Archive-enabling Tagging Using Progressive Barcodes

Steven Simske and Marie Vans; HP Labs; Fort Collins, Colorado

Abstract

We have previously described the Progressive Barcode, a high-density color barcode that changes over time.[4]In this paper we will describe how the progressive barcode works and its applicability to information workflows and archiving applications.

Introduction

Due to the ubiquity of high-resolution mobile cameras, the Data Matrix[1] and QR 2dimensional (2D) barcodes are being used now for many applications[2]. Color barcodes, often termed 3D barcodes, offer increased density over 2-dimensional barcodes[3], which can be taken advantage of to embed longer data strings in the same printed/displayed area. Additionally, color channels offer the possibility of containing multiple, distinct sets of data in the same "hybrid" mark.

The four-dimensional Progressive Barcode is a set of printed color marks that do not change in size as they are used to represent different stages (or "states") in a workflow. These barcodes support many different information lifecycles by allowing it to change through time.[4] Progressive barcodes can be also used to support two (or more) applications or services in the same object. One of these is usually standards-compliant and the other is usually proprietary or customized. Thus, a "hybrid" of two functions can be combined in a single mark.

These two sets of data include different densities of information. The first set is binary, with high contrast between the two binary encoding (usually black and white) tiles in the barcode. The second is N-ary, and utilizes color that is indistinguishable from white to the binary barcode reading software.

Building on previously published work[4], we demonstrate that archiving applications and services can be enabled by the progressive barcode. They are most effectively deployed when there are multiple types of information payloads needed for a single object—e.g. fields of archiving standards and versioning information as well as a variety of document/physical item workflow-related objects.

Progressive Barcodes

A progressive barcode changes as a one-way function of its current state. For example, if we start with a simple binary sequence {000000000000} and then move to a next state through the replacement of four 0's by four 1's. Then, two allowable next states are {001010001100} and {100110000100}. In general, if there are N 0's (zeroes) left to be changed into 1's (ones) and M 1's added to the next state, then we can write $N! / [M! (N-M)!]$ different next states, where $!$ is the factorial operator. For the progressive barcode, once a 0 has been changed into a 1, it cannot change back into a 0. Thus, each next state can be immediately compared to a previous state to see if it is logically a part of the same workflow, a necessary but not sufficient condition for security[5]. Figure 1 illustrates this concept.

These barcodes allow us to assign the statistical probability associated with any transition between two steps in a workflow based on how many bits are written and how many remain. If progression step i is defined as P_i, where the number of residual (0 bits) at the end of the workflow is NRB, and the number of initial unwritten bits is NIU, then governing equation for each step is:

$$(1)\quad \frac{N_{IU}!}{(N_{IU}-N_{RB})!\,N_{RB}!} \geq \prod_i P_i$$

P_i may be determined from, for example, the required statistical confidence that a next step cannot be randomly guessed multiplied by the total number of progressive barcodes of the current state that will be readable in the workflow. If the progressive barcode is binary, then the number of bits in the workflow is N_{RB}-N_{IU}. If there are N_C colors, then the number of bits increases to $[ln(N_C)/ln(2)]*(N_{RB}-N_{IU})$. The size (number of tiles) of the progressive barcode to be used in the workflow can be determined from these equations, along with the number of bits to write at each state.

Any number and combination of colors may be used to create progressive barcodes. However, for demonstration purposes we show six-color barcodes utilizing the pure printing colors: cyan (C), magenta (M), and yellow (Y). These can be later overwritten or overprinted to create three additional colors, red (R), blue (B), and green (G). Figure 2 demonstrates the concept of color progression. Each cell in the barcode starts out in a particular color state and can only progress accordingly. For example, if a cell is currently magenta, 'M', the next allowable state for the cell can be either blue, 'B', or 'R', red. It may not progress to green. Once a cell has reached the Black stage, it can no longer progress.[5]

To determine the absolute data content of a color tile, we consider each color tile to be independent. There are $\log_2(n)/\log_2(2) = \log_2(n)$ bits at any stage, where n=the number of colors. For example, if n=2, as for 2D QR-Code, then there is 1 bit per tile. For a color tile with six colors {RGBCMY}, there are 2.585 bits/tile. If there are eight colors allowed {RGBCMYWK}, there are obviously exactly 3.0 bits/tile. It should be noted that

there is a trade-off between reliability and data-density[3] when color is introduced into barcodes. Previously, a series of experiments on the effect of copying and restoration on color barcode payload density addresses this issue[3].

