Two Problems Believed to Exhibit Superunitary Behaviour Turn out to Fall within the Church-Turing Thesis

by

Abdollah Dadizadeh

A thesis submitted to the
Department of Computer Science
in conformity with the requirements for
the degree of Master of Science

Bishop's University
Sherbrooke, Quebec, Canada
January 2019

Abstract

The Turing machine has been throughout decades one of the widely accepted universal models of computation. Relatively recently however challenges of the universality of this model have been raised. Several problems that are at least apparently not solvable by Turing machines have been introduced. These problems belongs to different paradigms than the classical theory of computation. They are most often, but not always, associated to real time. We consider two such problems that apparently contradict the theory of universal computation model. The first problem is real time and was used to establish a strong hierarchy on the number of processors for real-time problems, thus contradicting the universality of the "uniprocessor" Turing machine. The second problem is not real time, but features constraints on the intermediate steps of the computation; it feature a sharp cut-off on the number of processors necessary to solve it, thus once mode contradicting the universality of the Turing machine. We find that instances of both these problems can be converted to CTL formulae such that the instance is solvable iff the respective CTL formula is satisfiable. Given that satisfiability for CTL is decidable, we thus show, in a however roundabout way, that these two problems do not violate the universality of the Turing machine. We believe that this technique can be applied to other problems contradicting the universality of the Turing machine, perhaps even all of them.

Acknowledgments

I would like to thank my supervisor Prof. Stefan D.Bruda, for encouragement, patient guidance and the brilliant ideas and advice he has provided throughout my time as a student. This work would have never been possible without his efforts and wisdom. I was lucky to have a supervisor who cared for the work we did and answered all the questions I had selflessly. I would also like to thank all the staff at Bishop's University, who were always there for me.

I also want to express my gratitude to my family for their continued support and encouragement, and especially to my sister whom was always there, supporting me emotionally throughout my studies.

Also I would like to thank my friends in the Department of Computer Science as well as the undergraduate Computer Science community, whose accepting and helping attitude made my life easier in difficult times.

Contents

Chapter 1

Introduction

The Turing machine was introduced by Alan Turing in 1936. Back then the idea looked simple but it was revolutionary: the machine has a tape which can store data and is modified through a simple mechanism. Since then the Turing machine has had a huge impact on computing research and indirectly practice and has been the foundation of many results. In particular, the machine has been the basis of the concept of computability [19] that is, the science of what can and cannot be computed. Indeed, the Church-Turing thesis [17, 21, 22] states that one particular Turing machine called the universal Turing Machine is capable of solving exactly all the problems that can be computed at all through mechanical computation. In particular, no other model of computation or actual computer can be more powerful than the Truing machine.

The Church-Turing thesis is not and cannot be a formal result, since one cannot anticipate future developments in the theory and practice of computing. It has however stood the test of time: While some more powerful models of computation have been developed over time, all of these models contain features that cannot be realistically implemented in practice, such as infinite precision real numbers [18]. On the other hand, all the models of computation that are considered realistic are equivalent with the Turing machine. On the practical side, all the computing devices developed to date have been found to be equivalent to the Turing machine. Sure some, such as quantum computers have the potential of solving some problems substantially faster than classical computing, but yet this does not make any difference in what can and cannot be computed. Note in passing that the Church-Turing thesis does not say anything about how fast or how slow a problem is solved. Computability is only concerned with whether a solution can be found in finite time; how long that time is is the realm of computational complexity.

Relatively recently some effort has been made to study computation beyond the Turing limit. We already mentioned the infinite precision real numbers [18], to which we can add the field of hyper-computation [10]; these models are strictly more powerful than the Turing machine, but contain features that are not realistically implementable in practical computation devices. Another direction, which is itself a subset of the so-called unconventional computation area [1, 2, 14] studies environmental setups that impose additional constraints on the computation being carried out, with the goal of finding computations that can be carried out within the given environmental constraints using some computational devices but not by classical devices or models including the universal Turing machine. The most common such a set of constraints is a real-time environment, in which the input arrives while the computation is carried out rather than being available in its entirety at the beginning of the computation, and the correctness of the output is defined not only in terms of the actual output being correct in respect to the given input, but also in terms of the time at which the output is produced. Note in passing that most of the time a real-time environment features real-time deadlines, though deadlines are not the only feature that makes a computation real time.

We focus in this thesis on a particular kind of unconventional computation which apparently goes beyond the power of the Turing machine. There are several papers, that describe computational tasks that apparently cannot be solved by a conventional computational device or Turing machine [5, 7, 10, 16] but can be readily solved within the given constraints when a parallel machine is used.

We take a fresh look at a couple of problems which exhibit such a "superunitary behaviour". The first problem is the pursuit and evasion on a ring [7], which was used to show that real-time problems apparently form a strong infinite hierarchy with respect to the numbers of processors necessary to solve them. The second problem is a constrained sorting problem [14]. The reason for our latter choice is that the problem is not real time, but instead imposes some restrictions on the machine state during the computation. In turn these restrictions apparently illustrate once again superunitary behaviour, in the sense that the problem was deemed solvable when a certain number of processors is made available, but cannot be solved with fewer processors.

In this thesis we show that there is nothing special about these two problems, meaning that they both fall into the classical computability theory after all. We establish this result using a fresh, logical perspective. Specifically, for each of the two problems we establish an equivalent formulation in CTL [13], a particular kind of temporal logic. That is, we show that for each instance of the problem at hand there exists a CTL formula such that the instance is solvable if and only if the equivalent CTL formula is satisfiable. Given that satisfiability for CTL is decidable [12, 20], it follows that the respective problem is after all solvable in the classical computability sense of the Church-Turing thesis.

The remainder of this thesis is organized as follows: We summarize the necessary preliminaries (Chapter 2) and then we review some of the existing work on superunitary behaviour (Chapter 3). Specifically, we summarize the two problems that will be our main focus namely, pursuit and evasion on a ring and constrained sorting. The main body of the thesis is contained in Chapters 4 and 5 where we develop equivalent formulations for the two problems mentioned above, respectively. We conclude in Chapter 6.

Chapter 2

Preliminaries

With an alphabet, finite set of symbols, Σ, the set of all the finite sequences of symbols (strings, words) is denoted by Σ^*. The length of a word σ is denoted by $|\sigma|$. Let $|\mathbb{N}| = \omega$ and note that $\omega \notin \mathbb{N}$ [9]. The set Σ^ω contains exactly all the words σ over Σ such that $|\sigma| = \omega$. For $A \subseteq \Sigma$, $|\sigma|_A$ denotes the length of σ after all the symbols that are not in A have been erased. We often write $|\sigma|_a$ instead of $|\sigma|_{\{a\}}$ for singleton sets.

2.1 Timed ω-Languages

If we have two finite or infinite words $\sigma = \sigma_1, \sigma_2, \ldots$ and $\sigma' = \sigma'_1, \sigma'_2, \ldots$ it can be said that σ' is a subsequence of σ ($\sigma' \subseteq \sigma$) iff (a) for each σ'_i there exists a σ_j such that $\sigma'_i = \sigma_j$, and (b) for any positive integers i, j, k, l such that $\sigma'_i = \sigma_j$ and $\sigma'_k = \sigma_l$, it holds that $i > k$ iff $j > l$.

The following summary is based on previous work [6] which is in itself an extension of earlier work [3]. The sequence $\tau \in N^\omega$, $\tau = \tau_1\tau_2\ldots$, is a time sequence if it is an infinite sequence of positive values, such that $\tau_i \leq \tau_{i+1}$ for all $i > 0$ (monotonicity). In addition, a finite of infinite subsequence of a time sequence, is also time sequence. A well-behaved time sequence is a time sequence $\tau = \tau_1\tau_2\ldots$ such that for every $t \in N$, there exists some finite $i \geq 1$ such that $\tau_i \geq t$ (progress). It follows that a time sequence may be infinite or finite, but a well-behaved time sequence, it is always infinite.

