QR codes: From a Survey of the State-of-the-Art to Executable eQR codes for the Internet of Things

Stefano Scanzio, *Senior Member, IEEE,* Matteo Rosani, Mattia Scamuzzi, and Gianluca Cena, *Senior Member, IEEE*

***Abstract*—QR codes are increasingly used in a plurality of scenarios, and research activities are being successfully carried out to improve this technology and widen its contexts of applicability. After an extensive survey of the state-of-the-art on the subject, this work presents the new, promising possibility to embed a programming language in a QR code.**

This new kind of executable QR codes, we named eQR codes, enable interaction with end users even in the absence of an Internet connection, and provide a sort of IoT paradigm where intelligence is embedded in the object tag in the form of a program. Among all the possible languages that can be embedded, this work focuses on a powerful but compact (in terms of QR code storage occupation) dialect, termed QRtree, which is aimed at implementing decision trees. The eQR code technology makes a new class of applications possible, e.g., providing hints for navigation or instructions for using rescue devices in places with no network coverage like mountains and caves. Smart interactive user manuals are enabled as well.

Besides defining the QRtree language and eQR code structure, this paper describes all the steps needed to generate eQR codes and to manage their execution in end-user devices. A simple yet realistic example and the related code are also presented, to practically show how this technology can be used to solve real-world problems. For the example, the QRtree version of the code takes $234\,\mathrm{B}$, less than one-half the size of an equivalent program in Python bytecode ($634\,\mathrm{B}$).

***Index Terms*—QR code, eQR code, executable QR code, programmable QR code, decision trees, compilers**

I. Introduction

THE technology behind the popular two-dimensional barcodes known as QR codes is now three decades old [1]. Nevertheless, it is still experiencing an endless spring due to the countless application contexts where it can be profitably applied. Popular examples include marketing [2], authentication [3], security [4], augmented reality [5], and transportation [6], to cite a few. Similarly to radio-frequency identification (RFID) [7], which constitutes one of the earliest enabling technologies for the Internet of Things (IoT), QR codes allow the quick identification of real-world objects, permitting them to be directly mapped in the cyberspace and "accessed" through the Internet. Typically, QR codes contain data that can be interpreted by an application: either numeric references, which have a similar function as linear barcodes (e.g., identifying or characterizing a particular object), or network references in the form, for instance, of a uniform resource locator (URL) that directly points to remote network resources. In the former case a network query is typically needed to retrieve the information associated with the object (unless it is stored in the device's memory), while in the latter accessing the relevant URL mandates that the device used for reading the QR code (e.g., a smartphone) is connected to a network (either the Internet or, at least, a local intranet).

Many research activities have been conducted in the past decades on QR codes, and the first contribution of this work is to provide a concise but comprehensive survey of the state-of-the-art about the latest innovations in this sector. The second, innovative contribution is to present a new type of QR code, we named executable QR (eQR) code, which enables interaction with the user by directly embedding a runnable program. The ability to include algorithms that can be executed locally is what differentiates eQR technology from traditional QR codes. This idea was preliminarily proposed in [8], which, to the best of our knowledge, is the only scientific work dealing with such a possibility at the time of writing. Basically, all devices that can be used for reading QR codes also work with eQR codes. Besides a camera with adequate resolution, local computation resources are needed to execute the embedded program. However, since this code is small by necessity, most existing smartphones likely meet these requirements.

Conventional QR codes already support interactivity. To this aim, an URL must be encoded that points to a suitable resource located on a web server—either an HTML page that embeds client-side scripts or a server-side program to be executed remotely. However, doing so requires a properly working network connection. Although users nowadays expect to be provided all-time with ubiquitous Internet connectivity, this is unfortunately not always the case. Many places still exist, even in modern towns, where decent connectivity is often unavailable, e.g., basements, old buildings, and also very crowded rooms. By exploiting eQR codes, the user retains the ability to interact with "dumb" physical objects, but no Internet connection is demanded as the algorithm is stored in the payload of the QR code. This makes the eQR technology a nice addition to the IoT paradigm, and enables a whole range of functions that otherwise could not be implemented with conventional QR codes. For short, it provides users a "quasi-IoT experience" in the absence of Internet.

One of the main problems of directly encoding programs in QR codes is their capacity. In fact, in the current version they can store 2953 bytes at most. For this reason, the primary goal of our work is to define a suitable representation of programs

S. Scanzio and G. Cena are with the National Research Council of Italy (CNR–IEIIT), Italy.
M. Rosani is with the National Research Council of Italy (CNR–IEIIT) and with Politecnico di Torino, Italy.
M. Scamuzzi is with Politecnico di Torino, Italy.
Corresponding author: S. Scanzio, e-mail: stefano.scanzio@cnr.it

(and the related translation process) to keep code footprint as compact as possible. The first method to limit the size of the generated QR code is to pose some constraints on the programming languages that can be encoded within it.

The eQR code format and content are based on the QRscript language, which permits to embed a variety of programming sub-languages (termed *dialects*). To this purpose, a header is included in QRscript to select the specific dialect encoded within the eQR code.

Besides dialect selection, this work also presents a specific language, we named *QRtree*, which has excellent characteristics from the point of view of compactness. Although the QRtree dialect only permits the implementation of decision trees, it suits a large number of scenarios derived from real needs. More specifically, this dialect permits defining programs that, by interacting with the user through a series of questions, select a specific leave of the tree that represents the final advice. The root of the tree corresponds to the first question, and subsequent questions permit navigating the tree structure until a leaf is eventually reached.

To understand the real potential of eQR codes one must remember once again that, although we are living in a connected world, stable Internet connectivity is not always available everywhere and to everyone. If the QR code is located in a place without Internet access, QRscript and, in particular, the QRtree dialect could be a valuable enabler in many application contexts. A first, concrete example, which will be employed in the following to illustrate the benefits provided by our technology, are mountain trails, where eQR codes can be placed close to the forks and in some specific positions along the route to enable hikers to understand which is the best path to follow. The program included in the eQR code asks the user a number of questions to guide her/him in performing the most suitable choice based on aspects like her/his tiredness, available time, and personal preferences.

A second example, which has to do with guided diagnostics and maintenance, are boats: if a failure (e.g., broken engine) is experienced in a place with no network coverage (e.g., in the middle of the sea), running the program embedded in an eQR code (attached, e.g., to the engine itself) could help solving the problem, either directly (through explicit instructions) or indirectly (by providing references to the pages in the instruction manual where a solution can be found). Generally speaking, any book can be made “smart” by including an eQR code at its beginning, near to the index. Doing so allows readers to interact with the book using their smartphone, e.g., to quickly jump to the desired page/section. Not relying on the Internet, in this case, ensures complete privacy when looking for specific topics.

Besides entertainment and education, eQR codes can be targeted to more serious (and potentially critical) applications, e.g., to help people to operate medical instrumentation and rescue devices like defibrillators in emergency conditions, even in contexts where there is (temporarily) no network connection. Petrochemical plants, installations in desert areas, high mountains, forests, and many other similar scenarios can take advantage of the eQR code technology.

The organization of the paper has the following structure:

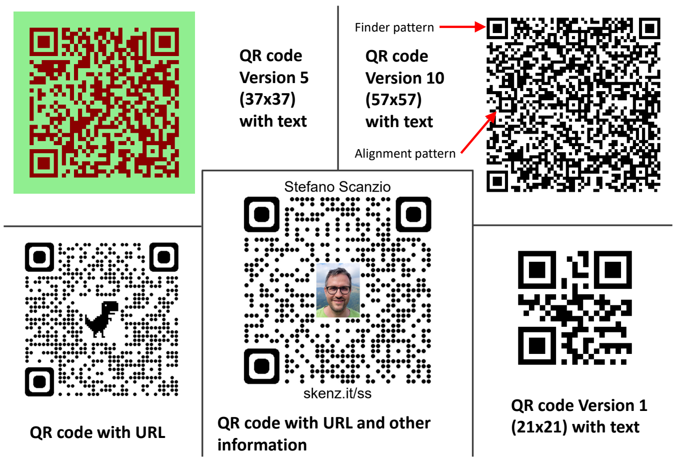

QR code Version 5 (37x37) with text
QR code Version 10 (57x57) with text
Finder pattern
Alignment pattern
Stefano Scanzio
skenz.it/ss
QR code with URL
QR code with URL and other information
QR code Version 1 (21x21) with text

Fig. 1. Examples of different types of QR codes.

Section II describes the existing QR code technology and analyzes the state-of-the-art; Section III introduces the new eQR code technology and the QRscript programming language it embeds; Section IV explains the QRtree dialect, while a real example of the steps needed to generate an eQR code and to execute it is shown in Section V; finally, Section VI draws some final remarks.

