Verification, Analysis and Synthesis of Safety Interlocks

Jonathan Jacky

Radiation Oncology Department RC-08
University of Washington
Seattle, WA 98195

jon@radonc.washington.edu

Technical Report 91-04-01

April, 1991

Abstract

Safety interlocks are mechanisms that prevent hazards. Complex interlocks often include programmable elements. Most safety interlock systems are designed using informal, intuitive methods and are verified by inspection. This paper describes some techniques that can be applied in a systematic method for developing safety interlock systems from formal statements of functional and safety requirements. Requirements expressed in English are represented as formulas in propositional logic that are related to liveness and safety properties. The techniques can be applied in any implementation notation; this paper uses the notation most commonly used for both nonprogrammable and programmable interlock systems, which is called the *ladder diagram*. One application is demonstrated: Proposed or existing implementations can be verified by comparing them to the requirements which the system's users believe to be appropriate. Approaches to two other applications are sketched: Existing implementations may be analyzed or "inverted" to reveal the requirements that they actually satisfy. Implementations may be synthesized from requirements. These techniques are complementary to standard digital design methods that emphasize minimization and normalization of given boolean formulas. The techniques are illustrated with an example drawn from a medical cyclotron used for radiation therapy.

1 Introduction

Safety interlocks are mechanisms that prevent hazards. For example, a radiation therapy treatment room is usually equipped with an interlock that prevents the treatment beam from being turned on when the door is open, in order to protect staff and visitors from scattered radiation. Interlocks in large process control systems can be far more complex than this example, involving numerous sensors and programmable elements.

Typical practice, recommended in guidelines provided to system designers [5], often delegates safety interlock functions to separate *protection systems* which are kept distinct from the subsystems that perform more complex control functions. This separation is supposed to simplify the design of the protection systems, make their verification feasible, and help ensure that essential safety-related functions continue to operate properly even if more complex control functions fail.

Most protection systems are designed using informal, intuitive methods and are verified by inspection. It is not unusual for them to be modified in the field. Usually the only documentation produced is that needed for implementation: circuit diagrams for nonprogrammable units, and source code for programmable devices. These practices are not consistent with modern recommendations for the development of safety-critical systems [9, 10, 11], which emphasize the need for an explicit statement of functional and safety requirements, and a development process that can be shown to produce an implementation that meets the requirements.

The limitations of these practices became apparent to us when we had to modify a complex interlock system created years earlier by others. This need often arises when control system functions must be modified, and the safety implications of the changes considered, or when the allocation of interlock functions among hardware subsystems must be changed. If the system incorporates many inputs and outputs, and the design has been optimized to minimize the number of elements used, the intended functions can be far from self-evident. The number of elements can make it impractical to perform exhaustive manual analysis.

This paper describes some techniques that can be applied in a systematic method for developing safety interlocks. Requirements expressed in English are represented as formulas in propositional logic that are related to liveness and safety properties. The method can be applied in any implementation notation; this paper uses the notation most commonly used for both nonprogrammable and programmable interlock systems, which is called the *ladder diagram*. One application is demonstrated: Proposed or existing implementations can be verified by comparing them to the requirements which the system's users believe to be appropriate. Approaches to two other applications are sketched: Existing implementations may be analyzed or "inverted" to reveal the requirements that they actually satisfy. Implementations may be synthesized from requirements. These techniques are complementary to standard digital design methods that emphasize minimization and normalization of given boolean formulas.

2 The technologies

The technologies and notations used for protection systems are unfamiliar to many computer scientists, so they are briefly reviewed in this section.

It is usual to implement complex control functions and protection functions using different technologies. A typical division of labor delegates control functions to general-purpose mini- or microcomputers programmed in high level languages, while the most critical protection functions are provided by nonprogrammable hardware. Protection functions of moderate complexity are implemented on small processors programmed in a special-purpose notation (of which more later).

The Clinical Neutron Therapy System at the University of Washington in Seattle is an example of this approach. It is a cyclotron and neutron therapy treatment facility that provides particle beams for cancer treatments with fast neutrons, production of medical isotopes, and experiments in medical physics and radiobiology [13, 6]. The operators' consoles and analog controls are largely managed by general-purpose mini- and microcomputers. The protection systems for the potentially hazardous operations of turning on the beam and moving the gantry and other massive components are implemented in relays. The on-off controls for the vacuum and cooling systems and many beamline elements, which include many safety and equipment protection functions, are provided by a special-purpose *programmable logic controller* (PLC).

2.1 Relays

Electromechanical relays are still the preferred technology for implementing critical protection functions in some process control applications, including accelerators [4, 6] This is due to their long history, ruggedness, and especially to the high confidence placed in the simple and effective testing methods that can be applied to these devices [4, 12].

Relay interlock circuits are not necessarily small. At the Seattle cyclotron facility, the circuits that enable the beam to turn on and motions to proceed include about 30 and 70 relays, respectively.