A timed ω-word over an alphabet Σ is a pair (σ, τ), where τ is a time sequence, and, $(\sigma, \tau) \in \Sigma^k \times N^k$, $k \in N \cup \{\omega\}$. Given a symbol σ_i from σ, $i > 0$ the associated value τ_i of the time sequence τ shows the time at which σ_i becomes available. A timed ω-word (σ, τ) is well behaved whenever τ is a well-behaved time sequence. A well-behaved timed ω-language over some alphabet Σ is a set of well-behaved timed ω-words over Σ.

Let (σ', τ') and (σ'', τ'') be two timed words over the alphabet Σ. Then we say that $(\sigma, \tau) = (\sigma', \tau')(\sigma'', \tau'')$ is a concatenation of (σ', τ') and (σ'', τ'') whenever:

1. τ is a timed sequence meaning that $\tau_i \leq \tau_{i+1}$ for any $i > 0$; both $(\sigma'_1, \tau'_1)(\sigma'_2, \tau'_2)\ldots$ and $(\sigma''_1, \tau''_1)(\sigma''_2, \tau''_2)\ldots$ are subsequences of $(\sigma_1, \tau_1)(\sigma_2, \tau_2)\ldots$; for any $i > 0$, there exists $j > 0$ and $d \in \{','' \}$, such that $(\sigma_i, \tau_i) = (\sigma^d_j, \tau^d_j)$.

2. For any $d \in \{','' \}$ and any positive integers i and j, $i < j$, such that $\tau^d_k = \tau^d_l$ for any $i \leq k < l \leq j$ there exists m such that, for any $0 \leq \iota \leq j - i$, $(\sigma_{m+\iota}, \tau_{m+\iota}) = (\sigma^d_{m+\iota}, \tau^d_{m+\iota})$.

3. For any positive integers i and j such that $\tau'_i = \tau''_j$ there exist integers k and l , $k < l$, such that $(\sigma_k, \tau_k) = (\sigma'_i, \tau'_i)$ and $(\sigma_\iota, \tau_\iota) = (\sigma'_j, \tau'_j)$.

Given n timed ω-words $\omega_1, \omega_2, \ldots, \omega_n$, $n > 1$, the notation $\omega = \prod_{i=1}^n \omega_i$ is a shorthand for $\omega = \omega_1, \omega_2, \ldots, \omega_n$ (that is, ω is the concatenation of all the words ω_i, $i > 0$). $L = \prod_{i=1}^n L_i$ is the timed ω-language obtained by concatenating the n timed $\omega-$languages L_1, $L_2, \ldots, L_n$.

Finally, the following "projection" operations will be used: Given some timed ω-word $\omega = (\sigma, \tau)$, let $\text{detime}(\omega) = \sigma$ and $\text{time}(\omega) = \tau$.

2.2 Temporal Logic

A temporal logic [13] is a particular kind of modal logic for specifying system properties as they change over time. Such a logic describes properties of sequences of the transitions between the states of a system. We specify certain properties rather than the full behavior of the system, such as “if x happens then y may happen in the future”, or “if x happens then y will happen in the future”, though it is not specified how long in the future. Time is not mentioned explicitly; instead the concepts of before, after, and eventually are used.

To build up expressions In temporal logic we describe the properties of the states of a system using atomic propositions and Boolean operators such as disjunction, conjunction and negation, just like propositional logic. Then to describe how the state of the system evolves over time a set of modal operators called temporal operators is used.

One popular family of temporal logics consists of CTL* [11, 13] , CTL (computation tree logic), and LTL (linear-time temporal logic) [15], CTL and LTL being strict subsets of CTL*. In this thesis we will only use CTL, but we define as usual CTL* first and then CTL as a restricted variant of CTL*.

The semantics for all temporal logics are typically defined in terms of Kripke structures [8, 13], which offer a model for computing systems. A Kripke structure M over a set AP of atomic propositions is a tuple $M = (S, Y, R, L)$ where S is a finite set of states, $Y \subseteq S$ is the set of initial states, $R \subseteq S \times S$ is a transition relation that must be total (that is, for every state $s \in S$ there exists a state $s' \in S$ such that $(s, s') \in R$), and $L : S \to 2^{AP}$ is a function that labels each state with the set of atomic propositions true in that state. A path in the Kripke structure $M = (S, Y, R, L)$ starting from a state s is a possibly infinite sequence of states $\pi = s_0 s_1 s_2 \ldots$ such that $s_0 = s$ and $(s_i, s_{i+1}) \in R$ for all $i \geq 0$.

CTL* formulae describe the properties of computation trees, which in turn describe all the possible paths originating from a given state as infinite trees rooted at the respective start state. Formulae use path quantifiers and temporal operators. The quantifiers are used to describe the branching structure of the computation tree. There are two path quantifiers namely A ("for all computation paths") and E ("for some computation path") and they specify that all of the multiple paths or some of the paths starting form a state (as the case might be) have some property. The temporal operators describe properties of a path through the tree. There are five basic operators: X ("next") which requires that a property hold in the second state of the path, F ("eventually") which is used to assert that a property will hold in some state on the path, G ("globally") which specifies that a property holds in every state on the path, U ("until") which requires that a property remains true until a second properties becomes true, and R ("releases") which specifies that a property must be true until another property becomes true (and thus releases the first from its obligations).

CTL* uses state formulae (which are true in a specific state) and path formulae (which are true along a specific path) with the following syntax:

- If $p \in AP$ then p is a state formula.
- If f and g are state formulae, then $\neg f$, $f \vee g$ and $f \wedge g$ are state formulae.
- If f is a path formula, then E f and A f are state formulae.
- If f is a state formula, then f is also a path formula.
- If f and g are path formulae, then $\neg f$, $f \vee g$, and $f \wedge g$ are path formulae.
- If f and g are state formulae, then X f, F f, G f, f U g, and f R g are path formulae.

CTL on the other hand is a restricted subset of CTL* where each of the temporal operators X, F, G, U, and R must be immediately preceded by a path quantifier (A or E). All the CTL formulae are thus state formulae [23]. With a ranging over AP and f, f_1, f_2 ranging over state formulae, we have the following syntax for CTL formulae:

$$\begin{aligned} f \quad = \quad & \top \mid \bot \mid a \mid \neg f \mid f_1 \vee f_2 \mid f_1 \wedge f_2 \mid \\ & \mathrm{AX}\, f \mid \mathrm{AF}\, f \mid \mathrm{AG}\, f \mid \mathrm{A}\, f_1 \,\mathrm{U}\, f_2 \mid \mathrm{A}\, f_1 \,\mathrm{R}\, f_2 \mid \\ & \mathrm{EX}\, f \mid \mathrm{EF}\, f \mid \mathrm{EG}\, f \mid \mathrm{E}\, f_1 \,\mathrm{U}\, f_2 \mid \mathrm{E}\, f_1 \,\mathrm{R}\, f_2 \end{aligned}$$

The CTL* semantics is defined with respect to Kripke structures. $K, s \models f$ stands for the state formula f being true in a state s of the Kripke structure K. This notation is extended naturally to path formulae as follows: $K, \pi \models f$ iff in the Kripke structure K the path formula f is true along the path π. The operator $\models$ is defined inductively as follows, where f and g are state formulae unless stated otherwise:

1. $K, s \models \top$ is always true and $K, s \models \bot$ is always false for any state s in K.

