A Formal Expression of the Safety and Functional Requirements of a Safety-Critical System

by

M M West, T F Buckley & P H Jesty
Division of Computer Science

December 1993

This report is an expanded version of Research Report 92.4: Pelican Safety Study.

† Dr Tom Buckley died during the latter part of the DRIVE Safely project.
This paper overviews the work of the EC funded DRIVE Safely project and some of the recommendations for development and production of safe Road Traffic Informatic systems. A selection of the methods currently identified as suitable for formal specification of safety-critical systems is reviewed. A formal specification of the functional and safety requirements for "Pelican" equipment is given in Z. This is used to identify deficiencies in the Department of Transport source specification. The implications of the lack of "formal methods technology" on the choice of a formal specification method for safety-critical systems is discussed.

1. Introduction

1.1. Background

The European Community DRIVE programme is stimulating development of advanced computer systems for road transport, both for use in vehicles and also in roadside equipment. New applications arising from DRIVE projects include route guidance systems and dynamic traffic management systems[1], and these can be seen to have safety implications for road traffic. The work of the DRIVE I project V1051‡, DRIVE Safely has involved the investigation of current and potential system and software engineering processes suitable for the development and production of safe Road Transport Informatic (RTI) systems. The end products of DRIVE Safely included a draft consultative document: Towards a European Standard: The Development of Safe Road Transport Informatic Systems (Draft 2)[2,3]. This draft covers system and software considerations as well as safety and certification criteria. The legal process by which all new vehicles and roadside equipment is approved by the Department of Transport is called Type Approval, and the ultimate intention is for the enhancement of the present Type Approval procedures so that they explicitly include evaluation of road transport systems containing embedded software.

1.2. Statement of Requirements

Documentation for Type Approval must include a statement of system requirements and, for a software embedded system with safety implications, this statement should include both safety aspects and a specification of the intended system behaviour. Leveson emphasises the importance of techniques which allow consideration of a system rather than just software in isolation, and the benefits of a formal framework for system requirements have been identified by other authors[4,5,6,7,8,9,10]. Leveson[11] has noted that a clear distinction must be made between the safety-critical and the mission orientated behaviour of a system and such a distinction has also been examined by Jahanian and Mok[6] and Lemos[12]. This paradigm is encapsulated in the Safety Life Cycle of the IEC draft standard[13]: for a safety related system, a particular set of non-functional requirements must be identified, viz the safety requirements. The safety requirements often specify what a system should not do, and are weaker than the functional requirements. Instead of attempting the more complex task of proving that the software implementation satisfies its original specification in terms of functional requirements, it is simpler to prove that it satisfies certain criteria for acceptable behaviour[14], the so called safety invariants. Indeed the safety invariants amount to the top level specification of a safety-critical system.

‡ The consortium for this DRIVE I project consisted of

The Institute of Traffic Safety, TUV Rheinland, Cologne, Germany
Program Validation Limited, Southampton, UK
The Safety Critical Computing Group, Leeds University, UK
TNO Road Vehicles Research Institute, Delft, The Netherlands.
1.3. Case Study: Pelican

The UK Department of Transport (DTp) issued an initial statement of requirements for Pedestrian Operated Traffic Signal Equipment (Pelicans)[15], and this document forms the basis for Type Approval. The Pelican Controller was selected as one of a series of case studies undertaken by DRIVE Safely[16, 17, 18] to aid the investigation into development methodologies. The case study was chosen partly because although non-trivial, it was small enough to be tackled in the time available (6 man months for the DRIVE work). Also, the DTp were concerned about equipment behaviour as there had been reports of its malfunctioning; the audible signal which acts as an indication to blind and partially sighted people when it is safe to cross had been sounding when the other lights were not working.

The group at Leeds University carried out an initial safety analysis of the Pelican equipment and produced formal specifications of the safety and functional requirements in both Z[19] and OBJ[20]. TNO concerned themselves with the safety aspects and with equipment malfunction[21], and the response from PVL principally concerned the presentation of the Z specification. The timing aspects of the Z (and OBJ) specifications were modelled by a complex series of constraints, so PVL thought it worth their while to produce an alternative specification which tackles the problem of timing in a different and simpler way[22]. Z and OBJ were chosen for formal specification because of the necessity of using a mature method with some technological support[23] for a safety-critical application; we wished to investigate both the possibilities and limitations of such methods.

In this paper we overview the work of DRIVE Safely and their recommendations as to development methods for Road Transport systems containing significant software. We then review some of the formal notations and methods currently used to specify requirements. A formal specification in Z of the safety invariants and a small fragment of the functional requirements for the Pelican equipment is then presented and the relationship between them demonstrated via animation[24] and proof outlines. The paper adopts some of the ideas from the PVL specification, in presenting an updated version of our original. In addition it presents some conclusions as to the suitability of Z for timed systems such as the Pelican and the implications this has for formal specification of safety-critical systems.

2. The Work of Drive Safely

Vehicle computer applications currently include airbags and integrated vehicle control systems. Airbags are designed to inflate rapidly to provide protection for drivers and passengers in the event of a frontal crash, the intention being to reduce both the severity of injuries as well as the number of deaths[25]. The equipment consists of crash-detecting sensors, a subsystem for inflating the airbag, warning device(s), as well as an Electronic Control Unit. The airbag application has direct implications for safety and, in consequence, sophisticated monitoring devices have been designed to ensure that the subsystem will operate correctly in a crash situation, and that the airbag will not inflate when not required, for this could itself cause a crash.

Complex integrated vehicle control systems are being developed that will enable the vehicle to respond electronically to the drivers demands, and in as safe a manner as possible[26, 27, 28, 29, 30, 31]. In order to meet exhaust emission and noise standards, the optimisation of the engine and fuel injection system is required and this is achieved electronically. Such systems control both the formation of the fuel/air mixture and the timing and manner of the ignition process. In addition, electronic transmission controls supervise the transmission sub-system for functionality and feasibility, ruling out driver control errors which could affect the safety of the vehicle. (For example, it is possible to prohibit an automatic transmission from unexpectedly shifting while cornering.) The interaction between the ignition process and transmission control is achieved by the availability of information concerning the ignition process, to the transmission control unit. The transmission control unit receives signals from the engine unit concerning engine speed and fuel injection timing signals, and calculates engine torque accordingly. Further integration of control units are proposed, for example an interconnected system comprising automatic transmission, antilock braking system, ignition
control and traction control. One suggested method of coping with the complex control processing required for such integrated control units is to provide a single-chip microcomputer that incorporates a real-time operating system as firmware[32]. The code for the operating system is installed permanently in read only memory and cannot be altered.

However the approach which examines vehicle-driver relationship in isolation is not adequate to solve the problems created by traffic, as the vehicle-driver exists in a traffic environment, and it is the whole traffic environment which must be considered. One of the intentions of DRIVE is to utilise embedded computer systems to maximise the efficiency of the existing road network, as well as improving safety and protecting the environment. These RTI systems are more strictly known as Advanced road Transport Telematic systems or (in the US) Intelligent Vehicle-Highway Systems. They also employ data communications and examples are ramp metering and route guidance. If a motorway is congested then ramp metering techniques can be used to restrict the number of vehicles entering it[33]. Operation of the metering system entails real time monitoring of the traffic situation on both the motorway and the slip road. This includes assessment of the vehicle queue in the slip road to determine if it is too long and if necessary allowing vehicles to leave it and join the motorway, for the length of the queue on the slip road is seen as being more significant than any ramp metering requirements. Communication between roadside equipment and on-board computers provides route guidance and warns about emergency situations[34, 35, 36, 37, 38]. Information relevant to recommended routes (such as traffic problems) is collected, processed and communicated to drivers. The way that drivers perceive and use the data is regarded as important, for a possible danger is that drivers may be confused if there is either too much information or it is badly presented.

The intention of the DRIVE programme is that programmable digital computer hardware and software (and associated data communications) should provide improved or new functionality to road transport systems to achieve increased safety and decreased congestion and pollution. The line of reasoning is that increased safety will be achieved by increasing sophistication and complexity. However complexity may in itself lead to design problems so that functionality cannot be equated with safety: the prime aim of the work undertaken by DRIVE Safely was to ensure that safety is not compromised by an uncontrolled proliferation of computer systems in road transport. The DRIVE Safely project culminated in a draft[2, 3] of a set of proposals for a European Standard. In drafting this document we imported ideas from other industrial sectors who utilise significant high integrity software including the defence industry[39, 40], the civil aircraft industry[41] and security[42]; we discovered no existing standard specific to RTI. In addition we used two draft standards produced by the International Electrotechnical Committee (IEC) : Draft - Software for Computers in the Application of Industrial Safety Related Systems[43] and Draft - Functional Safety of Programmable Electronic Systems: Generic Aspects[13]. These IEC generic standards were designed to form a template from which further application specific standards could be produced, and our draft proposals could lead to a sector standard for the road transport industry. The intention of Towards a European Standard is also to enhance the Type Approval process in Europe so that it explicitly includes embedded computer systems, and contained in the proposals are system and software considerations, as well as safety and certification criteria. In particular our document proposes the establishment of a European Certification Body. (The safety criteria are outlined in a later section of the paper.)

It has to be emphasised that these are proposals, the details of which can be altered in the light of discussions and the establishment of a consensus. This work will have to involve a wider body of interested organisations than the DRIVE Safely consortium, and a draft of the proposals has been forwarded to the European Committee for Standardisation (CEN) for consideration as a European Standard for RTI specific safety-critical systems.

The philosophy of the document is not to impede new methodologies by being too prescriptive about recommended methods. The intention is to build on accumulated experience; one proposal is to combine the jobs of Certification Authority with that of information disseminator. However the documentation of specification and design activities is regarded as of supreme importance and it is recommended that modern document writing guidelines[44, 45] should be followed for this.
Thus, in order to have confidence in the safety of a system we must have knowledge of its
development processes, and for this to be possible, adequate documentation is essential. Each
stage of the life-cycle will produce documentation that will be submitted to the Certification
Authority for their perusal. Documentation of sufficient quality and detail is seen as a benchmark
against which system safety and quality can be measured. Once the evaluators are satisfied that
the system has been designed according to their regulations, the system will be certified for
operational use.

Documentation for Type Approval must include a statement of system requirements and, for a
software embedded system with safety implications, this statement should include both safety
aspects and a specification of the intended system behaviour. In order for such a statement to be
both complete and unambiguous its expression in formal mathematical terms is "thoroughly
recommended" by Towards a European Standard. A review of some of the formal specification
techniques is contained in that document, together with tool support currently available.

3. A Formal Framework for Requirements

This section presents an outline of methods which have been identified as suitable for a formal
expression of requirements. However it is not confined to those reviewed in Towards a European
Standard. It is fairly wide, because of the variety of system architectures employed by RTI
systems. It includes first order approaches[5,6,7], modal logic[8,9,10] and Synchronous
Concurrent Algorithms[46]. We have also included a Z timing formalism[47] and some Z case
studies of timed systems. The approach is informal and aims to give an intuitive "flavour" of the
different formalisms. The object is to examine and compare their treatment of issues such as
timing, safety, causality, rather than present any exhaustive and lengthy survey; our choice
favours non-diagrammatic methods so that appropriate comparisons can be made, so we have not
included methods such as Petri Nets[4].

3.1. First Order Approach to Time

Any model of RTI systems (such as those described in the previous section) ought to allow an
expression of timing and two approaches contrasting sharply in their modelling of time are first
order and modal[48]. In the first order approach time is regarded as another variable: the
traditional mathematical view in which time is an independent variable and usually modelling is
via the differential calculus. The "first order approach to time" is one where quantification is over
time: \( \forall t : P(t), \exists t : P(t) \). For the remainder of the paper that is what we mean by a "first order
approach to time" and the following subsections present a few examples of this approach.

Requirements State Machine: Jaffe and Leveson[5] describe how a Requirements State Machine
(RSM) model of a process-control system can lead to a formal model in first order logic. The
intention of the formal model is to provide a minimum formal description which, it is suggested,
can be subsequently mapped onto a real-time specification language.