Adding color progression allows us to use the "static" data encoded within the black and white modules for standard purposes such as serial numbers, and product information while allowing a "separate channel" for encoding additional, workflow-related information that changes over the course of the workflow. For the static data, the off-the-shelf reader reads the black modules as normal and the rest of the modules, whether white or saturated non-white colors, as "white". Note that for color progression in this instance, colors are not allowed to progress all the way to black. Instead, the progression terminates at red, green or blue.

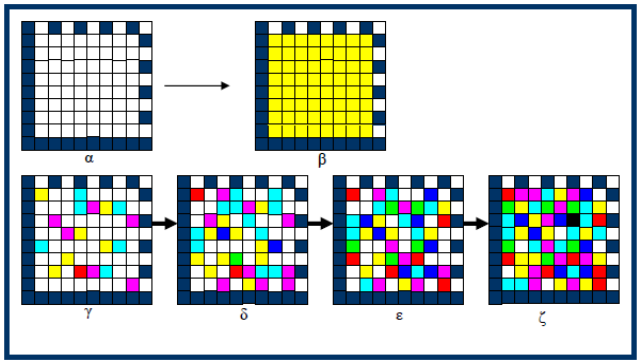

***Figure 1**: Figure 4 illustrates new data being added to a progressive barcode as it progresses through the workflow. The upper leftmost image (α) represents a barcode with only the non-payload indicia indicated. The non-payload indicia (NPI) are the perimeter pixels on all four sides and used for calibration. The yellow pixels shown on the upper row, center, image (β) are the data pixels which can be written to as part of the incremental writing process. In the lower row, the initial barcode is pre-filled with, in this case, 16 data bits as shown in the leftmost image (γ). Next, the three workflow stages from barcode, γ, result in incremental writing of pixels to the barcode (δ, ε, and ζ). Note that this figure is for illustration purposes and does not reflect the hybrid barcode.[4]*

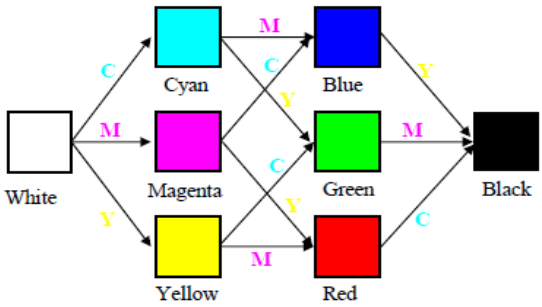

***Figure 2**. The basic lifecycle of a color tile, where the colors White, Cyan, Magenta, Yellow, Blue, Green, Red and Black are shorthanded as W, C, M, Y, B, G, R and K. [4]*

Progressive Barcodes for Archiving Applications

The amount of data that can be encoded in the first set of data – that is, black and white data – will be limited by the total number of black and white tiles available and the standards. URLs to websites are typically encoded into QR Codes while Data Matrix codes hold serial numbers, invoicing information or product numbers. While these are common uses of the technology, it is also possible to use them to encode other information; for example, xml fields for archiving purposes, ISBNs and other document-related information. The main characteristic of this information is that it remains static over time. A second channel of information can be encoded into the white tile channel, and this information could be used to track and store documents, for versioning or tracking document changes, and to assign ownership of documents. This "*hybridization*" allows for multiple services: open services and proprietary services. An example is the encoding of a website URL along with security/authentication services. These barcodes are readable by any off-the-shelf, QR Code and the Data Matrix code readers. The Data Matrix code demonstrates that a large quantity of data can be directly stored in the black-as-black and rest-as-white tiles. Figure 3 demonstrates this idea. Figure 5 demonstrates how these hybrid barcodes would work with progression in the color channel.

***Figure 3**: Demonstration of barcode with two channels. The black and rest-as-white will read with an off-the-shelf reader, the right Data Matrix will display an abstract and the left QR-Code displays a test message.*

Inference for Archiving Data

Progressive barcodes could be used for a series of incremental, related or linked codes which can be simultaneously secured as a multiplicity of items (rather than as individual elements). An obvious example would be items packaged in a carton which is placed on a pallet of like cartons and placed in a container. For archiving data, the example could include images contained by files that are contained within a directory on a specific computer, where each level had differing access permissions set.

