Delay-Insensitive Codes—An Overview

TOM VERHOEFF

Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
E-Mail address: mcvax.UUCP!eutrc3!wstomv

January 1987

Abstract. The problem of delay-insensitive data communication is described. The notion of delay-insensitive code is defined, giving precise conditions under which delay-insensitive data communication is feasible. Examples of these codes are presented and analyzed. It appears that delay-insensitive codes are equivalent with antichains in partially ordered sets and with all unidirectional error-detecting codes.

Keywords: Block codes — Delay-insensitive communication — Parallel data transmission — Self-timed systems

CONTENTS

1 PROBLEM DESCRIPTION

Consider a communication channel that consists of a number of tracks running from sender to receiver. The sender can use this channel to convey information to the receiver by rolling marbles along some of the tracks. The amount of time it takes a marble to travel from sender to receiver, however, is unknown and may even vary from track to track or from roll to roll. But it is nonnegative and finite, that is, marbles eventually do arrive after they have been rolled onto a track. The problem is to design a communication scheme for sender and receiver that is insensitive to variability in propagation delays.

The above problem description has not addressed certain issues that nevertheless are relevant. Let us narrow it down a bit. (a) One may assume that sender and receiver are in agreement about the identity of the tracks: the tracks are labeled. (b) To prevent the smashing of marbles the sender should roll at most one marble onto each track for the transmission of a single message. (c) When sequences of messages are to be transmitted, there arises the need for acknowledgment of the reception of a message. It can be implemented by a feedback track from receiver to sender. Acknowledgment is not considered to be part of the problem.

An example may further clarify the problem. Suppose there is the need to communicate messages taken from a set of n messages, numbered 1 through n. We use a channel with n tracks, labeled from 1 to n. Transmission of message with number k is accomplished by rolling a marble over the track labeled k, and no marbles over the other tracks. The receiver waits till he sees a marble, notices along which track it arrived, and knows that he has received the complete message, since there cannot be any other marbles on their way. Because of its resemblance to the one-hot assignment (Hollaar 1982), this very simple communication scheme is called the *One-Hot code*.

The One-Hot code is inefficient for large message sets, since it requires too many tracks. This is underscored by the following analysis. Given a channel with n tracks the sender has a choice of 2^n marble patterns: for each track he is free to roll a marble or not. To use only n of these patterns seems a big waste. Can you do better? Notice, however, that not all patterns can be used together. For instance, the pattern in which no marbles are rolled is useless in the presence of other patterns. The receiver would not be able to decide whether the empty pattern was sent. Initially he has received no marbles (the empty pattern), but some marbles may still be on their way transmitting a different pattern. How long should the receiver wait before deciding that the empty pattern was sent? In the next section precise conditions for delay-insensitive communication are formulated.

Originally this problem arose in the context of electronically implemented communication channels. A track corresponds to a signal wire that has two stable states, which will be called *Zero* and *One*. A marble on a track corresponds to a transition on a wire, i.e., to a change of state of a wire. The propagation speed of transitions is not always under complete control of the designer. This is especially the case for VLSI circuits (Seitz 1980, Udding 1986). The current trend is toward self-timed systems, where as little as possible is assumed about propagation delays. Hence, the interest in delay-insensitive communication schemes. It is also from this context that the requirement of ‘‘at most one marble on each track’’ comes. Multiple transitions on a wire can influence each other adversely, causing *transmission interference*.

If the One-Hot code is used on signal wires, as suggested above, then all combinations of wire states are feasible (as opposed to transition patterns). To limit the number of wire state patterns that may occur, one often resorts to *Return-to-Zero* or *Four-Phase* signaling (Seitz 1980). In this scheme all wires are initially in the *Zero* state, and each message is sent twice. The sole purpose of repeating a message is to bring the wires back to their *Zero* state. The name Four-Phase signaling derives from the fact that in practice each transmission is accompanied by an additional acknowledge phase, resulting in four phases per effective message.

We prefer the metaphor of marbles rolling on tracks because it avoids the possible confusion between transition (or marble) patterns and wire state patterns.

2 REQUIREMENTS FOR DELAY-INSENSITIVE CODES

In this section we formally define the notion of delay-insensitive code. It expresses precisely under what conditions a code can be used for delay-insensitive communication. We also introduce some additional terminology and present three code construction techniques. Finally, we mention some practically desirable properties of codes.

