Under consideration for publication in Formal Aspects of Computing

The Mechanical Generation of Fault Trees for Reactive Systems via Retrenchment II: Clocked and Feedback Circuits [1]

Richard Banach[1] and Marco Bozzano[2]

[1]School of Computer Science, University of Manchester,
Oxford Road, Manchester, M13 9PL, U.K.
banach@cs.man.ac.uk,
[2]FBK-IRST, Via Sommarive 18,
Povo, 38123 Trento, Italy
bozzano@fbk.eu

Abstract. The retrenchment approach to the mechanical construction of fault trees, introduced in the first paper for combinational logic circuits, is extended to handle clocked circuits and then feedback circuits. The temporal behaviour of clocked circuits is captured using their causal relations, and the potentially unbounded behaviour of cyclic circuits is decomposed into an iteration over their acyclic counterparts. The repercussions of all this for the theory of retrenchment are elaborated. For clocked circuits, the techniques we present allow glitches and other transient errors to be properly described. For feedback circuits, the plethora of behaviours that can occur, give rise to infinitary fault trees of an appropriate kind. All this paves the way for automated fault tree generation for reactive systems.

Keywords: Fault Tree Analysis, Fault Injection, Retrenchment, Mechanical Fault Tree Synthesis, Timed and Feedback Circuits

1. Introduction

In [BB10] hereafter referred to as PaperI, we introduced an approach to the mechanical construction of fault trees via retrenchment, focusing exclusively on the technically (and notationally) simpler case of pure (finite acyclic) combinational logic circuits, those which produce their outputs at the same instant that they consume their inputs. We covered the relevant details of retrenchment theory, and showed how the retrenchment simulation relation for suitably formulated retrenchments could be analysed to give a structured technique for generating nontrivially nested fault trees, illustrating this with an example. The informal example-led account was then suitably formalised, and shown to be

Correspondence and offprint requests to: Richard Banach, School of Computer Science, University of Manchester, Oxford Road, Manchester, M13 9PL, U.K. email: banach@cs.man.ac.uk

[1] Work partly supported by the E.U. projects ISAAC, contract no. AST3-CT-2003-501848, and MISSA, contract no. ACP7-GA-2008-212088.

sound and complete. The rigorous treatment did also clarify the details of the intermediate event insertion mechanism, so that the resolution trees of the basic analysis could be manipulated into a form closer to the demands of fault trees of a standard kind [VSD+02], whenever needed. We also tackled minimisation, essential for large examples in order to prevent the useful content of the fault tree from being obscured by huge numbers of superfluous cases. We showed that our structured technique presented several opportunities for on-the-fly minimisation, helping to reduce the *post hoc* subsumption minimisation workload, thereby potentially making some previously intractable examples tractable.

In this paper, we extend this work to the more interesting cases of synchronous circuits, both acyclic and cyclic. This requires revisiting the foundations of our theory, and reworking it in the context of streams of values. Since our aim is to capture analysis processes that must inevitably be finitary, we avoid dealing with the intrinsically infinite nature of streams as much as we can, in particular avoiding the heavier machinery of e.g. [Bc01] and related work. Once the foundations are in place, the resulting theory can be reinterpreted as a collection of statements about a collection of variables each of which takes values in a finite set, rather as for the combinational logic case — so the remaining theory can largely be taken over from the combinational logic case. At least that is the way it is for finite acyclic timed circuits. For the cyclic case, the potentially and unavoidably infinite behaviours that arise, necessitate an approach that cuts them up into finite pieces, and reassembles their infinitary properties from these finite pieces. So, there is an extra level of theoretical indirection to deal with, which adds some complexity to the process. Nevertheless, it ultimately yields a theory which is a smooth extension of the finite case.

We show that the extended theory can be developed into a structured approach for the generation of resolution trees for reactive systems. These resolution trees have a very general form that takes into account the relative timing of different events and feedback loops. They can be seen as a reference model that can be used for the generation of fault trees at different levels of abstraction. While keeping all the details may be prohibitive because of the complexity of a practical implementation, and the wealth of detail may also hinder the comprehension of the results, our resolution trees offer several opportunities for incorporating timing information in fault trees, in a controlled way that is both practicable and meaningful for safety engineers. This constitutes a significant improvement over the fault trees discussed in [BV07], where the structure of fault trees has been flattened as discussed in PaperI — the trees of [BV07] may be seen as an extreme simplification of our resolution trees where timing information has been completely abstracted away. As for the combinational case, when reviewed and properly interpreted by safety engineers, we expect the generated fault trees to be useful for identifying errors, either in system design or in formal models used for system design, and for complementing traditional safety assessment techniques.