2.2 Programmable Logic Controllers (PLC's)

PLC's are special-purpose computers intended for applications of moderate complexity that are dominated by on-off controls. They are very popular in industrial process control systems [14, 17] and are also used in accelerators [1, 6]. PLC's are usually based on general purpose microcomputers, but are packaged in rugged enclosures suitable for industrial environments and include signal conditioning and isolation so they can be connected directly to the controlled equipment. They usually include a programming environment for a special purpose language.

At the Seattle cyclotron facility, the PLC that controls the vacuum and cooling systems and many beamline elements includes over 150 inputs and 150 outputs.

2.3 Ladder diagrams

Relay interlock circuits and PLC programs are usually expressed in the same notation, called a *ladder diagram*. Ladder diagrams were derived from relay wiring diagrams, in order to make it easy for designers exprienced in the older technology to apply computers. As a result of this history, ladder diagrams are very popular in industry but little known among computer scientists.

Ladder diagrams are simply a way of encoding boolean logic expressions. They are an alternative to the gate representations used by electronics engineers. However, the normal form of a logical expression which is most convenient for translation into a ladder diagram may be different than the form that is most convenient for other digital technologies.

Figure 1 illustrates a simple ladder diagram with a single *rung*, that computes the propositional formula $D \Leftrightarrow (A \vee B) \wedge \neg C$ (where $\Leftrightarrow$ indicates double implication, "if and only if" (`iff`), $\wedge$ is conjunction (`and`), $\vee$ is disjunction (`or`), and $\neg$ is negation (`not`). The left and right *rails* represents a current source and sink, respectively. Elements A and B are *normally-open contacts* representing switches or relay contactors that admit the passage of current when asserted, and C is a *normally-closed contact* that prevents the passage of current when asserted. Element D is a *coil* which is energized when the other elements admit the passage of current between the rails. Coils are understood to represent outputs and diagrams are conventionally drawn with a single coil as the last element on the right in each rung.

Despite their lack of academic attention, ladder diagrams have features which many computer scientists admire. They are a visual, declarative, logic programming language, supported by graphic programming environnments which resemble CAD systems.

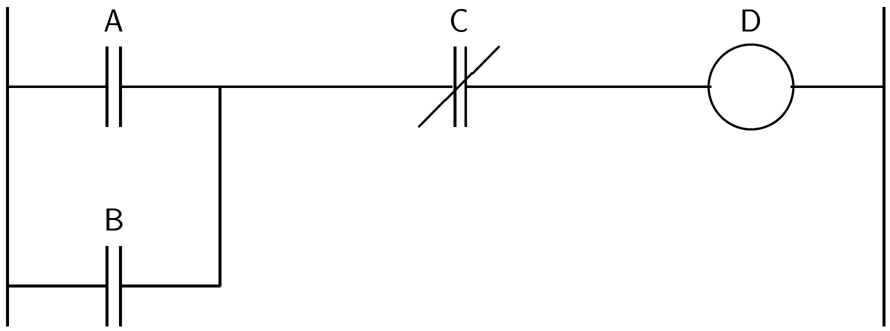

Figure 1: Ladder diagram that computes $D \Leftrightarrow (A \vee B) \wedge \neg C$.

However, ladder programming methods are primitive. Most tutorials [8, 14, 17] recommend intuitive methods and provide little guidance regarding verification. Recognition that standard digital design techniques can be applied to ladder programming is rare [16]. In particular, the concept that implementations might be derived from, or compared to, a formal representation of requirements is absent from this literature. The ladder diagrams are customarily the only detailed documentation provided for an interlock system.

The innovation proposed here is to consider the ladder diagrams to be an implementation of requirements expressed as formulas in propositional logic.

3 A simple example

Figure 2 illustrates a small example, based on features of our radiation treatment facility [6]. The cyclotron is isolated in a vault. When radio-frequency drive electronics RF are activated, the cyclotron accelerates particles down the beam line, which contains a moveable barrier. The barrier makes it possible to turn on the beam in two modes, which are essential for efficient use of the facility. When the barrier is closed, the cyclotron operator can adjust or "tune" the beam (using complex analog controls not discussed here), which is stopped in the barrier while therapy technologists are preparing the patient in the adjacent treatment room. After the technologists leave the treatment room and close the door, the barrier is opened and the particle beam continues into the treatment room, where it strikes the target and creates an external radiation beam that irradiates the patient.

