Generalization of VLC Decoding Trees-to-Processes Translation

Guan Y. Hong
Unitec/Department of Computing, Auckland, New Zealand
Email: ghong@unitec.ac.nz

A.C.M. Fong
Auckland University of Technology/School of Computing, Auckland, New Zealand
Email: afong@aut.ac.nz

***Abstract*—In an earlier paper, we presented a novel application of Communicating Sequential Processes (CSP) to the modeling of variable length code (VLC) synchronization. Different from the traditional encoding/decoding tree analysis, the new approach provides a very precise description of the synchronization mechanisms. Underpinned by strong mathematical principles, CSP is a process algebra for describing the patterns of communication and interaction between agents that interact via explicit message passing. Such properties have been adapted in the present context of understanding VLC synchronization mechanisms. As an extension of the novel approach, this paper presents a systematic translation from VLC encoding/decoding trees to processes, and describes further generalization of the models presented in our earlier paper. We therefore present this paper to ensure the scalability and applicability of the CSP approach of modeling in this context.**

***Index Terms*—modeling, variable length codes, synchronization, processes.**

I. Introduction

The benefits of self-synchronizing variable length codes (VLC) have been known for a long time e.g. [1] both in coding efficiency and in limiting error propagation. There have been numerous attempts of finding good VLC for different applications, typically using a tree-based approach or simulation e.g. [2], [3], [4].

However, the goal of finding universally good VLC remains elusive, not least because we need better understanding of the complex mechanisms at work. Some researchers even suggest that general understanding of the mechanisms of self-synchronization may be an unattainable goal e.g. [5] and [6].

In [7], we presented two alternative models of VLC synchronization using an adaptation of Communicating Sequential Processes (CSP) [8]. CSP is a well-established process algebra for describing the patterns of communication and interaction between agents that interact via explicit message passing. After years of development, CSP now has a rich mathematical theory supported by a number of automated tools e.g. for refinement checking. To ensure scalability and applicability of the approach in [7], this paper presents a systematic translation from VLC encoding/decoding trees to a process description amenable to analysis using CSP, particularly the CSP traces model. It further describes a generalization of the models presented in [7].

In particular, we develop a more concise description based on the CSP external choices to supersede the original models presented in our earlier paper. These latest results contribute towards the scalability and applicability of our general approach.

The rest of this paper is organized as follows. In Section 2, we present the basic components required for the systematic translation. The translation mechanism itself is presented as a 3-step procedure in Section 3. Section 4 then demonstrates the systematic translation with the aid of two examples, one of which is drawn from prior work by the authors and others. A further generalization is presented in Section 5. Finally, Section 6 concludes the paper.

II. Basic Components for Systematic Translation

Our systematic translation algorithm can be considered an adaptation of the standard nondeterministic finite automaton (NDFA) to deterministic finite automaton (DFA) translation from automata theory.

The basic components required are an initial state I, a set of intermediate error states ES (finite for statistically synchronizable VLC, i.e. resynchronization is guaranteed to happen in finite time), a terminal state S which corresponds to the point in time when synchronization is reestablished, a finite set of events known as an alphabet represented by Σ.

Typically, we choose to have $\Sigma = \{0, 1\}$ for binary codes, which is equivalent to *Bits* in Model A described in [7]. Finally, we need a set of transition relations R as defined below.

Manuscript received January 12, 2010; revised April 6, 2010; accepted May 27, 2010.
Corresponding author: ghong@unitec.ac.nz.

Definition 1 Set of Relations

Suppose X is the set of all states (i.e. $X = \{I\} \cup ES \cup S$). Then $R \subseteq X \times X \times \Sigma$ such that $(x_1, x_2, t) \in R$ means the transition $t \in \Sigma$ from $x_1 \in X$ to $x_2 \in X$ is permissible. In addition, it is possible to introduce non-determinism in the form of hiding to model unobservable transitions. Non-determinism can also arise from having multiple arcs leaving any given node.

III. A Three-Step Algorithm

We propose a three-step algorithm to perform systematic translation from decoding trees to processes. We shall then demonstrate the procedure with two application examples in the next Section.