The authors take as a case study a reactor tank; absolute time is represented by a clock of known
precision and thus the time of any input cannot be known more precisely than the granularity of
the clock. Startup and shutdown situations are examined as these are deemed particularly
important. The behaviour of the control subsystem is defined with respect to assumptions about
the behaviour of other parts of the system, sensors, for example. The authors intention is to
define important properties of real-time, process-control systems which must be specified and to
identify criteria (such as reachability criteria) which imply these properties. Thus a formal model
is built in order to be analysed for certain desired properties concerning behaviour.

The RSM is a Mealy machine with a state transition function mapping between states where the
next state depends on values and timings from sensors. Analysis of this function determines the
reachability of any state from a given state where reachability is formally defined in terms of
possible paths between states. One of the criteria is that the computer must never issue a
command which will move the process from a safe to an unsafe state; there must be no paths to
undesired hazardous states. Further, given that it is possible for the system to get into an unsafe
state by some outside event then it ought to be possible for the system to recover: every path from
a hazardous state must lead to a safe state.

Real Time Logic or RTL is a formal logic described by Jahanian and Mok as being particularly amenable to formalise timing considerations in safety analysis[6, 49]. The object is perceived as relating the safety assertion (the authors term for safety invariant) to the systems specification, and there are three possibilities. In the first case the safety assertion is derivable from and thus logically weaker than the systems specification. This means that the system is safe with respect to the behaviour denoted by the safety assertion. Secondly, if the safety assertion is unsatisfiable with respect to the system specification, it is inherently unsafe, and thirdly the negation of the safety assertion is satisfiable only under certain conditions so that additional constraints are needed to ensure system safety.

In order to capture timing and other behavioural aspects an event-action model is built which is then automatically formalised. Primitive and composite actions are described for an illustrative real-time system, viz a Martian Lander[6]. The object is to control acceleration, velocity and position of a space vehicle in order to achieve its safe landing. The event-action model is then formalised into RTL. RTL is a first order logic which is specific to systems where absolute timing is important as well as relative ordering of events. The authors justify their choice of RTL rather than temporal logics for specifying such systems with the fact that temporal logics are more concerned with relative ordering than with absolute timing. The modelling of the real-time system is characterised by the notions of events, actions, state predicates and timing constraints. Events are significant points of time or time markers and timing constraints assert the absolute time of certain events and hence specify the performance of the system. Event constants denote the start and stop of specific actions and state predicates denote the physical states of the system (eg whether the emergency landing mode, ELSM, is off or on) and may change as the result of an action or of an external event.

RTL reasons about the occurrences of events, and central to this is the occurrence function "@" where \( @(e, i) \) denotes the time of the \( ith \) occurrence of event \( e \). For example

\[
@((ELSM := F), 1) = 0
\]
captures the fact that initially ELSM is off. The systems specification and safety assertions for the Martian Lander are both represented in RTL and the safety assertions are shown to be consistent with the former in the following manner. The RTL formulae are transformed into formulae of Presberger arithmetic and a resolution based theorem prover is used.

Timed History Logic: Saeed[7, 12], adopts an approach to requirements analysis which shares some features with our own (as will be seen). If \( V_{P_i} \) is a set of possible values of state variable \( P_i \), then the state space, denoted \( \Gamma \), is such that:

\[
\Gamma = V_{P_1} \times V_{P_2} \times V_{P_3} \ldots V_{P_n}.
\]

In order for the model to express timing issues, the states of the system are related to time, \( T \), through a history function, \( H \), thus:

\[
H : T \rightarrow \Gamma,
\]
where \( T \) is modelled by the real numbers. Intervals are defined in terms of beginning and end points, and this concept is used to define modes of operation as a possible way of structuring specifications. A feature of the work described is that two levels of abstraction are distinguished: real world and controller. The safety analysis is performed at the real world level, the level of the physical process and its behaviour. The controller level consists of the sensors, actuators and any operator console. The concept of a universal history set is described, the set of all possible histories of the system. In addition different properties (relations) of these histories are identified, including relations holding over every time point of the system lifetime and relations which hold over every interval.

The system chosen to illustrate the method is a chemical plant (reaction vessel) where for a reaction between chemicals to take place, the temperature of the vessel must be above a certain level. In order for there not to be an explosion, the temperature must not rise above a certain specified value when the vessel contains chemicals. This constraint is formalised as a safety
invariant and the set of safe histories is defined as those histories satisfying it. This safety condition is then proven for all phases of operation; ie for all possible histories the vessel is safe.

The authors remark on the benefits of their work, one of which is the possibility of a rigorous relationship between a system and its environment. However they also identify a limitation in the lack of automated tools for analysis and simulation of the model.

3.2. Modal or Temporal Approach to Time

In the first order approach time is regarded as another variable; this contrasts with the modal or tense approach where (in general) there is no independent time variable. Universally and existentially quantified expressions over time such as $\forall t \ P(t)$, $\exists t \ P(t)$ are replaced by $\square P$ and $\Diamond P$ which express  *it is always the case that P*, *it is sometimes the case that P* respectively. The unary modal operators $\square$ and $\Diamond$ have these (as well as other) interpretations. An extensive introduction to the subject of time considerations in logic and computing is contained in a text by Galton[48] where the tense and non-tense approaches are compared, and different temporal logics overviewed; logics and models of real time have been surveyed more recently by Alur and Henzinger[50]. Galton describes a minimal tense logic and compares its expressiveness and clarity with equivalent first order expressions. The tense logic expressions can be thought of as "macros" of the first order expressions. The tense logic expressions are more like natural language and simpler; however there are some first order properties of temporal orderings which are not capable of re-expression in the minimal tense logic, which must then be extended. Further, for computer scientists, first order treatments lend themselves more readily to implementation as logic programs.

Galton further identifies an important philosophical difference between the two approaches in that in the first order approach to time anything which exists exists timelessly and time is regarded as no different in this respect to the three space variables; the notion of transience is not expressible.

*Safeness and Liveness* are important concepts when describing timed systems using temporal logic. The invariance operator $\square$ is related to safeness or invariance, thus *Nothing bad will ever happen* and the eventuality operator $\Diamond$ to liveness thus *Some functional property will eventually occur*. Examples of safety and liveness given by Barringer[8] are: *the reactor will never go critical* and *whenever the reactor is instructed to shut down, it will eventually do so*. These two properties are of a systems *behaviour*, where a behaviour can be defined as a sequence of states. A formal definition of safety and liveness with respect to a systems behaviour is given by Alpern and Schneider[51] who examine their topological characteristics.

*The Temporal Logic of Actions* or TLA is described by Abadi and Lamport[52] and includes the real time variable now as well as temporal operators. The timing properties are expressed separately from the non-timing, and the two conjoined to make the specification of the whole. The authors express the view that this separation makes real-time specifications easier to write and understand. In order to ensure that the real-time variable now never decreases, the following property is required:

$$ RT \equiv ( now \in \mathbb{R} ) \land \square [ now' \in ( now, \infty ) \lor ( now = now' ) ] $$

meaning that now is a real number and its new value now’ is always $\geq$ now. Both safety and liveness properties are analysed and liveness is expressed by the condition:

$$ NZ \equiv \forall t \in \mathbb{R}: \Diamond ( now > t ). $$

If time becomes an ordinary program variable then a real-time version of Zeno’s paradox can occur where time increases but is bounded. $NZ$ is the "non-Zeno" condition that now can get arbitrarily large, and not exhibit pathological properties such as never exceeding some upper bound.

The method is illustrated by both a queue and by a mutual exclusion protocol as examples of "closed systems", where the environment is represented as part of the system. The particular problems introduced by open systems (defined by the authors as those where the environment and system are separated) are discussed: a system can only behave correctly if the input from its
environment is correct (expected). The environment of open systems (by definition) cannot be controlled and it should not be possible to write a specification which implicitly controls the environment. The authors suggest a style particular to TLA in which it is possible to avoid such a paradox.

Interval Logic: An early representation of temporal knowledge and reasoning via intervals was Allen[53]. In this first order approach the interval is primitive and temporal relations are defined between intervals $t$, $s$ say, thus $t < s$ means the whole of interval $t$ is before the whole of interval $s$. There are seven relations in all between $t$ and $s$; $t$ before $s$, $t$ equals $s$, $t$ overlaps $s$, $t$ during $s$ etc. Properties can be "inherited" for if $P$ holds during $t$ and $t$ is contained in $s$ then $P$ holds during $s$[54]. However the concept of "event" is different from that described for RTL[6] where events are "points in time". In Allens work events describe an activity and hold over an interval and there is no way of expressing a "point in time".

A modal approach is taken by Moszkowski[55, 56, 57] viz "Interval Temporal Logic" (ITL) in which discrete intervals of time are primitive and occupied by a sequence of states over the time instants. Unary operators $\Box$ and $\Diamond$ can be intuitively understood as true on all subintervals and true on some subinterval when applied to some property of the state sequence. The logic expresses change over time. Moszkowski gives the advantage of using discrete time as corresponding most closely to our mental model; moreover we can control the granularity of the time model.

Of particular interest is 'chop' denoted " $\cdot$ " , where $B \cdot C$ is true for any interval which can be split into two sub-intervals, the first satisfying $B$ the second $C$. Hale[56] introduces Tempura which is a temporal logic language capable of implementing a prototype of ITL and Kono[57] presents work on an automatic theorem prover for Interval Temporal Logic.

Duration Calculus: An extension to ITL from discrete intervals to real-time is Duration Calculus[9] where state variables are introduced as total functions from time (the real numbers) to $\{0, 1\}$. The assignment $1$ denotes the condition that the system under consideration is in the state, and $0$ that the system is not in the state. Thus if $P(t)$ is such a function then the duration of $P(t)$ in the observation time interval $[a, b]$ is denoted in duration calculus by

$$\int_a^b P(t)$$

where $1 = b - a$, the interval length. An interval of non-zero duration is a "proper observation interval" where $1 > 0$. The calculus is thus able to take into account change over time of continuous variables, such as gas leaking from a burner. A case study example is such a gas burner where a formalised safety requirement is that

$$\int 1 \geq 60 \text{ sec.} \Rightarrow 20 \int \text{Leak} \leq 1.$$  

This means that when the duration of an interval is at least 60 seconds, then a leak in the gas burner must not last more than one twentieth of that time in total. A further denotation is where state $P$ holds throughout an observational interval ie predicate $\int P$ where the predicate which is true only for point intervals is $\int$ and

$$\int P \equiv (\int P = \int 1) \land (\int 1 > 0).$$

An induction rule is introduced which extends a hypothesis over adjacent intervals. The rules for implementation of the safety requirements are formally stated and the calculus is applied to prove that the implementation satisfies the requirements. (These implementation rules are, of course, functional requirements whose mission is safety.)

3.3. Modal Action Logic

The concept of causality is examined in detail by Allen[54] with the relationship between actions, agents and events. This first order formalism is mainly utilised in natural language processing and problem solving. A modal logic which also incorporates causal concepts is Modal Action Logic (MAL), a deontic extension to dynamic logic[58, 59, 10, 60], where deontic logic is a logic of permission and obligation. MAL underpins the work of the FOREST approach[61], a method
which incorporates both causal and temporal aspects and is intended to integrate with the phased approach of system development. The set of variables of a system are a state of information or world, and actions which effect transitions from one state to another are modal operators with as modal connectives. Furthermore, actions are performed by agents, for example

\[
\text{Pre-condition} \Rightarrow \{A, a\} \text{ Post-condition}
\]

means that if action \(a\) is performed by agent \(A\) in a state (world) in which Pre-condition is true, then the result is a state (world) in which Post-condition is true. \(\text{PER}\) and \(\text{OBL}\) are "deontic predicates" and axioms which involve them are local to some agent. Thus the fact that action \(a\) is permitted is denoted by \(\text{PER}(A, a)\) and that \(a\) is obliged is denoted by \(\text{OBL}(A, a)\). The notion of obligation overrides that of permission; if an action is obliged, then any permissions are suspended until the obliged action is completed. A variant of Allens Interval logic is used for temporal reasoning[61].