2. $K, s \models a$, $a \in AP$ iff $a \in L(s)$.

3. $K, s \models \neg f$ iff $\neg(K, s \models f)$.

4. $K, s \models f \wedge g$ iff $K, s \models f$ and $K, s \models g$.

5. $K, s \models f \vee g$ iff $K, s \models f$ or $K, s \models g$.

6. $K, s \models \mathrm{E}\ f$ for some path formula f iff there exists a path p starting at s such that $K, \pi \models f$.

7. $K, s \models \mathrm{A}\ f$ for some path formula f iff $K, \pi \models f$ for all paths p starting at s.

We use π^i to denote the i-th state of a path π, with π^0 being the starting (first) state. The semantics of the relation $\models$ for path formulae is the following:

1. $K, \pi \models \mathrm{X}f$ iff $K, \pi^1 \models f$ for any state formula f.

2. $K, \pi \models \mathrm{F}f$ iff $\exists i \geq 0 : \pi^i \models f$ for some state formula f.

3. $K, \pi \models \mathrm{G}f$ iff $\forall i \geq 0 : \pi^i \models f$ for every state formula f.

4. $K, \pi \models f \,\mathrm{U}\, g$ for state formulae f and g iff there exists $j \geq 0$ such that $K, \pi^j \models g$ and $K, \pi^i \models f$ for all $0 \leq i < j$. In other words, g must become true in some state s_j, and f must hold in all the previous states (from s_0 to s_{j-1}).

5. $K, \pi \models f \,\mathrm{R}\, g$ for any state formula f and g iff for all $j \geq 0$ if $K, \pi^i \not\models f$ for all $0 \leq i < j$ then $K, \pi^i \models g$ for all $0 \leq i < j$. In other words, g must remain true until f becomes true and releases g from its obligations.

The CTL semantics is then the immediate restriction to the CTL* semantics given by the restrictions imposed syntactically over the CTL formulae.

Chapter 3

Previous Work

We describe in this chapter two problems apparently exhibiting superunitary behavior. We will later develop logical models for these problems, thus showing that this is not the case.

The reason for choosing these particular problems is two fold: First, they come from different paradigms (real-time and constrained computations, respectively). Secondly, they are the basis of compelling arguments for the existence of computations beyond the Church-Turing thesis [7, 14].

3.1 Pursuit and Evasion on a Ring

The pursuit and evasion on a ring problem [7] was used to show that real-time problems form a strong infinite hierarchy with respect to the number of the processors necessary for solving them. That is, given any number of processors available for a real-time algorithm, there exist problems that can not be solved by that algorithm, but can be solved if we increase the number of the available processors. This happens even if each processor in the new increased set is arbitrarily slower than each of the processors in the initial set. This hierarchy is developed as a series of ω-timed languages L_k, $k > 0$. This problem is real time according to the formal definition presented in a previous paper [6]; this definition is in itself the generalization of several earlier definitions and matches very well the practical notion of real time.

The first language L_1 is defined as follows:

$$\begin{aligned} L_1 &= \{w \in L_0 L_u : \text{for any } i > 0, z_i(w) \subseteq z(w, c_i), \text{ and there exists some } t, t > 0, \\ &\qquad \text{and some } i_0, 0 \leq i_0 \leq r-1, \text{ such that } |s(w,t)|_a = |s(w,t)|_b\} \end{aligned}$$

with L_0, L_u, z_i, z, and s defined as follows.

L_0 contains words with all the symbols available at time 0 (the beginning of the computation) and showing the initial value of words that will be modified later in time:

$$L_0 = \left\{(\sigma, \tau) \mid \sigma \in \{a,b\}^r, \tau_i = 0 \text{ for all } 1 \leq i \leq r\right\}. \tag{3.1}$$

As the time passes, modifications to the initial word are introduced by the languages L_t, $t > 0$ defined as follows: $L_t = \{(\sigma, \tau) : |\sigma| = \text{j}, 1 \leq i \leq \text{p+1}, \sigma_1 \in \{+,-\}, \sigma_{2\ldots j} \in \{a,b\}^{j-1}, \tau_i = t \text{ for all } 1 \leq i \leq j\}$. A word in L_t stands for the change to the initial word happening at time t. Such a word in starts with a $+$ or $-$ as its first symbol, followed by at most p a and/or b symbols. The language L_u defines all the changes over the time. For some positive constant c:

$$L_u = \prod_{i>0} L_{ci}, \tag{3.2}$$

Let $w \in \{a,b\}^r$ and $u = u_0 u'$ such that $u_0 \in \{+,-\}$ and $u' \in \{a,b\}^j$, $j \leq p$. Then for some i (the "insertion point"), $0 \leq i < r$ the concept of the insertion modulo r at point i of u in w is defined as a function ins_r that receives three parameters (w, u, and i), then returns a new word w and a new i. How ins_r behaves is defined as follows: Let $i' = i + p$ if $u_0 = +$ and $i' = i - p$ otherwise. Then $ins_r(w, u, i) = (w', i' \bmod r)$ where w' computed as follows (with $\overline{x}$ denoting the reversal of the word x):

1. If $i' < 0$ (and so $u_0 = -$) then let $i'' = i' \bmod r$; then, $w' = \overline{u'_{0\ldots i}} w_{i+1\ldots i''+1} \overline{u'_{i+1\ldots j1}}$.

2. If $i' > i - 1$ (and thus $u_0 = +$) then $w' = u'_{r-i\ldots j-1} w_{i''+1\ldots i-1} u'_{0\ldots r-i-1}$.

3. Otherwise (that is, when $0 \leq i' < r$), let $i_1 = \min(i, i')$, $i_2 = \max(i, i')$, $x = u'$ if $u_0 = +$ and $x = \overline{u'}$ otherwise; then, $w' = w_{0\ldots i_1-1} x w_{i2+1\ldots r-1}$.

Define then the operator $\bigoplus$ as follows: the result of $(\sigma, i) \bigoplus_{j \in A} \sigma_j$ is the successive insertion modulo r of all the strings σ_j, $j \in A$ in order (from smaller to larger j) into σ starting at the initial insertion point i (which will change after each insertion based on the value returned by ins_r).

Consider a word $w \in L_0 L_u$, $w = w^0 \prod_{i>0} w^i$ with $w^0 \in L_0$ and $w^i \in L_{ci,i>0}$. For some time value t and some i_0,$0 \leq i_0 \leq r - 1$, let

$$s(w, t) = (\sigma^0, i_0) \bigoplus_{0 \leq ci \leq t} \sigma^i \tag{3.3}$$

where $\sigma^i = \text{detime}(w^i)$.

An algorithm A that receives w as an input and uses π processors, $\pi \geq 1$ (when $\pi = 1$ algorithm is sequential; otherwise it is parallel). may at any moment inspect (i.e., read from memory) some symbol from $s(w, t)$. In a parallel algorithm many processors may be inspecting at the same time different indices in parallel. For each processor q, $1 \leq q \leq \pi$ let ι_t^q be the most recent index inspected by the processor q up to time t. If some processor has inspected no symbols from $s(w, t)$ then by convention $\iota_t^q = -1$. Let I_t^t be the history of symbols inspected by the processor q up to time t that is, $I_t^q = \bigcup_{t' \leq t} \iota_{t'}^q \setminus \{-1\}$. Let now $lo = \min_{1 \leq q \leq \pi}(\iota_t^q)$, $hi = \max_{1 \leq q \leq \pi}(\iota_t^q)$, and $I = \bigcup_{1 \leq q \leq \pi}(I_t^q)$. Then we define $z(w, t)$, the acceptable insertion zone at time t as follows:

$$z(w, t) = \begin{cases} \{|i| 0 \leq i < r\} & \text{if } lo = -1 \\ \{|i| 0 \leq i < r, i \neq lo\} & \text{if } lo \neq -1 \wedge \exists j \notin I : j > hi \vee j < lo \\ \{|i| lo \leq i < hi\} & \text{otherwise} \end{cases} \tag{3.4}$$

The set of indices $z(w, t)$ has the following form: if all the indices in the contiguous sequence surrounded by the latest inspected indices have been inspected in the past, then all these indices are excluded from the acceptable insertion zone; otherwise, the acceptable insertion zone contains all of the indices except one of the inspected indices (the smallest).