A *code* is a pair (I, C) where

I is a finite set , (track indices)

C is a set of subsets of I . (code words)

The size of I is called the code’s *length* (it is the number of tracks in the channel), and the size of C is called the code's *size* (it is the number of messages in the message set that is coded). Notice that the code's size is at most two to the power the code's length.

A code word is an element of C and it indicates along which tracks marbles will be sent for the transmission of the corresponding message. The size of a code word x is also called its *weight* (in marble units) and it is denoted by $\mathbf{w}(x)$.

Sometimes it is convenient to represent subsets of I by their characteristic function. The characteristic function of x, $x \subseteq I$, is a mapping from I into $\{0, 1\}$, taking on the value 1 in the elements of x and 0 otherwise. If the elements of I are somehow ordered, then a code could be represented by a set of fixed-length bit vectors (words over $\{0, 1\}$). There is an obvious notion of code isomorphism involved, but we shall not pursue it explicitly here.

Terminology reminding of any of these representations may be used interchangeably. For instance, the notion of code length and code word derives from the bit vector representation. Notice that in the characteristic function and bit vector representations there is a slight asymmetry between 0 and 1: a 1 corresponds to a track with marble, a 0 to an empty track.

A code (I, C) is called *delay-insensitive* when

$$(\mathbf{A}\, x, y : x \in C \wedge y \in C \wedge x \subseteq y : x = y)\ , \tag{DI}$$

that is, when no code word is contained in another code word. This exactly captures the conditions under which the receiver is able to decide unambiguously when he has completely received the code word transmitted by the sender. If C would contain two (hence, distinct) code words x and y with $x \subseteq y$, then the receiver might (temporarily) see x when either x or y was sent (viz. if the tracks $y \backslash x$ are slow). But the receiver cannot know which is the case at the time he sees x. If the code is DI then the complete-reception-detection strategy for the receiver is simply to wait until he sees a code word, since no more marbles can be on their way after that. Reception detection boils down to code membership test.

The One-Hot code from Section 1 consists of all singletons over I, that is, of all words of unit weight. Its size equals its length and it is obviously delay-insensitive (satisfying DI).

It appears that DI codes are equivalent to antichains in a partially ordered set (cf. Section 3) and to all unidirectional error-detecting (AUED) codes (Bose and Rao 1982, Theorem 5). All DI codes that we present were already known as AUED codes, although most of them were rediscovered independently.

Code (I, C) is called *subcode* of code (I, D) when C is a subset of D. Obviously, a subcode of a DI code is itself a DI code. Although this may seem to be a useless construction, there are situations where subcodes are preferable to their parents (see below and Section 5).

The *complement* of code (I, C) is the code $(I, \overline{C})$, where $\overline{C}$ is defined by

$$\overline{C} = \{ I \backslash x \mid x \in C \} . \qquad \text{(complement)}$$

When I is understood, $I \backslash x$ is sometimes written as $\overline{x}$. Note that complementing $(I, \overline{C})$ results in (I, C) again (complementation is idempotent). From

$$x \subseteq y \equiv \overline{y} \subseteq \overline{x} \qquad (2.1)$$

it follows that a code is DI if and only if its complement is DI. In the bit vector representation complementation corresponds to replacing all 0's and 1's in code words by 1's and 0's respectively. Hence, the asymmetry between 0 and 1, as emphasized above, is irrelevant for DI codes.

The complemented code has the same length and size as the original code. The complement of the One-Hot code might be called the *One-Cold code*. It consists of all words with a single 0.

The *concatenation* (sometimes also called *direct sum*) of codes (I, C) and (J, D), with $I \cap J = \emptyset$, is the code $(I \cup J, CD)$, where CD is defined by

$$CD = \{ x \cup y \mid x \in C \wedge y \in D \} . \qquad \text{(concatenation)}$$

Concatenation is symmetric and associative. The length of the concatenated code is the sum of the lengths of the composing codes, its size is the product of the sizes of the components. A small exercise in set calculus shows that if both codes are DI then so is their concatenation.