The need for proof support is recognised and the MAL prover[60] identified as suitable. The paper describing the MAL prover uses it to establish safety properties for a railway-signaling system. A safety property of the signaling system is presented here to illustrate MAL. A railway layout is composed of track segments, each of which is provided with an identifier. Train movements are governed by signals which themselves are controlled by interlocks. If there is a train occupying a track segment (ts say) immediately following a signal (s say) then the interlock (agent) does not permit the action of setting s to indicate "proceed". This safety property is intended to prevent a further train running into the back of the one occupying ts and is expressed:

\[
\forall t : \text{train} \bullet (on(t, ts) \Rightarrow \neg\text{PER}(\text{set\_signal}(s, \text{proceed}), \text{interlock}))
\]

The method of capturing requirements using agents and associated actions, permissions and obligations supports the construction of the requirements specification. A further advantage over other methods is seen[10] as the ability to characterise both normative and non-normative behaviours so that error situations can be analysed. An example specification which illustrates this feature is a library system with the usual facilities for library users to borrow books. The criterion for normativeness is given as a property of states, rather than as a property of a sequence of states for this is seen by the authors as superior in facilitating error-recovery. The criteria for normativeness utilises both permission and obligation; for example permitted actions when performed on normative states lead to normative states. One kind of non-normative state is where a borrower has not returned or renewed a book by its due date and error recovery is subsequently achieved by means of the action issue\_fine. In order to represent such a temporal constraint the authors specify a new agent big\_ben, counting hours and days by means of natural numbers. The actions and permissions of the new agent are distinct from those of library. Thus the action strike\_hour changes the hour/ time and is specific to big\_ben.

### 3.4. Synchronous Concurrent Algorithms

Synchronous Concurrent Algorithms (or SCA’s) can provide a model for many kinds of architectures and algorithms including all types of serial and parallel computers and digital controllers. SCA’s provide a framework for expressing relationships between timing and processes, and are not associated with any one formal system. SCA’s are based on a network of modules and channels, synchronised by a global clock. Streams of (input) data processed by the network emerge as infinite streams of (output) data. \(PR(A)\) (primitive recursion over abstract data type \(A\)) is a mathematical formalism used to describe SCA’s[46,62] which is chiefly algebraic and where the relationships between objects of interest are defined by equations[63].

In the simplest case \(A\) is a data set \(\{a_i\}\) and we assume that each module of the network is capable of processing a vector \((a_1, a_2, .. a_n)\) of data and of holding the computed item. At any given time cycle the module simultaneously accepts vector \((a_1, a_2, .. a_n)\), and outputs the data item computed previously. It subsequently processes \((a_1, a_2, .. a_n)\) to produce a new data item \(a_n\), which is held until the next cycle. In order for synchronisation, there is an upper bound, \(t_b\) for the module to complete its processing cycle, ie one "step". The modules receive their data precisely when they require it, and each component of the vector arrives simultaneously at the
module which requires it as input. Some (source) modules are connected to sources of input to
the network, and some (sink) modules to outputs from the network. One channel only is
associated with each source and sink module although channels may fork to produce more than
one copy of data. In Figure 1, if vector \((a_1, a_2, \ldots, a_n)\) is available at time \(t\) to module \(m\) then two
copies of \(a_m = f_m(a_1, \ldots, a_n)\) are output by \(m\) at time \(t + 1\).

**Synchronous Concurrent Algorithm: Figure 1**

\[ m \]
\[ a_1 \]
\[ a_2 \]
\[ a_n \]
\[ f_m \]
\[ a_m \]
\[ a_m \]

The clock measures discrete time \(\{0, 1, 2, 3, \ldots\}\). A data stream over set \(A\) from some source
module \(i\) will be of the form \(a_i(t), a_i : T \rightarrow A\) where \(a_i(t)\) is the data item supplied by source \(i\) at
time \(t\). Assuming \(n\) sources, at time \(t\) a vector
\[ a(t) = (a_1(t), a_2(t), a_3(t) \ldots a_n(t)) \]
is available for processing as part of an input stream \(s\) where \(s\) is defined as a total function over
time:
\[ s : T \rightarrow A^n. \]

In general the network processes sets of such streams so if a set of input streams is \([ T \rightarrow A^n]\)
and a set of output streams is \([ T \rightarrow A^m]\) then the processing of the stream can be captured by the
function
\[ \Phi : [ T \rightarrow A^n ] \rightarrow [ T \rightarrow A^m ]. \]

The mathematical formalism underpinning the above is primitive recursion over algebras. (A
proof assistant for primitive recursion has been developed by Eker and Stavridou[64].) The
following example which illustrates primitive recursion is taken from Harman and Tucker[62].

The programmers model (PM) of a microprocessor transforms states of the microprocessor,
where a state comprises the contents of the machines registers and memory which are visible to
programmers. This is a more convenient model for a microprocessor application than a stream
transformer (SM) which maps input streams to output streams. The model PM is of a lower level
of abstraction than SM. If \(\sigma\) is the initial state then its next state will be \(\text{comp}(\sigma)\) at time \(t = 1, \text{comp}^2(\sigma)\) at \(t = 2\,\text{, and \,} \text{comp}^t(\sigma)\) at time \(t\), where \(\text{comp} : C \rightarrow C\) is the nextstate function. The
machine is the captured by
\[ \text{COMP} : T \times C \rightarrow C \]
where \(C\) is the set of states and COMP is a simultaneous primitive recursive function defined by:
\[ \text{COMP}(0, \sigma) = \sigma, \]
\[ \text{COMP}(t + 1, \sigma) = \text{comp}((\text{COMP}(t, \sigma)). \]

The verification of the microprocessor consists of a further specification at a lower level of
abstraction, ie one which involves design decisions, viz, the abstract circuit design, AC. In order
to relate the two specifications (levels of abstraction), the concept of re-timing is required. (A
re-timing is a mapping between two clocks.) This is because the speed of clock \(T\), related to
function COMP, may not correspond to real-time, for each state-to-state transformation will take
a differing amount of time to execute. The theory outlined above is applied to the PDP-8
microprocessor.

### 3.5. The Choice of Z for the Specification

This section first gives some examples of the use of Z for real-time and other timed systems and
next explains the reasons for the choice of Z. The first example given is a time framework in Z
which compares with ours in some respects, although the work was undertaken independently.

**Interval Logic in Z:** An interval model of time in Z has been developed by Coombes and McDermid[47] in which the interval is the basic unit and represented as a given set, INTERVAL. The time model which is formalised has a definite beginning but no end and is dense. Relations between intervals are axiomatically defined, for example before, during, in a manner akin to Allen. However (and unlike Allen) it is possible for an interval to be a single point, and this is accomplished by the condition that it is an interval during which no other interval can occur.

A subset INTSET of non-overlapping intervals is defined (which may be infinite) and further operations defined with respect to INTSET. In order to accommodate several clocks, a GRID (set of points on the time line) is constructed from a set of point intervals, and its granularity related to a specific INTSET. The history (or trace) of variable X is represented by a generic function, Itrace, with domain INTSET and range the type of X. The authors claim that the concept of time grids enables the specification to be looked at from multiple viewpoints, a necessary feature for describing distributed systems. A monitoring system consisting of two asynchronous channels and a comparator provides an example to illustrate the use of the timing model and the history function Itrace.

**Diagnostic X-Ray Machine:** The real-time kernel of a diagnostic X-ray machine has been specified in Z by Spivey[65] and models states that the kernel could occupy and events which take it from one state to another. The set of processes is defined, and processes which handle interrupts are distinguished from background processes, both of which are supported by the kernel. The kernel provides scheduling facilities and this is modelled by the Z specification. Priorities are associated with processes via a partial injection thus:

\[ \text{handler} : \text{ILEVEL} \rightarrow \rightarrow \text{PID}_1 \]

where ILEVEL is a finite set including all the priorities, PID$_1$ is the set of processes excepting the null process. The (interrupt) handler is a partial injection and captures the fact that not every priority level is associated with a handler and an interrupt handler is associated with a unique priority. States that single processes may occupy are specified, together with events such as an interrupt or the selection of a new current (background) process.

Spivey discovered that the specification appeared to allow a possible deadlock when no process was allowed to run. However it also appeared that the control software ruled out the combination of circumstances for this occurrence and moreover a hardware timeout was provided in case of failure. Thus patients using the machine would be protected. Nevertheless, the flaw in the design of the kernel does mean that the machine’s robustness is affected.

Deficiencies of the model are identified, eg there is no way of specifying timing or fairness, and the author suggests ways of remedying these. However the specification would then lose its simplicity and clarity. For example to add timing information in an adequate manner would require a much lower level of abstraction.

**Oscilloscope:** Delisle and Garlan specify an oscilloscope in Z[66]. The natural numbers model absolute time, and the integers model the voltage input to the oscilloscope and the horizontal and vertical coordinates on the screen. This is because the oscilloscope has both finite time and voltage resolution. Certain distinguished events are trigger events, and these are defined. The traces observed on the oscilloscope screen are defined relative to the timing of the trigger events. The oscilloscope’s architecture is captured by the Z specification and the authors remark that although this model is a rather idealised one it does provide insight into user requirements and to design. For example it allows the definition of a waveform, as a "bag" of time-voltage pairs. This definition exposes what is common to all waveforms, so that the many varieties can be handled more easily.

**Cyclotron:** A clinical Cyclotron control system has been specified in Z by Jacky[67]; the cyclotron provides fast neutrons for cancer treatment and has obvious safety-critical implications. The main specification includes control laws and safety assertions (in the same schema) and provides Z specifications for safety interlocks. The interlocks are pre-conditions for operation schemas. For example before the neutron beam is turned on, a protective door must be closed.
Timing considerations are not formalised for the authors do not consider Z a suitable notation for this.

The Integration of Z with Other Methods: In order to cope with the particular problems of timed systems, various extensions to the Z notation have been either developed or proposed. There are many examples of object-oriented extensions of Z and we give a brief description of one example in the next paragraph. MaMooZe[68] is an object-oriented extension of Z which also incorporates modal action logic. The refinement of state-based systems such as Z to link them with CSP[69] has also been suggested[70, 71]. There is also an "Event Calculus" which apparently resembles process algebras such as CSP and which links Z with a formal diagram notation[72].

Object-Z[73] adds the notion of Class to Spivey's Z type framework, where an object is an instance of a class. Events are operations (schema names) together with pre- and post-operational states. Of particular interest to our purpose is a class history, which is a sequential record of the operations undergone by an object: a sequence of events. Possible histories are constrained by the introduction of temporal operators including ◦ and □ where □P signifies that P holds for every event in the history.

The Choice of Z: Z was chosen for formal specification because we wished to explore both the possibilities and limitations of methods which were already used in industrial projects[74, 75]. It is increasingly the view by many authorities[76, 23] that the language chosen for safety-critical applications should be both reasonably mature and well understood by a significantly large enough Industrial Community. An advantage of mature methods such as Z is that they possess tool support such as type checkers and proof assistants. Z is currently undergoing standardisation. A particular recommendation is for animation of the formal specification[77] and a method of animating Z via Prolog had been developed by West and Eaglestone[24]. The paper is summarised below.

Animation may be used to derive the consequences of the specification, and to demonstrate the functionality of the target system. An appropriate example for safety-critical cases might be the set of inputs which could give rise to an erroneous or hazardous output. The draft IEC standard on industrial safety-related systems[43] states that the aim of prototyping/animation is

To check the feasibility of implementing the system against the given constraints. To communicate the specifier’s interpretation of the system to the customer, in order to locate misunderstandings.