Any translated model TM of processes can be represented by a tuple comprising the above five items, i.e. $TM = (I, ES, S, \Sigma, R)$ or more simply $TM = (X, \Sigma, R)$. In other words, once the five items are defined, TM immediately follows. Our 3-step procedure for systematic translation from a VLC decoding (or encoding) tree to processes is as follows.

1. Identify the components that make up TM as described in Section 2, i.e. $TM = (I, ES, S, \Sigma, R)$. In particular, I , ES and S will represent the processes, R will represent events and Σ are $Bits$ in the present context.
2. Optionally construct a transition diagram, if it helps in visualizing the various events and processes.
3. Write a set of recursions to describe the various processes.

The results will be a set of recursions that completely describe the synchronization process of the VLC under consideration.

IV. Application Examples

We now illustrate the application of the proposed 3-step algorithm with two examples.

A. Example 1

Using the code set C_{01} in [7] as an example, the procedure for systematic translation is as follows. In Step 1, the components that make up the tuple are identified. From the decoding tree in Fig. 1, we can immediately identify I and ES, and $\Sigma = \{0, 1\}$, which is equivalent to $Bits$. We can also have the following

$S = \{A, B, C, D, E\}$ and

$R = \{(I , ES_1, 0), (I , ES_2, 1), (ES_1, S, 0), (ES_1, S, 1), (ES_2, S, 0), (ES_2, ES_3, 1), (ES_3, S, 0), (ES_3, S, 1)\}$,

or more simply

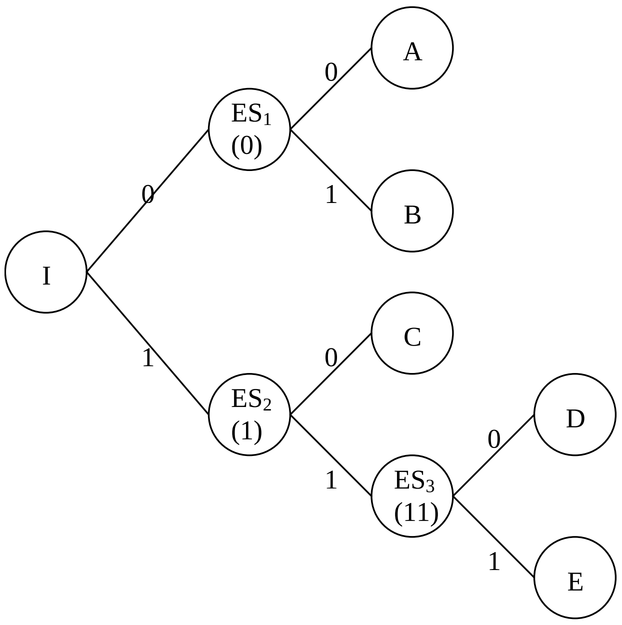

Figure 1. Decoding tree of code set C_{01}

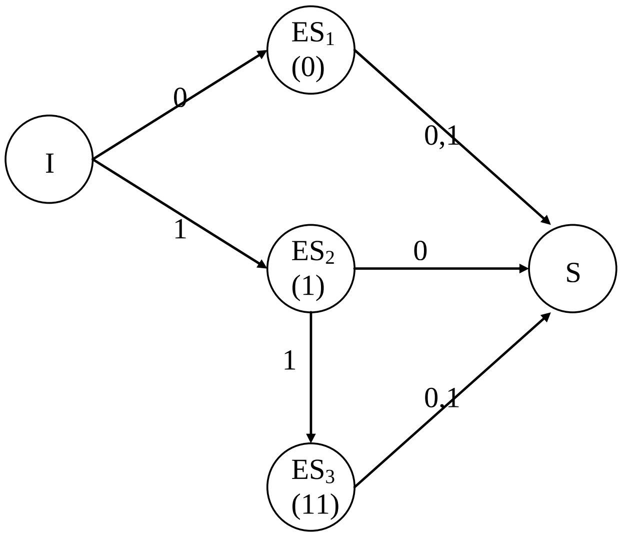

Figure 2. Model A (event = bit) transition diagram for code set C_{01}

$R = \{(I , ES_1, 0), (I , ES_2, 1), (ES_1, S, \Sigma), (ES_2, S, 0), (ES_2, ES_3, 1), (ES_3, S, \Sigma)\}$.