The concatenation of n (disjoint) One-Hot codes of length two yields a DI code of length $2n$ and size 2^n. This code is known as the *Double-Rail code* (Seitz 1980). Notice that all code words of the Double-Rail code have the same weight, viz. n. For $n > 2$ it gives a (substantial) size improvement over the One-Hot code of the same length. For instance, over a channel with 20 tracks the One-Hot code can transmit only 20 different messages, compared to 1024 for the Double-Rail code. There are, however, about one million patterns available on 20 tracks. The largest DI code of length 20 utilizes 184,756 of these patterns, it will appear below as Sperner code.

Before presenting and analyzing some other DI codes it is important to realize that not just any DI code is practically useful. The only aspect we have emphasized so far is large code size for a given code length. Measures for this aspect are the code's *rate* R and its *redundancy* r as defined by

$$R = \frac{\log M}{n} \ , \qquad \text{(rate)}$$

$$r = n - \log M \ , \qquad \text{(redundancy)}$$

where n is the code's length, M is the code's size, and the logarithms have base 2. Note that $0 \le R \le 1$ and $0 \le r \le n$ hold. Good codes have a rate close to 1 and a small redundancy.

Equally important aspects are ease of encoding (by sender), membership test, and decoding (the latter two by receiver). Encoding and decoding on their own are not well-defined concepts. They heavily depend on the message space to be coded. We will consider the message space to consist of all bit vectors of a certain length, because that seems to be the most prevalent practical case. Ease of encoding and decoding is now reflected by the complexity of the mapping from message vectors to code vectors and vice versa.

The One-Hot code is very simple as far as membership test is concerned, since all code words have unit weight. Encoding and decoding of One-Hot codes require somewhat more complicated circuitry, unless its length is very small. For the length-two One-Hot code simply map the message vectors 0 and 1 onto the code vectors 10 and 01 respectively.

By the way, note that the membership test need only work under the circumstance that the sender transmitted a code word and not something else. Hence, for a subcode the same membership test can be used as for its parent, although this need not always result in the most efficient membership test. What makes a subcode interesting, however, is that it may allow a better encoding/decoding scheme than its parent code.

The Double-Rail code has a very simple encoding/decoding scheme and membership test, because it is the concatenation of length-two One-Hot codes. The membership test consists of the conjunction of membership tests for the One-Hot components. Encoding is accomplished by encoding each message bit separately via its One-Hot encoding, and decoding works similarly.

3 k-OUT-OF-n CODES AND SPERNER CODES

A *k-out-of-n code* is a code of length n that consists of all words of weight k, thereby generalizing the One-Hot code (which is obtained by taking $k=1$). Obviously this is a DI code: if code word x is a subset of code word y then x equals y, since x and y are known to have the same weight.

The size of the k-out-of-n code is the binomial coefficient n **choose** $k = n!/(k!(n-k)!)$. The membership test is straightforward: $\mathbf{w}(x)=k$. Except for $k=1$ and $k=n-1$, no simple encoding scheme is known. The complement of the k-out-of-n code is an $(n-k)$-out-of-n code.

A DI code is called *maximal* if it cannot be extended without violating the DI property. All k-out-of-n codes are maximal in this sense. A word z of length n that is not in the k-out-of-n code has a weight different from k. Therefore, either it contains a code word (if $\mathbf{w}(z)>k$) or is contained in one (if $\mathbf{w}(z)<k$). Hence, the word z cannot be added to the code without violating the DI property. Notice that the Double-Rail code of length $2n$ is a subcode of the n-out-of-$2n$ code. This Double-Rail code is not maximal for $n>1$.

A DI code is called *optimal* if there does not exist a DI code of the same length containing more code words. Sperner (1928) has completely characterized the optimal DI codes. They turn out to be the (n **div** 2)-out-of-n codes and their complements. We therefore refer to these codes also as *Sperner codes*. The Sperner code of length 20 has size 20 **choose** $10=184{,}756$.

Applying Stirling's approximation,

$$n! \sim n^n e^{-n}\sqrt{2\pi n} \ , \qquad \text{(Stirling)}$$

one obtains

$$2n \textbf{ choose } n = \frac{(2n)!}{(n!)^2} \sim \frac{2^{2n}}{\sqrt{\pi n}} \ . \qquad (3.1)$$

This shows that the even-length Sperner codes utilize a relatively large number of the 2^{2n} available patterns. As is readily verified, the size of an odd-length Sperner code is half that of the Sperner code of length one more. The redundancy of a Sperner code of length n is, therefore, approximately

$$\tfrac{1}{2}\log n + \tfrac{1}{2}\log \pi/2 \ . \qquad (3.2)$$

The remainder of this section will be devoted to proving that the Sperner codes are the optimal DI codes. The proof we present is not completely new, but in spite of its simplicity it does not seem to be well-known. We also refer the reader to Greene and Kleitman (1978) for alternative proofs. The proof given by Freiman (1962) is essentially the same as that of Sperner (1928).