The distinction between an animation and a prototype is discussed more fully in the above paper, but a brief summary is thus: it lies in the level of abstraction achieved and the scope of the facilities for deriving consequences. A prototype is a version of the required system itself, perhaps with non-functional requirements eased, whereas an animation is concerned with an abstraction of the required system. The abstraction may be created from formulae or Horn clauses, and the system object may be modelled using mathematical objects such as typed sets. The scope of the prototype is simply that of demonstrating the functionality of the required system, whereas an animation should make apparent the logical relationships within the specification[78]. The paper by West and Eaglestone explains the reasons why deriving theorems from formal specifications (although necessary) are not sufficient for making apparent such relationships. In order for the specification to stand up to critical examination by the customer (or user) there would be the need for an expert to reformulate the customers expressed query.

Animation is important for early validation of the specification and is a form of early testing; indeed the results of the animation can be set aside and compared with the final tests for the equipment[77, 79, 80]. Our tool was used for validation of the specification and to aid understanding. It provided for early identification of safe and unsafe states and as a tutorial introduction to the Z time framework; its use will be described in a later section.

4. System Requirements: Pelican Equipment

In order to establish the feasibility of our proposals, several case studies were undertaken by DRIVE Safely. The Pelican case study was the initial idea of the DTp[18] as they were concerned
about the statement of requirements for Pelican equipment: *Pedestrian Operated Traffic Signal Equipment (Pelican) : MCE0125a and MCK 1053*[15] which forms the basis for Type Approval in the UK. There had been reports of Pelican equipment malfunctioning; the audible signal had been sounding when the vehicle and pedestrian lights were not working. A problem for the DTp is that *Type Approval* principally involves black box testing of a prototype and quality control during production and does not taken account of the differences between electro-mechanical systems and systems with embedded software. Further, Pelican equipment is purchased by local authorities who may interface it with other equipment and/or modify it.

In this section the system requirements contained in *MCK 1053* are outlined and this is followed in the next by the Z specification of the equipment, which is based on it. This activity helped us provide a critique of the document for the DTp as well as aiding us in our study of methodologies. *MCK 1053* contains functional and safety requirements which are interspersed with the electrical and other specifications, and for this reason there was some difficulty in its interpretation. Additionally, some parts were ambiguous, and indeed the safety requirements were interpreted quite differently by individuals who worked independently on them. The authors of the document evidently did not have a programmable electronic controller in mind, but rather an electromechanical system in which certain events were physically constrained from ever happening.

The Z specification which follows captures the control aspects of Pelican equipment during the *fixed vehicle period* mode, when the equipment operates to a predetermined fixed period cycle on receipt of a pedestrian demand (see Table 1). State durations are not shown, but once a Pelican controller has been set up for a particular location, timings are then fixed in line with "Road Circulars" issued by the UK Department of the Environment. Extra safeguards are provided for equipment malfunction (for example, the combination of vehicle green and pedestrian greenman) as a redundant backup for the enforcement of the safety invariants. In addition there is a monitoring requirement for the red vehicle lights. If the equipment is found to be malfunctioning unsafely, then the policy of the UK DTp is to disable the display of any light or audible signal until corrective maintenance is available. This is then the *fail safe* state of the system, and its provision minimises the possibility of an accident if an error occurs, for this state is one which is guaranteed to satisfy the safety requirements but perhaps no others[11].

<table>
<thead>
<tr>
<th>STATE (period)</th>
<th>SIGNAL TO VEHICLES</th>
<th>SIGNAL TO PEDESTRIANS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Steady Green</td>
<td>Red Standing Man</td>
</tr>
<tr>
<td>2</td>
<td>Steady Amber</td>
<td>Red Standing Man</td>
</tr>
<tr>
<td>3</td>
<td>Steady Red</td>
<td>Red Standing Man</td>
</tr>
<tr>
<td>4</td>
<td>Steady Red</td>
<td>Green Walking Man</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Audible signal to pedestrian)</td>
</tr>
<tr>
<td>5</td>
<td>Steady Red</td>
<td>Flashing Green Man</td>
</tr>
<tr>
<td>6</td>
<td>Flashing Amber</td>
<td>Flashing Green Man</td>
</tr>
<tr>
<td>7</td>
<td>Flashing Amber</td>
<td>Red Standing Man</td>
</tr>
</tbody>
</table>

*Table 1 : Adapted from MCK 1053*

5. **Formal Representation of Functional and Timing Requirements**

This section presents a fragment of the Z specification of the Pelican equipment. Some comparisons are made in a later section with the methods of section 3. As the DTp document did not allocate functionality between hardware and software, the Z specification does not anticipate any particular hardware/software architecture. The normative states of the Pelican are first related to the signal configurations of the various periods of Table 1. Next, a time framework is developed in Z and the sequencing of the table states related to time by means of a history function.
5.1. The States of the Pelican

We relate the states of the Pelican to the light and audible signals displayed to vehicles and pedestrians. States are represented as abstract objects belonging to the set \( \text{SIGNALSTATE} \).

\[ \text{SIGNALSTATE} \]

The set represents all the states of the Pelican, and in order to have a robust specification, this includes non-normative states. The vehicle and pedestrian light colours, and the audible signal status are each represented by a set:

\[
\begin{align*}
\text{VCOLOUR} & := \text{green} \mid \text{red} \mid \text{amber} \mid \text{flashamber} \mid \text{vl\_off} \\
\text{PCOLOUR} & := \text{greenman} \mid \text{redman} \mid \text{flashgreen} \mid \text{pl\_off} \\
\text{BLEEPER} & := \text{bleep} \mid \text{nobleep}
\end{align*}
\]

where \( \text{vl\_off} \) and \( \text{pl\_off} \) represent the fact that the vehicle and pedestrian lights can be dark.

5.2. Representation of the Table States

The signal configurations specified by Table 1 are such that there are unique vehicle and pedestrian light colours associated with each state, and the vehicle light colours are modelled by a partial function \( \text{vlightshow} \) thus:

\[
\text{vlightshow} : \text{SIGNALSTATE} \rightarrow \text{VCOLOUR}
\]

Each state of the table is a member of the domain of \( \text{vlightshow} \) and for example \( \text{vlightshow}(s_1) = \text{green} \) means that \( s_1 \) displays green to vehicles in period 1 of the table. Pedestrian light colours and the status of the audible signal are respectively modelled by partial and total functions:

\[
\begin{align*}
\text{plightshow} & : \text{SIGNALSTATE} \rightarrow \text{PCOLOUR} \\
\text{audio} & : \text{SIGNALSTATE} \rightarrow \text{BLEEPER}
\end{align*}
\]

Thus the state corresponding to period 1 of the table, and denoted \( s_1 \) is captured by the schema \text{Period1}.

\[
\text{Period 1}
\]

\[
\begin{align*}
\text{vlightshow} : \text{SIGNALSTATE} \rightarrow \text{VCOLOUR} \\
\text{plightshow} : \text{SIGNALSTATE} \rightarrow \text{PCOLOUR} \\
\text{audio} : \text{SIGNALSTATE} \rightarrow \text{BLEEPER} \\
s_1 : \text{SIGNALSTATE}
\end{align*}
\]

\[
\begin{align*}
\text{vlightshow}(s_1) &= \text{green} \\
\text{plightshow}(s_1) &= \text{redman} \\
\text{audio}(s_1) &= \text{nobleep}
\end{align*}
\]

The other periods of the table can be similarly expressed; for example:

\[
\text{vlightshow}(s_2) = \text{amber} \ , \ \text{plightshow}(s_2) = \text{redman} \ , \ \text{audio}(s_2) = \text{nobleep}
\]

is the predicate of \text{Period2}. The other periods are represented by \text{Period3} , \text{Period4} , \text{Period5} , \text{Period6} , \text{Period7} . The states specified by Table 1 can be represented as a conjunction of the period schema, thus:

\[
\text{Table 1} \equiv \text{Period} 1 \land \text{Period} 2 \land \text{Period} 3 \land \text{Period} 4 \land \text{Period} 5 \land \text{Period} 6 \land \text{Period} 7.
\]

The sequencing of these table states is represented in \( Z \) in a later section.
5.3. Initial State

Prior to the equipment being switched on, all signals are off. The equipment returns to this state (denoted $s_8$) if an error is discovered, for example if there is a power failure. The schema which captures this condition is $InitState$. The conjoined expressions in the predicate of $InitState$ which capture the configurations of $s_8$ are

$$
\begin{align*}
 vl\text{lightshow}(s_8) &= \text{vl\_off} \\
 pl\text{ightshow}(s_8) &= \text{pl\_off} \\
 audio(s_8) &= \text{nobleep}
\end{align*}
$$

The schema $NormalStates$ models the table and initial states and these are defined as the normative states of the controller, thus:

$$
NormalStates \equiv \text{Table 1} \land InitState
$$

describes $s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8$. The functional properties of $vl\text{ightshow}$, $pl\text{ightshow}$ and $audio$ ensure that these states are distinct from each other.

5.4. Functional Requirements: Timing Aspects

An adequate model of timing constraints should be capable of expressing the functional requirements of the system, with respect to timing and sequencing of signals, and their response to external events. In particular, some representation of absolute time is important when specifying traffic control equipment. In order to achieve this end, time is measured from some origin by the natural numbers, $(TIME = N)$ where each natural number represents a unit of time which must be stated: our unit is one tenth of a second. The unit of time then represents the granularity of the model.

Thus the Z timing model can be visualised as a series of snapshots taken at unit intervals starting from some absolute point in time. Each snapshot records events in time and the signal configuration when the snapshot is taken. In order to capture the asynchronous events associated with the system, such as the wait button being pressed or a power failure, these events are synchronised so that they are deemed to occur at the next snapshot. The approach differs from that normally adopted in Z specifications in that pre-operational and post-operational variables are not coupled and compared. Instead we model various constraints on values within time intervals, so that we examine a system "history".

This history is considered from the moment the equipment is switched on, $\text{switchon}$ through powering up and through its runtime operation to its disablement and possible maintenance and reset, $\text{reset}$. Runtime operation can end either because the equipment is switched off remotely or if an error occurs. The end of runtime operation (for whatever reason) is denoted $\text{switchoff}$. After runtime operation there is a period of time (specified as less than half a second) during which the equipment is powered down to when it is disabled, $\text{offline}$ after which the equipment enters state $s_8$ for an unspecified time period, to $\text{reset}$. This paper concerns itself chiefly with the runtime operation of the Pelican, i.e. the period beginning $\text{online}$ and ending $\text{switchoff}$.

5.5. Time Framework : State Transitions

We consider any non-empty time sequence from $\text{online}$ to $\text{switchoff}$ to be punctuated by a finite set of distinguished times, called $transitions$,

$$
\text{transitions} : \mathbb{I} \times TIME
$$

This set identifies the absolute times when the Pelican potentially changes its signal configuration. The set of transitions gives rise to a set of non-intersecting intervals, bounded by the transition times, where each time interval is occupied by a signal state. In fact, there would be a period of time necessary for the transition, but this is disregarded in this model, as the unit time period is many orders of magnitude greater. Any analysis of the transition period would require consideration of the theory of real numbers and is beyond the scope of the paper. The history function is similar in concept to $I\text{trace}$ of Z interval logic[47]. However we relate a time interval to a state rather than to a variable, as in the latter. Also the time model captured by Z interval logic is closer to (although not identical with) the notions of the theory of real numbers.
If we represent a time interval as an ordered pair: \((\text{start}, \text{finish})\) then for example \((t_1, t_2) \mapsto s_1\), \((t_2, t_3) \mapsto s_2\) means that \(s_1\) occupies time period \(t_1 \leq t < t_2\), and that \(s_2\) occupies time period \(t_2 \leq t < t_3\).