Starting from L_1 as the "one-dimensional" definition (see Section 3.1.1 for the reason behind this name) we can extend this definition to "k dimensions", $k > 1$ by defining the language L_k as follows: Fix $k > 1$, $p > 0$, $r > 2p$, let $r' = kr$, and let $L'_0 = \{(\sigma, \tau) : \sigma \in \{a, b\}^{r'}, \tau_i = 0 \text{ for all } 1 \le i \le r'\}$. L'_0 is similar with L_0, but now we have k regions each consisting of r symbols.

Further let $\mathbb{N}_k = \{\text{enc}(i) : 1 \le i \le k\}$, where enc is a suitable encoding function from $\mathbb{N}$ to $\{I\}^*$, for some symbol $I \notin \Sigma$. The actual form of enc is not important, except that $|\text{enc}(j)| \le j$ for any $j \in \mathbb{N}$, and that enc^{-1} is defined everywhere and computable in finite time (these properties clearly hold for any reasonable encoding function). Define $L_t^{\mathbb{N}} = \{(\sigma, \tau) : \sigma \in \mathbb{N}_k, \tau_i = t \text{ for all } 1 \le i \le |\sigma|\}$. Then the multidimensional version of L_t is $L'_t = L_t^{\mathbb{N}} L_t$. Additionally to the direction of insertion and the word to be inserted (as in the one-dimensional case), a word in L'_t is now providing a "dimension" (from 1 to k) along which the insertion takes place. Finally, as before let $L'_u = \prod_{i>0} L'_{ci}$ for a given constant $c > 0$. The k-dimensional language L_k will be a subset of $L'_0 L'_u$.

Given some word $w \in \{a, b\}^{r'}$, let $w = w(1)w(2)\ldots w(k)$, where $|w(i)| = r$, $1 \le i \le k$, and call each $w(i)$ a segment of w. Let $w \in \{a, b\}^{r'}$, and $u = u'u''$ with $u' \in \mathbb{N}_k$ and $u'' \in \Sigma^j$, $1 \le j \le p + 1$, $u''_1 \in \{+, -\}$, and $u'_{2\ldots p} \in \{a, b\}^{j-1}$. Then, for some i, $0 \le i \le r - 1$, define

$$(w, i) \bigotimes u = \left(\prod_{j=1}^{d-1} w(j) \right) ((w(d), i) \bigoplus u'') \left(\prod_{j=d+1}^{k} w(j) \right)$$

where $d = \text{enc}^{-1}(u')$. The operator $\otimes$ defines the insertion modulo r in one of the k segments, as follows: the two components of the word which is going to be inserted are u' (encoding a number from 1 to k) and u'' (denoting the actual actual word to be inserted). The operator then inserts (modulo r) u'' into the segment of w, given by u'. The operator $\otimes$ is defined analogously to (and based on) $\bigoplus$.

For some word $w \in L'_o L'_u$ ($w = w^o \prod_{i>0} w^i$ with $w^0 \in L'_o$ and $w^i \in L'_{ci}$, $i > 0$) and for some i_0, $0 \leq i_0 \leq r-1$, let

$$s'(w,t) = (\sigma^0, i_0) \bigotimes_{ci \leq t} \sigma^i \tag{3.5}$$

where $\sigma^i = \text{detime}(w^i)$, $i \geq 0$. Relation (3.5) is a generalization of Relation (3.3) to k dimensions, with only one of these dimensions being modified by the current insertion.

As before, consider an algorithm A which receives some word $w \in L'_o L'_u$ as input and uses π processors. Then for some $t \geq 0$ define $z^j(w,t) = z(w(j),t), 1 \leq j \leq k$, with $z(w(j),t)$ defined as before, except for the following change: if at time t some processor inspects an index outside $s(w(j),t)$, then $\iota^q_t(j) = -1$ and $I^q_t(j) = \emptyset$.

Putting everything together, let $z'(w,t) = \bigcup_{j=1}^{k} z^j(w,t)$ and call $z'(w,t)$ the acceptable insertion zone at time t. We now define L_k as follows:

$$\begin{aligned} L_k \;=\; & \{w \in L'_o L'_u : \text{ for any } i > 0, z'_i(w) \subseteq z'(w, ci), \text{ and there exists some } t, \\ & t > 0, \text{ and some } i_0, 0 \leq i_0 \leq r-1, \text{ such that } |s'(w,t)|_a = |s'(s,t)|_b\} \end{aligned} \tag{3.6}$$

with $s'(w,t)$ and $z'(w,t)$ as defined before and $z'_i(w)$ denoting the set of indices whose values are modified by the timed sub-word w^i of w, $w^i \in L_{ci}$, $i > 0$ [7].

Proposition 1 **[7]** *There exists no (2n-1)-processor parallel deterministic real-time algorithm that accepts $L_n, n \geq 1$. There exists a 2n-processor parallel deterministic real-time algorithm that accepts L_n and that uses arbitrarily slow processors, $n > 1$.*

3.1.1 An Intuitive Problem Statement

In order to facilitate the understanding of the pursuit and evasion problem described above we will consider the following intuitive formulation which is also mentioned in the original paper [7] and is also alluded to in the previous section: Using geometry as a tool, each word $w(i)$ of $s'(w,t)$, $1 \leq i \leq k$ will become a circle, where $w(i)_{r-1}$ is (conceptually) adjacent to $w(i)_0$. Each update $u'u''$ replaces j consecutive symbols in the "circle" specified by u', starting from the current insertion point and moving from that position to the right or the left, based on the value of u''_0 ($+$ for right, $-$ for left). The function ins_r is designed to perform exactly this kind of insertion (or modification of the respective circle). Imagine now that these modifications are performed by a "pursuee" whose movements are given as input to an algorithm (the "pursuer"). Note in passing that the pursuee has a topmost "velocity" of p/r-th of the circles circumference per time unit. The modulo operation makes the two ends of w conceptually adjacent, thus the pursuit space becomes (conceptually) a circle.

The algorithm that accepts L_k (that is, the "pursuer") should inspect all the symbols that are modified during the insertion process. The algorithm is successful if it "catches up" with the input that is, is able to determine the content of all the circles before they are further modified. In other words, the pursuer has to match the moves of the pursuee [7]. The "velocity" of the accepting algorithm is determined by its ability to inspect symbols stored in memory, and thus this velocity is directly proportional to the speed of the processors(s) used by the algorithm.