Our proof will consist of two parts. First, we show that the Sperner codes are optimal. Then we show that there are no other optimal codes. The proof is based on the theory of *partially ordered sets* (posets). Recall that a poset is a set together with an *order relation*, the latter being a reflexive, antisymmetric, and transitive relation. We start with some definitions.

Let I be a finite set and let P be the power set of I, that is, P consists of all subsets of I. The pair $(P, \subseteq)$ is the poset that will concern us here. Set inclusion is the order relation. *Comparability* (under this order relation) of x and y, $x \in P$ and $y \in P$, is denoted by $x \mathbf{li}\, y$:

$$x \mathbf{li}\, y \equiv x \subseteq y \vee y \subseteq x \ . \qquad \text{(comparable)}$$

Code (I, C) is a DI code, if and only if C is a subset of P and any two (distinct) elements are incomparable. Such a totally unordered set is called an *antichain* in poset terminology:

$$(\mathbf{A}\, x, y : x \in C \wedge y \in C \wedge x \neq y : \neg(x \mathbf{li}\, y)) \qquad \text{(antichain)}$$

Delay-insensitive codes and antichains in our poset are equivalent. In the study of antichains it is helpful to understand the totally ordered sets, or chains, of our poset. The set L, $L \subseteq P$, is called a *chain* (in the poset) when

$$(\mathbf{A}\, x, y : x \in L \wedge y \in L : x \mathbf{li}\, y) \ . \qquad \text{(chain)}$$

Furthermore, L is called a *line* (of the poset) when it is a maximal chain, that is, when L is a chain to which no element can be added without violating the chain condition. The notion of a chain also has a meaning in the original problem context. Consider the intermediate results that the receiver observes, while he is waiting for the arrival of a complete message: these words form a chain.

Define *Lines* as the set of all lines in our poset. $Cone(x)$ denotes the set of all lines through x, for $x \in P$:

$$Cone(x) = \{ L \in Lines \mid x \in L \} \ . \qquad \text{(cone)}$$

Let us now do some analyzing and counting. The number of elements in a set V will be denoted by $\#V$. We shall, however, continue to write $\mathbf{w}(x)$ instead of $\#x$ if $x \in P$. Define the numbers n, m, and

$\overline{m}$ by

$$n = \#I \ , \quad m = n \ \mathbf{div} \ 2 \ , \quad \overline{m} = n - m \ . \tag{3.3}$$

The Sperner codes over I are the m-out-of-n and $\overline{m}$-out-of-n codes, both have size n **choose** m.

Each line of our poset contains exactly one element of every weight in the range from 0 to n. Each line, therefore, has $n+1$ elements and contains $\varnothing$ and I, the only elements of weight 0 and n respectively. The other elements, between $\varnothing$ and I, can be chosen in $n!$ ways. Hence, we have

$$\#Lines = n! \ . \tag{3.4}$$

To count the number of elements in $Cone(x)$ for some $x \in P$, observe that one can choose the elements between $\varnothing$ and x in $\mathbf{w}(x)!$ ways, and those between x and I in $(n - \mathbf{w}(x))!$ ways. We therefore have

$$\#Cone(x) \ = \mathbf{w}(x)!\,(n - \mathbf{w}(x))! \ = \#Lines/(n \ \mathbf{choose} \ \mathbf{w}(x)) \ . \tag{3.5}$$

By the way, notice that $Cone(\varnothing) = Cone(I) = Lines$. Let c denote the size of a smallest cone:

$$c = (MIN \ x : x \in P : \#Cone(x)) \ . \tag{3.6}$$

From (3.5) and the fact that ‘‘binomial coefficients peak in the middle,’’ we know that

$$c = \#Lines/(n \ \mathbf{choose} \ m) = m!\,\overline{m}! \ . \tag{3.7}$$

Put differently: the smallest cones are exactly those for which x has weight m or $\overline{m}$:

$$(\#Cone(x) = c) \ \equiv \ (\mathbf{w}(x) = m \ \vee \ \mathbf{w}(x) = \overline{m}) \ . \tag{3.8}$$