The state occupying a time interval is unique and this is captured by the partial function \(\text{scenario}\).

\[\text{scenario} : (\text{TIME} \times \text{TIME}) \rightarrow \text{SIGNALSTATE}\]

An example of \(\text{scenario}\) is

\[\text{scenario} = \{(70,120) \mapsto s_6, (120,140) \mapsto s_7, (140,340) \mapsto s_1, (340,370) \mapsto s_2, (370,390) \mapsto s_3, (390,420) \mapsto s_4, (420,430) \mapsto s_5\}\]

where the integers represent tenths of a second. It can be seen that our example \(\text{scenario}\) has transition times as follows,

\[\text{transitions} = \{70, 120, 140, 340, 370, 390, 420, 430\}\]

To denote a set of intervals we use a set of ordered pairs \(\text{transsuccessor}\), denoting the successor transition, so that

\[\text{dom scenario} = \text{transsuccessor}\]

This set of ordered pairs must be a function

\[\text{transsuccessor} : \text{TIME} \rightarrow \text{TIME}\]

for in (\(\text{start}, \text{finish}\)) each \(\text{start}\) must have a unique \(\text{finish}\). For example

\[\text{transsuccessor} = \{(70,120), (120,140), (140,340), (340,370), (370,390), (390,420), (420,430)\}\]

The domain of \(\text{transsuccessor}\) is \(\text{transitions}\) less its largest member, hence

\[\text{dom transsuccessor} = \{70, 120, 140, 340, 370, 390, 420\}\]

and its range \(\text{transitions}\) less its smallest member, hence:

\[\text{ran transsuccessor} = \{120, 140, 340, 370, 390, 420, 430\}\]

where

\[\text{transsuccessor}(t) > t, t \in \text{dom transsuccessor}\]

The strict "\(>\)" means that intervals have non-zero duration.

We now have a formal framework for time, and a relationship of a time interval to the state occupying it. At any given time, the system state must be unique so that there should be no overlapping of time intervals; this is proved in the next section. The schema which captures this is called \(\text{RunTime}\).
### RunTime

| online, switchoff : TIME | (RT1) |
| transitions : \(F \text{TIME}\) | (RT2) |
| transsuccessor : TIME \(\rightarrow\) TIME | (RT3) |
| scenario : (TIME \(\times\) TIME) \(\rightarrow\) SIGNALSTATE | (RT4) |

- \(\forall t_1 : \text{transitions} \bullet (\text{online} \leq t_1 \leq \text{switchoff})\) (RT3)
- \(\text{dom transsuccessor} = \text{transitions} - \{\text{switchoff}\}\) (RT4)
- \(\text{ran transsuccessor} = \text{transitions} - \{\text{online}\}\) (RT5)
- \(\forall t_2 : \text{dom transsuccessor} \bullet \text{transsuccessor}(t_2) > t_2\) (RT7)

Note that both \(t_1, t_2\) in the quantified expressions are of type \(\text{TIME}\). This is because in expression (RT3) \(\text{transitions}\) is a finite subset of \(\text{TIME}\), and in expression (RT4), since \(\text{transsuccessor}\) is a partial function from \(\text{TIME}\) to \(\text{TIME}\), its domain is a subset of \(\text{TIME}\).

### 5.6. Properties of the Time Framework

An important requirement of our framework is that for any given time interval between successive transitions the system state should be unique: there should be no overlapping of time intervals. This is rigorously argued from expressions (RT1)-(RT7) in the predicate of RunTime and it relies on principles of set theory and in particular properties of finite sets of natural numbers (these being the \(\text{types}\) of the variables in the predicate). We first show that the function \(\text{transsuccessor}\) repeatedly applied to \((\text{online})\) generates the rest of the set \(\text{transitions}\) and the required result follows as a corollary.

Let \(\text{transitions}\) be represented by the ordered set \(\{t_0, t_1, \ldots, t_m\}\) where

\[
t_i < t_{i+1} \quad \text{for} \quad 0 \leq i \leq m - 1, \quad \text{and} \quad t_0 = \text{online}, \quad t_m = \text{switchoff}, \quad m = \#\text{transitions} - 1.
\]

**Lemma 1**

\[
\forall i : 0 \ldots m - 1 \bullet \text{transsuccessor}(t_i) = t_{i+1}
\]

**Proof**

The proof is by contradiction. We therefore assume that:

\[
\exists i : 0 \ldots m - 1 \bullet \text{transsuccessor}(t_i) \neq t_{i+1}
\]

and let \(j\) be the smallest such index for which the above holds.

Then for all \(i \geq j + 1\)

\[
\text{transsuccessor}(t_i) > t_{j+1}; \quad \text{[from (RT7)]}
\]

for \(i = j\)

\[
\text{transsuccessor}(t_j) \neq t_{j+1}; \quad \text{[Hypothesis]}
\]

for the remaining values of the index : \(i < j\)

\[
\text{transsuccessor}(t_i) < t_{j+1}; \quad \text{[Hypothesis]}
\]

Thus we have

\[
\forall i : 0 \ldots m - 1 \bullet t_{j+1} \neq \text{transsuccessor}(t_i)
\]

or

\[
t_{j+1} \notin \text{ran transsuccessor} \quad \text{[Definition]},
\]

which contradicts (RT5). Hence the proof follows.

The following are corollaries of Lemma 1.
(1) Transsuccessor is monotonic: if $t_1 < t_2$ then $\text{transsuccessor}(t_1) < \text{transsuccessor}(t_2)$.

(2) A bijection can be set up between integers $0, \ldots, m$ and the elements $\text{online}, \ldots, \text{transsuccessor}^m(\text{online})$ of the set $\text{transitions}$ and so $\text{transsuccessor}$ has an inverse. The existence of an inverse is necessary for the causal relationship between a pedestrian pressing the wait button and the illumination of the wait light (although not explicitly required for this paper).

(3) There is no overlapping of the intervals between transitions so that: $\exists t_1 : \mathbb{N} \cdot t_1 \in \text{transitions}$ and $\text{transsuccessor}(t) > t_1 > t$.

(4) If for $1 \leq n \leq m$, we define $i_n = (\text{transsuccessor}^{n-1}(\text{online}) \mapsto \text{transsuccessor}^n(\text{online}))$, then by (RT6) $\text{dom scenario} = \text{transsuccessor} = \{i_n\}$

5.7. The Current State

Using the definition of Corollary (4), there is no overlapping of the $i_n$ so that each time value $\text{now}$ between $\text{online}$ and $\text{switchoff}$ is associated with a unique interval of the set $\{i_n\}$, and hence with a unique state, $\text{currstate}$, of $\text{SIGNALSTATE}$. We can therefore construct a function associating a state with every time instant between $\text{online}$ and $\text{switchoff}$:

$\text{whichstate} : \text{TIME} \mapsto \text{SIGNALSTATE}$

with values defined by means of the schema $\text{WhichState}$

<table>
<thead>
<tr>
<th>RunTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{currstate} : \text{SIGNALSTATE}$</td>
</tr>
<tr>
<td>$\text{now} : \text{TIME}$</td>
</tr>
<tr>
<td>$\text{whichstate} : \text{TIME} \mapsto \text{SIGNALSTATE}$</td>
</tr>
</tbody>
</table>

$\exists t_1, t_2 : \text{TIME} \cdot$

$\text{online} \leq \text{now} < \text{switchoff}$

$\text{whichstate}(\text{now}) = \text{currstate}$

This schema was animated in order to make the time framework more meaningful, and this is explained further in a later section and in the Appendix.

5.8. Conditions for Runtime Operation

The next two schema give two conditions which are necessary for the runtime operation of the equipment. They are that state durations are fixed for a particular site, and that the red vehicle light monitor must indicate that it is working. Each state from the table is given a duration appropriate to the site, and this relationship is modelled by the function:

$\text{statedur} : \text{SIGNALSTATE} \mapsto \text{TIME}$

However this stricture may not apply to the last state of runtime operation, which could well be truncated after the discovery of some fault. The constraint on the duration of states during runtime operation is modelled by the schema $\text{FixedDur}$.
The specified operation of the equipment depends on the functioning of the red vehicle lights, for at least one red vehicle lamp must be working for each approach. The equipment is monitored to ensure this is so, and RedOk captures the indication of the monitor: it thus represents a refinement of the requirements regarding the red vehicle light. The given set

\[
\text{BULB} ::= \text{working} \mid \text{blown}
\]
distinguishes the case where there is at least one red vehicle lamp operational, \text{working}, from the case where all are non-operational, \text{blown}. At any instant of the Pelican runtime, the bulb(s) are either \text{working} or \text{blown}, and this is modelled by the function:

\[
\text{redmonitor} : \text{TIME} \rightarrow \text{BULB}.
\]
The domain of \text{redmonitor} consists of all the instants between \text{online} and \text{switchoff}, and for all instants with the possible exception of \text{switchoff}, the monitor must indicate \text{working}. The exception holds because at that instant, the monitor may indicate that the bulbs have \text{blown}.

\[
\forall t : \{\text{online} \ldots \text{switchoff} - 1\} \bullet \text{redmonitor} (t) = \text{working}
\]

5.9. Operational Sequence

The working of the equipment is closely associated with the operation of the wait light, which registers either no request, or that a pedestrian must wait.

\[
\text{REQUEST} ::= \text{pwait} \mid \text{norequest}
\]
The condition of the wait light is modelled by a function:

\[
\text{waitlight} : \text{(TIME} \times \text{TIME}) \rightarrow \text{REQUEST}
\]
which tells us in which time intervals the wait light registers a request, and in which there is no request. The instants of time when the pedestrian presses the button are captured by a finite subset of time \text{button}:

\[
\text{button} : \text{IF TIME}.
\]

The relationship between the operation of wait button and the illumination of the wait light was represented in the complete specification[16] but is not replicated here. For simplicity we consider it the case that the wait light is associated with the whole of a period, if it is on for part of it. In addition, a pedestrian demand is induced artificially as the equipment is turned on and \text{Start} captures this first state of runtime operation.
Start

---

**RunTime**

**Table 1**

- \( \text{waitlight} : (TIME \times TIME) \rightarrow REQUEST \)
- \( t_0 : TIME \)

---

\[
\begin{align*}
\text{dom waitlight} &= \text{transsuccessor} \\
\text{transsuccessor}(\text{online}) &= t_0 \\
\text{scenario}(\text{online} \mapsto t_0) &= s_6 \\
\text{waitlight}(\text{online} \mapsto t_0) &= \text{pwait}
\end{align*}
\]

The schema denoted TState1 shows possible transitions from the green vehicle period (period 1), the next state depending on whether or not the wait light is on or off. If it is on, the next state is \( s_2 \), and the wait light remains on through periods 2 and 3 until pedestrians cross in period 4. Otherwise the next state is \( s_1 \), and there is no change in state; the transition is then notional, denoting the fact that the specified state duration has lapsed. This feature is known as stuttering\( [81] \) where the change is to a hidden, internal state. In the case of the Pelican the internal change could be the re-starting of the clock registering the elapsed time for \( s_1 \).