To illustrate the concept of insertion modulo r we refer to one of the original examples for the one-dimensional case [7] shown graphically in Figure 3.1, where $r = 8$ and $p = 3$. Initially we have $s(w,t) = bbbbbbbb$, with the insertion point of $i = 1$. The word $s(w,t)$ is represented here as a circle (as explained above), with 8 identified locations that corresponds to the eight symbols stored in $s(w,t)$. These locations are labeled with their indices (inside the circle) and with the values stored therein (outside). The first circle shows the insertion of the word $u = -a_1a_2a_3$ (all the as are the same symbol, the subscripts provided solely for illustration purposes). The pursuee moves to the left (or counterclockwise), rewriting symbols at indices 1, 0, and 7, in this order. Afterward the new insertion point becomes $i = 7$, and $s(w,t+c) = a_2a_1bbbbba_3$. Consider now that the next word to

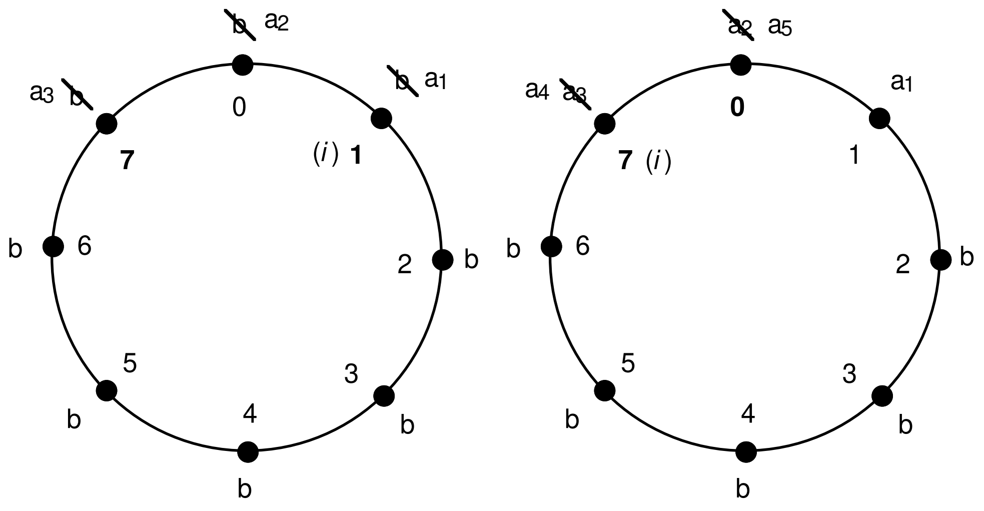

Figure 3.1: Pursuit and evasion in one circle.

be inserted is $u = +a_4a_5$. The indices whose values are modified are 7 and 0. This process is illustrated in the second circle. The final result is $i = 0$ and $s(w, t + 2c) = a_5a_1bbbbba_4$.

Even when two pursuers are slow compared with the pursuee, as long as they inspect a contiguous segment instead of "jumping" around the circle, they are able to define the acceptable insertion zone $z(w, t)$ which consists only of the not yet visited indices between their current position. Given that the pursuee is only allowed to modify indices inside $z(w, t)$, the pursuers are able to "corner" it (and are thus able to complete their task) by progressively narrowing $z(w, t)$. They are thus able to "catch" the pursuee despite the possible speed disadvantage. On the other hand, a single pursuer does not get the benefit of the acceptable insertion zone (which in this case contains only one index), and so is unable to catch the pursuee unless it is faster.

The k-dimensional geometric version can be explained intuitively as an extension of the one one-dimensional version as in Figure 3.2. There are k circles, each of length r. Each collection of k identical indices (one on each circle) is connected by a special path; there are r such paths, and they are shown using thin lines between the circles. As soon

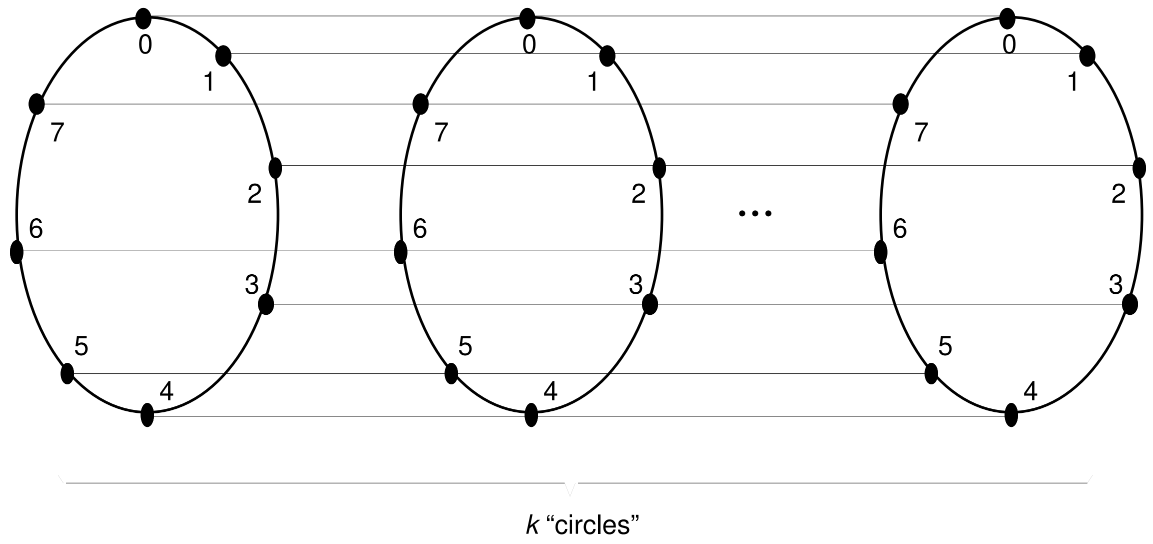

Figure 3.2: Pursuit and evasion through k dimensions.

as the pursuers use these paths they loose the advantage of the acceptable insertion zone, whereas the pursuee can use them at will. If $2k$ pursuers do not jump, between the circles or within a circle, then they can establish one acceptable insertion zone per circle and progressively narrow it until the pursuee is caught. If on the other hand there are $2k - 1$ pursuers or less, there will be at least one "unguarded" circle, whose acceptable insertion zone only consists of a single index, and so the pursuee cannot be caught, meaning that the content of the circles cannot be verified.

3.2 A Constrained Sorting Problem

The following problem was proposed as an illustration of the absence of universality in computation and is stated as follows [14]: We are given an array called A, consisting of the n locations $A[0], A[1], \ldots, A[n-1]$ capable of storing one integer each. It is requested that the values stored in the array be sorted in-place into nondecreasing order (so that when the algorithm terminates we have $A[0] < A[1] < \cdots < A[n-1]$) subject to the following constraint: After each step of the sorting algorithm, no three consecutive integers satisfy $A[j] > A[j+1] > A[j+2]$ for any $0 \leq j \leq n-3$. For brevity in what follows we will call this problem constrained sorting.

Sorting is ubiquitous in computing. This variant of the ages old sorting problem has one extra twist in that no computation step is allowed to leave three consecutive values in the wrong order; the way in which the algorithm progresses matters just as much as the end result. This being said, there are no restrictions on the parallel or sequential model being used or on the algorithm being deployed on that model.

A parallel algorithm using $n/2$ processors on a shared memory parallel machine can solve the problem handily by means of pairwise swaps applied to the input array [14]. It is however claimed [14] that a parallel machine with fewer than $n/2$ processors, including a 1-processor that is, sequential machine, will fail to solve the problem since it will fail to solve the problem for all possible $n!$ permutations of the input and at the time satisfy the on-going condition at every computational step. A particular unfavorable situation is the input being sorted in the wrong order (that is, $A[0] > A[1] > \cdots > A[n-1]$); it is claimed [14] that any algorithm using fewer than $n/2$ processors will fail to satisfy the on-going constraint in its very first step.

In the end, the authors who introduced this problem claim that the result contradicts the common belief that any computation by any model of computation can be effectively simulated by the standard Turing machine, and thus the Church-Turing thesis is invalidated.