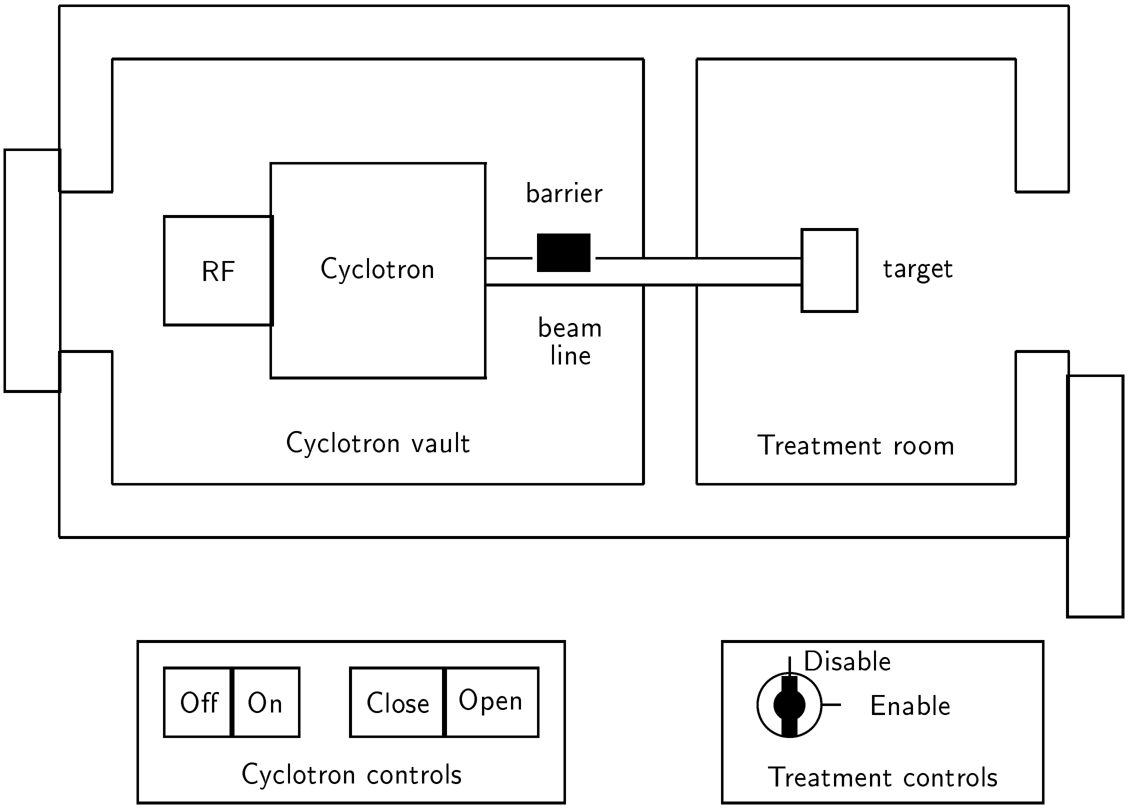

Figure 2: Cyclotron, treatment room, and control panels

Subsystem	*Outputs*	*Control inputs*	*Sensor inputs*
Cyclotron RF drive	RF	ON OFF	
Beam line barrier	BARRIER	OPEN CLOSE	OPENED CLOSED
Cyclotron vault door			VAULT
Treatment keyswitch		DISABLE ENABLE	
Treatment room door			TREAT

Table 1: Interlock system inputs and outputs

The cyclotron operator's controls include ON/OFF, a two-position toggle that energizes the drive electronics, and OPEN/CLOSE, a two-position toggle that controls the barrier. These controls are connected to four separate inputs, one for each position of each switch. It is good practice to provide a separate input for positive indication of each position, rather than depending on a single input for both (which requires assuming, for example, that the barrier is supposed to be CLOSED if OPEN is not asserted). Each of pair of inputs affects one output, which is asserted to turn on the RF or open the barrier, respectively. In this simplified example, the absence of an active output signal is understood to result in the affected component dropping into its safe position: the RF turns off and the barrier closes.

Additional inputs are also included, because the potentially unsafe actions of turning on the drive or opening the barrier must sometimes be prevented. Therapy technologists control a DISABLE/ENABLE keyswitch which must be enabled to allow the beam to reach the treatment room. Other inputs come from sensors rather than operators' control panels. Separate sensors determine whether the barrier is OPENED or CLOSED. Two sensors are needed for a positive indication because the barrier can be in an intermediate position. The shielding doors to the vault and treatment room are opened and closed under manual control, but there are VAULT and TREAT door sensors which are asserted when the doors are fully closed. Only one sensor is needed for each door because they are considered to be open whenever they are not completely closed.

The complete set of inputs and outputs is summarized in Table 1. It is useful to distinguish *control inputs* which are directly manipulated by the operators and *sensor inputs* which detect internal conditions that may not be directly controlled or observed by the operator.