The reference model idea deserves further comment. The analysis technique we propose for building the resolution tree is exhaustive, but because of its theoretical foundations, it is also detailed and complete. As noted already, the totality of detail may inhibit its use *verbatim* for large examples, but this does not stand in the way of its use as an overriding conceptual framework, from which more practical approaches may be derived by selectively forgetting some of the details of the full technique. In particular, this can better illuminate how different practical approaches relate to one another via the different subsets of the full information that they forget. In the sequel, our derivations are always targeted at the fault trees derived from the analysis, since these are an industry standard artifact, but we discuss the relationship with the precursor resolution tree adequately in each case. These relationships anyway turn out to be straightforward enough for the examples we treat.

Bearing this in mind, the rest of the paper is as follows. In Section 2 we discuss fault tree analysis for synchronous circuits. In Section 3 we develop the techniques for describing the behaviour of acyclic clocked circuits and their I/O streams (which are already potentially infinitary objects) in terms of finite pieces (their causal relations), showing how these compose. This perspective is crucial if the description of system behaviour is to remain finite — it also eliminates explicit state from the description of systems. In Section 4 we elaborate the consequences of this framework for retrenchment.

We then turn our attention to fault tree generation for acyclic clocked circuits in Section 5. We treat an example closely related to the example of PaperI, and show that the extension of the combinational framework can now readily deal with glitches (transient faults lasting for only one, or for very few clock ticks), and transient phenomena in general. We also comment briefly on the theoretical aspects.

We then turn to the cyclic case. Although the underlying retrenchment theory for this quite a bit more complicated than for the acyclic case, it so happens that the practical algorithms that emerge from the theory are a very mild extension of those for the acyclic case. So to aid readability, we delegate the theory to appendices, and focus on the practical aspects in the body of the paper.

Thus, in Section 6 we examine fault tree generation for cyclic systems. We treat a cyclic example that builds on our earlier ones. For finite cyclic structures which are capable of infinite behaviours, provided the inputs conform to a regular structure, the set of infinite behaviours has a regular structure too. We see that the fault tree derived for our cyclic example has such a regular structure, which is captured by finite means using back-links in a finite fault tree

— evidently this extends the notion of fault tree in [VSD+02]. We also discuss the impact of feedback on the formal treatment of fault tree generation. Section 7 then discusses related work. Section 8 concludes and outlines future work.

Several appendices cover material sidelined from the main body of the paper. Appendix A covers the calculation of the within relation for skew-sequentially composed indexed dataflow retrenchments. Then, Appendix B is concerned with the more complicated technical details of the formulation of system models in the cyclic case. It starts by describing how the acyclic clocked formalism can be extended to cover cyclic circuits. Appendix C then explores the more drastic consequences of this for retrenchment. Now while the extension of retrenchment theory from the instantaneous case to the timed acyclic case is mild, the extension for cyclic systems, particularly when oriented for backwards fault analysis, is technically much more demanding. Thus, while for the preceding extensions we could just quote results from elsewhere, in the cyclic case, the novelty we encounter merits the inclusion of a discussion of the proof of soundness of the backwards infinitary retrenchment. We also comment on composition mechanisms for these more complex cyclic objects.

N. B. Although this paper is self-contained, those aspects dealt with in detail in PaperI but also needed here are given rather tersely. So, a familiarity with PaperI, while not indispensable, is nevertheless desirable.

2. Fault Tree Analysis for Synchronous Circuits

In this section we resume the discussion on the role of fault trees in reliability engineering given in PaperI, focusing on the issues that arise in presence of time, both in the acyclic and the cyclic cases. Although traditionally fault trees do not provide strong facilities for expressing time-related behaviour, there are several ways in which timing information can be incorporated into FTA [VGRH81, VSD+02]. We describe below a few possibilities.