Chapter 4

A CTL Formulation of Pursuit and Evasion on a Ring

The pursuit and evasion problem is a compelling case for superunitary behavior, in that the number of processors themselves make a difference, irrespective of the performance of the individual processors , meaning that the $2n$ processors may be even slower than the $2n - 1$ bunch and yet be able to solve more instances. However, we will show now that the problem is in fact within the Church-Turing thesis. To do so we follow a somehow convoluted but nonetheless clear path: we develop a method of obtaining a CTL formula out of every instance of the problem such that the formula is satisfiable iff the problem instance is solvable.

4.1 High Level Description

The logical formulation will essentially state that the pursuers are free to move around and leave marks in their trails subject to the original restrictions, while the pursuee , for a bit of variation hereinafter named “the bird”, is allowed to move into any unmarked, adjacent location. The CTL formula thus obtained is satisfiable iff there is a sequence of moves by the pursuers that trap the bird in a single location and unable to move, as desired.

It may be tempting to track the moves of the bird and its pursuers throughout the circles, but this approach has a number of pitfalls; such a tracking is difficult (perhaps impossible) to implement given the limitations of CTL, and it is certainly not in the spirit of a logical specification of an evolving system. Instead, we will involve the following ingredients in our specification:

1. We specify the possible moves of the pursuers.

2. We then specify the moves that are available to the bird in the current circle, subject to the past moves of the pursuers.

3. Then we specify the possible “inter-circle” moves of the bird.

This specification will result in a “rolling condition”, meaning a condition that will hold for all the states until the end condition of the bird being trapped is met. That is, we start from the following meta-formula:

$$(\text{“The bird is not trapped”})\ \mathrm{U}\ (\text{“The bird is trapped”})$$

Then we further refine the formula according to the description of the rolling condition above to the following:

$$\begin{array}{llll} \mathrm{E} & (\mathrm{E} & (\mathrm{E} & ((\mathrm{EX}\ \text{“The bird can move in the circle”})\vee \\ & & & \;\;(\mathrm{EX}\ \text{“The bird can move to another circle”})) \\ & & \mathrm{U} & (\text{“The bird is trapped in one circle”}) \quad) \\ & \mathrm{R} & & (\mathrm{EX}\ \text{“The bird can move to another circle”}) \quad) \\ \mathrm{U} & & & (\text{“There are no move circles to go”}) \end{array} \tag{4.1}$$

In other words, “The bird is not trapped” consists of two parts: the bird is not trapped in the current circle, or the bird can move to at least one other circle.

For the sake of a more concise presentation we will give symbolic names to the various propositions used in the CTL formula, as follows. Note that some of those propositions have already been encountered above, while some others will come into play later during the subsequent refinement.

- $\mathcal{P}$ = "The bird is caught"
- $\mathcal{F}$ = "The bird can move in the circle"
- $\mathcal{G}$ = "The bird cannot move in the circle"
- $\mathcal{O}$ = "The bird can move to another circle"
- $\mathcal{N}$ = "The bird cannot move to another circle"

Recall from Section 3.1 that each circle has r positions, and that there are k circles overall.

4.2 Trapping the Bird

We will focus first of the movements happening in one circle. For this purpose C_i will be a variable that is true iff the bird is at position i in the respective circle, $0 \leq i < r$. Later on we will also use $C_{j,i}$ to specify the complete position of the bird in the ensemble of k circles; that is, $C_{j,i}$ will be true iff the bird is at position i in the circle k, $0 \leq i < r$ and $1 \neq j \leq k$.

The best strategy for the pursuers is to move away from each other, thus catching the bird in a pinch. However, this is not the only winning strategy; for example, one pursuer can stay put while the other one advances, and the bird will still get trapped ; though it will take more time for this to happen. It will in the other hand make no sense for the pursuers to jump around, as this will eliminate their advantage. From the point of view of the logical description however, none of the above make any difference; one only need to set the conditions up such that a winning strategy will win and of course a loosing strategy will not. To simplify the formulation however, we will not restrict the movement of the pursuers to adjacent locations in the circle; while this deviates from the original formulation (in which jumping cannot lead to the capture of the bird), but is nonetheless without loss of generality since (a) choosing not to jump is an obvious particular situation in this jumping is otherwise allowed, and (b) jumping cannot succeed in those cases that lack a successful strategy that does not involve jumping.

Recall that in essence the pursuers leave marks as they move along, more precisely, they define the acceptable insertion zone based on their movement. We will represent these marks using the variables L_i, $0 \leq i < r$, one for each location in the circle. The possible values of these variables are P, standing for the respective position being either marked, or occupied by a pursuer, and A, which stands for the position being available for the bird. Figure 4.1 shows how the circle is thus divided into a marked region and an unmarked one. While we do not technically impose such a division, this is done without loss of generality and so can and will be assumed. Note incidentally that later on we will use a second set of marks names $L_{j,i}$ which refer to position i in circle j.

We can now formulate various useful properties using the marks. For example, to specify that the bird can move to any place that is not marked we can write EX $\bigvee_{i=0}^{r-1}(C_i \wedge L_i = A)$. We note however that the bird is more constrained, in that it can only move to adjacent locations. Therefore we can state the condition $\mathcal{F}$ ("The bird can move in the circle") as follows: for the bird not to be trapped in the circle it must be able to move into one of the two adjacent locations. That is,

$$\mathcal{F} = \mathrm{EX}((C_{x+1 \bmod r} \wedge L_{x+1 \bmod r} = A) \vee (C_{x-1 \bmod r} \wedge L_{x-1 \bmod r} = A))$$

It is also useful to introduce the condition that makes the bird trapped in a circle. This is technically $\neg\mathcal{F}$, but we believe that an explicit formulation makes the overall picture

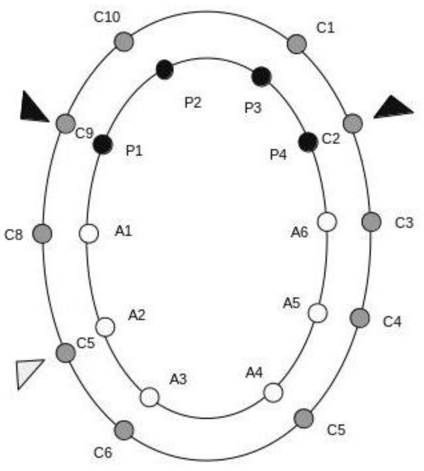

Figure 4.1: Boolean variables for the logical formulation of pursuit and evasion in a single circle.

clearer:

$$\mathcal{G} = \text{EX}\ (C_{x+1 \bmod r} \wedge (L_{x+1 \bmod r} = P)) \wedge (C_{x-1 \bmod r} \wedge (L_{x-1 \bmod r} = P))$$

Indeed, the bird is trapped if it is sandwiched between the two pursuers (or their marks).

To extend this construction to multiple circles, we must specify the condition $\mathcal{F}$ for each of the k circles. In particular, we need a new set of variables like C_i and L_i for each of the k circles. We will therefore name the new variables by adding a subscript identifying the circle, such that $C_{j,i}$ represents location i in circle j and $L_{j,i}$ represents the label of the location i in circle j. The new index j will range from 1 to k. An example for the $C_{j,i}$ family of variables with two circles having 10 locations each is given in Figure 4.2.