Inference is the relationship between objects and their containers. Suppose that we have individual objects, such as images, signified by A, which are packaged together into a file, signified by B. Multiple files are contained in folders, signified by C. Finally, multiple folders are on each computer, signified by D. Thus, D contains multiple C, where p is the number of C in D: a relationship we indicate by $D_n(C_m,C_{m+1},C_{m+2}, \ldots, C_{m+p-1})$. Each C contains s number of B: a relationship we indicate by $C_q(B_r,B_{r+1},B_{r+2}, \ldots, B_{r+s-1})$. Finally, each B contains v number of A: a relationship we indicate by $A_t(B_u,B_{u+1},B_{u+2}, \ldots, B_{u+v-1})$. It is clear from this that the inference model, where $\mathcal{F} \leftarrow \mathcal{G}$ indicates containment of set $\mathcal{G}$ by the container $\mathcal{F}$. In this example, then, we have the following inference relationship:

$$D \leftarrow \{C\} \leftarrow \{B\} \leftarrow \{A\}$$

Where {} implies a set of 1 or more contained items. Typically the set {} is of more than one item, although not always; for example, a package or "electronic package" can be separately

labeled from an item inside it (for example a directory with a single file in it).

We can then designate the number of items in a set $\vec{G}$ by $n(\vec{G})$. Next, we wish to know how many tags (barcodes) are required to tag n(A) items when they are inferred to B, C and D containers as described above. The following ratios are very important: n(A)/n(B), n(B)/n(C), and n(C)/n(D). We will create a series of cryptographically secure tags, in sequence, and assign them to the containers they are representing as follows:

(D) (C) (B) A….A (B) …. (B) A….A (B) (C) …. (C) …. (C) (D)

In other words, if we have AA…A as the individual items and B are the larger units around these, C the larger units holding the B units, et cetera, then we use two tags (barcodes) on each enclosing container to mark the start and end of the set of items within the container…e.g. if n(A) = 4 and n(B) = 3, then for one C container, tags are assigned as follows:

C B A A A A B B A A A A B B A A A A B C

To label the associated C, B and A items in this example, we need n(A)*n(B)*n(C) tags for the "A" items; that is, 4*3*1 = 12. We also need 2*n(B)*n(C) tags for the "B" items, and 2*n(C) tags for the "C" items.

This continues in perpetuity. Suppose we have M levels of containment – that is, M=alphabetic (levels of containment, where M is for Matryoshka, since it is a Russian doll model) – then we need one identifier each for the innermost units and two each for each container. The overall number of identifiers required, nIDs_required, is:

$$n_{IDs\_required} = \prod_{j=A}^{M} n(j) + 2 * \sum_{i=B}^{M} \prod_{j=i}^{M} n(j) \quad (2)$$

Thus, if M=4 (for the A, B, C, D example above):

$$n_{IDs\_required} = n(A) * n(B) * n(C) * n(D) + 2 * [n(B) * n(C) * n(D) + n(C) * n(D) + n(D)] \quad (3)$$

For the N=3 [n(A)=4; n(B)=3; n(C)=1] example:

$$n_{IDs\_required} = n(A) * n(B) + n(C) + 2 * [n(B) * n(C) + n(C)] \quad (4)$$

Thus,

$$n_{IDs\_required} = 4*3*1 + 2*[3*1+1] = 12 + 8 = 20 \quad (5)$$

This total of 20 includes 12 primary tags for the 12 individual items, 2 each for the three B containers, and 2 for the C container.

This model for inference, while simple, allows a wide variety of possibilities. If the tags on the individual items {A} cannot be read – for example if they are labeled with barcodes and inside a folder that cannot be read during due to permission issues – then the tags on the folder B, would correspond to the start and end of the sequence of tags corresponding to the two on container B and all of the individual tags on the A items within. Figure 6 is an illustration of how this might work for document storage. Figure 4 is an example using tags from GS1, a global organization that develops standards for supply chain workflows. The figure is based on a single GS1 tag that is readable via inference even at the pallet level.

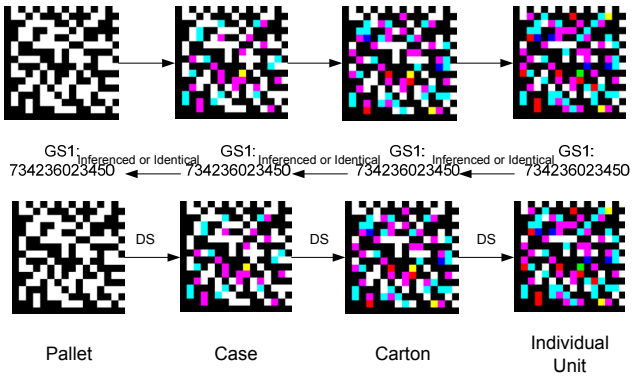

***Figure 4**. Example use of inference using GS1tags.*[4]

Instead of just wanting to know the range of values in a container, we wish to ensure that multiple items can infer to the same container in a statistically meaningful way. We also wish this relationship to be established relatively (without connection to an on-line database) and absolutely (by connecting to the on-line database).