Fig. 3 shows a ladder diagram intended to meet the requirements in this example, as might be designed by intuitive methods. The first rung calculates an intermediate result, SAFE (representation of intermediate results as outputs is a common ladder programming idiom). SAFE indicates that the treatment room door is closed and the key is in the ENABLE position. If both ENABLE and DISABLE are somehow asserted, SAFE will not be asserted. The second rung causes the barrier to open if the OPEN toggle is asserted, the CLOSE toggle is not asserted, and SAFE is asserted. This conservative approach prevents the barrier from opening when the treatment room is unsafe, whether or not the beam is on. The third rung turns on the RF drive when ON is asserted, OFF is not asserted, the vault door is closed, and *either* the barrier is not OPENED *or* the treatment room is SAFE. This rung contains an error, as will be explained subsequently.

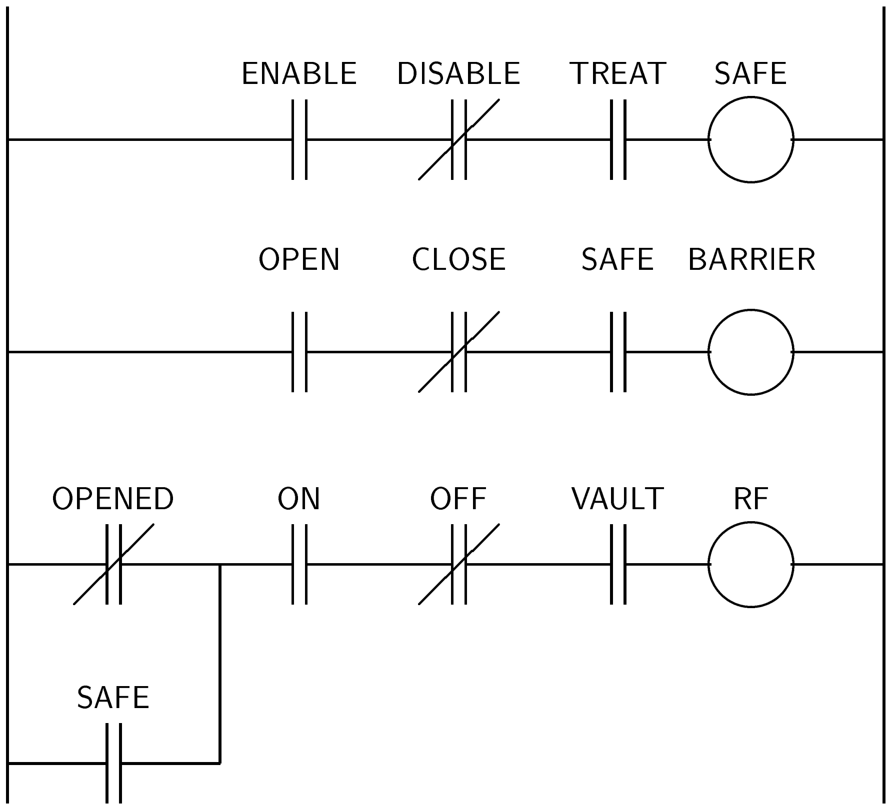

Figure 3: Erroneous ladder program for cyclotron and treatment room interlocks

This solution was obtained intuitively and verified by inspection. The following sections describe how the design and verification process can be formalized.

4 Formalization

An interlock system can be represented by a set of inputs and outputs, where each element in both sets may assume the values *true* and *false*. Such a system may potentially assume many different states, one state for each possible combination of assigments of *true* and *false* to all the inputs and outputs. The purpose of an interlock system is to prevent the occurence of states which are considered to be hazardous. This is accomplished by building the system such that all the outputs are determined by appropriately chosen formulas involving the inputs.

This paper limits consideration to interlock systems that can be expressed in *combinational* logic: their outputs are solely a function of their inputs at the same moment, without regard for the past history of inputs. Consequently, they can be represented by propositional logic formulas where outputs are not fed back to behave as inputs. Almost all interlocks in our cyclotron facility [6] can be represented in this way, even when they are intended to protect operations that are largely sequential. If detecting the effect of some output is required for an interlock function, it is good practice to provide a separate sensor for that purpose, instead of assuming that the commanded effect has been achieved.

4.1 Design formulas

Any combinational ladder program can be expressed as a set of propositional formulas, one for each rung, which we call *design formulas*. Each design formula consists of a single output on the left of a double implication `iff` ($\Leftrightarrow$), with an expression on the right that only includes the connectives `and` ($\wedge$), `or` ($\vee$), and `not` ($\neg$). Outputs may appear on the right side of a design formula, as long as no outputs are fed back to inputs, directly or indirectly.