A straightforward approach is to incorporate time-related information into the textual description of intermediate events. As an example, an intermediate event could be described as ‘relay contacts are closed for more than 3 seconds’, or ‘motor fails to start within 5 seconds’. Similarly, external (house) events inside the fault tree can be used to qualify branches of the tree that deal with different modes or phases of operation. For instance, it is possible to use such events to distinguish the operation of an aircraft when it is on the ground or in flight. A more structured way of adding timing information to fault trees is by using an *inhibit gate* with an associated *conditioning event* [VSD+02]. An inhibit gate can be used to constrain the ways faults are propagated inside the fault tree. In particular, an input fault can be propagated to the output of an inhibit gate only if the corresponding condition is enabled. Hence, time-related information can be incorporated into the conditioning event. The use of inhibit gates is common when developing ‘state of system’ faults. Moreover, safety engineers may use *exposure intervals* in order to qualify the period during which components are exposed to failures. The use of exposure intervals is common to qualify the exposure to failure of basic events in the fault tree, and it is required for quantitative evaluation. The exposure interval may depend on a number of factors. Typically it involves considering the total time a given component is relied upon during service, and on maintenance tasks. For instance, in avionics, the exposure interval for a component can be defined as the time between successive inspections, or the duration of a flight if the component is inspected after each flight.[2] Finally, as mentioned in PaperI, dynamic gates may be used to model timing dependencies between events in a fault tree, see [DBB92]. Examples of dynamic gates are priority AND, which enforces an order of occurrence between input events, and spare gates, modeling timing dependencies between spare units. A few example fault tree structures are sketched in Fig. 1.

The presence of loops and feedback complicates the generation of fault trees considerably. Traditionally, loops are not allowed inside fault trees, so loops must be resolved by safety analysts. Different strategies can be used to resolve logic loops. A common strategy [VSD+02] prescribes that, when a feedback loop involving signals exchanged between two subsystems is present, the analysis should be limited to the internal failures of each subsystem, and the feedback signal should not be followed. Typically, this amounts to performing an *ad hoc* abstraction of the system, which is often captured as a state-of-system fault in the resulting fault tree. Alternative techniques for resolving logical loops may be adapted to the specific system at hand. In general, resolving logical loops relies on the safety engineer’s expertise, and requires understanding of the model as a whole, thus breaking the traditional ‘divide-and-conquer’ strategy used to construct fault trees.

The techniques presented in this paper generalize the ones presented in PaperI to the case of dynamic and cyclic systems. As in PaperI, the resolution tree being constructed follows system structure. Moreover, the construction of

[2] Note that the exposure interval could be reduced, e.g. when the component is not being used throughout the flight, or when it can fail only under specific circumstances. An example of this is given by a pool of *cold spares* units, which are allowed to fail only when activated — one at a time in a specific order [DBB92].

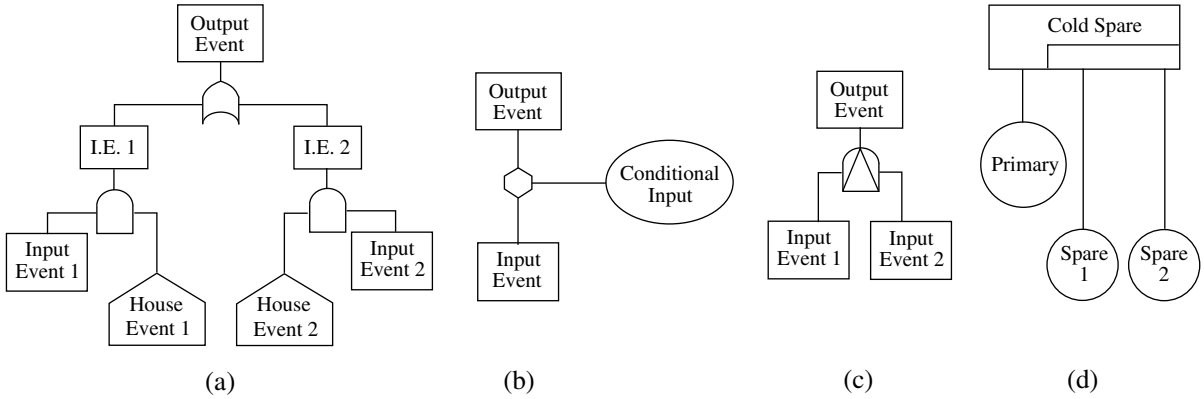

Fig. 1. Some example FT structures modelling different kinds of timed dependencies: (a) a fault tree structure with house events, (b) a conditioning event associated with an inhibit gate, (c) a priority AND gate, (d) a cold spare gate.