Relative inference is established when there is a mechanism for associating a container with an item and vice versa. Progressive barcodes are a good example of such a mechanism as they meet the criteria for unambiguous, statistically-separable relative inference.

Since the binary strings (or "unique IDs") have a one-way function moving forward – binary 0 can convert to binary 1 or stay binary 0, while binary 1 cannot convert to binary 0 – there is appropriate containment of the item by the container. As a consequence of this, the item I is shown to be contained by the container, C, with the relationship I + A = C subject to the binary relationships I&A=0, I&C=I, and A&C=A. Thus, I=10000101 can be contained by C=11001101 since C=11001101 is I=10000101 + A=01001000 and I&A=0, I&C=I, and A&C=A.

The binary strings can be made non-ambiguous through an explicit set of non-collision rules in the database. The minimum set of rules is that each individual item *I* must have a fully unique ID. A non-collision data set is enforced for this. Thus, the overall set of all I, or {I}, is non-colliding. Because of the relationship I + A = C subject to the binary relationships I&A=0, I&C=I, and A&C=A (as described above), we ensure that the sets {I} and {C} are mutually non-colliding by design (unless A=0).

Statistical separation between different values for {C} can be ensured by requiring a set of rules about the different Hamming distances between sets. The Hamming distance (Hd) between two binary strings, BS1 and BS2, is the sum of all string indices j for which BS1(j) XOR BS2(j) is equal to 1; that is:

$$Hd = \sum_{j=1}^{length(BS_1)=length(BS_2)} BS_1(j) \oplus BS_2(j) \quad (6)$$

The Hd can be used to eliminate even non-colliding new values for I, A or C if, in the context of the existing sets of I and C binary strings these new values lead to unwanted similarity between items and/or containers. The statistics of separation are based on Hd and also the ability to "guess" a legitimate prior or next state (that is, guess "I" from "C" or "C" from "I"), as described in the Appendix and as applied in the below.

Relative inference, therefore, is validated independently of the database. We need only show that I&A=0, I&C=I, and A&C=A, and that the number of binary 1's for I, A and C – that is, n1(I), n1(A) and n1(C) – are the same for any set of purportedly related items and/or containers. While this approach does not "prove" that the individual binary strings are accurate, it does show whether or not a set of purportedly related binary strings actually fit a model.

Obviously, this model prevents casual tagging of a set of items through assignment of random binary strings. It does not, however, prevent reverse engineering of A and even C from a large enough set of items {I}.

Absolute inference is a form of inference that requires approval of two or more binary strings simultaneously, with or without the overlying relative inference model. The easiest means of absolute inference is the simple association of two binary strings in a database. Another form of absolute inference is when two strings are related to each other through an algorithm or process; for example, if C=digital signature(I) or C=scrambling_algorithm(I), etc.

Conclusions

Progressive barcodes are ideal mechanisms for applications and services that can take advantage of two channels of information encoded into a single mark that does not change size over time. They are suitable for document workflow tracking and archiving solutions where data for one channel does not change (serial numbers, document IDs, MARC and other archiving information) is encoded in the black and rest-as-white tiles and the channel represented as color tiles can change over time for tracking, security, and other purposes. Using inference, it is possible to contain objects inside other objects and verify their authenticity without having to open the container.

References

[1] International Standard ISO/IEC 16022:2006(E), Second edition 2006-09-15, "Information technology – Automatic identification and data capture techniques – Data Matrix bar code symbology specification," 142 pp., 2006.

[2] International Standards Organization (ISO). 2006 B. International Standard ISO/IEC 18004:2006(E), Second edition 2006-09-15. Information technology – Automatic identification and data capture techniques – QR Code 2005 bar code symbology specification, 114 pp.

[3] Simske, S.J., Aronoff, J.S., Sturgill, M.M., Villa, J.C. 2008. Spectral pre-compensation and security deterrent authentication, in Proceedings of the International Conference on Digital Printing Technologies, NIP24, Pittsburgh, Pennsylvania, pp. 792-795.

[4] S J. Simske, A.M. Vans and B. Loucks. 2012. Incremental Information Objects and Progressive Barcodes, in Proceedings of the Digital Fabrication and Digital Printing Conference, NIP28, Quebec City, Quebec, Canada, pp. 375-377.