II. QR CODES

Conventional QR codes, also known as quick response codes, are a two-dimensional barcode technology invented quite a long time ago (in 1994) by the Denso Wave Automotive company to track vehicles during manufacturing. The most recent version of the standard dates back to 2015 [9]. QR codes improve over previous-generation one-dimensional barcodes by increasing recognition speed and storage capacity.

A. Technology

The main idea behind QR codes is to encode information on a printable support, customarily in the form of a grid of black and white square dots (see, e.g., Fig. 1), which can be easily read by the camera of widespread commercial devices like smartphones and tablets. The information contained in the QR code is then extracted from the acquired image and converted into a binary representation. As depicted in the figure, specific *finder patterns* and *alignment patterns* are included that the reading device software exploits to identify the orientation and the position of the QR code within the captured image.

This approach can be used to encode various types of information like text, URLs, Wi-Fi network information, etc., by storing four types of data: numeric, alphanumeric, binary, and kanji. The latter is intended for encoding Japanese symbols. The maximum amount of information that can be stored in the QR code is related to the selected data type. For eQR codes, numeric or binary codings were chosen to store the compiled QRscript program as a sequence of bits. In the case of numeric coding, the sequence of bits is interpreted as a base-10 number.

A concept of version is defined for QR codes to specify their size, in the range from 1 (21×21 matrix) up to 40 (177×177

matrix). For every version, four correction levels are possible: L (low), M (medium), Q (quartile), and H (high). They can be used to increase the overall robustness at the expense of the storage capacity. Version 40, with a low error correction level, is the combination that maximizes the storage capacity, that in this configuration is equal to 2953 bytes for binary coding and 7089 digits for numeric coding. By setting the appropriate version and compression level, a compromise between storage capacity and correction capability can be reached, depending on the characteristics of the application. Fig. 1 shows some examples of QR codes, including version 1, version 5, version 10, and other less usual formats. Refer to Fig. 1 of [8] for an example of a QR code using version 40.

B. State-of-the-Art

For the analysis of the state-of-the-art, all the journal papers that contain the string "QR code" or "QR codes" in the title and that were published in the past 5 years (between 2019 and April 2023) by the main publishers, including IEEE, Elsevier, Springer, Wiley, and MDPI, were analyzed, for a total of 115 papers. Then, we selected a subset including some of the most representative works, divided in turn between those focused on application contexts (26 papers) and those related to research trends (30 papers). Within each category, the main areas of interest/topics were identified.

1) Application contexts: QR code technology has been successfully applied in a variety of application contexts, including anti-counterfeiting, augmented reality, automatic configuration, fast reading, identification, localization, maintenance, medical, payment, recycling, safety, security, teaching, traceability, and tourism. This wide range of applications demonstrates that, nowadays, QR code usage embraces many different fields, making this technology increasingly pervasive both in daily use and in specialized contexts.

Selected works are summarized in Table I where, for every application context, the related papers are listed. For each paper, a brief description is also provided.

2) Research trends: Besides applications, intense research activity is currently ongoing aimed at improving the QR code technology. Table II reports a selection of research trends organized by the main topics. Starting from the end of the table, *security* is the topic with the largest number of contributions. Its main purpose is to make this technology more secure, for instance by embedding additional and possibly hidden information within the QR code. Research identified with the topic *printing type* is related to the use of special inks or to the printing of QR codes on unusual surfaces. The insertion of *multiple information* in the same QR code and techniques to *increase capacity* are the two sides of the same coin, and they are aimed to increase the amount of information that can be packed in a conventional QR code. The proposal and experimentation of new techniques to *improve recognition* quality is the typical research activity that aims to enhance the usability of this technology. *Anti-counterfeiting* and *improve other technologies* are other specific research activities pertaining to QR codes. Instead, *improve beauty* is a relevant research topic aimed at embedding QR codes

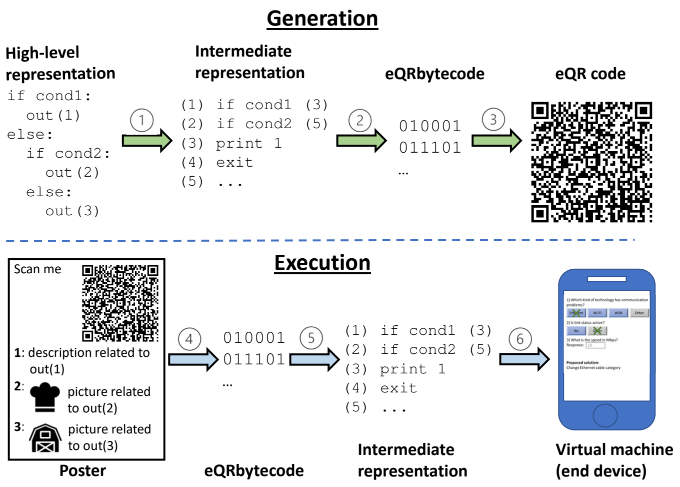


Fig. 2. Generation and execution chain for eQR codes.

in other pictures in order to make them more beautiful and hence more suitable to be adopted, for example, for advertising purposes. The work about eQR codes presented here, as well as in [8], belongs to the category of the new research trends. They have been categorized under the *programmability* topic because they are aimed at defining the best ways to make QR codes executable.

III. QRSCRIPT AND EQR CODES

An eQR code is a QR code that embeds a specific programming language denoted QRscript.

A. Generation and execution chain

Fig. 2 highlights the steps involved in the *generation* of an eQR code (on the top) and its subsequent *execution* on the end-user device (on the bottom).

The steps of the *generation* chain are typically performed by the developer of the application (a company, an association, an individual, etc.) and lead to the creation of the eQR code, which can be printed, painted, displayed, or visualized in some way to the end user. The first step of this chain (arrow ①) is the translation from a *high-level representation* (e.g., programming language or graphical representation) to an *intermediate representation*. Although a small illustrative example is provided in Section V, this translation step is out of the scope of this work, because complete control on the selection of the high-level representation must be provided to the user, to the point that the program could be written using the intermediate language representation directly. In other words, a potentially large number of high-level representations can be mapped to the same intermediate code. In the second generation step, which is represented by arrow ②, the intermediate representation is mapped to a binary representation, which was named *eQRbytecode*. Inserting the *eQRbytecode* inside an eQR code (represented by arrow ③) is quite trivial, since suitable libraries are publicly available that can be used for generating conventional QR codes.

In the inverse direction, denoted the *execution* chain, an eQR code that is visualized in some way to the end user

TABLE I
MAIN APPLICATION CONTEXT WHERE QR CODES ARE EXPLOITED.

Topic	Ref.	Description
Anti-counterfeiting	[10]	Implement and improve the quality of an anti-counterfeiting system through the use of QR codes
Augmented reality	[11]	QR codes for the personalization of objects displayed by virtual reality
Automatic config.	[12]	QR codes used to auto-configure Industrial Internet of Things sensor networks
Fast reading	[13]	Mobile application that allows reading densely placed QR codes
Identification	[14]	QR codes and robotic arms to automatically manage books in a library
Localization	[15]	QR codes used to improve the localization of mobile robots
	[16]	Use of QR codes placed in specific points to track the position of a person
	[17]	Use and placement of QR codes to enable navigation in indoor environments
Maintainance	[18]	QR codes are used to perform predictive maintenance and self-calibration of a robotic arm
Healthcare	[19]	Embedding of an ECG signal within a QR code
	[20]	Use of QR codes for authenticating medical images in a blockchain framework context
Payment	[21]	Improving security when QR codes are used for payment
	[22]	How the use of QR codes for mobile payments is perceived by users
	[23]	Use of QR codes to manage E-Wallets
Recycling	[24]	Use of QR codes to manage radioactive waste
	[25]	Combined use of QR codes and blockchains to build a recycling platform
Safety	[26]	Management of construction safety through QR codes
Security	[27]	Embed a secret within the QR code
	[28]	Manage encryption and decryption through watermarking in medical applications
	[29]	A comprehensive study about QR code applications from the point of view of security and privacy
Teaching	[30]	Integration of QR codes in teaching material and in classrooms to improve the quality of education
	[31]	Integrate QR codes and text to improve the learning of English as a foreign language
Traceability	[32]	Use of blockchains, explainable artificial intelligence, and QR codes for food traceability
	[33]	A review on the possibility of embedding information on manufactured parts
	[34]	Use of traceability to track vegetable supply chain
Tourism	[35]	QR codes for exploring mount Etna (volcano)

(e.g., printed on posters or stickers, possibly with additional information) is translated to a binary representation (arrow ④). The application executing on the end-user device (e.g., a smartphone) has to recognize and extract the eQR/QR code contained in a digital image (e.g., acquired using the camera of the smartphone), perform the related error correction algorithms, and obtain the sequence of bytes representing the eQRbytecode. Again, many libraries are currently available to perform this step. The eQRbytecode is then translated into an intermediate representation (arrow ⑤) and, finally, the intermediate representation is executed by an application (a sort of virtual machine) running in the end-user device (arrow ⑥). While running the encoded program, this virtual machine interacts with the end user through an input/output interface (e.g., the touch screen of the smartphone). This part of the translation chain is not analyzed in detail in this work, because the implementation of the user interface and the characteristics of the virtual machine strongly depend on the needs of both end users and the application developer.