the tree in the timed and cyclic cases follows the same basic rules illustrated in PaperI. While in the acyclic case the tree is always guaranteed to be loop-free, in the case of systems with feedback, we are required to extend the basic routines with some kind of 'loop detection' mechanism which avoids infinite unfolding of the model.[3] This is achieved by using back-links to finitely represent an infinite number of possible behaviours inside a finite tree.

Clearly, the introduction of back-links extends the notion of resolution tree of PaperI. Given that resolution trees can be transformed into fault trees, this also suggests an extension of the traditional notion of fault tree given in [VSD+02], by admitting logical loops. In Section 6.5 we briefly discuss the issue of transforming resolution trees into more traditional fault trees, by resolving logical loops and forgetting timing information in a controlled way, and we compare that approach with what is currently available in safety analysis platforms based on symbolic formal verification techniques, such as FSAP [BV07]. This transformation also addresses the problem of the size and complexity of the generated trees, making them suitable for use by safety analysts (see also the discussion in Section 8). As for PaperI, the construction of a resolution tree may offer opportunities for on-the-fly minimisation, when the generation of a minimised fault tree is required. Minimisation rules are discussed in Section 5.

3. Acyclic Synchronous Systems and their Compositions

We now set up the technical machinery for dealing with clocked and feedback circuits. We start with an informal description, giving formal definitions later. As in PaperI, we describe our synchronous hardware system models using collections of input/output transformers, one transformer per component or collection of components, depending on the granularity of the description. An I/O transformer is a relation from the system's inputs to the system's outputs. Given that we are dealing with *synchronous* hardware systems, we can expect the delays in the input/output behaviour of components or subsystems to be just integer multiples of the clock delay, so that simple indexing is sufficient to keep track of all the different values that are needed. Thus the inputs and outputs will form *streams*, i.e. sequences over the time index set. At the least insightful level of abstraction, the behaviour of a circuit is just a relation from the input streams to the output streams in their entirety.

Henceforth we will assume that all circuits are **time invariant**, i.e. that observed behaviour depends on relative time intervals between input and output values, rather than absolute time indexes.[4] We will also assume that that circuits are **total** as relations from inputs to outputs, i.e. in I/O terminology, they will be **input ready**. This is useful later, and is invariably the case in practice. Also, all data types we use are **finite**, as one would expect in a digital hardware context.

For now, all circuits are acyclic. Time invariance, acyclicity and the finite nature of digital components, make it reasonable to assume that the overall relation from input streams to output streams decomposes into independent

[3] Moreover the loop detection mechanism must be appropriately sensitive to any inputs present.

[4] The single deviation to time invariance that we tolerate is when we inject transient faults into an otherwise time invariant system. Time invariance then implies that if we displaced the injection of the transient faults (along with other inputs) forwards or backwards in time, then the resulting behaviour of the system as a whole would also be similarly displaced.

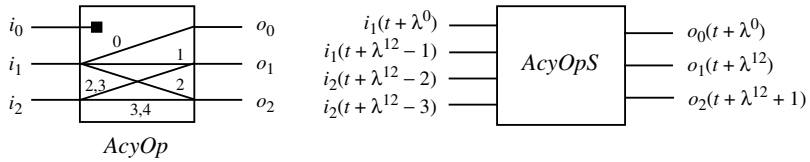

Fig. 2. An acyclic circuit *AcyOp*, and its causal relation *AcyOpS*.

pieces, called **irreducible subrelations**, where each irreducible subrelation is a relation from a few input values extracted from the input streams at a few time indexes, to a few output values injected into the output streams at a few later time indexes. An irreducible subrelation is therefore just a relation (which we refer to as the underlying relation of the irreducible subrelation). However, in order to connect the relation to the context of a circuit which consumes and produces values over time, we need also a specification of how particular input values of the relation are drawn from the input streams read by the circuit, and particular output values of the relation are inserted into output streams written by the circuit — all taking into account any relative delays that may be needed.[5] The overall relation from the circuit's input streams (taken in their entirety) to the circuit's output streams (taken in their entirety) then becomes just the patching together of occurrences of the relevant irreducible subrelations at various times.