One final observation is required before we can prove the optimality of the Sperner codes. If line L belongs to both $Cone(x)$ and $Cone(y)$, then both x and y lie on L, and, hence, they are comparable. That is, the cones of incomparable elements of P are disjoint:

$$\neg(x \ \mathbf{li} \ y) \ \Rightarrow \ Cone(x) \cap Cone(y) = \varnothing \ . \tag{3.9}$$

Let C be an antichain in our poset. We prove that it has at most n **choose** m elements.

$\#C$

$=$ { counting }

$(SUM \ x : x \in C : 1)$

$\leq$ { (3.6) }

$(SUM\ \ x : x \in C : \#Cone(x)/c)$

$=$ { distribution }

$(SUM\ \ x : x \in C : \#Cone(x))/c$

$=$ { (3.9), using that C is an antichain }

$\#(\cup x : x \in C : Cone(x))/c$

$\leq$ { for all $x \in C$: $Cone(x) \subseteq Lines$ }

$\#Lines/c$

$=$ { (3.7) }

n **choose** m .

This also gives an upper bound on the size of a DI code. The Sperner codes attain this upper bound. Therefore, if (I, C) is a DI code, then its size is at most the size of a Sperner code over I. The Sperner codes are optimal.

Because the upper bound is attained by the Sperner codes, we have only equalities in the above derivation for the antichain C, in the case that (I, C) is an optimal DI code. In particular this means that the two estimates are exact. Concerning the first estimate we derive

$(SUM\ \ x : x \in C : 1) = (SUM\ \ x : x \in C : \#Cone(x))/c$

$\equiv$ { calculus }

$(\mathbf{A}\ x : x \in C : \#Cone(x) = c)$,

$\equiv$ { (3.8) }

$$(\mathbf{A}\ x : x \in C : \mathbf{w}(x) = m \ \vee\ \mathbf{w}(x) = \overline{m}) . \tag{3.10}$$

To paraphrase: If a DI code over I is optimal then all the cones of its elements have minimum size, and, hence, all its words have weight m or $\overline{m}$. As far as we now know—assuming only the first estimate to be exact—the weights may still be mixed, or some words of these weights may be missing from the code.

Concerning the second estimate we derive

$$\#(\cup x : x \in C : Cone(x)) = \#Lines$$

$\equiv$ { calculus, using $Cone(x) \subseteq Lines$ }

$$Lines \subseteq (\cup x : x \in C : Cone(x))$$

$\equiv$ { set calculus }

$$(\mathbf{A}\ L : L \in Lines : (\mathbf{E}\ x : x \in C : L \in Cone(x))) \ .$$

$\equiv$ { definition of $Cone(x)$ }

$$(\mathbf{A}\ L : L \in Lines : (\mathbf{E}\ x : x \in C : x \in L)) \tag{3.11}$$

$\equiv$ { separate proof, see below }

$$(\mathbf{E}\ k : 0 \le k \le n : (\mathbf{A}\ x : x \in P : x \in A \ \equiv \mathbf{w}(x) = k)) \ . \tag{3.12}$$

To paraphrase: If a DI code over I is optimal then all lines intersect it, and, hence (not trivially), it consists of all words of a certain weight k.

Combining (3.10) and (3.12) immediately yields: If a DI code over I is optimal then it consists either of all words of weight m or of all words of weight $\overline{m}$, implying that it is a Sperner code over I.

There is still one proof obligation to be dealt with: the equivalence of (3.11) and (3.12). It is obvious that (3.12) implies (3.11), since each line contains words of every possible weight. We now prove that (3.11) implies (3.12).

Assume (3.11) to hold. Let y be an element of C with minimal weight. Take k equal to $\mathbf{w}(y)$. We shall show

$$(\mathbf{A}\ x : x \in P \wedge \mathbf{w}(x) = k : x \in C) \tag{3.13}$$

by mathematical induction on $\mathbf{w}(y \setminus x)$. From this (3.12) follows, because an antichain cannot contain other elements than all those of some fixed weight (recall that the k-out-of-n codes are maximal).