The QRscript programming language specification only concerns the translation processes from the intermediate representation to eQRbytecode (arrow ②) and vice-versa (arrow ⑤).

B. QRscript and eQR codes specification

The *QRscript header* is the first part of the eQRbytecode, and specifies how to interpret the following part of the eQR code (Fig. 3). It is composed of five elements: *padding*, *continuation*, *security*, *URL*, and *dialect*. The QRscript header is directly followed by the *dialect code*, which represents the encoded program. A detailed description is available in the specification document [65].

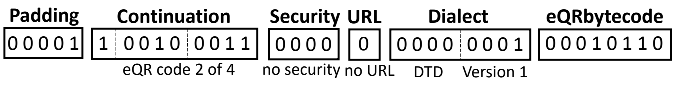


Fig. 3. QRscript header.

1) Dialect: Starting from the end, the *dialect* element specifies the programming language (i.e., the dialect) embedded in the eQR code. The ability to support more than one dialect permits to more effectively face the scarce storage capacity of QR codes. Specific dialects can be defined that provide different trade-offs between compactness and expressive power, which suit different application contexts. It is worth pointing out that any increase in the computational complexity for decoding and executing the program embedded in the eQR code due to the specific type of dialect (and the related instruction set) is, from our perspective, irrelevant. In fact, the small size of the code that can be stored in an eQR code makes the effort for dealing with it always negligible compared to the computational power of the target device (e.g., a smartphone). At the time of writing, the only dialect we defined is the *QRtree dialect*, which is aimed at implementing a decision

TABLE II
MAIN RESEARCH TRENDS ABOUT QR CODES (REFERENCE [**] REPRESENTS THIS PAPER).

Topic	Ref.	Description
Programmability	[**]	Embedding a program in a QR code to make it executable
Anti-counterfeiting	[36]	Placing hidden information within the QR code to prevent its copy/falsification
	[37]	Use of visual features combined with QR codes to guarantee the authenticity of a product
Improving beauty	[38]	Generation of artistic QR codes, which are embedded in a picture
	[39]	Embedding a QR code within a micrography image
	[40]	Embedding a QR code within a picture
	[41]	Inserting a QR code in a picture by using deep learning technology
Improving other technologies	[42]	Integrating QR codes and RFID technologies to decrease the complexity of RFID circuitry
Improving recognition	[43]	Using an adaptive method based on binarization to improve recognition quality
	[44]	Improving recognition quality when QR codes are placed in an uneven surface
	[45]	Reducing the size of information in a QR code to improve recognition performance
	[46]	A deblurring method is used to improve acquisition quality of QR codes
	[47]	Improving recognition quality of QR code by fastly restoring out-of-focus blurred images
	[48]	Removing perspective by identifying the vertex of a QR code
	[49]	Improve the performance in recognizing multiple QR codes in a picture
Increasing capacity	[50]	Increasing the capacity of a QR code by means of lossless compression
	[51]	Using colored QR codes to improve the storage capacity
	[52]	Improving the encoding efficiency of Chinese characters based on the use frequency
Multiple information	[53]	Embedding three layers of information within one QR code
	[54]	Embedding two layers of information within one QR code
	[55]	Placing a QR code inside another QR code and reading both with different angles
Printing type	[56]	QR codes printed with ink sensitive to pH for quality monitoring of food freshness
	[57]	QR code is printed on a sand core surface for traceability
	[58]	Printing the QR code by caving it on the surface of an object
Security	[59]	Embedding a secret within the QR code by using colors
	[60]	Embedding a secret within the QR code
	[61]	Embedding a secret within the QR code for authentication
	[62]	Using QR codes and singular value decomposition to encrypt images
	[63]	Embedding privacy information by using two layers in the QR code
	[64]	Protection for potential adversarial learning attacks using QR codes

tree inside the eQR code. In this work, it is described and analyzed starting from Section IV.

The dialect selector is encoded on (at least) 4 bits, followed by (at least) other 4 bits that specify its *version*. Both are coded using the *exponential encoding* (EC), which permits to encode integer quantities on a number of bits that grows exponentially. For instance, the current version (1) of the QRtree dialect is encoded as `0000 0001`. This representation technique is used in many other parts of the QRtree dialect.

2) Exponential encoding: The simple idea behind EC consists in doubling the number of bits used to encode an unsigned integer every time the *initially* allocated space (i.e., 4 bits for the QRtree dialect) is not enough to store it. For integer values between 0 and 14, the number of bits of the representation is 4 (e.g., $5_{10} = 0101_{\mathrm{EC}}$). Values between 15 and 29 are encoded on 8 bits, where the first 4 bits are set to 1111 (e.g., $18_{10} = 11110011_{\mathrm{EC}}$). Values between 30 and 284 are encoded on 16 bits, where the first 8 bits are set equal to 11111111 (e.g., $123_{10} = 1111111101011101_{\mathrm{EC}}$), and so on.

This representation guarantees both compactness and extensibility: 1) the number of bits used to encode any value is directly related to the value itself; 2) in theory, there is no upper limit on the values that can be encoded.

3) URL: In those situations when the end-user device has Internet access, the ability to exploit an *URL* to run a remote application in the place of the QRscript might be a more appropriate solution. Clearly, remote applications have almost no constraints about program size, and are likely to provide (much) better user interaction. Moreover, since they run on a remote web server, they could include a range of information types, like videos, sounds, and images, that can be hardly integrated within an eQR code for space reasons. This explains why we foresaw the option to include an URL within an eQR code. In particular, the URL bit identifies the presence of the URL. If the bit is 1, the subsequent bits encode the URL using UTF-8 encoding [66]. The character *end-of-text (EXT)* is used as the string terminator.

At the application level, it is possible to configure a specific behavior once the selected URL has been requested, which refers back to the execution of the eQR code, allowing the possibility of deactivating the URL even after its generation.

4) Security: Properly managing *security* is of utmost importance for eQR codes, mainly to ensure their *authenticity* and *integrity*. The execution of a fake eQR code with malicious code may cause serious damages, possibly leading users to perform potentially dangerous actions. For certain kinds of applications, where the information included in the eQR code should be only accessible by selected end users (e.g., privacy-protected data), *encryption* could be optionally enabled.

The security element in the QRscript header exploits the exponential coding with an initial size equal to 4 bits, which represents the *security profile* used for the specific eQR code. The security profile indicates: which type of security is enabled; the parts of the eQR code that are involved; the specific algorithm (e.g., RSA); and the length of the digital signature that is possibly included within the eQR code. Security is disabled when the security profile is set to `0000`. The application developer is in charge of defining a mapping between the values the security element can assume and security policies. This mapping is not defined in the specification document, and consequently it may differ for different applications.

5) Continuation: To overcome the size limit of QR codes, QRscript supports the concatenation of more than one eQR code. To this extent, the *continuation* element of the QR script header has been defined, which permits to identify the different sub-eQR codes to be merged, and in particular their overall number and order. After some basic operations for reconstructing a new header and concatenating the dialect codes, a new (concatenated) eQRbytecode is generated that can be directly executed by the end-user device. This makes it easy to double and even triplicate the maximum allowed size of eQR programs. In some contexts the application could be split over different eQR codes, placed in different locations and containing different pieces of program, which in their entirety enable the resolution of a joint decision problem.

6) Padding: The first bit of the eQRbytecode header identifies the presence of padding, which is needed only for QR codes with binary encoding in the case the length of the eQRbytecode is not an exact multiple of 8 bits. Differently from communication protocols, padding is added at the beginning of the eQR code (and not at its end), since it must be completely independent of the dialect that is embedded within the eQR code. In particular, `1` means no additional padding, whereas `01`, `001`, `0001` mean 1, 2, and 3 additional padding bits, respectively, and so on (the maximum size of the padding element is 8 bits). In the case of numeric coding, the eQRbytecode is expressed as a decimal number and the presence of padding does not affect conversion because leading 0s are not significant.

IV. QRtree dialect

QRtree is a dialect specifically aimed at coding a *decision tree* within an eQR code. A decision tree is a structure composed of three kinds of nodes, namely *decision*, *end*, and *chance* nodes. Chance nodes are not considered in QRtree, because they represent probabilistic decisions, which are not relevant for the application contexts to which this dialect applies. Decision nodes are the vertex of the tree, and represent the conditions that allow the user to choose among the different branches of the tree, based on its answers. End nodes are the leaves of the tree, which provide responses to the user. A simple example of a decision tree is shown in Fig. 4.a.