What makes an irreducible subrelation *irreducible*, is the property that it cannot be expressed as a parallel composition of two simpler relations. This irreducibility property and basic causality, imply that the latest of the input values (drawn from the input streams) in an irreducible subrelation can occur no later than the earliest of the output values (inserted into the output streams). The irreducibility property implies that an irreducible subrelation might only describe a *part* of the functionality of the circuit of interest, in which case the overall behaviour would be given by more than one of them, acting in a parallel composition over the set of input streams, forming the **causal relation** of the circuit.

N. B. Note the terminological convention we will use below: circuits (i.e. pieces of hardware, regarded as such) have input and output *signals*, while irreducible subrelations (i.e. the behaviour of the said pieces of hardware), and the causal relations they form, have input and output *values*, assembled as necessary into *streams*. We will use the same names for the I/O signals (in the hardware picture) and for the I/O streams (in the relational picture).

Let us illustrate these ideas in Fig. 2. On the left is a circuit *AcyOp* (short for *AcyclicOperation*); it has three input signals $i_0, i_1, i_2,$ and three output signals $o_0, o_1, o_2,$. The interior of *AcyOp* indicates relationships between input signals and output signals. Thus, i_0 is not used in forming any output (i.e. there is no dataflow from i_0 to any output), indicated by the small black square. The numbers labelling the lines in the interior of *AcyOp* indicate the delays (in terms of clock ticks) from the relevant input signal to the relevant output signal. Thus the response of o_0 to i_1 is instantaneous (delay 0, labelling the line from i_1 to o_0). Likewise there are dependencies with delays 2 ticks and 3 ticks from i_2 to o_1 (label $2,3$ on the line from i_2 to o_1). And so on.

Merely noting the delays along paths does not yet tell us which parts of the functionality of *AcyOp* are independent and which are coupled. The more informative causal relation *AcyOpS* of *AcyOp* does this. (Notationally, in the remainder of the paper, a final *S* in a relation name indicates that it is the name of a causal relation or irreducible subrelation.) *AcyOpS* is illustrated on the right of Fig. 2. The two independent parameters λ^0 and λ^{12} are reference points in time, relative to which we read from the input streams and write to the output streams. The fact that there are two such parameters, by assumption independent, indicates that the functionality of *AcyOp* splits into two decoupled parts, two irreducible subrelations. (In terms of the circuit *AcyOp*, we would expect to see it consisting of two independent subcircuits, or equivalent).

N. B. At this point it is profitable to mention a notational convention we will adhere to in the rest of the paper. Regarding reference parameters such as λ^0 and λ^{12}, their superscripts will always be derived from the labels of the output streams associated with the (unique) independent irreducible subrelations they belong to — this uniqueness is stipulated formally in Def. 3.4.2 below. Thus λ^0 is associated with output stream 0; and λ^{12} is associated with output streams 1 and 2, (never stream twelve, as we never explicitly deal with more than a small number of different streams in this paper).

Regarding λ^0, we see that $i_1(t+\lambda^0)$ and $o_0(t+\lambda^0)$ are coupled using λ^0, and no other input or output values are

[5] We say *specification*, rather than give a particular mechanism, since the connection between the parameters of an irreducible subrelation and the streams they are linked to may be described in many ways. Def. 3.4 gives a formal, though cumbersome, mechanism. However in the rest of the paper we use a lighter, more informal descripion.