We then apply Step 2 and construct a transition diagram to aid visualization. This is as shown in Fig. 2. Finally, Step 3 is applied to describe the processes resulting in the set of recursions as shown in (1).

$$\begin{aligned} I &= 0 \to ES_1 \square 1 \to ES_2 \\ ES_1 &= x : Bits \to S \\ ES_2 &= 0 \to S \square 1 \to ES_3 \\ ES_3 &= x : Bits \to S \end{aligned} \tag{1}$$

The set of recursions in (1) completely describe the synchronization process of the code set C_{01}.

B. *Example 2*

As a further example, suppose the code set C_{02} has decoding tree as shown in Fig. 3. We can immediately identify *I* and *ES*, and $\Sigma = \{0, 1\}$, which is again equivalent to *Bits*. We can also have the following

$S = \{A, \ldots, F\}$ and

$R = \{(I, ES_1, 0), (I, ES_2, 1), (ES_1, S, 0), (ES_1, ES_3, 1), (ES_2, S, 0), (ES_2, ES_4, 1), (ES_3, S, \Sigma), (ES_4, S, \Sigma)\}$.

Next, we apply Step 2 and construct a transition diagram as shown in Fig. 4. Finally, Step 3 is applied resulting in the set of recursions as shown in (2).

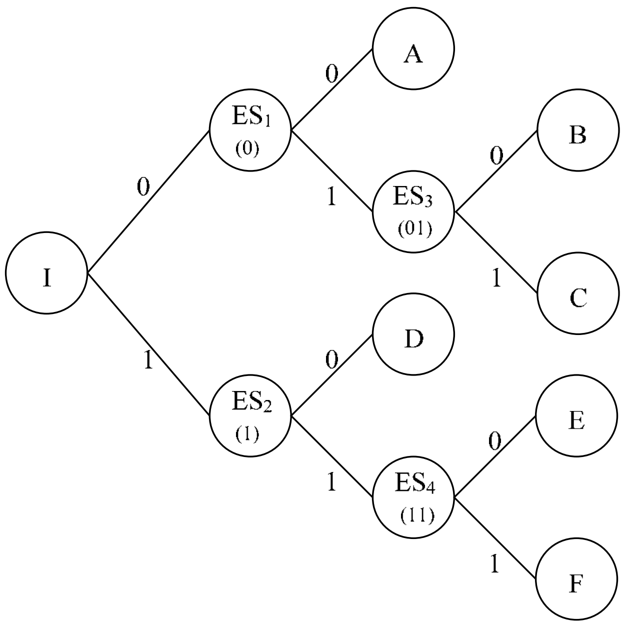

Figure 3. Decoding tree of code set C_{02}

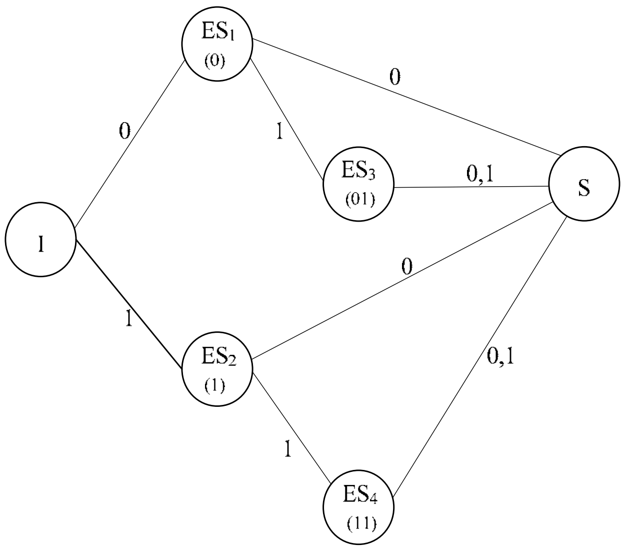

Figure 4. Transition diagram for code set C_{02}

$$\begin{aligned}
I &= 0 \to ES_1 \square 1 \to ES_2 \\
ES_1 &= 0 \to S \square 1 \to ES_3 \\
ES_2 &= 0 \to S \square 1 \to ES_4 \\
ES_3 &= x : Bits \to S \\
ES_4 &= x : Bits \to S
\end{aligned} \qquad (2)$$