---

**TState 1**

**RunTime**

**Table 1**

- \( \text{scenario} : (TIME \times TIME) \rightarrow \text{SIGNALSTATE} \)
- \( \text{waitlight} : (TIME \times TIME) \rightarrow REQUEST \)

---

\[
\begin{align*}
\text{dom waitlight} &= \text{transsuccessor} \\
\forall t_1, t_2, t_3 : & \text{transitions } \Rightarrow \text{transsuccessor}(t_1) = t_2 \\
\text{transsuccessor}(t_2) &= t_3 \\
\text{scenario}(t_1 \mapsto t_2) &= s_1 \\
( \text{waitlight}(t_1 \mapsto t_2) = \text{pwait} \Rightarrow \\
\text{scenario}(t_2 \mapsto t_3) &= s_2 \land \text{waitlight}(t_2 \mapsto t_3) = \text{pwait} \\
( \text{waitlight}(t_1 \mapsto t_2) = \text{norequest} \Rightarrow \\
\text{scenario}(t_2 \mapsto t_3) &= s_1 )
\end{align*}
\]

Plainly, the transitions from the other states of the table can similarly be represented in Z. Let them be denoted TState2, TState3, TState4, TState5, TState6, TState7, where TState7 captures the transition from \( s_7 \) to \( s_1 \) and for \( 2 \leq n \leq 6 \), TState\( n \) captures the transition from \( s_n \) to \( s_{n+1} \). The wait light can indicate \text{norequest} or \text{pwait} in periods 1, 5, 6, 7; in periods 2 and 3 it must indicate \text{pwait} and in period 4 it must indicate \text{norequest}. However if it indicates \text{pwait} in periods 1, 5, 6, 7 it must indicate \text{pwait} in the next period. For example the quantified expressions (TS4)-(TS6) in TState 1 have the following equivalent in TState 7:

\[
\begin{align*}
\text{scenario}(t_1 \mapsto t_2) &= s_7 \Rightarrow \text{scenario}(t_2 \mapsto t_3) = s_1 \\
\text{waitlight}(t_1 \mapsto t_2) &= \text{pwait} \Rightarrow \text{waitlight}(t_2 \mapsto t_3) = \text{pwait}
\end{align*}
\]

We can combine the constraints for runtime operation with the operational requirements and capture the runtime operation of Pelican equipment by a schema denoted RunTimeOp:

\[
\text{RunTimeOp} \equiv \text{FixedDur} \land \text{RedOk} \land (\text{Start} \land \text{TState1} \land \text{TState2} \land \text{TState3} \land \text{TState4} \land \text{TState5} \land \text{TState6} \land \text{TState7})
\]

There are further conditions for runtime operation which are not represented formally here; for example the DTp require provisions for a power cut and for the overriding of the Pelican operation from a remote control site.
5.10. Initial and Powering Down Modes

We have covered the period of time (ie online ≤ t < switchoff) during which the Pelican is operating in runtime mode, and this can plainly be extended to cover the time intervals switchon ≤ t < online, when the equipment is powering up and in state s8, and switchoff ≤ t < reset when the equipment is being run down and disabled after failure, or for some other reason. For switchoff ≤ t < offline, it may be in an error state. The DTp specifies this period as being within 0.5 second (5 of our units). From offline ≤ t < reset, it is disabled and also in s8. (The attributes of s8 are defined in InitState.)

The next section formalises the safety requirements for the Pelican; these are distinct from the functional requirements, and are much simpler, as will be seen. We subsequently explain how the above functional requirements fit into the Pelican safety framework.

6. Formal Expression of Safety Requirements

6.1. Safety Areas in DRIVE

The work undertaken by DRIVE Safely has been outlined in a previous section and culminated in a consultative document: Towards a European Standard: The Development of Safe Road Transport Informatic Systems (Draft 2) which provides guidelines for the engineering of RTI systems. Thus the successful performance of the end-product depends on adequate Hazard Analysis and the correct identification and documentation of any associated safety requirements. Subsequent implementation of the system must take account of these safety requirements; they add an extra dimension to system development. This philosophy has been encapsulated in the safety life-cycle model of the IEC[13] and forms the basis of development and evaluation criteria proposed by DRIVE Safely. Its scope includes all the stages in the equipments lifetime, from initiation to disposal. Hazard Analysis identifies the hazards associated with the system, and subsequent Risk Assessment attaches a risk factor to each hazard. Specification of each safety related (sub) system must include both functional and safety aspects. These deal with the functions that the sub-system has to perform, and detail how the sub-system will be made safe.

In order for a safety assessment to be made, the boundaries of the system appropriate for safety analysis must first be identified. This system may include a vehicle plus driver or comprise (for example) a set of traffic lights plus interacting vehicles, pedestrians, cyclists etc. A safety analysis is then undertaken to discover both the hazards associated with the system and the criticality and likelihood of an accident, (risk) associated with each hazard. During the CEC DRIVE I programme a group of projects who formed a "Safety Task Force"[3] identified three basic areas of road traffic safety and the first, system safety, involves the safe functioning of equipment under all conditions. An example might be the safe functioning of a vehicle braking system under all road and vehicle conditions. The response times associated with this safety area are usually less than one second. The second safety area usually involves response times of several seconds: human-machine interaction safety (HMI) is associated with the reaction of a human individual to the information displayed by the equipment and their responses to the equipment. An example might be a route guidance system which must convey information to a driver without creating too much of a distraction from the primary task of driving: its interface with the driver should be carefully considered for these safety implications. Traffic safety is associated with the whole road traffic infrastructure (e.g. road layout) and it can often take a very long time to demonstrate the effectiveness, or otherwise, of a new feature. An example of this might be a ramp metering system on a motorway. However it is not always possible to categorise the safety aspects of any proposed system under a single heading. For example there are road traffic implications in guiding a driver along certain routes.

These three areas of traffic safety must be considered during all phases of the system life-cycle of road transport equipment, and it can be seen that this entails expertise beyond that usually associated with systems engineering (in requiring behavioural responses to traffic situations, for example). From these considerations are derived the possible hazards and safety criteria of the equipment which contains the embedded software. The DRIVE II project PASSPORT is further
refining these proposals by developing a new methodology which provides a structured approach to preliminary hazard identification[82, 83].

6.2. Controllability Categories
After a list of hazards associated with the system has been established, a risk must be assigned to each hazard, and the notion of controllability category is the mechanism for this[3]. Five levels of controllability have been identified and the highest level is one where the effects of a hazard are not normally controllable by the road user and where the hazard is likely to result in a serious accident (usually including fatalities): it is uncontrollable. Examples include the jamming of the steering mechanism, complete brake failure, or traffic lights simultaneously showing green at two intersecting roads. The lowest level is one where the effects of the hazard are unlikely to affect safety, eg the failure of taped music. The highest confidence levels are required of the most safety-critical products; for example a system with a hazard which is normally uncontrollable should have the greatest care associated with its development. For this reason each of the five controllability categories is associated with an Integrity Level: the higher the controllability category, the higher the level of integrity of the equipment. In particular it is only by having a detailed knowledge of the development process that the Certification Authority can have assurance that the system fulfills its safety requirements at the required integrity level. This knowledge will be gained through the documentation provided. Documentation of sufficient quality and detail is seen as a benchmark against which system safety and quality can be measured. The underlying philosophy is that by using suitable methods for system development those hazards which have the most serious consequences as measured by controllability should have a low probability of occurrence ie should be extremely improbable. However we did not attempt to quantify the failure rate (probability) for any given controllability category; the research is not yet sufficiently well developed. Current reliability models are suitable for predicting only modest levels of reliability; they cannot be used to predict the high levels of reliability required for safety-critical software[84].

6.3. Fault Tolerance
The recommended method for designing safety into a system is the elimination of any fault which might result in a hazard, that is fault avoidance. However this council of perfection is not always realisable so the supplementary approach is to build fault tolerance into the equipment. This approach acknowledges the possible occurrence of a fault in a component of the system but builds in measures for detection of such a fault. After detection the equipment should safely recover from the fault; however full functionality may not be restored. A fail safe state is one which is guaranteed to satisfy the safety requirements but perhaps no others; for example all traffic lights are on red. For vehicles, no fail safe state may be available so it is desirable to design for stepped down performance to keep the vehicle operational, the so called “limp home strategy”[85]. An example might be the transfer to a conventional braking system after failure of the antilock braking system.

6.4. Safety Areas for the Pelican
The safety area we concentrated on was principally system safety, in the sense that it involved the controllability of an accident given various categories of equipment malfunction. To identify the Pelican safety requirements requirements, a safety analysis was performed on the system in order to discover possible hazards and their causes. The Pelican system safety analysis involved the principal hazard associated with pedestrian crossings, the situation where a pedestrian and vehicle attempt to negotiate the crossing simultaneously, possibly leading to an accident. Types of system malfunction which could lead to this hazard include direct conflict of lights and deviation of the equipment from the specification leading to confusion of individuals. Those cases where there was no malfunction but involved some element of HMI, were not examined.
6.5. Pelican Safety Analysis

Further analysis of the causes of the principal hazard was undertaken using a fault tree technique[86] as in Figures 2 and 3. The first fault tree (Figure 2) concerns the case when there is direct conflict of lights, for example the pedestrian sees *greenman* while vehicles see *green*.

**Figure 2: Fault Tree 1**

![Fault Tree 1 Diagram]

At this stage any attempt at development of the end nodes of either of the fault trees would anticipate design decisions; however the nodes should be expanded when these decisions have been made. Causes of signal conflict can include factors outside direct control considerations such as a bulb blowing or being damaged and there is a monitoring feature for vehicle red bulbs.

Fault tree 2 denotes the more ambiguous area of confusion of pedestrian or driver where the duration of a particular state is different from expected or the *sequence* of states is incorrect, period 1 followed by period 4 for example. The node labelled "Illegal combination" denotes the cases where the combination of light and vehicle colours is not a specified one. Thus the state is non-normative although there may be no direct conflict.

**Figure 3: Fault Tree 2**

![Fault Tree 2 Diagram]

The following two subsections formalise the safety requirements which correspond to Fault Trees 1 and 2.

6.6. Safety Requirements: Direct Conflict

The risk of an accident is high given direct conflict of lights: the effects of this hazard would normally not be controllable by the road user. The DTp specification[15] states that all
controllers shall be equipped with facilities to prohibit

(1) The display of a green signal for pedestrians with a green signal for vehicles.

(2) The display of a steady green signal for pedestrians without the display of a red signal to vehicles.

(3) The operation of the audible signals unless the steady red signal to vehicles (Period 4 of table 1) is displayed

"Interlock requirements" for (1) are suggested as follows:

" Alternatively 2 independent circuits shall be provided to monitor the signal leads. These monitor circuits shall act independently of each other and shall detect if any error voltage is present greater than 20% of the nominal voltage on the signal leads."

In order to implement (2) a red vehicle lamp monitor is required to operate during the red vehicle period but no corresponding requirements are indicated for the audible signal to implement (3), and no monitoring requirements for other lamps. It could be said that the monitoring of the audible signal is an implicit requirement of (3) but this is not explicit; possibly the audible signal was not part of the original Pelican equipment but was added on at a later date. If any of the above errors are detected, then the Pelican equipment is required to shut down, and return to the initial state of the system, when the lamps are all off, for recall that this initial state, $s_8$ is defined by the DTp as the fail safe state of the system.

The above three requirements were initially used as the basis for three specifications of the non-timing safety requirements of the controller, the three safety invariants (there were two in Z and one in OBJ). However after they were reviewed, an amended set was produced, taking into account the fact that ambiguities in the DTp document had led to differences in the three interpretations. We also decided that all three sets of safety invariants were incomplete as none had interpreted pedestrian green to include both green and flashing green.

Thus the amended version included two conjoined predicates:

\[
\begin{align*}
\text{plightshow}(safestate) &= \text{greenman} \Rightarrow \text{vlightshow}(safestate) \neq \text{green} \\
\text{plightshow}(safestate) &= \text{flashgreen} \Rightarrow \text{vlightshow}(safestate) \neq \text{green},
\end{align*}
\]

whereas the original had only included the first. This version presents the safety invariants for a single state of the system, safestate, and is denoted NoConflict.

\[
\begin{align*}
\text{NoConflict} \\
\text{vlightshow} : \text{SIGNALSTATE} &\rightarrow \text{VCOLOUR} \\
\text{plightshow} : \text{SIGNALSTATE} &\rightarrow \text{PCOLOUR} \\
\text{audio} : \text{SIGNALSTATE} &\rightarrow \text{BLEEPER} \\
\text{safestate} : \text{SIGNALSTATE}
\end{align*}
\]

\[
\begin{align*}
\text{safestate} \in \text{dom vlightshow} \\
\text{safestate} \in \text{dom plightshow} \\
\text{safestate} \in \text{dom audio} \\
\text{plightshow}(\text{safestate}) &= \text{greenman} \Rightarrow \text{vlightshow}(\text{safestate}) \neq \text{green} \\
\text{plightshow}(\text{safestate}) &= \text{flashgreen} \Rightarrow \text{vlightshow}(\text{safestate}) \neq \text{green} && (\text{NC1}) \\
\text{plightshow}(\text{safestate}) &= \text{greenman} \Rightarrow \text{vlightshow}(\text{safestate}) = \text{red} && (\text{NC2}) \\
\text{audio}(\text{safestate}) &= \text{bleep} \Rightarrow \text{vlightshow}(\text{safestate}) = \text{red} \\
&\quad \wedge \text{plightshow}(\text{safestate}) = \text{greenman} && (\text{NC3})
\end{align*}
\]

Note that all configurations which satisfy conjunct (NC3) also satisfy conjunct (NC1). However (eg) vehicle amber and pedestrian green satisfy (NC1) but not (NC3); (NC3) implies (NC1). Although (NC1) appears redundant it has not been omitted as it was an explicit and separate DTp requirement. In any case redundant safeguards are desirable in case of failure of one of them.
6.7. Safety Requirements: Confusion of Driver or Pedestrian

The fault tree of Figure 3 includes nodes identifying timing malfunctions such as either incorrect sequencing of states or incorrect state duration. The manner in which a timing malfunction might contribute to an accident is not so clear as in the case where signals directly conflict. To decide, for example, the controllability of an accident given an incorrect sequence of states requires a knowledge of human reaction to traffic situations. Such in-depth knowledge of human response to timing malfunctions was outside the scope of the project, and there were no requirements for monitoring for incorrect sequencing in the DTp document. Similarly there were no requirements for monitoring for incorrect combination of states, other than as outlined above.