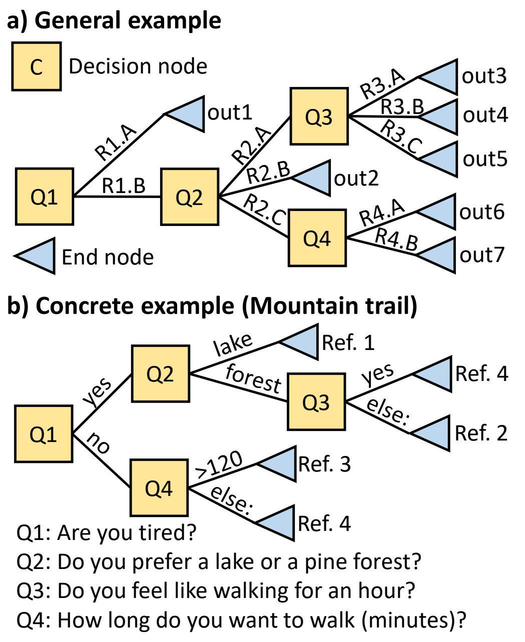


Fig. 4. Two examples of decision trees: a) general example showing their structure; b) specific and concrete example regarding mountain trails (it will be expanded and discussed in detail in Section V).

It is worth pointing out that, even if the QRtree was conceived to efficiently embed a decision tree in an eQR code, other (different) languages could be also encoded using the intermediate representation defined by the QRtree dialect. However, due to the absence of variables and backward jumps, most high-level programming languages cannot be represented through QRtree. In other words, the intermediate language of QRtree is not *Turing-complete*. We remark again that the QRtree dialect and the corresponding specification document [67] define only the conversion rules between intermediate language and eQRbytecode, which in the context of this dialect is named *eQRtreebytecode*.

The example that will be illustrated in Section V, sketched in Fig.4.b, considers an eQR code aimed to guide hikers on a mountain trail. The high-level programming language reported in Fig. 6 can be easily translated into the intermediate representation of Fig. 7. Details about the QRtree dialect in such intermediate representation and its translation to the eQRbytecode of Fig. 8, as well as the reverse process, are thoroughly explained in the four subsections below.

A. QRtree structure

The part of an eQR code that contains the QRtree dialect is divided into two sections: *QRtree header* and *QRtree body* (or *code*). The first section is optional, and permits to (re)configure the encoding rules of the instructions included in the subsequent code section.

The QRtree dialect does not permit the definition of variables. There is only one implicit variable, which is named `v_input`, that is defined at the beginning of the execution and refers to the value acquired by the end user by the most recent input instruction. In the case of decision trees, the content of this variable is needed to select the branches of the tree that lead to a final output represented by an end node. In particular, v_input can be compared with constants (i.e., literals), which consist in *strings*, *integers*, and *real* values.

B. Data types

The main data type of the QRtree dialect is the *string*, as it is used in both input and output instructions. Strings are quite space-consuming, and hence much attention was devoted to encoding efficiency in dialect definition. In particular, the first two bits of a string identify the type of the string: ASCII-7 (00), UTF-8 (01), and DICT (10). The ASCII-7 [68] encoding relies on 7-bit characters, hence it only supports English-like alphabets. This limitation is overcome by the UTF-8 [66] encoding, which is a variable-length encoding (8 to 32 bits are needed for every character) where the final length depends on the dimension of the used character set.

The DICT type refers to predefined strings that are identified through numeric references in output instructions (instead of the whole string). Three types of dictionaries are defined, identified by the bits that follow in the string, namely *global* (00), *specific* (01), and *local* (1). As highlighted in the specification document [67], if not all the three types of dictionaries are used, more compact representations are possible. Global dictionaries are generic dictionaries that contain frequently used words that suit a multitude of applications (e.g., "yes", "no", "maybe", etc.). Instead, specific dictionaries are typical of a given application context (e.g., "instruction manuals for boats", "mountains", "caves", etc.), and specific commands in the QRtree header can be used to activate them. Finally, local dictionaries are stored in the QRtree header itself. Typically, they can be used for strings repeated many times in the eQR code, which are stored only once in the QRtree header. Detection of such strings can be performed automatically during translation, by analyzing the occurrence of the strings within the eQR code and generating the corresponding local dictionary accordingly. The reference to a specific string within a dictionary is encoded as an unsigned integer using the exponential encoding described in Subsection III-B2.

Signed integer literals are encoded using the *two's complement* [69] on either 16 bits (INT16 type) or 32 bits (INT32 type). Instead, real literals can use either the half-precision (16 bits, FP16 type) [70] or the conventional single precision (32 bits, FP32 type) floating-point representations.

C. QRtree header

The *QRtree header* is optional, and its presence is indicated by setting the first bit of the eQRtreebytecode to 1. It consists of a list of 6 different types of commands, each one encoded on 3 bits with the exponential encoding. A specific HEADER_END command is defined as the terminator of the list.

More in detail, INT_TYPE and FLOAT_TYPE commands are used to select a specific representation for integers (INT16 or INT32) and reals (FP16 or FP32) values. This saves space, since specifying the representation of every single number is no longer required. The command DICT_TYPES is used to activate *global* and/or *specific* dictionaries, whereas DICT_SPEC_TYPE permits to select which *specific* dictionaries (one or more) are active among those defined by the application developer, specified using the JSON format [67]. Finally, the DICT_LOCAL command permits to embed the *local* dictionary within the eQR code by using the specific syntax defined in [67].

D. Code

The *code* section follows the QRtree header, and contains the list of instructions that make up the program.

1) Intermediate representation: From the point of view of the intermediate representation, the seven instructions defined by the QRtree dialect are reported in Table III. An example of intermediate representation, which refers to the application example described in Section V, can be found in Fig. 7. It uses a *three-address* code, in which every instruction is identified/labeled with an integer in increasing order.

All the four input/output instructions (input, inputs, print, and printex) display information for the user on the screen, either a <string> (defined in Subsection IV-B) or a <reference>. The binary value <type> is used to distinguish between the two cases.

The option to output a *reference* (an *exponentially encoded* unsigned integer that points to an external source, e.g., a printed description) is a valuable feature of the QRtree dialect. One of the benefits of eQR codes, and QR codes in general, is that they can be printed on a physical support with some extra information next to them, like text or images. On the one hand, this can help users to understand how to use the eQR code and what is its purpose. On the other hand, references permit to move some parts of the program (e.g., long strings) outside the eQR code, leading to a consistent saving in terms of space. For example, strings related to explanatory text (descriptions/instructions) can be printed on the same sheet as the eQR code instead of being embedded within it. In this case, the program can simply output an indication to the user, inviting her/him to look at a certain reference. Besides strings, references can point to other kinds of information that cannot be easily embedded within the eQR code, like pictures and maps.

Another interesting point concerns input. In the typical case of *direct* input (instruction inputs), the answer is fed directly by the user through a textual interface. For *indirect* input (instruction input), the answer is obtained, for instance, by means of decision buttons that, for the execution step (arrow ⑥ of Fig. 2), are automatically derived from the if instructions of the intermediate representation. Indirect input permits to design more user-friendly user interfaces for the applications executed on end-user devices.

The goto instruction is related to unconditional jumps, while if and ifc instructions refer to conditional jumps. In particular, the if instruction compares the last string provided by the user, stored in v_input, with its <string> argument. If they match, a jump is performed to the corresponding label. Similarly, the ifc instruction performs a comparison between v_input and its operand <op>, using relational operator <rel_op>, chosen among == for equality, != for

TABLE III
INSTRUCTIONS DEFINED FOR THE INTERMEDIATE REPRESENTATION OF THE QRTREE DIALECT.

Instruction	Syntax and description
`input`	`input <type> (<string>|<reference>)` Print a `<string>` or a `<reference>` on the screen, followed by a *indirect* input. Entered value stored in `v_input`.
`inputs`	`inputs <type> (<string>|<reference>)` Print a `<string>` or a `<reference>` on the screen, followed by a *direct* input. Entered value stored in `v_input`.
`print`	`print <type> (<string>|<reference>)` Print a `<string>` or a `<reference>` on the screen.
`printex`	`printex <type> (<string>|<reference>)` Print a `<string>` or a `<reference>` on the screen. After that, the execution is terminated.
`goto`	`goto (x)` Jump to the instruction identified with the label `(x)`.
`if`	`if <string> (x)` Jump to the instruction identified with the label `(x)`, if the last entered value `v_input` is equal to `<string>`.
`ifc`	`ifc <rel_op> <op> (x)` Jump to the instruction identified with the label `(x)`, if the comparison `v_input <rel_op> <op>` is true. Available `<rel_op>` are == (equal), != (not equal), <=, >=, <, and >.