Again, the last two error states ES_3 and ES_4 behave the same way. In fact, ES_3 and ES_4 behave just like RUN_{Bits}. So, one might be interested to know *traces*(RUN_{Bits}). In addition, notice that *S* behaves in a similar way as the *SKIP* process.

Once *TM* is in place, we can further define bit sequences (e.g. *INPUT* as seen by a decoder), codewords or synchronizing sequences, code sets, etc. as follows. First, a trace *tr* is defined as $tr = \langle t_1, t_2, \ldots, t_k \rangle$ where $t_i \in \Sigma, i = 1, \ldots, k$ and $\langle\ \rangle$ denotes the empty trace. Generalizing $Bits^*$, one can have Σ^*, which represents the set of all possible traces. Next, a connection φ in *TM* is defined as $[(x_1, t_1), (x_2, t_2), (x_3, t_3), \ldots, (x_n, t_n)]$. Typically, we are interested in any connection φ that goes from the initial state to the terminal state, such that $x_1 = I$, $x_n = S$, $x_2, x_3, \ldots x_{n-1} \in ES$ and $(x_i, x_{i+1}, t_i) \in R$.

All synchronizing sequences can be obtained from *TM* as any trace $tr = \langle t_1, t_2, \ldots, t_k \rangle$ such that a connection φ exists in *TM*. For example, the traces $\langle 0,0 \rangle, \langle 0,1 \rangle$, $\langle 1,0 \rangle$, $\langle 1,1,0 \rangle$ and $\langle 1,1,1 \rangle$ all correspond to synchronizing sequences for the code set C_{01}. This same formulation can also be applied to the translation from an encoding tree to processes. In this case, valid codewords can be generated from *TM* by obtaining traces *tr*.

V. Further Generalization

We now consider the scalability of the model descriptions first presented in [7]. In principle, arbitrarily large code sets can be described precisely through the kind of generalization presented in this section, although in practice we would have a limited number of source symbols for encoding.

The following discussion and examples will make it apparent that this kind of generalization ensures scalability (and therefore applicability) of the model descriptions for practically useful code sets.

Fig. 5 illustrates a generic decoding tree that is applicable to any exhaustive binary code which has been the subject of this study. It should be obvious that, in principle, the decoding tree of the form shown in Fig. 5 can represent any arbitrarily large code set simply by extending the branches.

The decoding tree shows a single initial state *I*, and a number of intermediate error states *ES*. Recall that all