Base: $\mathbf{w}(y \setminus x) = 0$. Since x and y have the same weight, they are equal in this case. Hence, $x = y \in C$.

Step: $\mathbf{w}(y \setminus x) > 0$. Then $y \setminus x$ is nonempty. Let $i \in y \setminus x$. Since x and y have the same weight, $x \setminus y$ is also nonempty. Let $j \in x \setminus y$. Define the words z and x' by $z = x \cup \{ i \}$ and $x' = z \setminus \{ j \}$. We now have $x \subseteq z$ and $x' \subseteq z$. Furthermore, we know $\mathbf{w}(z) = k+1$ and $\mathbf{w}(x') = k$. We derive

$$y \setminus x'$$

$=$ { definition of x' }

$y \setminus (z \setminus \{ j \})$

$=$ { $j \notin y$ on account of $j \in x \setminus y$ }

$y \setminus z$

$=$ { definition of z }

$y \setminus (x \cup \{ i \})$

$=$ { set calculus }

$(y \setminus x) \setminus \{ i \}$.

Because $i \in y \setminus x$, we now have shown $\mathbf{w}(y \setminus x') = \mathbf{w}(y \setminus x) - 1$. This allows us to apply the induction hypothesis to x', which yields $x' \in C$.

Consider a line L through z and x (such a line exists, since $x \mathbf{\ li\ } z$). By assumption (3.11) L intersects C. L does not intersect C in the word z or a larger word, since $x' \in C$ is a proper subset of z and C is an antichain. There are no words between z and x because their weights differ by one. Therefore, L intersects C in x or a smaller word. But all words smaller than x have weight less than k, while by the choice of k all words of C have weight at least k. Conclusion: $x \in C$.

This concludes the proof that the Sperner codes are the optimal DI codes.

4 BERGER CODES

From the preceding section we know that the Sperner codes are the optimal DI codes. It seems, however, that there does not exist an easy encoding scheme for them (van Overveld 1985). The Berger codes, which we present in this section, are somewhat smaller than the Sperner codes, but their encoding is very simple. They are also an example of good codes that are not subcodes of the k-out-of$n$ codes.

Let E be an encoding scheme that maps data (or message) vectors of length m onto the code words of some code with length n, $n \geq m$. Of course, E is a one-to-one mapping. The encoding E is called *systematic* if for each data vector d, $E(d)$ is the concatenation of d and some vector s of length k, $k = n - m$. This is written as $E(d) = d : s$ (concatenation is here denoted by a colon). We call d and s

the *data* and *synchronization* parts respectively. In the case of a DI code the synchronization part must enable the receiver to detect the complete arrival of a message.

A code is called *separable* when it allows a systematic encoding or when it is isomorphic to such a code. This section will deal only with separable codes. A systematic encoding is completely specified by the way in which the synchronization part depends on the data part. We denote this mapping from data to synchronization part by $sync$. Data vector d is, thus, encoded as $d:sync(d)$. Notice that decoding a separable code is very simple, but that the complexity of encoding and membership test depends on the expression for $sync$.

A mapping $sync$ also defines a code, viz. the set of words obtained by applying the corresponding encoding to all data vectors of a fixed length m. We call this the code *generated* by $sync$ (for data length m). As an example consider the mapping $sync(d)=\bar{d}$. The corresponding encoding maps data vectors of length m onto code words of length $2m$ and weight m. The code generated by $sync$ is a DI code, because it is a subcode of the Sperner code of length $2m$. This code is isomorphic to the Double-Rail code. Hence, the Double-Rail code is separable.

We now investigate the conditions that $sync$ must satisfy in order to generate a DI code. Throughout this section m will be the length of data vectors and k the length of their $sync$ images. Define C as the code generated by $sync$, it has length $n=m+k$.

Let x and y be data vectors of length m. We derive

$$x:sync(x)\subseteq y:sync(y) \Rightarrow x:sync(x)=y:sync(y) \tag{4.1}$$

$\equiv$ { property of concatenation, using that x and y have the same length }

$$x\subseteq y \wedge sync(x)\subseteq sync(y) \Rightarrow x=y \wedge sync(x)=sync(y)$$

$\equiv$ { $sync$ is a mapping }

$$x\subseteq y \wedge sync(x)\subseteq sync(y) \Rightarrow x=y$$

$\equiv$ { proposition calculus }

$$x\subseteq y \wedge x\neq y \Rightarrow \neg(sync(x)\subseteq sync(y)) \tag{4.2}$$

$\Rightarrow$ { set calculus }

$$x\subseteq y \wedge x\neq y \Rightarrow sync(x)\neq sync(y)\ .$$

So, if C is DI, then all elements in a chain (cf. Section 3) of data vectors have different $sync$ images.