given relative to λ^0. This indicates that there is an irreducible subrelation $AcyOpS^0(i_1(t+\lambda^0), o_0(t+\lambda^0))$ of *AcyOpS* involving just these values. Note that in order to make all this precise, we have to associate the first input position of relation $AcyOpS^0$ with the input stream i_1 and delay 0 (relative to λ^0) of circuit *AcyOp*; also we have to associate the first output position of relation $AcyOpS^0$ with the output stream o_0 and delay 0 (relative to λ^0) of the circuit. In the formal Def. 3.4 below, the association between input positions and input-stream-plus-delay pairs is generically handled by the function $INstr^\gamma$ (Def. 3.4.1.(a)), and the association between output positions and output-stream-plus-delay pairs is generically handled by the function $OUTstr^\gamma$ (Def. 3.4.1.(b)). (Note that for convenience, for an irreducible subrelation, we count input delays negatively and output delays positively, (both of them from the relevant λ parameter). Therefore, by the causality mentioned above, both input and output delays can be specified as positive numbers.)

Likewise, regarding λ^{12}, we see that input values $i_1(t+\lambda^{12}-1), i_2(t+\lambda^{12}-2), i_2(t+\lambda^{12}-3)$ and output values $o_1(t+\lambda^{12}), o_2(t+\lambda^{12}+1)$ are given relative to λ^{12}. This indicates that there is another irreducible subrelation involving them, $AcyOpS^{12}(\langle i_1(t+\lambda^{12}-1), i_2(t+\lambda^{12}-2), i_2(t+\lambda^{12}-3)\rangle, \langle o_1(t+\lambda^{12}), o_2(t+\lambda^{12}+1)\rangle)$, in which all the input values are aggregated using angle brackets, and similarly for the output values (a convention we maintain for the rest of the paper). Thus for $AcyOpS^{12}$, the input association function $INstr^{12}$ maps the first input position of the relation $AcyOpS^{12}$ to input stream i_1 and delay 1, the second input position of $AcyOpS^{12}$ to input stream i_2 and delay 2, and the third input position of $AcyOpS^{12}$ to input stream i_2 and delay 3. Also the output association function $OUTstr^{12}$ maps the first output position of the relation $AcyOpS^{12}$ to output stream o_1 and delay 0, and the second output position of $AcyOpS^{12}$ to output stream o_2 and delay 1. In the preceding, all the delays are relative to λ^{12} itself.

In this manner, *AcyOpS*, which gives the functionality of the circuit *AcyOp* in the context of the input signals i_0, i_1, i_2, and the output signals o_0, o_1, o_2,, is given by the composition of $AcyOpS^0$ and $AcyOpS^{12}$ — plus one extra detail. We note that both $AcyOpS^0$ and $AcyOpS^{12}$ draw values from the input stream i_1. Since these are values from the same input stream, if it happens that $\lambda^0 = \lambda^{12}-1$, the values used must be the same. Thus we arrive at the formulation of *AcyOpS* in terms of $AcyOpS^0$ and $AcyOpS^{12}$:

$$\begin{aligned}
&AcyOpS(\langle i_1(t+\lambda^0), i_1(t+\lambda^{12}-1), i_2(t+\lambda^{12}-2), i_2(t+\lambda^{12}-3)\rangle, \langle o_0(t+\lambda^0), o_1(t+\lambda^{12}), o_2(t+\lambda^{12}+1)\rangle) \equiv \\
&\quad AcyOpS^0(i_1(t+\lambda^0), o_0(t+\lambda^0)) \wedge \\
&\quad AcyOpS^{12}(\langle i_1(t+\lambda^{12}-1), i_2(t+\lambda^{12}-2), i_2(t+\lambda^{12}-3)\rangle, \langle o_1(t+\lambda^{12}), o_2(t+\lambda^{12}+1)\rangle) \wedge \\
&\quad (\lambda^0 = \lambda^{12}-1 \Rightarrow i_1(t+\lambda^0) = i_1(t+\lambda^{12}-1))
\end{aligned} \tag{1}$$