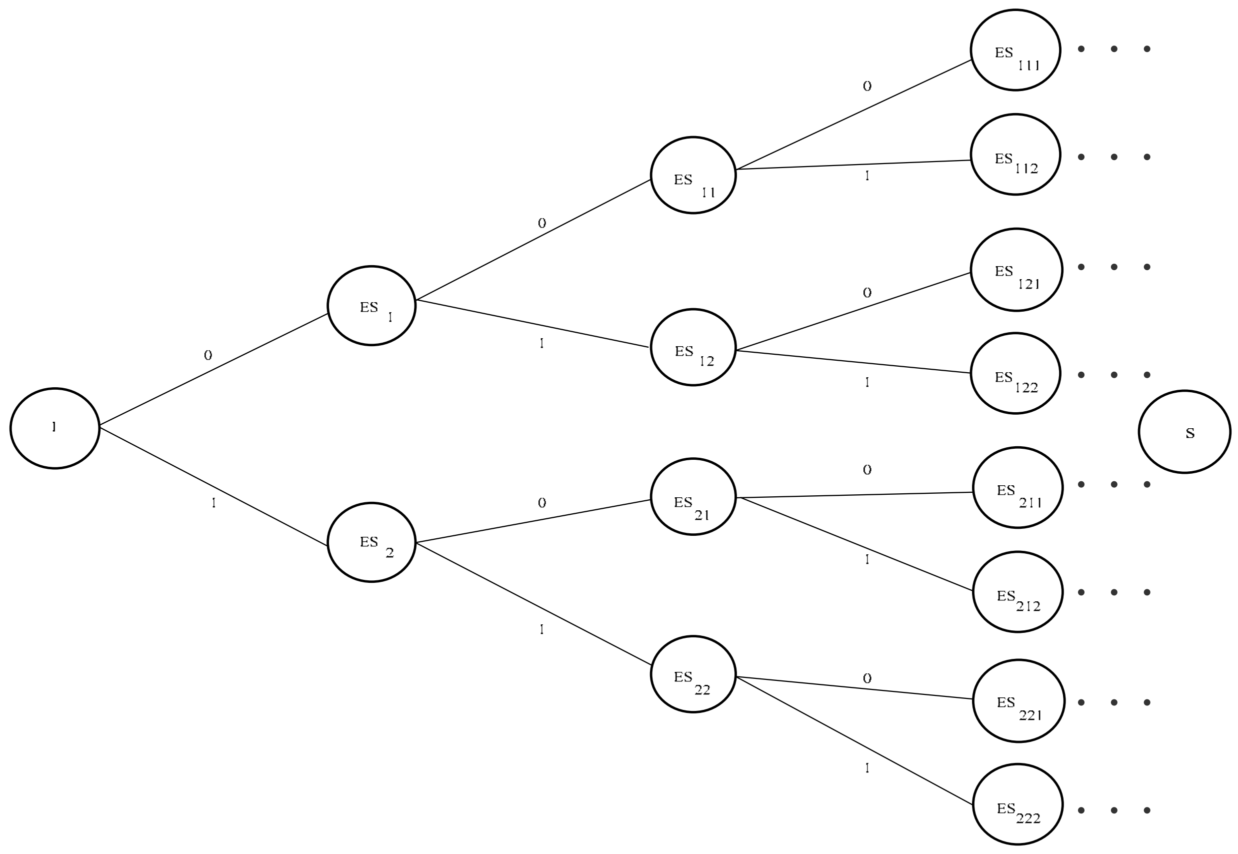

Figure 5. Generic decoding tree

code sets considered in this study are assumed to be statistically synchronizable (resynchronization is guaranteed in finite time). This means eventually all possible paths through the *ES* will lead to the terminal synchronized state (represented by a single state *S* in the diagram).

As usual, when applying the process-oriented way of thinking, each of the nodes in Fig. 5 represents a process and the possible events are 0 and 1. At this stage, we can write a set of recursions to completely describe the decoding process. Alternatively, we can simply write (3) for the process *I*, which effectively describes the entire decoding process in a concise manner.

$$I = 0 \rightarrow (0 \rightarrow (\ldots S) \square 1 \rightarrow (\ldots S)) \square 1 \rightarrow (0 \rightarrow (\ldots S) \square 1 \rightarrow (\ldots S)) \quad (3)$$

Here, each occurrence of $(\ldots S)$ represents iterative branching until *S* is reached. The examples shown in the last Section of this paper are therefore particular instances of this generalized formulation. Specifically, by performing appropriate pruning we can obtain the decoding tree for any code set that might be of interest, including the examples presented earlier (namely the code sets C_{01} and C_{02} as in Figs. 1 and 3).