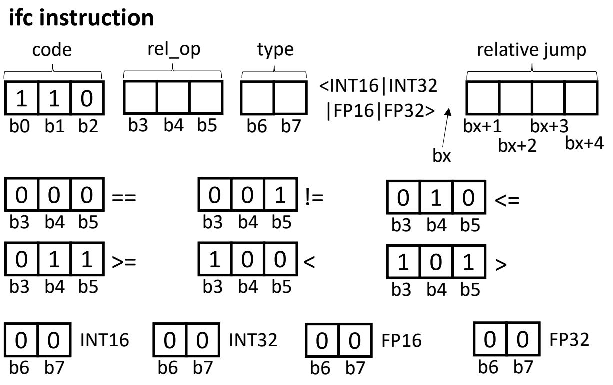


Fig. 5. Conversion rules from intermediate representation to eQRtreebytecode of the `ifc` instruction.

inequality, <=, >=, <, and > for order. Again, if the result of the comparison is true, a jump to the label is performed. The operand is one of the numeric datatypes defined in Subsection IV-B, namely, `INT16`, `INT32`, `FP16`, or `FP32`. The virtual machine running in the end-user application is in charge of performing the required data type conversions of `v_input` to the type of the operand.

Since the QRtree dialect does not support loops, only forward jumps have been defined, i.e., those which point to instructions that follow the current one.

2) eQRtreebytecode: A set of rules is defined for converting instructions from the intermediate representation to the binary eQRtreebytecode and vice-versa. The first three bits identify the instruction. Pattern `111` was left unused to deal with possible extensions of the instruction set. Every instruction has its own conversion rules, which are specific to its behavior and the operands involved in its execution.

As a first, simple example, the `goto` instruction is identified by the bit pattern `100`, which is followed by a relative jump specified by the number of instructions to skip, coded using the *exponential encoding*. Thus, the eQRtreebytecode fragment `100 0000` means to jump to the next instruction.

The most complex instruction is `ifc`, whose conversion rules are reported in Fig. 5. In detail, the initial `110` pattern identifies the `ifc` instruction, the next three bits identify the relational operator to be used (`rel_op`), followed by two bits that identify the `type` of the operand, either `INT16`, `INT32`, `FP16`, or `FP32` (depending on the settings in the QRtree header, more compact notations are possible for `type`). The `type` field defines how to interpret the following bits, which encode the literal, and what type of automatic conversion must be applied to `v_input`. Finally, exactly as in the `goto` instruction, we find the relative jump to be performed when the result of the comparison is true.

Details about the translation rules of the other instructions are reported in the QRtree dialect specification document [67].

V. APPLICATION EXAMPLE

As a concrete, simple example of the QRtree dialect, an eQR code is illustrated below that, once suitably placed on a mountain trail, helps hikers to choose a specific destination among several alternatives, according to their capabilities and preferences.

A simple compilation chain named `QRtree` [71], which relies on `python` and implements all the mandatory features listed in the QRtree specification document [67], was developed and released under the GPL-3.0 license. All the steps of the example described in this section can be reproduced by executing the file named `ex01-IEEE_IoT-J_mountain_routes.txt`, which is located in the `examples` directory of the `QRtree` software. In the same directory another application example is included, named `ex02-IEEE_IoT-J_defibrillator.txt`, which is aimed to guide the user in correctly operating a defibrillator. The availability of easy-to-use technology like eQR codes could make the difference in case of emergency.

A. eQR code Generation

The idea behind this example is to guide a hiker in the selection of the destination that best fits her/his current conditions and wishes, by asking a series of specific questions. In the

```
input "Are you tired?"
if "yes":
  input "Do you prefer a lake or a pine forest?"
  if "lake":
    print "Pay attention to the cross with the busy road"
    print 1 # Reference to lake indications
    exit
  else if "forest":
    input "Do you feel like walking for an hour?"
    if "yes":
      print 4 # Ref. to Madonna della Neve hut (1595 meters)
      print "You will find the forest on the way"
      exit
    else:
      print 2 # Reference to pine forest indications
      exit
else if "no":
  inputs "How long do you want to walk (minutes)?"
  ifc > 120:
    print 3 # Reference to Rivetti hut (2150 meters)
    exit
  else:
    print 4 # Ref. to Madonna della Neve hut (1595 meters)
    exit
```

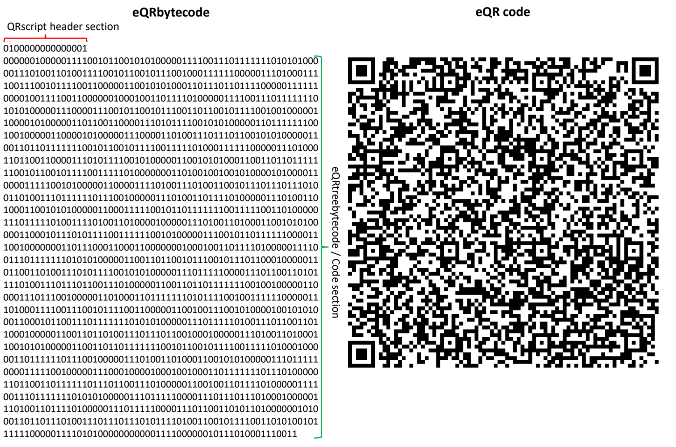
Fig. 6. Example of a high-level programming language that describes the decision tree for helping mountain hikers.

generation chain sketched in Fig. 2, the application designer develops a decision tree, using a high-level programming language, that asks the hiker a number of questions and then provides the related advice. With reference to Fig. 6, the first question: “Are you tired?” permits to check the level of fatigue of the hiker. In the case the user is tired (“yes” response), the program proposes two alternatives within a short walking distance, “lake” or “forest”. The first option leads to reference “1”, while in the case the answer is “forest” the program stored in the eQR code further investigates if the user is possibly willing to walk for one more hour (by means of the question “Do you feel like walking for an hour?”). The related indications, for the “yes” and “no” (better, not “yes”) answers, are provided as references “4” and “2”, respectively.. Conversely, if the user is still lively (i.e., the answer to the first question is “no”), the program asks: “How long do you want to walk (minutes)?”, which requests her/him to enter a number expressed in minutes. This leads to recommending the “Alfredo Rivetti hut” (reference “3”) if the answer is greater than 120 (two hours), or the “Madonna della Neve hut” otherwise. Starting from this description, the intermediate representation reported in Fig. 7 can be automatically obtained by using the rules reported in Section IV and in the specification documents [65], [67].

The result of the conversion from the intermediate representation (in terms of the corresponding eQRbytecode binary representation) and the generated eQR code is reported in Fig. 8. As can be seen, the QRscript header starts with the padding `01`, while continuation is disabled (the next bit is set to `0`). Also security is disabled (`0000`), as well as the URL (`0`). The dialect is “QRtree” (`0000`), and its version is 1 (`0001`). The length of the eQRbytecode for this simple decision tree is 1872 bits (234 bytes), which corresponds to just 7.9% of the maximum capacity of a QR code. Most of the space is occupied by strings. It is worth observing that real applications are typically more complex, which likely means that they need to exploit a larger portion of the available space of the QR code.

```
(0) input "Are you tired?"
(1) if "yes" (4)
(2) if "no" (18)
(3) goto (22)
(4) input "Do you prefer a lake or a pine forest?"
(5) if "lake" (8)
(6) if "forest" (11)
(7) goto (17)
(8) print "Pay attention to the cross with the busy road"
(9) printex 1
(10) goto (17)
(11) input "Do you feel like walking for an hour?"
(12) if "yes" (15)
(13) printex 2
(14) goto (17)
(15) print 4
(16) printex "You will find the forest on the way"
(17) goto (22)
(18) inputs "How long do you want to walk (minutes)?"
(19) ifc > 120 (21)
(20) printex 4
(21) printex 3
```

Fig. 7. A possible intermediate representation obtained from the high-level programming language of the example.