Because the longest chains of data vectors have $m+1$ elements, *sync* must produce at least $m+1$ different images. Hence,

$$2^k \geq m+1 \qquad (4.3)$$

is a necessary condition for C to be DI. Notice that in the code generated by $sync(d)=\overline{d}$ (isomorphic to the Double-Rail code) all *sync* images are different.

To obtain the largest possible DI code (for a given length n) one should take k as small as possible, while still satisfying (4.3). It turns out that even for the smallest such value of k, many choices of *sync* are available that satisfy (4.2). One particularly elegant choice is the following. Instead of taking $sync(d)=\overline{d}$ as above, we define *sync* by

$$sync(d)= \text{ the } k\text{-bit binary representation of } \mathbf{w}(\overline{d}), \qquad (4.4)$$

that is, the number of 0's in d written as a k-bit binary number. It generates the *Berger code* (Berger 1961, Freiman 1962), sometimes also called *sum code*. Notice that $\mathbf{w}(\overline{d})$ ranges from 0 to $m+1$ and has a k-bit binary representation, because (4.3) holds.

Let us first prove that the Berger code is a DI code (x and y are data vectors):

$x : sync(x) \subseteq y : sync(y)$

$\equiv$ { property of concatenation, using that x and y have the same length }

$x \subseteq y \wedge sync(x) \subseteq sync(y)$

$\Rightarrow$ { definition of *sync* and property of binary representation }

$x \subseteq y \wedge \mathbf{w}(\overline{x}) \leq \mathbf{w}(\overline{y})$

$\equiv$ { property of complementation }

$x \subseteq y \wedge \mathbf{w}(x) \geq \mathbf{w}(y)$

$\equiv$ { set calculus }

$x = y$.

The membership test for the Berger code is also simple. The word $d : s$ is a code word if and only if $s = sync(d)$, i.e., if and only if s is the binary representation of the number of 0's in d.

Let us now consider the size of the Berger code. It is, of course, 2^m. How does this compare to 2^n? In the best case we have $m+1=2^k$, we thus get

$$2^m = 2^{n-k} = 2^n/(m+1) \geq 2^n/n \quad . \tag{4.5}$$

The worst case occurs for $m = 2^{k-1}$, yielding $2^m \geq 2^n/2n$. The redundancy of the Berger code is k, which is approximately $\log n$. According to (3.2) this is roughly twice that of the Sperner code of the same length.

For example, the Berger code of length 19 is obtained by taking $m = 15$ and $k = 4$, so it has 32,768 code words. The Sperner codes of length 19 utilize 19 **choose** $9 = 92{,}378$ code words of the roughly half a million words available.

The complement of a Berger code is also a separable DI code. Only for $m = 2^k - 1$ is this again a Berger code. In that case the code is invariant under complementation, or *self-complementary*: the complement of each code word is itself a code word. Notice that, by their construction, the Berger codes are optimal among the separable DI codes (Freiman 1962). But they are not the only optimal separable DI codes. By the way, also notice that the Sperner codes of length at least 3 are not separable.

5 KNUTH CODES

This short section presents the main idea behind a family of DI codes recently designed by Knuth (1986). These codes are subcodes of the Sperner codes but they allow a relatively simple encoding and nevertheless are not much smaller. We describe only the least sophisticated Knuth code.

A word x of length m is called *balanced* when $\mathbf{w}(x) = m$ **div** 2. To encode the data vector d of length m it is first written as $x : y$ such that $x \,\bar{:}\, y$ is balanced (this is always possible, consider x's of increasing length). To make the encoding one-to-one, the length of x is separately encoded in a fixed-even-length balanced vector z. The encoding for x is discussed below. The data vector d now has encoding $x \,\bar{:}\, y : z$, which is also balanced. The receiver can decode such a code word by first determining the extent of complementation from z, then splitting the data part accordingly, and complementing its first half.