With this notation, the condition $\mathcal{O}$ ("The bird can move to another circle") can be stated as follows:

$$\mathcal{O} = \text{EX}\ (C_{j,i} \wedge \bigvee_{1 \leq n \leq k, n \neq j} L_{n,i} = A)$$

Indeed, we specify that some position i in some other circle than the current one is available, which means that the bird can move to that position (meaning, to the same position

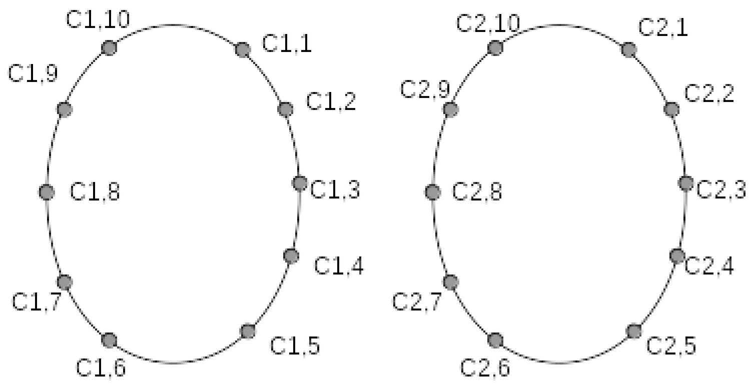

Figure 4.2: Example of location labeling in multiple circles.

i but in a different circle).

Similarly, we can specify that the bird cannot move to another circle as follows:

$$\gamma = \text{EX}\ (C_{j,i} \wedge \bigwedge_{1 \leq n \leq k, n \neq j} L_{j,i} = P)$$

Indeed, this means that the bird had no place *i* to go in any other circle.

We thus reach the following expression for the condition $\mathcal{P}$ ("The bird is trapped"):

$$\begin{aligned}\mathcal{P} \quad = \quad & \text{EX}((C_{x+1 \text{ mod } r} \wedge (L_{x+1 \text{ mod } r} = P)) \wedge (C_{x-1 \text{ mod } r} \wedge L_{x-1 \text{ mod } r} = P)) \wedge \\ & (C_{j,i} \wedge \bigwedge_{1 \leq n \leq k, n \neq j} L_{j,i} = P)\end{aligned}$$

Plugging everything together in the meta-formulation of the problem namely, Expression 4.1 will produce a CTL formula that is equivalent with the respective instance of the pursuit and evasion on a ring problem.

4.3 Equivalence between Pursuit and Evasion Instances and the Derived CTL Formulae

It is worth emphasizing that we use two sets of variables C and L, one with a single subscript (and referring to the situation in some implicit circle) and another with two subscripts, which identify explicitly the circle we refer to.

The circles represents a the word in the language L_t, with all the changes up to time t incorporated. The goal of the pursuers is to analyze this word in time before the next changes occur. Given the direct "translation" to CTL used in this chapter it is quite immediate that the CTL formula corresponding to a particular instance as developed here specifies the same thing. Indeed, we have specified logically the situation in which the entities (bird and pursuers) are sufficiently constrained so that the pursuers can succeed iff they succeed by following a successful strategy strategy in the original problem. In particular, $2k$ pursuers will continue to be successful while $2k - 1$ pursuers will not.

More specifically, for the "only if" part of the equivalence we note that whenever the pursuers are able to catch the bird in the original problem they will need to "corner" it that is, reduce the acceptable insertion zone to one location (the one with the bird on it). Clearly, the formula $\mathcal{P}$, which is the second argument of the outermost U operator in Expression 4.1 specifies the same thing: the locations adjacent to the bird are blocked ($\mathcal{G}$), as are all the other equivalent locations in the other circles ($\mathcal{N}$). In the meantime, the first argument of the outermost U operator specifies the acceptable behavior of the bird: It can move inside the current circle ($\mathcal{F}$) as much as it wishes (even if moving to another circle is also an option) for as long as such a move is available ($\mathcal{G}$ as the second argument of the innermost U), or it can move to a different circle if this is possible ($\mathcal{O}$ as the second argument of R).

In addition the statements used in Expression 4.1 are clearly equivalent to the processing taking place in the original problem. $\mathcal{F}$ ensures that the bird can only move to an adjacent location and only if that location is not marked, while $\mathcal{O}$ is true iff the bird moves to the same location in a different circle and once more the new location is not marked. At the same time, $\mathcal{G}$ specifies that the two locations adjacent to the location of the bird are both marked, so that the bird cannot move in the current circle, and $\mathcal{N}$ specifies that if the location of the bird in the current circle is i then all the other locations i in all the other

$k - 1$ circles are marked, so that the bird cannot leave the current circle. Finally, $\mathcal{P}$ is a conjunction of $\mathcal{G}$ and $\mathcal{N}$, thus being equivalent to the bird being trapped.

The fact that the bird is eventually caught is ensured by the semantics of p U q which is true only if the second argument q becomes eventually true at some point.

The "if" part of the equivalence is established in the same manner as the "only if" part above, indeed, most of the argument above is based on equivalence rather than implication, except for one thing namely, the management of the acceptable insertion zone. This is the only major thing that we have not specified logically. More precisely, the original problem specifies that this zone disappears as soon as the pursuers jump, yet this is not the case in the CTL formulation. This lack of management is however without loss of generality. Indeed, we note that jumping pursuers can catch the bird very quickly by just jumping to adjacent positions. However, not jumping will still produce the same result, only slower: instead of jumping the pursuers can just move to the same adjacent positions sequentially, going through all the intermediate positions. In other words, jumping is just a shortcut and so it does not make any difference since we are interested in solving the problem and not in the time it takes for the solution to become available; therefore not managing the acceptable insertion zone will not loose generality.

In all, what we have achieved is a direct logical formulation of the possible ways that the the system pursuers can take to catch the bird, as desired.

Chapter 5

A CTL Formulation of Constrained Sorting

While the immediate algorithmic approach fails to solve the constrained sorting problem if fewer than $n/2$ processors are available, the very nature of the problem statement seems to suggest that a temporal logic approach is particularly suitable for this problem. Indeed, the requirement that a certain condition is maintained throughout all the computational steps appears to be particularly suited for the use of the G (or perhaps U) operator.

Sure enough, that turns out to be indeed the case. We start by noticing that the following condition needs to be met at each computational step:

$$\psi = \neg \left(\bigvee_{i=0}^{n-3} A[j] > A[j+1] > A[j+2] \right)$$

Indeed, any violation of this condition implies that there are three consecutive values in the array that are in decreasing order, which is explicitly forbidden by the problem statement.

On the other hand, we can just as easily state the following condition that has to be met in order for the problem to be solved:

$$\phi = A[0] < A[1] < \cdots < A[n-1]$$

Putting everything together we need to specify that ψ is continuously true until the problem is solved that is, until ϕ becomes true. We thus reach the following natural logical translation of a constrained sorting instance: EX(ψ U ϕ) that is,

$$\mathrm{EX}\left(\neg\left(\bigvee_{i=0}^{n-3} A[j] > A[j+1] > A[j+2]\right)\ \mathrm{U}\ A[0] < A[1] < \cdots < A[n-1]\right)$$

The initial X operator is needed because the initial state of the array A stores the input values, which do not necessarily observe any constraint; the "no three consecutive values in decreasing order" constraint applies from the next step on instead.

5.1 Equivalence between Constrained Sorting Instances and the Derived CTL Formulae

The equivalence is quite immediate, since the environment and desired result of the computation are faithfully described logically: The end result is that the array is sorted, clearly equivalent to ϕ. That this state is reached is ensured by the semantics of the U operator which requires that its second argument becomes true eventually. Before ψ becomes true ϕ needs to hold , again by the semantics of U. However, ϕ is the negation of the condition that is explicitly forbidden by the original problem, and so ϕ holding until the array is sorted is equivalent to the forbidden condition never happening during the sorting process, as desired.

What happens when the sequence is sorted to begin with? It can be argued, and we did so here however implicitly, that if this is the case then the sequence will remain sorted in all the subsequent states and so ψ releases ϕ from its obligations starting from the next (second) state and thus the situation continues to fall within the scope of our CTL formula. It can be argued however that the formula should be true from the very beginning to maintain a strong equivalence with the original problem. This argument can be addressed by strengthening the CTL formula from $\mathrm{EX}(\psi \ \mathrm{U} \ \phi)$ which we establish above to

$$\phi \vee \mathrm{EX}(\psi \ \mathrm{U} \ \phi)$$

With this formulation, if the sequence is already sorted then ϕ holds in the start state and that makes the formula true irrespective of the second operand of the disjunction. Otherwise, the first operand of the disjunction is false and so the truth value of the formula is determined by the truth value of the second operand as described above.