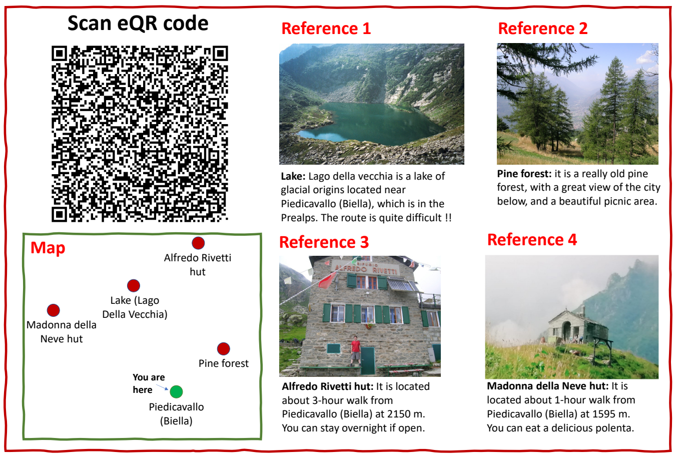
```
0100000000000001
000000100000111100101100101010000011110011101111111010101000
001110100110100111100101100101110010001111110000011101000111
100111001011110011000001100101010001101110110111100000111111
000010011110011000000100010011011110100000111100111011111110
101010000011100001110010110010111001101100101111001001000001
100001010000011011001100001110101111001010100000110111111100
100100000110000101000001110000110100111011101100101010000011
001101101111111001011001011110011111010001111110000011101000
110110011000011101011110010100000110010101000110011011011111
110010110010111100111110100000001101001001001010000101000011
000011111001010000011000011110100111010011001011101110111010
011010011101111111011100100000111010011011110100000111010011 0
100011001010100000110001111100101101111111001111100110100000
111011111010011110100110100001000001110100110100011001010100
000110001011101011110011111100101000001110010110111111000011
100100000001101110001100011000000001000100110111101000001111 0
011101111111010101000001100110110010111001011101100010000011
011001101001110101111001010100000111011111000011101100110101
111010011101110110011101000001100110110111111100100100000110
000111011100100000110100011011111110101111001001111110000011
101000111100111001011110011000001100100111001010000100101010
001100010110011101111111010101000001110111110100111011001101
100010000001100110110100111011101100100010000011101001101000 1
100101010000011001101101111111001011001011110011111010001000
001101111110111001000001110100110100011001010100000111011111
000011111001000000111000100001000100100011011111110111010000 0
110110011011111101110110011101000001100100110111101000001111
001110111111101010100000111011111000011101110111010001000001
110100110111101000001110111110000111011001101011010000001010
001101101110100111011101110101111010011001011110011010100101
11111000001111010100000000000011110000000101110100011100 11
```

Fig. 8. The eQRbytecode obtained from the intermediate representation and the corresponding eQR code.

B. eQR code Execution

The *execution* chain permits to run the program embedded in an eQR code. The eQR code to be executed is typically embedded in a poster (other kinds of support are also possible) together with additional information, including pictures, textual descriptions, and maps. For example, the poster depicted in Fig. 10 includes a *map* that shows hikers where the various reachable places are located. The program considered in the current example outputs one of four possible references. For every reference, the poster includes a picture that visually shows the destination and some text that provides additional information. The poster could be placed, in the form of an indication sign, near a crossroad that permits to reach the four destinations.

Let us assume that there is no Internet coverage. When the end user (a hiker, in the example) scans the eQR code with the camera of her/his (unconnected) smartphone, a suitable app (pre-installed in the device) transforms the content of the eQR code into the binary eQRbytecode representation, which can be directly translated into the intermediate representation. Starting from it, execution is possible by means of, e.g., a suitable



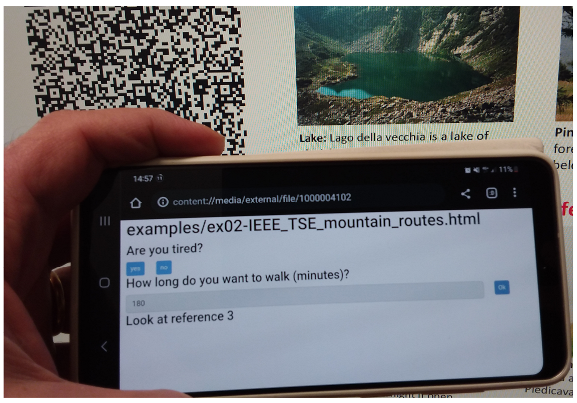
Lake: Lago della vecchia is a lake of glacial origins located near Piedicavallo (Biella), which is in the Prealps. The route is quite difficult !!

Pine forest: it is a really old pine forest, with a great view of the city below, and a beautiful picnic area.

Alfredo Rivetti hut: It is located about 3-hour walk from Piedicavallo (Biella) at 2150 m. You can stay overnight if open.

Madonna della Neve hut: It is located about 1-hour walk from Piedicavallo (Biella) at 1595 m. You can eat a delicious polenta.

Fig. 9. Sample eQR code printed on a sheet of paper along with additional information (map, images, text). The poster can be used as indication sign.



examples/ex02-IEEE_TSE_mountain_routes.html

Are you tired?

How long do you want to walk (minutes)?

180

Look at reference 3

Fig. 10. Instance of execution of the QRscript (QRtree dialect) embedded in the sample eQR code on a smartphone (virtual machine).

virtual machine. In our proof-of-concept, eQR execution relies on a JavaScript application that can be run on any browser. Although this implementation is clearly not optimized (we are not pursuing a fully-engineered commercial product at this time), it can be tested on both mobile platforms (Android and iOS) and conventional devices (Windows, Linux, and macOS).

The screenshot in Fig. 10 refers to a specific execution instance of the eQR code. The application initially asks the hiker if she/he is tired. Following the negative answer ("no"), the application asks how many minutes she/he intends to keep walking. Since the response is 180, which is greater than 120, the application outputs a message that invites to take a look at reference 3, which corresponds to the "Alfredo Rivetti hut".

Although extremely simple, the above example shows all the steps involved in both the generation and the execution chain that apply to a concrete scenario concerning tourism, including a sample user interaction. Similarly to QR codes, which were conceived for industrial applications but can now be found everywhere in our lives, the possibilities offered by eQR codes are likely uncountable. Contexts in the real world that can benefit from this technology include methods to help people to easily find solutions to their problems (or simply the information they are looking for) when an Internet connection is lacking, and even to improve their safety by providing unambiguous guided instructions when needed.

C. Comparison

Since the room available in QR codes is scarce, the most interesting metric for assessing solutions like ours is code size. To the best of our knowledge, there are at the moment no competing solutions conceived for similar purposes. For this reason, we compared the size of the eQR code for the above example (see Fig. 8) with a semantically equivalent program (i.e., which does exactly the same things: the same sequence of questions are asked to the user and, for the same answers, the same advice is provided) written in Python. For the latter, the size of the source code is $537\,\mathrm{B}$ (i.e., occupation is 18.2%), while the related bytecode takes $634\,\mathrm{B}$ (21.5%). It is worth pointing out that, when the size of data is small (as in our case), generic compression techniques like ZIP and RAR are ineffective. Conversely, QRtree usage managed to shrink code size to $234\,\mathrm{B}$ (7.9%), which is noticeably less than alternatives and permits packing more complex algorithms inside an eQR code. It should be noted that, the lower the overall size of the strings, the higher the relative space saving achieved by eQR codes compared to other encodings.

VI. Conclusions

As highlighted by the analysis we performed on the state-of-the-art about QR codes, reported at the beginning of this work, the past years witnessed a promising trend concerning the research activities on this technology. The ability to embed an executable program (denoted QRscript) inside them, and a formal and agreed definition of eQR codes, are an essential step that further enhances QR codes and brings them new life. QRscript was conceived bearing extensibility in mind, and permits embedding a variety of dialects. In this work, the QRtree dialect is defined that specifically permits to encode decision trees. Its usability and potential have been illustrated through a real-life application example.

eQR codes close a significant gap regarding the ability to embed algorithms in a QR code, in such a way to enable interactive behavior, which somehow resembles the Internet of Things, also when Internet is unavailable and both the system (object) and its user (human being) are offline. For this reason we believe that they have the potential to know a wide diffusion in the coming years. Future work includes, possibly, the standardization of the specification documents about eQR codes referred to in this work and the definition of new dialects, including a general-purpose language that fully supports structured algorithms.

Acknowledgment

Stefano Scanzio is grateful to Prof. Silvano Rivoira for all the years they spent teaching together the course of Formal Languages and Compilers at the Politecnico di Torino, and for all he learned from him on this subject. This work derives in part from that knowledge.

References

[1] S. Tiwari, "An Introduction to QR Code Technology," in *International Conference on Information Technology (ICIT 2016)*, 2016, pp. 39–44.

[2] C. Teuta, S. P. Payal, A. Ramesh, and T. Sakaguchi, "QR Code: A New Opportunity for Effective Mobile Marketing," *Journal of Mobile Technologies, Knowledge and Society*, vol. 2013, p. ID748267, 2013.

[3] C. Chen, "QR Code Authentication with Embedded Message Authentication Code," *Mobile Networks and Applications*, vol. 22, pp. 383–394, 2017. [Online]. Available: https://doi.org/10.1007/s11036-016-0772-y

[4] K. Saranya, R. Reminaa, and S. Subhitsha, "Modern applications of QR-Code for security," in *IEEE International Conference on Engineering and Technology (ICETECH 2016)*, 2016, pp. 173–177.