The DTp document references a British Standard Specification for Road Traffic Signals[87] which mainly concerns constructural and optical requirements. There is also a requirement for testing timing durations, but no monitoring requirement for when the equipment is actually operating. However a simple requirement for state duration might be that each duration should have a minimum value. Thus the normative states have a minimum \( \text{mintime} \) time associated with them and this is captured by the following schema, TimeBounds.

\[
\text{TimeBounds} = \\
\{ \text{RunTime} \} \\
\quad \text{mintime} : \text{TIME} \\
\forall t_1, t_2 : \text{transitions} \\
\quad \text{transsuccessor}(t_1) = t_2 \Rightarrow (t_2 - t_1) \geq \text{mintime}
\]

7. Animation of the Pelican Specification

A method of animating Z via Prolog had been developed by West and Eaglestone[24]. The animation aided the safety validation of the Pelican and showed that the normative states are also \textit{safe}. The results of the animation have been supported by proof for these significant cases. Animation was used both to demonstrate the relative weakness of the safety requirements and (tutorially) to aid explanation of the time framework.

7.1. Relative Weakness of Safety Requirements

We present here the use of animation to aid the safety validation of the normative/non-normative states of the system. Thus a small model in Prolog demonstrates that the states specified by NormalStates satisfy the safety invariants and illustrates the fact that the safety invariants are weaker than the formally expressed functional requirements. In other words there are error states whose functional assignments differ from the normative states: \( \{ s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8 \} \) which nevertheless satisfy the formally expressed safety constraints defined in NoConflict. A small model of SIGNALSTATE contains both normative and non-normative states. The non-normative states \( \text{error}_1, \text{error}_2 \) have the following configurations associated with them:

\[
\text{vlightshow(error}_1) = \text{green}, \text{plightshow(error}_1) = \text{greenman}, \text{audio(error}_1) = \text{nobleep},
\]

\[
\text{vlightshow(error}_2) = \text{red}, \text{plightshow(error}_2) = \text{pl_off}, \text{audio(error}_2) = \text{nobleep}.
\]

The normative and non-normative states are suitably modelled in Prolog, using manufactured rules which simulate Z schema by capturing their mathematical structure. The Prolog includes a model of the schema NoConflict, and the strategy is to use the theorem proving mechanism of the logic program to show which of the states in the model satisfy the safety invariants. The code which models the NoConflict schema is presented in the Appendix, together with a fragment of the required data instantiations.
The functional assignments of states $s_1, \ldots, s_8$, as defined in $\text{NormalStates}$ satisfy the formally expressed safety constraints defined in $\text{NoConflict}$. This is shown by giving $\text{safestate}$ functional assignments equal to those of the normative states. Querying the Prolog (which models the Z logic) as to whether $s_1$ satisfies the safety constraints returns an affirmative answer, and this same affirmative answer is given when the other normative states are queried.

However $\text{error}_1$ does not satisfy the safety invariants, and this is verified when the Prolog is queried. Recall that the safety invariants are weaker than the functional specification, so that although $\text{error}_2$ is a non-normative state, it does satisfy the safety invariants. Thus $\text{NoConflict}$ is weaker than $\text{NormalStates}$.

7.2. Animation of the Time Framework

The schema $\text{WhichState}$ has been animated (see the Appendix). The animation was for tutorial purposes - to aid understanding of the time framework. Queries were put as to which state is in operation at a given time, and at which time or times a given state is in operation. The data used is the same as that contained in the example scenario, in a previous section. Specific times were chosen within the bounds $\text{online} (= 70)$ and $\text{switchoff} (= 430)$ as to which state was associated with that time, eg for $\text{now} = 70$, what is $\text{currstate}$? The Prolog gives the required state and illustrates the case that there is only one state associated with a time, but there are may possible times for certain states. Prolog for the schema is given in the Appendix together with some of the data instantiations and some sample queries.

8. Relationship of Safety to Functional Requirements

The safety of the Pelican has principally been related to the configurations of individual states (ie chiefly the first fault tree) and their compliance with schema $\text{NoConflict}$. A demonstration of the safety of specified configurations was presented in a previous section via animation. However the relationship between Z and Prolog is manufactured and not formal; it is necessary to supplement the validation with proof. Further, we must define reachability with respect to our formal time framework and show that the normative (safe) states are the only reachable states during all but a small period of the Pelican history. The required proofs will be outlined later in the section.

The safety conditions arising from the second fault tree have involved the duration of the signals and were modelled in schema $\text{TimeBounds}$. It can be seen that the schema $\text{FixedDur}$, which requires that each state involved in the runtime operation must have a given duration, also ensures that the duration must have a certain minimum value provided that the given durations are compatible with the minimum value. Thus $\text{RunTimeOp}$ (which depends on $\text{FixedDur}$) implies the safe duration of the runtime states with the possible exception of the last state which may have been truncated.

8.1. Reachability of the Normative States

During the interval, $\text{switchoff} \leq t < \text{offline}$, while powering down, the controller may be in an error state, or sequence of states. We shall now show that at all other periods in the equipment history, within the time intervals $\text{online} \leq t < \text{switchoff}$ and $\text{offline} \leq t < \text{reset}$ the equipment is in one of the states specified by schema $\text{NormalStates}$.

Period $\text{online} \leq t < \text{switchoff}$: The operation of Pelican equipment is as specified in $\text{RunTimeOp}$ and we need to show that the set

$$\text{tablestates} \equiv \{ s_1, s_2, s_3, s_4, s_5, s_6, s_7 \}$$

is closed with respect to this operation. We accordingly define the reachable states of the equipment for any given scenario to be the range of scenario. We now wish to show that a consequence of the schema $\text{RunTimeOp}$ is that the table states are the only reachable states in $\text{online} \leq t < \text{switchoff}$; thus for any given system behaviour, the range of scenario is a subset of tablestates. Noting that the system behaviour depends on the operational sequencing of the wait light, we present a proof sketch.
Lemma 2
\[ \text{ran scenario} \subseteq \text{tablestates}. \]

Proof (Sketch)
The proof is by induction over the ordered set of intervals \( \{i_n\} \), defined in Lemma 1, Corollary (4), where \( \{i_n\} = \text{dom scenario} \), for \( 1 \leq n \leq \#\text{transsuccessor} - 1 \). We shall accordingly assume that
\[ \text{scenario}(i_n) = \sigma, \sigma \in \text{tablestates} \]
and then identify the state occupying the next sub-interval \( (i_n + 1) \). From TState 1 [(TS5), (TS6)] if \( \sigma = s_1 \), then \( i_{n+1} \) is occupied by either \( s_2 \) or \( s_1 \), depending on the status of the wait light, for from (TS1) \( \{i_n\} = \text{dom waitlight} \).

In a similar manner it can be shown from the predicates of TState 2, TState 3, TState 4, TState 5, TState 6, TState 7 that if \( i_n \) is occupied by any of the other table states, then \( i_{n+1} \) is also occupied by a table state. Thus
\[ \text{scenario}(i_n) \in \text{tablestates} \Rightarrow \text{scenario}(i_{n+1}) \in \text{tablestates}. \]
The base case is that \( i_1 \) is occupied by \( s_6 \) (from Start). Thus \( \text{scenario}(i_n) \in \text{tablestates} \) for all \( i_n \in \text{dom scenario} \) and the proof follows.

8.2. Safety of Reachable States

In order to prove the system is functionally safe we need to prove the safety of the reachable states, both table and initial, \( s_8 \). It is necessary to prove that the functional assignments of \( vlightshow, plightshow \) and \( audio \) for each of the reachable (normative) states \( s_1, s_2, \ldots, s_8 \) are consistent with the predicate of \( \text{NoConflict} \). We observe that each of the normative states is a member of the domains of \( vlightshow, plightshow \) and \( audio \) so it is left to prove that the constraints (NC1) . . (NC4) are satisfied. Taking \( s_4 \) as an example the left hand side of implication (NC1) is \( \text{true} \) and the right hand also \( \text{true} \), so implication (NC1) is \( \text{true} \). In a similar manner (NC2), (NC3), (NC4) can be shown to be satisfied by \( s_4 \) and a proof constructed for the other normative states.

Note that we have not attempted to prove any safety property of the functional sequence of states, only that each of the functionally reachable states is safe in terms of configuration and duration.

9. Discussion

9.1. Assessment of the Z Specification

We have presented a formal specification in Z of the safety and functional requirements for "Pelican" equipment using a UK Department of Transport source document and some guidelines from a DRIVE Safely consultative document: Towards a European Standard [2, 3]. Some (but not all) necessary proof arguments have been presented to demonstrate the internal consistency and safety of the specification. The Pelican equipment is modelled in such a way that its states are related both to equipment attributes and to a formal time framework. The states of the Z specification are modelled as abstract objects, whose relationships to light and audible signals are represented as functions (\( vlightshow \) etc.) and both normative and non-normative states are included in the representation.

Time is modelled by the natural numbers and intervals constructed by means of a set of transition times. Further, a unique state is defined for each of these intervals. We have proved that the constructed intervals do not overlap, thus it is possible to define a unique state at any given time point. A function \( \text{whichstate} \) relates the current state of the system to a time variable \( now \), the value being obtained from the history function \( \text{scenario} \). As described, the notion of \( \text{interval} \) is not primitive, but constructed from time; it is essentially a pair of times denoting the beginning and end of the interval. This approach can be compared with the more complex one of Z interval
logic[47] for the relationship of state to time interval, *scenario*, is akin to the function *Itrace* of the Z interval logic. However *Itrace* depends on intervals which are primitive objects and further, these intervals are closed as they include both endpoints. Also *Itrace* captures the history (ie trace) of a single variable rather than a state.

**Safety and Liveness:** A comparison was made in a previous section of the first order approach to time with temporal logic. Ours is a first order approach and compares with Saeed[7] in the use of a history mechanism to relate states with time. Further, the set of possible histories satisfying the schema *RunTimeOp*, if regarded as a sequence of states, is akin to the set of program behaviours defined by (for example) Alpern and Schneider[51]. A separate safety specification is presented in Z which provides safety criteria for states of the system, and this can be compared with the notion of safety (invariance) in temporal logic. Similarly the notion of liveness (eventuality) can be compared with the concept of a behaviour history: equipment behaviour is either cyclical or repeatedly in Period 1, depending on pedestrian demand.