The extent of complementation ranges from 0 to m, thereby constituting $m+1$ messages. These can, for instance, be encoded by an even-length Sperner code. That this encoding is not so simple does not matter too much, because the number of values to be encoded is relatively small. It could be done by table-lookup.

For example, the Sperner code of length 6 has 20 code words. Hence, data vectors of length m, $m \le 19$, can be encoded into code words of length $m+6$. Actually, this also holds for $m = 20$, because when m is even the data vector never needs to be complemented entirely.

Let us briefly analyze the size of these Knuth codes. Consider the Sperner code of length $2k$. From (3.1) we know that its size is approximately $2^{2k}/\sqrt{\pi k}$, which, in the best case, equals $m+1$. For the size of the corresponding Knuth code of length $n = m+2k$ we, therefore, have

$$2^m = 2^{n-2k} = \frac{2^n}{(m+1)\sqrt{\pi k}} \ge \frac{2^n}{n\sqrt{\pi k}} \tag{5.1}$$

Hence, the size of these Knuth codes is only slightly less than that of the Berger codes of the same length. The redundancy is roughly

$$\log n + \tfrac{1}{2} \log \log n + \tfrac{1}{2} \log \pi/2 \; . \tag{5.2}$$

Of course, a number of variations on this encoding scheme are possible. For these we refer the reader to Knuth (1986).

6 CONCLUSION

We have formulated the problem of delay-insensitive data communication. We have given necessary and sufficient conditions for codes to be usable for delay-insensitive message transmission. It turned out that these conditions are equivalent to the ones defining antichains in partially ordered sets and all unidirectional error-detecting (AUED) codes. A number of delay-insensitive codes have been constructed and analyzed. The following table summarizes these DI codes (logarithms have base 2).

DI Code (length n)	**Redundancy** (approximately)	**Rate** ($n \rightarrow \infty$)
One-Hot	$n - \log n$	0
Double-Rail	$n/2$	0.5
Knuth	$\log n + \frac{1}{2} \log \log n + \frac{1}{2} \log \pi/2$	1
Berger	$\log n$	1
Sperner	$\frac{1}{2} \log n + \frac{1}{2} \log \pi/2$	1

Important practical aspects of codes are their size (measured, for example, by redundancy or rate), encoding-decoding scheme, and membership test. The One-Hot codes are among the most trivial DI codes, but they are very inefficient for larger code lengths. The Double-Rail and Berger codes are separable and their encoding is simple. Especially the Berger codes look promising for the use in delay-insensitive circuits. The Sperner codes are the optimal DI codes, but no easy encoding scheme is known. The Knuth codes are large subcodes of the Sperner codes which do allow a relatively simple encoding.

Acknowledgments. Thanks is, in the first place, due to Huub Schols for providing an elegant characterization of the conditions for delay-insensitive communication. This has greatly simplified many arguments. Henk van Tilborg pointed out the relation with unidirectional error-detecting codes.

References

Berger JM (1961) A note on error detection codes for asymmetric channels. Inf Control 4: 68-73

Bose B, Rao TRN (1982) Theory of unidirectional error correcting/detecting codes. IEEE Trans Comput C-31: 521-530

Freiman CV (1962) Optimal error detection codes for completely asymmetric binary channels. Inf Control 5: 64-71

Greene C, Kleitman DJ (1978) Proof techniques in the theory of finite sets. In: Rota G-C (ed) Studies in combinatorics. The Mathematical Association of America, pp 22-79

Hollaar LA (1982) Direct implementation of asynchronous control units. IEEE Trans Comput C-31: 1133-1141

Knuth DE (1986) Efficient balanced codes. IEEE Trans Inf Theory IT-32: 51-53

van Overveld WMCJ (1985) On arithmetic operations with m-out-of-n codes. Computing Science Notes, No 85/02, Dept of Math and Comp Sci, Eindhoven University of Technology, The Netherlands

Seitz CL (1980) System timing. In: Mead CA, Conway LA, Introduction to VLSI systems. Addison-Wesley, Reading MA, chapter 7

Sperner E (1928) Ein Satz über Untermengen einer endlichen Menge. Math Z 27: 544-548

Udding JT (1986) A formal model for defining and classifying delay-insensitive circuits and systems. Distributed Computing 1: 197-204