks," *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, June 1970.
- [10] E.F. Codd, "Recent investigations in relational database systems," in *Proceedings of the IFIP Congress*, 1974.
- [11] F. Esponda, E.D. Trias, E.S. Ackley, and S. Forrest, "A relational algebra for negative databases," Technical report TR-CS-2007-18, University of New Mexico, 2007.
- [12] F. Esponda, "Negative Surveys," ArXiv Mathematics e-prints: math/0608176, Aug. 2006.