*Causality* is via events or time values, rather than via agents and actions as in *MAL*. For example there is a causal link between the times the button is pressed and, with certain provisos, the status of the wait light and it is this status which determines the identity of the next state after transition. This seems a natural means of expressing cause and effect in linking an action (pressing a button) to a result (subsequent permission to cross the road). However although the time value *switchoff* apparently determines the end of normal operational mode, the agent which causes this event could well be the failure of the red vehicle bulbs, and the subsequent change in status of *redmonitor*. Thus the causative link is not explicit, and blurs the distinction between an action and its timing.

We have also outlined a proof that the functional behaviour satisfies the safety conditions: the functionally reachable states are safe. The reachability criteria are similar to those described by Leveson[5] and the safe behaviour of the equipment can also be compared to the "safe histories" of Saeed[7]. However invariance is "over all intervals" and it is a state which is safe or unsafe, rather than any sequence of states. This assumption is adequate for the Pelican (and there is no indication in the DTp document as to what is a safe sequence, apart from the normal cycle) but could be considered inadequate for many applications. For physical systems the measure of change of a sequence of parameters is very significant and should be considered as a criterion for safe behaviour, as it is in the temporal logic definition.

Animation[24] has been used to demonstrate that the safety requirements are weaker than the functional requirements: all normative states are safe but there are some non-normative states which are also safe. It has been suggested by Leveson[11] and other authors that, while a verification of code implementation against the functional requirements may not be possible because of the relative complexity of the latter, a proof that it satisfies its safety conditions may be more feasible. Thus, if the safety requirements are formally expressed, there is the possibility of a proof that the final code implementation is safe. (This was not attempted for the Pelican, as we did not have access to the software.) In developing the formal specification we identified ambiguity and incompleteness in the DTp specification and these are outlined in the next subsection.

**9.2. Deficiencies in the DTp Specification**

The DTp document was evidently written with an electro-mechanical implementation in mind and during the time when the Pelican requirements were being formally specified there was much discussion as to its interpretation, and much difficulty in extracting its functional requirements[16]. It was not clear that the prohibition of *green* also included flashing green, and the exact constraint as to the sounding of the audible signal was also unclear. Also, there are many possible non-normative configurations of lights and audible signals. Some of the configurations are prohibited by the safety conditions but (as demonstrated by the Prolog animation) there are many that are not. These many configurations were not considered in the DTp specification. Another deficiency is the lack of a definition of a safe sequence of states; if the *only* safe sequence is the normal operational one, then this should be stated and safeguarded.
against error. These difficulties underlined the importance of documentation of such requirements, and that modern document writing guidelines[44, 45] should be followed. For safety-critical systems a formalisation of the safety requirements should be attempted in as full a manner as possible. Where the formalised safety requirements are used as specifications for safety monitors or other redundant sub-systems, it is necessary that the safety constraints should be unambiguous. This then facilitates the design of the appropriate sub-systems. The requirement regarding the restrictions of the audible signal to period 4 is arguably the most important of the three and a safeguard which prohibits it should be clear and explicit. There is potentially great danger to blind or partially-sighted persons if it were to sound in the wrong period, and the logic of this requirement indicates that its implementation requires the detection of an audible signal error[21].

The above recommendations were reported to the DTp[16] and we were later informed that a new specification had been initiated. Also, we have heard that steps were taken to make safe the Pelican audible signal on site where necessary.

9.3. Formal Methods Technology

Z is a mature method, well-understood by a significant Industrial Community and possessing tool support. We have illustrated the use of animation in our specification, supplemented by the use of proof: proof-support (eg in HOL) is available for Z as well as automated type checkers. However in comparing the treatment of some properties of safety-critical system specifications with our Z specification, it is apparent that other formalisms have some superior features to our own. Although our time framework is adequate for the case study it would be too rigid for some complex systems. RTI systems where measurement of continuous variables is significant (such as a vehicle ignition system) would require an approach where time is captured by the real numbers rather than the naturals, such as TLA. An interval logic such as ITL[56] or duration calculus[9] is more expressive in its treatment of change, while MAL appears to capture causal aspects, and furthermore is integrated into a systematic method of capturing requirements[61] (ie FOREST). The SCA formalism[46], viz PR(A), is radically different from the others, being chiefly algebraic. It is capable of modelling many different architectures and is simple to use; the ability to support local clocks (re-timing) is practical. However we are not aware of any safety-critical case-studies.

While the methods described above are more expressive, they have been used predominantly in research environments and do not (yet) possess the tools that would fit them for use in an industrial environment. An exception is the interval model of time in Z[47] which is apparently capable of capturing requirements for all types of asynchronous distributed system and this may remedy the situation. However the question remains as to its capability of capturing safety invariants: this would presumably mean the expression of an invariant relation over traces of several different variables. Extensions of the Z notation[73, 68] and the use of Z with other methods such as CSP[71] are also more expressive than standard Z.

In a study of formal methods, safety-critical systems and standards by Bowen and Stavridou[23], the transition from research to methods and tools that are "fit for purpose" for the needs of industry is termed formal methods technology. They stress the fact that procedures such as proof must be enabled by suitable technology before they are appropriate for industrial use, and it is the users of such methods who should drive the technology. In their study these authors suggest that mature formal methods (despite some problems) can and should be used to produce safer software. At least half the software safety standards they reviewed reflect the importance and maturity of formal methods.

10. Summary and Conclusions

We have overviewed the work of the EC funded DRIVE Safely project, and some of their recommendations for development and production of safe RTI systems. A selection of the methods currently identified as suitable for formal specification of safety-critical systems were
reviewed, including first order, temporal and modal logics and SCA’s. Some examples of the use of Z for timed systems were also briefly described.

A formal specification in Z of the functional requirements for "Pelican" equipment was given with respect to a time framework which was also in Z. Three areas of traffic safety identified within DRIVE were also described, and the formalised safety requirements given in Z for the Pelican. We were subsequently able to show via animation and proof that apart from a short period after malfunction, when it may be in an unknown, unsafe state, the equipment states satisfy the safety invariants at all stages of equipment life. This has implications for system implementation in that it may be feasible to show that the completed system satisfies the safety requirements where it is not possible to show it satisfies the functional requirements. In the process of the task of developing the formal specification we discovered deficiencies in the DTp source document which were reported; the deficiencies have (we have heard) been rectified by the DTp.

The above underlined the need for adequate documentation and the necessity for formalising requirements (particularly involving safety). However this still leaves the problem that methods such as Z which are technologically mature with some tool support are deficient in the desirable features of some of the currently available research while the newer methods still lack technical support and maturity.

11. Acknowledgements

This work was partly supported by the CEC funded DRIVE programme.

The authors would like to thank Denton Clutterbuck and William Marsh of PVL (Southampton), John Murdoch, Pat Hill and many other colleagues of Leeds University for a number of suggestions and comments during the development of this work.
References

33

(1986).


Appendix: Animation of Z Specification, in Prolog

The translation is based on some manufactured rules, giving a mapping between a subset of Z and Prolog: the code for the schema depends on a Prolog library of set theory code, which is given in the appropriate reference. Our small model of SIGNALSTATE contains both normative and non-normative states:

$$\{ s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, error_1, error_2 \},$$

and each element is provided with a functional value in $v_{lightshow}$, $p_{lightshow}$ and $audio$. We also present a small example of a system history:

$$scenario = \{ (70,120) \mapsto s_6, (120,140) \mapsto s_7, (140,340) \mapsto s_1, (340,370) \mapsto s_2, (370,390) \mapsto s_3, (390,420) \mapsto s_4, (420,430) \mapsto s_5 \}.$$

where the integers represent tenths of a second. It is possible in this animation model to give several instantiations for each identifier, and a query returns a suitable combination.

A1: Animation of NoConflict Schema

The following is the code for NoConflict, and a fragment of the data instantiations required.

```
schema_type([Vlight,Plight,Audio,Safestate], noconflict) :-
  /*signature*/
  givenset(S,signalstate), givenset(V,vcolour),
  givenset(P,pcolour), givenset(Bl,bleeper),
  varname(Vlight, vlight), partial_function(Vlight,S,V),
  varname(Plight, plight), partial_function(Plight,S,P),
  varname(Audio, audio), total_function(Audio,S,Bl),
  varname(Safestate, safestate), member(Safestate, S),

  /*predicate*/
  dom(Safestate, Vlight), dom(Safestate, Plight), dom(Safestate, Audio),
  /*statements involving implication have been replaced by their logical equivalents*/
  /* involving negation and conjunction*/
  \+ (member([Safestate,greenman],Plight), member([Safestate,green],Vlight)),
  \+ (member([Safestate,flashgreen],Plight), member([State,green],Vlight)),
  \+ (member([Safestate,greenman],Plight), \+ member([Safestate,red],Vlight)),
  \+ (member([Safestate,bleep],Audio),
  \+ (member([Safestate,greenman],Plight), member([Safestate,red],Vlight))).

/* data for given sets */
givenset([s1,s2,s3,s4,s5,s6,s7,s8, error1, error2], signalstate).
givenset([green,red,amber,flashamb, offv], vcolour).
givenset([greenman,redman,flashgreen, offp],pcolour).
givenset([bleep,nobleep],bleeper).

/* data for type individuals */
varname(s1,safestate).
varname(s8,safestate).
varname(error1,safestate).
varname(error2,safestate).

varname([ [s1, green], [s2, amber], [s3, red], [s4, red], [s5, red], [s6, flashamb],
  [s7, flashamb], [s8, offv],[error1, green ],[error2,red ]], vlight ).
varname([ [s1, redman], [s2, redman], [s3, redman], [s4, greenman], [s5, flashgreen],
  [s6, flashgreen], [s7, redman], [s8, offp], [error1, greenman], [error2, offp ]],plight).
```
A2: Animation of WhichState Schema

schema_type([Scenario, Now, Online, Switchoff, State, Transitions, W, Transsuccessor], whichstate) :-

schema_type([Online, Switchoff, Transitions, Transsuccessor, Scenario], runtime),
givenset(Time, time), givenset(S, signalstate),

varname(W, ws), partial_function(W, Time, S),
varname(Now, now), member(Now, Time),
varname(Switchoff, switchoff), varname(Online, online),
varname(State, currstate), member(State, S),

Online =< Now, Switchoff > Now,
member(Now, State, W),
dom([T1, T2], Scenario), T1 =< Now, Now < T2,
member([[T1, T2], State], Scenario).

schema_type([Online, Switchoff, Transitions, Transsuccessor, Scenario], runtime) :-
givenset(Time, time), givenset(S, signalstate),

varname(Switchoff, switchoff), member(Switchoff, Time),
varname(Online, online), member(Online, Time),
varname(Transitions, transitions), subset(Transitions, Time),
varname(Transsuccessor, transsuccessor),
    partial_function1(Transsuccessor, Time, Time),
varname(Scenario, scenario), partial_function2(Scenario, Time, Time, S),

member(Switchoff, Transitions), member(Online, Transitions),
        Online < Switchoff, 

setof( K1, dom( K1, Transsuccessor ), Domtrans ),
setof( K2, ran( K2, Transsuccessor ), Rantrans ),

cHECK_ALL_INTIME(Transitions, Switchoff, Online),

dELETE(Switchoff, Transitions, Domtrans),
delete(Online, Transitions, Rantrans),

setof( K3, dom( K3, Scenario ), Domscen ),
set_equal( Domscen, Transsuccessor),
check_all_indomtrans(Domtrans, Transsuccessor).

varname( [[70,120],s6], [[120,140],s7],[[140,340],s1], [[340,370],s2],
        [[370,390],s3], [[390,420],s4], [[420,430],s5] ], scenario ).
varname(70, online).
varname(430, switchoff ).
The queries shown demonstrate the fact that there is only one state associated with any time between $t \geq 70$ and $t < 430$ and there is no state defined for $t \geq 430$. However there may be many times associated with a given state.

?- schema_type([_,70,_,_,S,_,_,_], whichstate).
S = s6 ?
no

schema_type([_,430,_,_,S,_,_,_], whichstate).
no

schema_type([_,Now,_,_,s6,_,_,_], whichstate).

Now = 70 ?

Now = 80 ?
yes