The design formulas for the example (section 3, Fig. 3) are:

$$\begin{array}{lll} D1: & SAFE & \Leftrightarrow & ENABLE \wedge \neg DISABLE \wedge TREAT \\ D2: & BARRIER & \Leftrightarrow & OPEN \wedge \neg CLOSE \wedge SAFE \\ D3: & RF & \Leftrightarrow & ON \wedge \neg OFF \wedge VAULT \wedge (\neg OPENED \vee SAFE) \end{array}$$

It is useful to transform the set of design formulas to a set of *expanded design formulas* where intermediate results such as TREAT are expanded, so that only actual outputs appear, and all outputs are expressed in terms of inputs. In our example the expanded design formulas are:

$$
\begin{array}{lll}
ED2: & BARRIER & \Leftrightarrow \quad OPEN \wedge \neg CLOSE \wedge (ENABLE \wedge \neg DISABLE \wedge TREAT) \\
ED3: & RF & \Leftrightarrow \quad ON \wedge \neg OFF \wedge VAULT \wedge (\neg OPENED \vee (ENABLE \ldots))
\end{array}
$$

(Note these are numbered for ease of comparison with the original design formulas).

Each expanded design formula must be true in all system states, so the separate design formulas can be conjoined into a single design formula that must always be true. We call this the system's *combined expanded design formula*. In our example, the combined expanded design formula EDC is:

$$EDC: \quad ED2 \wedge ED3$$

4.2 Safety requirements

There are two kinds of safety requirements. *Safety interlock requirements* should be enforced automatically by the interlock system. These requirements are derived from English sentences such as: "The beam cannot turn on if the cyclotron vault door is open." Safety interlock requirements are formalised as implications, with some operating condition as the antecedent (left side) of the implication and the safety conditions that must be satisfied in that operating condition as the consequence (right side).

In our example, we have two safety interlock requirements:

$$
\begin{array}{lll}
S1: & RF & \Rightarrow \quad VAULT \\
S2: & RF \wedge \neg CLOSED & \Rightarrow \quad ENABLE \wedge TREAT
\end{array}
$$

Safety interlock requirements mean, "If the outputs are energized in a certain configuration, then sensors indicate that the system is safe." Implication (`if`, $\Rightarrow$) gives us just the meaning we want, since it is false only in the case where the outputs are energized but the safety sensors do not indicate safety; if the outputs are not energized, it does not matter what the sensors indicate. It may be helpful to note the equivalence of the form $o \Rightarrow s$ with $\neg i \Rightarrow \neg s$ ("If the sensors say the system is unsafe, then the output is off") and $\neg o \vee s$ ("The output is off, or the sensors indicate safety").

The second kind of safety requirement is the *safety control requirement.* These requirements concern facilities that must be made available at the operators' consoles and are derived from sentences such as, "Pressing the OFF button will turn off the RF drive." Safety control requirements are implications, with the pertinent control input on the left side and the affected output on the right side, to indicate that the operator can (for example) unconditionally turn off the beam or close the barrier:

$$
\begin{array}{lll}
S3: & OFF & \Rightarrow \neg RF \\
S4: & CLOSE & \Rightarrow \neg BARRIER
\end{array}
$$

It is important to establish that there are no peculiar states in which these controls do not work as intended. They are as critical as the automatic interlocks. We depend on the operators to manually achieve a safe state if certain fault conditions occur, for example if the patient is observed to move during a treatment while the beam is on.

Every safety requirement must be satisfied in all system states. Therefore, the separate safety requirements can be conjoined into a single safety requirement that must always be true. We call this the system's *combined safety formula.* In our example, the combined safety formula SC is:

$$SC: \quad S1 \wedge S2 \wedge S3 \wedge S4$$

4.3 Other requirements

In order to synthesize designs, additional information is needed. Most of these additional requirements are similar to liveness requirements, because they describe events that should happen. *Activation formulas* are derived from sentences such as, "Pressing the ON button can turn on the RF drive." In our example they are:

$$
\begin{array}{lll}
A1: & RF & \Rightarrow ON \\
A2: & BARRIER & \Rightarrow OPEN
\end{array}
$$

These formulas are implications with the output on the left and the corresponding control input on the right. This correctly represents the case where (for example) the operator presses the ON button but the RF does not turn on, because some interlock prevents it.

Every activation formula must be true in every system state, so they can be conjoined into a combined activation formula. In our example it is:

$$AC: \quad A1 \wedge A2$$

Finally, there are *output mode formulas.* These are conjunctions representing combinations of outputs that must be achievable in some operating mode. They are derived from sentences such as, "It is possible to turn on the beam in the treatment room." In our example they are:

$$
\begin{array}{ll}
M1: & RF \wedge \neg BARRIER \\
M2: & RF \wedge BARRIER
\end{array}
$$

Unlike the other formulas, output mode formulas are not true in every system state, so it is not meaningful to conjoin them. There are many system states where none of the output mode formulas are true and there is usually no state where all the output mode formulas are true.

5 Verification

The verification problem is to show that a proposed or existing design is consistent with requirements. A necessary preliminary for verification is that each output of interest must be identified, and represented in a combined expanded design formula of the form described in section 4.1 At this stage it is necessary to confirm that each output can be represented by combinational (not sequential) logic.