[5] Vans, Marie, Steven Simske, and Brad Loucks. "Progressive Barcode Applications." NIP & Digital Fabrication Conference. Vol. 2013. No. 1. Society for Imaging Science and Technology, 2013.

Author Biography

Steve Simske is Director and Chief Technologist for the Content Solutions Lab in HP Labs, where he oversees the development of document ecosystem and security solutions, brand protection, education, print production, and other printing and personalized systems research. He holds a PhD in electrical engineering from the University of Colorado where he was also a postdoctoral fellow in aerospace engineering.

Marie Vans is currently a Research Scientist with Hewlett-Packard Labs in Fort Collins, Colorado. Her main interests are security printing and document analytics. She has a Ph.D. in computer Science from Colorado State University. She is also currently a student in the Master's program in the Department of Information at San José State University where she is focusing on the use of virtual worlds for distance education.

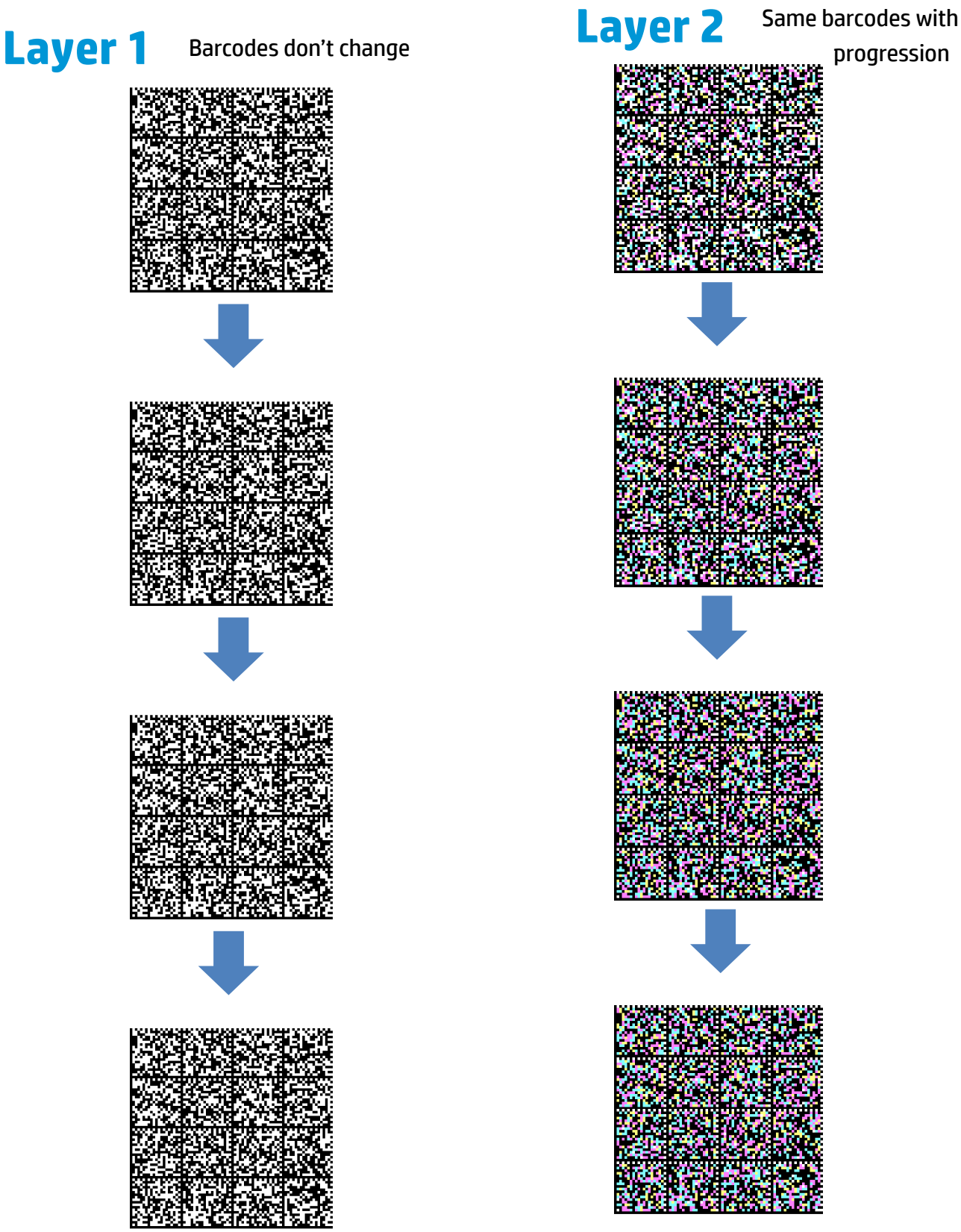

***Figure 5**. Example progressive barcode progression showing two layers: 1. Static information, 2. Proprietary information*