To save space, in future we abbreviate time index expressions like $t+\lambda^{12}-1$ by writing the λ parameter as a subscript to t, so that $t+\lambda^{12}-1$ becomes $t_{\lambda^{12}}-1$. In this manner (1) becomes:

$$\begin{aligned}
&AcyOpS(\langle i_1(t_{\lambda^0}), i_1(t_{\lambda^{12}}-1), i_2(t_{\lambda^{12}}-2), i_2(t_{\lambda^{12}}-3)\rangle, \langle o_0(t_{\lambda^0}), o_1(t_{\lambda^{12}}), o_2(t_{\lambda^{12}}+1)\rangle) \equiv \\
&\quad AcyOpS^0(i_1(t_{\lambda^0}), o_0(t_{\lambda^0})) \wedge \\
&\quad AcyOpS^{12}(\langle i_1(t_{\lambda^{12}}-1), i_2(t_{\lambda^{12}}-2), i_2(t_{\lambda^{12}}-3)\rangle, \langle o_1(t_{\lambda^{12}}), o_2(t_{\lambda^{12}}+1)\rangle) \wedge \\
&\quad (\lambda^0 = \lambda^{12}-1 \Rightarrow i_1(t_{\lambda^0}) = i_1(t_{\lambda^{12}}-1))
\end{aligned} \tag{2}$$

A further convention is at work in (1) and (2). We stipulate that while input values may be shared between different irreducible subrelations, output values may not. Thus, as noted above and formalised in Def. 3.4.2, each output stream belongs to a unique irreducible subrelation. We therefore use a partitioning of the output signal names to distinguish the different irreducible subrelations of a circuit or of its associated causal relation. In (1), the set of output signal names is $\{o_0, o_1, o_2\}$ and the partition of interest is $\{\{o_0\}, \{o_1, o_2\}\}$, corresponding to the two λ values used, λ^0 and λ^{12}. As we have been doing already, instead of writing the elements of the partition, $\{o_0\}$ and $\{o_1, o_2\}$, in full when used as irreducible subrelation names (or as labels for the associated λ values), we shorten the notation, and instead write just 0 and 12 respectively in the superscripts of $AcyOp^0$ and $AcyOp^{12}$. Since this labelling task is the *only* use we have for the partition of output signal names, we are free to view the labels as either the partition elements themselves, or as elements of some other set (such as the set of strings {"0","12"}) in bijection with them. The wording of Def. 3.4.2 sanctions either view.

For non-instantaneous circuits, memory elements typically play a crucial role, introducing the kinds of delay that we have been dealing with into a circuit's behaviour. In fact the combination of instantaneous but otherwise arbitrary I/O relations and pure memory elements is sufficient to cover all cases that can be described by the kind of causal

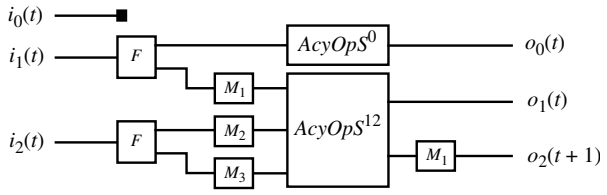

Fig. 3. Realising the causal relation *AcyOpS* of Fig. 2 via a sequential logic circuit of instantaneous elements and memory elements.

relation that we have been discussing. To see this, let us illustrate the implementation of the causal relation *AcyOpS* of (1) by these means. Fig. 3 shows a circuit that achieves this, assuming combinational logic circuits that instantaneously implement (the underlying relations) $AcyOpS^0$ and $AcyOpS^{12}$. In Fig. 3, the F elements are instantaneous fanouts, and the M_k are memory elements that introduce k units of delay into the data path. On the input side, the fanouts and delays ensure that the instantaneous circuits for $AcyOpS^0$ and $AcyOpS^{12}$ receive inputs appropriately scheduled from the input streams i_0, i_1, i_2 (N. B. i_0 is just discarded), while the M_1 element on the output side of $AcyOpS^{12}$ ensures that the second output of $AcyOpS^{12}$ is not delivered to the environment till after one additional unit of delay, as demanded by (1). Fig. 3 corresponds to a specific setting of both λ^0 and λ^{12} to 0. Since we assume time invariance, this is sufficient.

The above discussion persuades us that using irreducible subrelations could be beneficial in terms of efficiency: focusing on one irreducible subrelation at a time prevents having to drag irrelevant data around a calculation, which is particularly beneficial in the context of the VLSI circuits of today. However it will turn out to be the case that irreducibility of relations, as a technical property, will not be needed in the derivations to follow. Thus **irreducibility is a convenience rather than a necessity.**

Let us now formalise the preceding ideas.