5.1 Verifying safety

Usually verification concentrates on safety. To verify safety, confirm that all design formulas are consistent with all safety requirements, by conjoining the combined expanded design formula with the *negation* of the combined safety requirement, to obtain the *safety verification formula*. In our example, the safety verification formula SV is:

$$SV: \quad EDC \wedge \neg SC$$

Safety is verified by determining that the safety verification formula is *not* satisfiable. Any assignment of values $true$ and $false$ to variables that satisfies SV (makes SV evaluate to $true$) represents an unsafe state: a combinations of inputs and outputs that is permitted by the design formulas, but violates the safety requirements.

Applying this test to our example, we find that our safety verification formula can be satisfied. The unsafe states are:

BARRIER	ON	OFF	OPENED	VAULT	RF	CLOSED	ENABLE	TREAT
F	T	F	F	T	T	F	F	*
F	T	F	F	T	T	F	T	F

(where T is *true*, F is *false*, and "*" represents "don't care" conditions. OPEN, CLOSE and DISABLE are always "don't care" conditions).

These are states where the beam is on in the vault, the treatment room is not completely safe, the output that opens the barrier is not energized but *the barrier is not closed.* This situation can occur because $\neg OPENED$ is not equivalent to $CLOSED$. After the output that holds the barrier open is activated or deactivated, the barrier always travels through an intermediate position for several seconds while neither the OPENED nor CLOSED sensor is asserted. During part of this interval the beam could reach the treatment room.

The error can be corrected by changing $\neg OPENED$ to $CLOSED$ in the interlock program. The corrected rung in the ladder diagram appears in Fig. 4 (compare to Fig. 3), and the corrected design formula D3 is:

$$D3: \quad RF \quad \Leftrightarrow \quad ON \wedge \neg OFF \wedge VAULT \wedge (CLOSED \vee SAFE)$$

With this correction in place, the new safety verification formula is not satisfiable, indicating that the new design is consistent with the safety requirements.

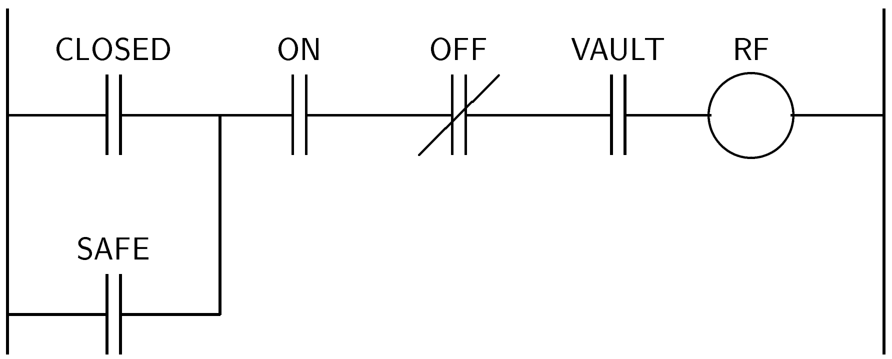

Figure 4: Corrected rung in ladder program for cyclotron and treatment room interlocks

5.2 Verifying other properties

Safety is difficult to verify by informal inspection because it requires consideration of all possible states. Other requirements can be verified by identifying a single state that satisfies each requirement. Designers can usually ensure this. Nevertheless, we note that these requirements can also be formally verified, as follows:

Form the conjunction of each output mode requirement with the combined activation requirement and the combined expanded design formula. In our example there are two such *activation verification formulas*:

$$\begin{array}{ll} AV1: & M1 \wedge AC \wedge EDC \\ AV2: & M2 \wedge AC \wedge EDC \end{array}$$

Each activation verification formula must be satisfiable. It is necessary to test each formula separately; in general, it is not possible to satisfy more than one mode requirement at a time.

6 Analysis

6.1 Analyzing requirements

One can ask whether a given set of requirements are consistent: is there any implementation that can satisfy them? It is possible to answer this question by forming a set of propositions where each output mode requirement in turn is conjoined with all the activation requirements and safety requirements. In our example these *combined requirement formulas* are:

$$\begin{array}{ll} R1: & M1 \wedge AC \wedge SC \\ R2: & M2 \wedge AC \wedge SC \end{array}$$

The requirements are consistent if all the requirement formulas are satisfiable.

One can also ask whether the formal requirements (the formulas) faithfully capture the designer's intent. This question can be illuminated by examining the set of assignments that satisfy each combined requirement formula. These assignments list the input configurations that can occur in each output mode. Designers can compare these with their expectations. In particular, it is useful to consider the "don't care" inputs that can assume any value without affecting the outputs in a particular mode.

6.2 Analyzing existing designs

It is sometimes necessary to determine the functions of an existing interlock system, where the only available documentation is the ladder diagram. If the system incorporates many inputs and outputs, and the design has been optimized to minimize the number of elements used, the intended functions can be far from self-evident.