For example, we can write the following for the code sets C_{01} and C_{02}, respectively.

$$I = 0 \rightarrow (0 \rightarrow A \square 1 \rightarrow B) \square 1 \rightarrow (0 \rightarrow C \square (1 \rightarrow (0 \rightarrow D \square 1 \rightarrow E))) \quad (4)$$

$$I = 0 \rightarrow (0 \rightarrow A \square 1 \rightarrow (0 \rightarrow B \square 1 \rightarrow C)) \square 1 \rightarrow (0 \rightarrow D \square 1 \rightarrow (0 \rightarrow E \square 1 \rightarrow F)) \quad (5)$$

Equations (4) and (5) therefore summarize the decoding and synchronization mechanisms of C_{01} and C_{02} precisely. From these processes, we can easily deduce synchronizing sequences by recording the traces of these generalized processes up to the required length of such sequences.

VI. Conclusion

This paper has presented a systematic translation to ensure that the models developed in [7] can be applied to VLC in general. Using well-established automata theory, this paper has shown that translation from VLC decoding trees to processes can be mechanized and generalized.

Examples have been presented to illustrate how the translation can be applied to derive process representations that completely describe the synchronization of VLC. We have also demonstrated how further generalization can be applied to cover any generic binary codes as illustrated in Fig. 5. With this, future research will focus on a range of interesting situations, such as when an observed sequence of events (e.g. bits, characters or codewords) may be received in error.

ACKNOWLEDGMENT

The authors wish to thank the reviewers who have provided valuable comments for the improvement of this work.

REFERENCES

[1] B. Rudner, "Construction of minimum redundancy codes with optimum synchronization property", *IEEE Trans. Inform. Theory*, vol. 17, pp. 478–487, 1971.

[2] Y. Takishima, M.Wada, and H. Murakami, "Error states and synchronization recovery for variable length codes", *IEEE Trans. Commun.*, vol. 42 (2/3/4), pp. 783–792, 1994.

[3] G. R. Higgie, "Database of best T-codes", *IEE Proc-Comput. Digit. Tech.*, vol. 143, pp. 213–218, 1996.

[4] A. C. M. Fong and G. R. Higgie, "Using a tree algorithm to determine the average synchronization delay of self-synchronizing T-codes", *IEE Proc. Comput. Digit. Tech.*, vol. 149(3), pp. 79–81, 2002.

[5] M.R. Titchener, "The synchronization of variable-length codes", *IEEE Trans. Inf. Theory*, vol. 43(2), pp. 683–691, 1997.

[6] G. Zhou and Z. Zhang, "Synchronization recovery of variable-length codes", *IEEE Trans. Info Theory*, vol. 48(1), pp. 219–227, 2002.

[7] A. C. M. Fong and A. Simpson, "Using CSP to model the synchronization process of variable length codes", *IEE Proc. Commun.*, vol. 153(2), pp. 195–200, 2006.

[8] C. A. R. Hoare, *Communicating Sequential Processes*. Prentice Hall International Series in Computer Science. Prentice Hall, 1985.

Guan Y. Hong is currently senior lecturer in the Department of Computing at Unitec New Zealand. She was formerly an IT consultant and senior lecturer at Massey University. Her research interests include analysis and implementation of information systems, digital communications and software quality and reliability. She has published numerous research papers in leading journals and conference proceedings in these areas.

Dr. Hong is a member of the IEEE and has been active in the organization of major international conferences and participates in the peer review process of leading research journals.

A. C. M. Fong received the BEng(Hons.) degree in electronics and computing and the MSc degree in electrical engineering from Imperial College London, the MSc degree in computing from the University of Oxford and the PhD in electrical engineering from the University of Auckland.

He is currently professor of computer engineering at Auckland University of Technology, New Zealand. His previous appointments include associate professor at Nanyang Technological University and Engineer with the Motorola Corporate Research and Technology Center. He has published well over one hundred research papers in leading international journals and conferences, such as *IEEE Trans. Knowledge and Data Engineering*, *IEEE Trans. Multimedia* and *IEEE Trans. Evolutionary Computation*. He is the lead author of the book Multimedia Engineering (Wiley UK, 2006, ISBN: 978-0-470-03019-6). His research interests include internet and multimedia technology, digital communications and software engineering.

Prof. Fong is a senior member of IEEE and a Chartered Engineer registered in the UK. He has served on the organizing committees of numerous major international conferences and serves on the editorial boards of several international journals.