[5] T.-W. Kan, C.-H. Teng, and W.-S. Chou, "Applying QR Code in Augmented Reality Applications," in *Proceedings of the 8th International Conference on Virtual Reality Continuum and Its Applications in Industry*, ser. VRCAI '09. New York, NY, USA: Association for Computing Machinery, 2009, pp. 253–257. [Online]. Available: https://doi.org/10.1145/1670252.1670305

[6] S. L. Fong, D. Wui Yung Chin, R. A. Abbas, A. Jamal, and F. Y. H. Ahmed, "Smart City Bus Application With QR Code: A Review," in *IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS 2019)*, 2019, pp. 34–39.

[7] Z. Meng, Z. Wu, and J. Gray, "Rfid-based object-centric data management framework for smart manufacturing applications," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2706–2716, 2019.

[8] S. Scanzio, G. Cena, and A. Valenzano, "QRscript: Embedding a Programming Language in QR codes to support Decision and Management," in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2022, pp. 1–8.

[9] ISO Central Secretary, "Information technology — Automatic identification and data capture techniques — QR Code bar code symbology specification," International Organization for Standardization, Geneva, CH, Standard ISO/IEC 18004:2015, 2015. [Online]. Available: https://www.iso.org/standard/62021.html

[10] Y. Yan, Z. Zou, H. Xie, Y. Gao, and L. Zheng, "An IoT-Based Anti-Counterfeiting System Using Visual Features on QR Code," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6789–6799, 2021.

[11] P.-Y. Lin, W.-C. Wu, and J.-H. Yang, "A QR Code-Based Approach to Differentiating the Display of Augmented Reality Content, journal = Applied Sciences," vol. 11, no. 24, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/24/11801

[12] S. S. Madsen, A. Q. Santos, and B. N. Jørgensen, "A QR code based framework for auto-configuration of IoT sensor networks in buildings," *Energy Informatics*, pp. 46–volume = 4, url = https://doi.org/10.1186/s42 162–021–00 152–w, doi = 10.1186/s42 162–021–00 152–w, 2021.

[13] X. Tian, S. Qin, B. Jiang, Y. Gao, and X. Wang, "Fast Batch Reading Densely Deployed QR Codes," *IEEE Transactions on Mobile Computing*, vol. 22, no. 3, pp. 1507–1520, 2023.

[14] X. Yu, Z. Fan, H. Wan, Y. He, J. Du, N. Li, Z. Yuan, and G. Xiao, "Positioning, Navigation, and Book Accessing/Returning in an Autonomous Library Robot using Integrated Binocular Vision and QR Code Identification Systems," *Sensors*, vol. 19, no. 4, 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/4/783

[15] S.-H. Bach, P.-B. Khoi, and S.-Y. Yi, "Application of QR Code for Localization and Navigation of Indoor Mobile Robot," *IEEE Access*, vol. 11, pp. 28 384–28 390, 2023.

[16] T. Liu, J. Kuang, W. Ge, P. Zhang, and X. Niu, "A Simple Positioning System for Large-Scale Indoor Patrol Inspection Using Foot-Mounted INS, QR Code Control Points, and Smartphone," *IEEE Sensors Journal*, vol. 21, no. 4, pp. 4938–4948, 2021.

[17] J. Yan, J. B. Lee, S. Zlatanova, A. A. Diakité, and H. Kim, "Navigation network derivation for QR code-based indoor pedestrian path planning," *Transactions in GIS*, vol. 26, no. 3, pp. 1240–1255, 2022. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/tgis.12912

[18] Y. Zhang, W. Zhu, and A. Rosendo, "QR Code-Based Self-Calibration for a Fault-Tolerant Industrial Robot Arm," *IEEE Access*, vol. 7, pp. 73 349–73 356, 2019.

[19] P. Mathivanan and A. Balaji Ganesh, "QR code based color image cryptography for the secured transmission of ECG signal," *Multimedia Tools and Applications*, vol. 78, pp. 6763–6786, 2019. [Online]. Available: https://doi.org/10.1007/s11042-018-6471-x

[20] W. Wen, Y. Jian, Y. Fang, Y. Zhang, and B. Qiu, "Authenticable medical image-sharing scheme based on embedded small shadow QR code and blockchain framework," *Multimedia Systems*, vol. 29, pp. 831–845, 2023. [Online]. Available: https://doi.org/10.1007/s00530-022-00999-3

[21] Y. Zhou, B. Hu, Y. Zhang, and W. Cai, "Implementation of Cryptographic Algorithm in Dynamic QR Code Payment System and Its Performance," *IEEE Access*, vol. 9, pp. 122 362–122 372, 2021.

[22] B. A. Eren, "QR code m-payment from a customer experience perspective," *Journal of Financial Services Marketing*, 2022. [Online]. Available: https://doi.org/10.1057/s41264-022-00186-5

[23] F. A. A. Ramli, M. I. Hamzah, S. N. Wahab, and R. Shekhar, "Modeling the Brand Equity and Usage Intention of QR-Code E-Wallets," *FinTech*, vol. 2, no. 2, pp. 205–220, 2023. [Online]. Available: https://www.mdpi.com/2674-1032/2/2/13

[24] J.-W. Lee, J.-Y. Jeong, J.-J. Kim, H.-S. Park, and S. Chae, "A Note on the Design of Waste Management System Using QR Code for Radioactive Waste," *Sustainability*, vol. 14, no. 15, 2022. [Online]. Available: https://www.mdpi.com/2071-1050/14/15/9265

[25] E. Borandag, "A Blockchain-Based Recycling Platform Using Image Processing, QR Codes, and IoT System," *Sustainability*, vol. 15, no. 7, 2023. [Online]. Available: https://www.mdpi.com/2071-1050/15/7/6116

[26] J.-S. Kim, C.-Y. Yi, and Y.-J. Park, "Image Processing and QR Code Application Method for Construction Safety Management," *Applied Sciences*, vol. 11, no. 10, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/10/4400

[27] M. Alajmi, I. Elashry, H. S. El-Sayed, and O. S. Farag Allah, "Steganography of Encrypted Messages Inside Valid QR Codes," *IEEE Access*, vol. 8, pp. 27 861–27 873, 2020.

[28] J. Liu, J. Han, K. Fu, J. Jia, D. Zhu, and G. Zhai, "Application of QR Code Watermarking and Encryption in the Protection of Data Privacy of Intelligent Mouth-Opening Trainer," *IEEE Internet of Things Journal*, vol. 10, no. 12, pp. 10 510–10 518, 2023.

[29] H. A. M. Wahsheh and F. L. Luccio, "Security and Privacy of QR Code Applications: A Comprehensive Study, General Guidelines and Solutions," *Information*, vol. 11, no. 4, 2020. [Online]. Available: https://www.mdpi.com/2078-2489/11/4/217

[30] S. N. Abdul Rabu, H. Hussin, and B. Bervell, "QR code utilization in a large classroom: Higher education students' initial perceptions," *Education and Information Technologies*, vol. 24, pp. 359–384, 2019. [Online]. Available: https://doi.org/10.1007/s10639-018-9779-2

[31] S. Kuru Gönen and G. Zeybek, "Using QR code enhanced authentic texts in EFL extensive reading: a qualitative study on student perceptions," *Education and Information Technologies*, vol. 27, pp. 2039–2057, 2022. [Online]. Available: https://doi.org/10.1007/s10639-021-10695-w

[32] S. Bhatia and A. S. Albarrak, "A Blockchain-Driven Food Supply Chain Management Using QR Code and XAI-Faster RCNN Architecture," *Sustainability*, vol. 15, no. 3, 2023. [Online]. Available: https://www.mdpi.com/2071-1050/15/3/2579

[33] M. Usama and U. Yaman, "Embedding Information into or onto Additively Manufactured Parts: A Review of QR Codes, Steganography and Watermarking Methods," *Materials*, vol. 15, no. 7, 2022. [Online]. Available: https://www.mdpi.com/1996-1944/15/7/2596

[34] Y. Dong, Z. Fu, S. Stankovski, S. Wang, and X. Li, "Nutritional Quality and Safety Traceability System for China's Leafy Vegetable Supply Chain Based on Fault Tree Analysis and QR Code," *IEEE Access*, vol. 8, pp. 161 261–161 275, 2020.

[35] F. Pasquaré Mariotto, F. L. Bonali, A. Tibaldi, E. De Beni, N. Corti, E. Russo, L. Fallati, M. Cantarero, and M. Neri, "A New Way to Explore Volcanic Areas: QR-Code-Based Virtual Geotrail at Mt. Etna Volcano, Italy," *Land*, vol. 11, no. 3, 2022. [Online]. Available: https://www.mdpi.com/2073-445X/11/3/377

[36] T. Wang, H. Zheng, C. You, and J. Ju, "A Texture-Hidden Anti-Counterfeiting QR Code and Authentication Method," *Sensors*, vol. 23, no. 2, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/2/795

[37] Y. Yan, Z. Zou, H. Xie, Y. Gao, and L. Zheng, "An IoT-Based Anti-Counterfeiting System Using Visual Features on QR Code," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6789–6799, 2021.