The formalism proposed here suggests an approach to "inverting" existing designs to obtain their implicit requirements. First, for each output of interest the ladder diagrams are translated to design formulas (section 4.1) and then converted to expanded design formulas by substituting and expanding intermediate results. Then, the right side of each extended design formula is converted to *disjunctive normal form* (DNF). Each disjunct represents a mode in which in which that output can turn on.

7 Synthesis

The synthesis problem is to create the design formulas, given the requirements. We would like a synthesis method that can be shown to always generate a correct design (given consistent requirements). Moreover, the design formulas must be useful in the chosen implementation technology. For ladder diagrams, this means they must take the form of the design formulas of section 4.1.

The formalism proposed here suggests an approach to synthesizing designs from requirements. The various requirements are first collected together into combined requirements formulas as in section 6.1. Then, one may apply standard normalization and minimization techniques to produce an economical design for the chosen implementation technology. For ladder diagrams, one might convert the requirements for each output to DNF, and then factor out conjunctions that appear in more than one disjunct or rung, representing these intermediate results as coils, as illustrated in Fig. 3.

8 Current status and future work

We have only examined a few simple examples. Further work is needed to demonstrate the usefulness of these techniques for realistic systems.

8.1 Select a boolean manipulation package

The example in this paper was checked using a small program that tests for satisfiability by generating and searching truth tables. This is not a practical method for systems of realistic complexity, which will require software that efficiently manipulates boolean expressions. A central task is to efficiently solve the boolean satisfiability problem; many approaches are reviewed in [15]. Also required are efficient transformations to normal forms such as DNF and efficient treatment and compact presentation of "don't care" conditions. A package based on the *binary decision diagram* (BDD) representation of boolean expressions [3, 2] would seem well-suited to this application.

8.2 Consider realistic systems

The actual beamlines that inspired the example in this paper are far more complex [6]. The cyclotron feeds three beam lines, and each treatment room can be used in two modes (there is a less protected mode used for maintenance and testing, with no patient present). Instead of a single barrier, each beam line includes six independently moveable elements; the rules governing each depend on the currently active beamline and its mode.

It will be important to determine whether systems of realistic complexity can be handled by available boolean manipulation packages.

An important pracitical issue derives from our requirement that interlock logic must be combinational, not sequential. This means that outputs cannot be fed back to inputs. In fact feedback is a common ladder programming "idiom" used to obtain a toggle effect from momentary pushbuttons. It is usual to install pairs of momentary pushbuttons rather than toggle switches for practical reasons (wiring is easier if duplicated controls must be offered at several locations, etc.). Feeding an output back to its input achieves a "latching" or "self-hold" effect, similar to a flipflop. All such occurences can be replaced by toggle switches for purposes of analysis, thus removing the feedback without substantively affecting functional behavior (Fig. 5). In fact, our example has already been transformed in this way. All of the controls shown on the cyclotron operator's panel (Fig. 2) are actually momentary pushbuttons, and Figs. 3 and 4 already include the transformation illustrated in Fig. 5.

It will be important to determine whether it is practical to perform transformations to remove nonessential feedback, and to learn whether there are interlock functions where sequential logic is essential.

8.3 Develop synthesis and analysis techniques

(I hope to demonstrate a sound synthesis procedure in the final version of the paper)

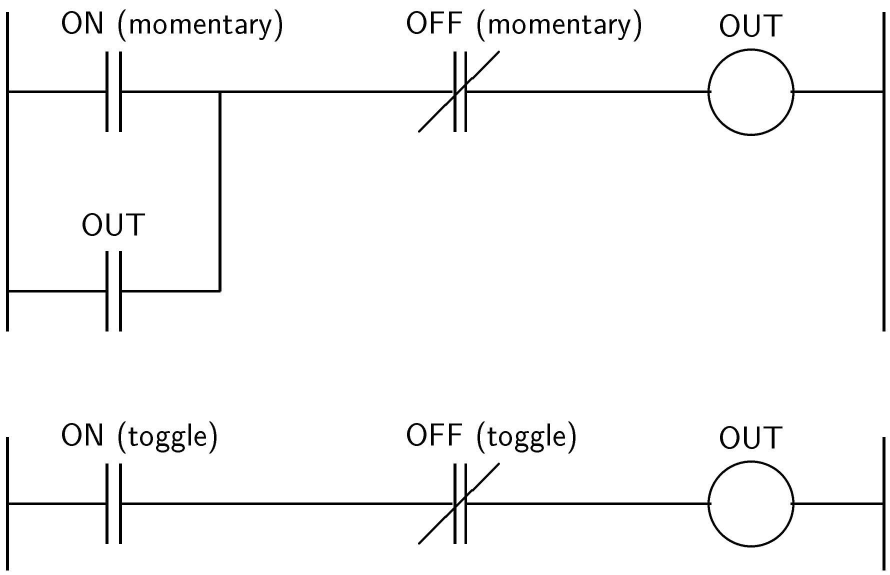

Figure 5: Equivalent sequential and non-sequential ladder diagrams

8.4 Refine the notion of completeness

Jaffe and Leveson [7] have proposed a definition of the completeness of system requirements. It is expressed in predicate calculus, so it should not be difficult to cast it in propositional terms as well. Its relevance to the techniques proposed here will be considered, to determine whether an explicit test of completeness (in their sense) can be described.