Chapter 6

Conclusions

We succeeded in bringing two problems that support significant findings in the area of superunitary behavior back into the Church-Turing thesis fold. We did so in a somehow roundabout manner, by formulating every instance of these problems as equivalent CTL formulae. In other words, solving each of these problems is equivalent to determining that the equivalent CTL formula is satisfiable. This plus the fact that satisfiability problem for CTL is decidable proves our point.

The first problem namely, pursuit and evasion on a ring is an interesting problem because it addressed for the first time a common criticism to the superunitary behavior effort: Many problems apparently exhibiting superunitary behavior are solvable when n processors are available, but they turn out to be equally solvable when a single processor is available but that processor is n times faster than the original processors being considered. The pursuit and evasion problem does not have this limitation; indeed, the processors (pursuers) can be arbitrarily slow and the superunitary behavior still manifests itself. This feat is accomplished by asking the algorithm to handle data that arrives in real time and also by requiring that input data be constrained by the behavior of the algorithm that tries to cope with it. These features are hardly rare in the real world of computing: real-time computations are everywhere, and input data sensitive to the computation that processes it (or the other way around) abound, most notably in industrial control processes whose primary purpose is to interact with and thus control or constrain equipment. In addition, the pursuit and evasion problem establishes a very strong result: no matter how many processors we have at our disposal, we can always come out with an instance that overwhelms them.

Constrained sorting is another notable problem: It is very simple, it features superunitary behavior, and is not even real time. We chose it mostly for the last feature, since superunitary behavior examples come overwhelmingly from the real-time domain. The problem features constrains that should be observed not just at the end of the computation but all the time while the computation is being carried out. Again, there is nothing unusual about such a requirement, as situations like this do appear in real-world computing systems [14].

The papers that introduce these two problems show that they are not solvable unless enough resources (namely, processors) are available, further implying implicitly or explicitly that this violate the Church-Turing thesis. Based on these two problems and many other similar ones it was often concluded that the thesis only holds for "classical" problems, where the input is all available at the beginning of the computation, the output should be made all available at the end of the computation, and the computation itself is not constrained time-wise or indeed in any other way, except for providing the correct output for the given input.

What happened in reality is that the proofs that these two problems are unsolvable unless enough resources are available hold only for direct, algorithmic approaches. Somehow surprisingly, taking the roundabout approach of converting these problems to a temporal logic formulation and then solving that is far more successful and turns out to invalidate the proof of unsolvability. This also invalidates the aforementioned direct, algorithmic approach, meaning that there must exist classical, sequential algorithms running on say, the universal Turing machine, that solve these two problems. Turing wins this particular round.

A quick look at the many other superunitary behavior problems would appear to suggest that they too are amenable to similar approaches. Whether this is indeed the case however remains to be established. We believe that investigating a general, logic-based framework of superunitary behavior is the way to continue this investigation rather than tackling these problems one by one; this way Turing may win or lose a whole match in a single swoop.

In the end the Church-Turing thesis remains a… thesis, which can be disproved but can never be proven. Temporal logic is not a panaceum and has its own limitations. For example our CTL formulations look clean and simple in the end but have been a challenge mainly because neither CTL nor CTL* can express all the regular properties [4]. We believe our work suggests two, apparently contradictory but equally promising lines of investigation. On one hand, the general, logic-based framework for superunitary behavior as we know it today is worth pursuing as we already mentioned earlier. On the other hand, digging deeper into the limitations of temporal logic has the potential of revealing new, stronger problems that feature superunitary behavior. However, one needs to be careful in this approach, for where temporal logic fails some other Turing-complete model of computation might succeed.

Bibliography

[1] S. G. AKL, *Unconventional wisdom: Superlinear speedup and inherently parallel computations*, International Journal of Unconventional Computing, 13 (2018), pp. 283–307.

[2] S. G. AKL AND S. D. BRUDA, *Parallel real-time optimization: Beyond speedup*, Parallel Processing Letters, 9 (1999), pp. 499–509.

[3] R. ALUR AND D. L. DILL, *A theory of timed automata*, Theoretical computer science, 126 (1994), pp. 183–235.

[4] R. AXELSSON, M. HAGUE, S. KREUTZER, M. LANGE, AND M. LATTE, *Extended computation tree logic*, in International Conference on Logic for Programming Artificial Intelligence and Reasoning, Springer, 2010, pp. 67–81.

[5] R. BERGER, *The undecidability of the domino problem*, no. 66 in Memoirs of the American Mathematical Society, American Mathematical Society, 1966.

[6] S. D. BRUDA AND S. G. AKL, *Towards a meaningful formal definition of real-time computations*, in Proceedings of the Fifteenth International Conference on Computers and Their Applications, Citeseer, 1999.

[7] ——, *Pursuit and evasion on a ring: An infinite hierarchy for parallel real-time systems*, Theory of Computing Systems, 34 (2001), pp. 565–576.

[8] E. M. CLARKE AND I. A. DRAGHICESCU, *Expressibility results for linear-time and branching-time logics*, in Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems), Springer, 1988, pp. 428–437.

[9] J. H. CONWAY, *On numbers and games*, AK Peters/CRC Press, 2000.

[10] B. J. COPELAND AND D. PROUDFOOT, *Alan turings forgotten ideas in computer science*, Scientific American, 280 (1999), pp. 98–103.

[11] R. DE NICOLA AND F. VAANDRAGER, *Three logics for branching bisimulation*, Journal of the ACM (JACM), 42 (1995), pp. 458–487.

[12] M. J. FISCHER AND R. E. LADNER, *Propositional modal logic of programs*, Journal of Computer and Systems Sciences, 18 (1979), pp. 194–211.

[13] O. GRUMBERG, E. CLARKE, AND D. PELED, *Model checking*, 1999.

[14] N. NAGY AND S. AKL, *Time inderterminacy, non-universality in computation, and the demise of the church-turing thesis*, Tech. Rep. 011-580, Queen's University, 2011.

[15] A. PNUELI, *The temporal semantics of concurrent programs*, Theoretical computer science, 13 (1981), pp. 45–60.

[16] R. M. ROBINSON, *Undecidability and nonperiodicity for tilings of the plane*, Inventiones mathematicae, 12 (1971), pp. 177–209.

[17] J. E. SAVAGE, *Models of computation*, vol. 136, Addison-Wesley Reading, MA, 1998.

[18] H. T. SIEGELMANN, *Computation beyond the Turing limit*, Science, 268 (1995), pp. 545–548.

[19] A. M. TURING, *On computable numbers, with an application to the entscheidungsproblem*, Proceedings of the London mathematical society, 2 (1937), pp. 230–265.

[20] M. Y. VARDI AND L. STOCKMEYER, *Improved upper and lower bounds for modal logics of programs: Preliminary report*, in Proceedings of the seventeenth annual ACM symposium on Theory of computing (STOC 85), Lecture Notes in Computer Science, 1985, pp. 240–251.

[21] WIKIPEDIA, *Church-Turing thesis*. https://en.wikipedia.org/wiki/Church-Turing_thesis.

[22] D. WOOD AND D. WOOD, *Theory of computation*, Harper & Row New York, 1987.

[23] R. ZUO, *On the equivalence between computation tree logic and failure trace testing*, Master's thesis, Bishops University, 2018.