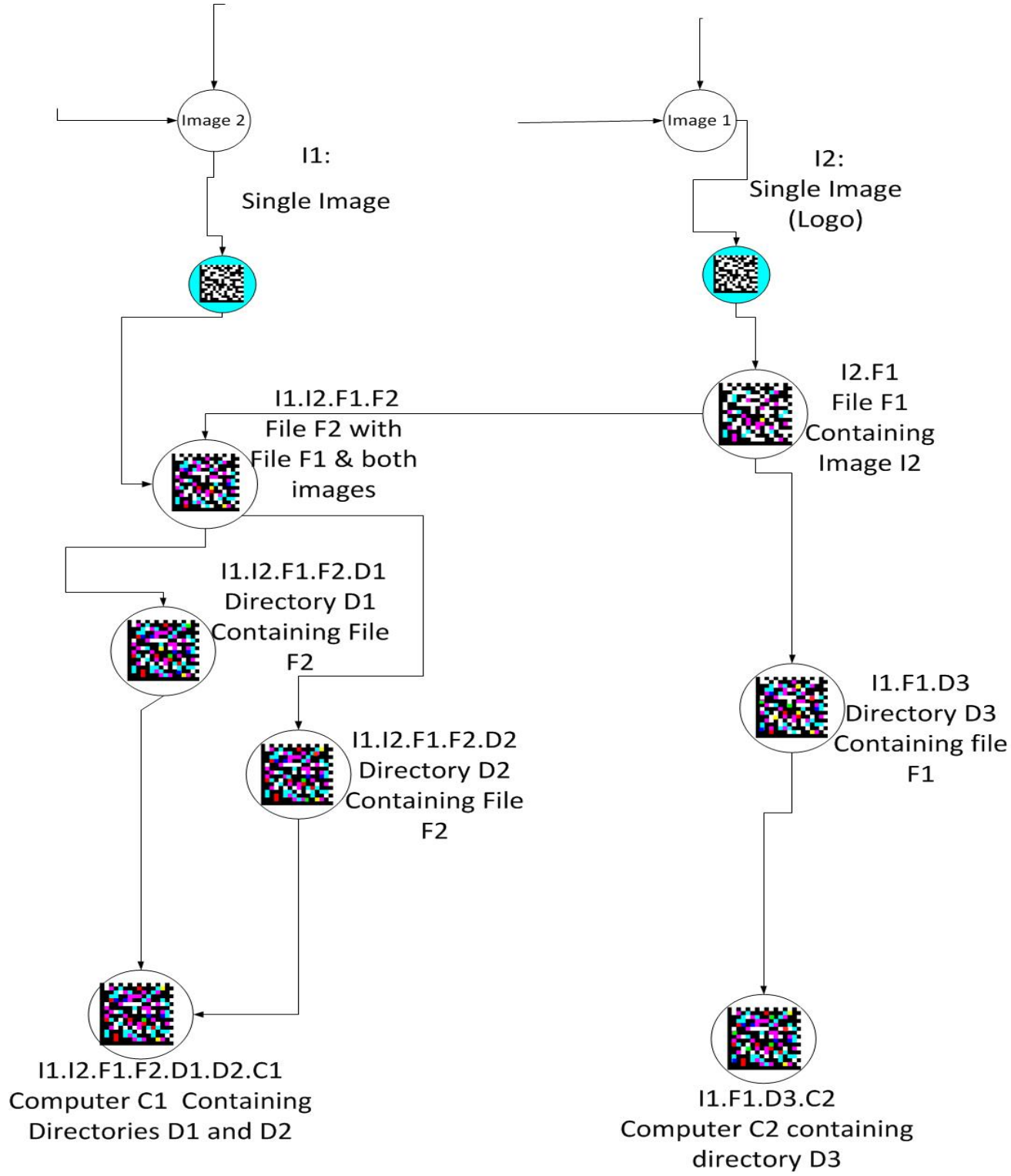

***Figure 6**. Example progression showing how inference can be used for document storage. Please note that the readability of these and other barcodes in this paper after printing is not possible to guarantee.*