(I hope to have this by the final version also)

9 Discussion

This report shows how simple applications of elementary logic might be useful in an area where informal, intuitive approaches still dominate practice.

These techniques are complementary to the standard digital design methods that emphasize minimization and normalization of given boolean formulas. The emphasis in this report is on relating boolean formulas to propositions which are closer to natural language statements of requirements.

This paper is illustrated with ladder diagrams, but the same techniques could be applied in other technologies, such as programmable logic devices (PLD's). They are also relevant to programs written in high-level languages. However, special-purpose application languages, such as ladder diagrams, can make verification easier because they more closely resemble the formalism in which the requirements are expressed.

10 Acknowledgements

The author thanks Ruedi Risler for sharing his knowledge of the cyclotron facility. The author thanks Tod Amon, Gaetano Boriello, and Carl Ebeling for commenting on an earlier version of this paper, and acquainting the author with BDD's and other digital techniques.

References

[1] Romain C. Agostini, Loy Barker, Jim Hodgers, Daryl Reagan, and Helmut V. Walz.

PLC-based interlock system for superconducting magnets. *IEEE Transactions on Nuclear Science*, 36(5):1741 – 1745, October 1989.

[2] Karl S. Brace, Richard L. Rudell, and Randal E. Bryant. Efficient implementation of a BDD package. In *27th ACM/IEEE Design Automation Conference*, pages 40 – 45. IEEE Computer Society, June 1990.

[3] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677 – 691, August 1986.

[4] T. Dickinson. Radiation safety systems at the NSLS. In Eric R. Lindstrom and Louise S. Naylor, editors, *Proceedings of the 1987 IEEE Particle Accelerator Conference: Accelerator Engineering and Technology, Volume 3*, pages 1605 – 1607. IEEE, 1987.

[5] Great Britain Health and Safety Executive. *Programmable Electronic Systems in Safety Related Applications. Volume 1: An Introductory Guide. Volume 2: General Technical Guidelines.* Her Majesty's Stationery Office, London, 1987.

[6] Jonathan Jacky and Cheryl P. White. Testing a 3-D radiation therapy planning program. *International Journal of Radiation Oncology, Biology and Physics*, 18:253–261, January 1990.

[7] Matthew S. Jaffe and Nancy G. Leveson. Completeness, robustness, and safety in real-time software requirements specification. In *11th International Conference on Software Engineering Proceedings*, pages 302–311. IEEE Computer Society, 1989.

[8] Allan G. Kern. Batch process automation in a PLC: Software design is the key. *ISA Transactions*, 29(2):33–45, 1990.

[9] Nancy G. Leveson. Software safety: What, why and how. *ACM Computing Surveys*, 18(2):125–163, June 1986.

[10] Nancy G. Leveson. Software safety in embedded computer systems. *Communications of the ACM*, 34(2):34 – 48, February 1991.

[11] David L. Parnas, A. John van Schouwen, and Shu Po Kwan. Evaluation of safety-critical software. *Communications of the ACM*, 33(6):636 – 648, June 1990.

[12] R. J. Post. Some considerations of interlocking and safety circuits for radiotherapy apparatus. *International Journal of Radiation Engineering*, 1(2):169–191, 1971.

[13] Ruedi Risler, Jüri Eenmaa, Jonathan P. Jacky, Ira J. Kalet, Peter Wootton, and S. Lindbaeck. Installation of the cyclotron based clinical neutron therapy system in Seattle. In *Proceedings of the Tenth International Conference on Cyclotrons and their Applications*, pages 428–430, East Lansing, Michigan, May 1984. IEEE.

[14] Howard P. Rosenof and Asish Gosh. *Batch Process Automation: Theory and Practice.* Van Nostrand Reinhold Company, New York, 1987.

[15] H. Simonis and M. Dincbas. Propositional calculus problems in CHIP. In H. Kirchner and W. Wechler, editors, *Algebraic and Logic Programming: Second International Conference*, pages 189 – 203, Nancy, France, October 1990. Lecture Notes in Computer Science, No. 463.

[16] John Sloan. Advanced PLC program design. In *WESCANEX '87: Western Canada's First Programmable Control Systems Conference and Exhibition*, pages 45 – 48. IEEE, 1987.

[17] Robert E. Wilhelm, Jr. *Programmable Controller Handbook*. Hayden Books, Indianapolis, Indiana, 1985.