[38] M. Xu, H. Su, Y. Li, X. Li, J. Liao, J. Niu, P. Lv, and B. Zhou, "Stylized Aesthetic QR Code," *IEEE Transactions on Multimedia*, vol. 21, no. 8, pp. 1960–1970, 2019.

[39] S.-H. Hung, C.-Y. Yao, Y.-J. Fang, P. Tan, R.-R. Lee, A. Sheffer, and H.-K. Chu, "Micrography QR Codes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 9, pp. 2834–2847, 2020.

[40] M.-J. Tsai and S.-L. Peng, "QR code beautification by instance segmentation (IS-QR)," *Digital Signal Processing*, vol. 133, p. 103887, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1051200422005048

[41] M.-J. Tsai, H.-Y. Wu, and D.-T. Lin, "Auto ROI mask R-CNN model for QR code beautification (ARM-QR)," *Multimedia Systems*, 2023. [Online]. Available: https://doi.org/10.1007/s00530-022-01046-x

[42] J. Sun, K. Shrestha, H. Park, P. Yadav, S. Parajuli, S. Lee, S. Shrestha, G. R. Koirala, Y. Kim, K. A. Marotrao, B. B. Maskey, O. C. Olaoluwa, J. Park, H. Jang, N. Lim, Y. Jung, and G. Cho, "Bridging R2R Printed Wireless 1 Bit-Code Generator with an Electrophoretic QR Code Acting as WORM for NFC Carrier Enabled Authentication Label," *Advanced Materials Technologies*, vol. 5, no. 2, p. 1900935, 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/admt.201900935

[43] R. Chen, W. Li, K. Lan, J. Xiao, L. Wang, and X. Lu, "Fast Adaptive Binarization of QR Code Images for Automatic Sorting in Logistics Systems," *Electronics*, vol. 12, no. 2, 2023. [Online]. Available: https://www.mdpi.com/2079-9292/12/2/286

[44] G. Papp, M. Hoffmann, and I. Papp, "Embedding QR Code onto Triangulated Meshes using Horizon Based Ambient Occlusion," *Computer Graphics Forum*, vol. 41, no. 1, pp. 29–45, 2022. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14394

[45] H.-C. Liu, T.-R. Chou, C.-S. Lu, and H.-C. Wang, "Improving readability by modifying graphic QR code microstructure," *Electronics Letters*, vol. 57, no. 23, pp. 879–881, 2021. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/ell2.12301

[46] R. Chen, Z. Zheng, J. Pan, Y. Yu, H. Zhao, and J. Ren, "Fast Blind Deblurring of QR Code Images Based on Adaptive Scale Control," *Mobile Networks and Applications*, vol. 26, pp. 2472–2487, 2021. [Online]. Available: https://doi.org/10.1007/s11036-021-01780-y

[47] R. Chen, Z. Zheng, Y. Yu, H. Zhao, J. Ren, and H.-Z. Tan, "Fast Restoration for Out-of-Focus Blurred Images of QR Code With Edge Prior Information via Image Sensing," *IEEE Sensors Journal*, vol. 21, no. 16, pp. 18 222–18 236, 2021.

[48] H. Eugênio Gonçalves, L. Xavier Medeiros, and A. Coutinho Mateus, "Algorithm for Locating the Vertices of a QR Code and Removing Perspective," *IEEE Latin America Transactions*, vol. 19, no. 11, pp. 1933–1940, 2021.

[49] R. Chen, H. Huang, Y. Yu, J. Ren, P. Wang, H. Zhao, and X. Lu, "Rapid Detection of Multi-QR Codes Based on Multistage Stepwise Discrimination and A Compressed MobileNet," *IEEE Internet of Things Journal*, pp. 1–1, 2023.

[50] A. Mohammed Ali and A. K. Farhan, "Enhancement of QR Code Capacity by Encrypted Lossless Compression Technology for Verification of Secure E-Document," *IEEE Access*, vol. 8, pp. 27 448–27 458, 2020.

[51] Y. Huang, P. Cao, and J. Li, "Research on multiplexed colour QR code with direct readability," *Electronics Letters*, vol. 58, no. 8, pp. 309–311, 2022. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/ell2.12433

[52] B. Kang, J. Jia, W. Gao, and N. Zhang, "Research on Improved Character Encoding Methods Based on QR Code," *Chinese Journal of Electronics*, vol. 28, no. 6, pp. 1170–1176, 2019. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/cje.2019.07.005

[53] S. Liu, Z. Fu, and B. Yu, "Rich QR Codes With Three-Layer Information Using Hamming Code," *IEEE Access*, vol. 7, pp. 78 640–78 651, 2019.

[54] T. Yuan, Y. Wang, K. Xu, R. R. Martin, and S.-M. Hu, "Two-Layer QR Codes," *IEEE Transactions on Image Processing*, vol. 28, no. 9, pp. 4413–4428, 2019.

[55] G.-J. Chou and R.-Z. Wang, "The Nested QR Code," *IEEE Signal Processing Letters*, vol. 27, pp. 1230–1234, 2020.

[56] Y. Xu, Z. Liu, R. Liu, M. Luo, Q. Wang, L. Cao, and S. Ye, "Inkjet-printed pH-sensitive QR code labels for real-time food freshness monitoring," *Journal of Materials Science*, vol. 56, pp. 18 453–18 462, 2021. [Online]. Available: https://doi.org/10.1007/s10853-021-06477-x

[57] L. Song, W. Liu, X. Zou, H. Huo, P. Guo, Y. Yu, and C. Wen, "Research on a Traceability Process of Sand Core Information by Printing QR Code on Sand Core Surface in the Casting Production Process," *International Journal of Metalcasting*, vol. 15, pp. 1476–1482, 2021. [Online]. Available: https://doi.org/10.1007/s40962-021-00572-0

[58] H. Peng, P. Liu, L. Lu, A. Sharf, L. Liu, D. Lischinski, and B. Chen, "Fabricable Unobtrusive 3D-QR-Codes with Directional Light," *Computer Graphics Forum*, vol. 39, no. 5, pp. 15–27, 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14065

[59] J.-S. Pan, T. Liu, B. Yan, H.-M. Yang, and S.-C. Chu, "Using color QR codes for QR code secret sharing," *Multimedia Tools and Applications*, vol. 81, pp. 15 545–15 563, 2022. [Online]. Available: https://doi.org/10.1007/s11042-022-12423-z

[60] P.-C. Huang, C.-C. Chang, Y.-H. Li, and Y. Liu, "Efficient QR Code Secret Embedding Mechanism Based on Hamming Code," *IEEE Access*, vol. 8, pp. 86 706–86 714, 2020.

[61] L. Xiong, X. Zhong, N. N. Xiong, and R. W. Liu, "QR-3S: A High Payload QR Code Secret Sharing System for Industrial Internet of Things in 6G Networks," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 10, pp. 7213–7222, 2021.

[62] M. Wang, X. Yang, X. Meng, Y. Wang, Y. Yin, and G. Dong, "Multi-image encryption based on QR code and singular value decomposition ghost imaging," *Journal of Optics*, vol. 51, pp. 841–850, 2022. [Online]. Available: https://doi.org/10.1007/s12596-021-00813-9

[63] Z. Fu, Y. Cheng, S. Liu, and B. Yu, "A new two-level information protection scheme based on visual cryptography and QR code with multiple decryptions," *Measurement*, vol. 141, pp. 267–276, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0263224119303549

[64] Y.-W. Chow, W. Susilo, J. Wang, R. Buckland, J. Baek, J. Kim, and N. Li, "Utilizing QR codes to verify the visual fidelity of image datasets for machine learning," *Journal of Network and Computer Applications*, vol. 173, p. 102834, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804520303040

[65] S. Scanzio, M. Rosani, M. Scamuzzi, and G. Cena, "QRscript specification," *arXiv*, pp. 1–13, Mar. 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2403.04708

[66] F. Yergeau, "UTF-8, a transformation format of ISO 10646," RFC 3629, Nov. 2003. [Online]. Available: https://www.rfc-editor.org/info/rfc3629

[67] S. Scanzio, M. Rosani, M. Scamuzzi, and G. Cena, "QRtree - Decision Tree dialect specification of QRscript," *arXiv*, pp. 1–32, Mar. 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2403.04716

[68] International Organization for Standardization, "Information technology - ISO 7-bit coded character set for information interchange," International Organization for Standardization, Geneva, ISO/IEC Standard 646, 1991.

[69] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 5th ed. Morgan Kaufmann Publishers, 2014, Section 2.4.2: Two's complement representation of numbers.

[70] "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019.

[71] S. Scanzio, M. Rosani, and M. Scamuzzi, "QRtree software," *GitHub*, Mar. 2024. [Online]. Available: https://github.com/eQR-code/QRtree