Definition 3.1. Let S and T be sets. A relation $R : S \leftrightarrow T$ is a subset of $S \times T$. A relation R is irreducible iff it is impossible to find $R_1, R_2, S_1, S_2, T_1, T_2$ such that $R_1 : S_1 \leftrightarrow T_1$ and $R_2 : S_2 \leftrightarrow T_2$ are relations, and $S = S_1 \times S_2$, $T = T_1 \times T_2$, $R = R_1 \times R_2$ all hold. $R : S \leftrightarrow T$ is reducible iff it is not irreducible.

If a relation $R : S \leftrightarrow T = R_1 \times R_2$ is reducible, with $R_1 : S_1 \leftrightarrow T_1$ and $R_2 : S_2 \leftrightarrow T_2$, and $S = S_1 \times S_2$, $T = T_1 \times T_2$ then whenever (s_{11}, t_{11}) and (s_{12}, t_{12}) are elements of R_1 (and are distinct), and (s_{21}, t_{21}) and (s_{22}, t_{22}) are elements of R_2 (and are distinct), then all four combinations: $((s_{11}, s_{21}), (t_{11}, t_{21}))$ and $((s_{11}, s_{22}), (t_{11}, t_{22}))$ and $((s_{12}, s_{21}), (t_{12}, t_{21}))$ and $((s_{12}, s_{22}), (t_{12}, t_{22}))$ are in R. Finding such pairs of pairs, and showing that one of the combinations is missing, is often an efficient way of showing irreducibility of R (for the given way of decomposing R).

Definition 3.2. Let S and T be sets, and let $R : S \leftrightarrow T$ be a relation from S to T. Suppose $S = S_1 \times S_2 \times \ldots \times S_m$ and $T = T_1 \times T_2 \times \ldots \times T_n$. Then an input position of R is an element of $\{1 \ldots m\}$, and an output position of R is an element of $\{1 \ldots n\}$.

The input and output positions of an irreducible subrelation constitute the means by which the underlying relation of an irreducible subrelation (of a circuit of interest) can be connected to the input and output signals of the circuit (this being done formally in Def. 3.4.4).

Definition 3.3. A stream $St : \langle a \ldots b \rangle \to D$ is a function whose range D is a data type (for us, a set), and whose domain $\langle a \ldots b \rangle$ is a segment of the integers, where a is finite or $-\infty$, b is finite or $+\infty$, and $a \leq b + 1$, all hold.[6]

The formal definition of a stream just given enables the required input values of an occurrence of an irreducible subrelation to be located, and the necessary output values of the occurrence to be placed. Item 4 of Def. 3.4, next, handles the formal details.

Definition 3.4. An acyclic circuit behaviour for a time interval $\langle t_{\mathsf{init}} \ldots t_{\mathsf{fin}} \rangle$ (where t_{init} is finite or $-\infty$, t_{fin} is finite or $+\infty$, and $t_{\mathsf{init}} \leq t_{\mathsf{fin}} + 1$), consists of a set of input streams ST_{in}, a set of output streams ST_{out}, and a set of irreducible subrelations $\{R^\alpha, R^\beta \ldots\}$, subject to the following conditions.

1. Each irreducible subrelation R^γ is associated with three things:

[6] For any finite $c \in \mathbb{Z}$, $\langle c+1 \ldots c \rangle$ denotes the empty segment of integers.

(a) a function $INstr^{\gamma}$ that maps each input parameter position of its signature to (a pair consisting of):

i an input stream in ST_{in} where the data type of the stream is the data type of the input parameter of (the underlying relation of) R^{γ},

ii a delay $\theta \in \mathbb{N}$ (which depends on the input parameter position),

(b) a function $OUTstr^{\gamma}$ that maps each output parameter position of its signature to (a pair consisting of):

i an output stream in ST_{out} where the data type of the stream is the data type of the output parameter of (the underlying relation of) R^{γ},

ii a delay $\phi \in \mathbb{N}$ (which depends on the output parameter position),

(c) a reference index λ^{γ} permitted to range over a segment of integers $\langle \lambda^{\gamma}_{\mathsf{init}} \ldots \lambda^{\gamma}_{\mathsf{fin}} \rangle$.

2. (The labels) $\alpha, \beta \ldots$ (are in bijective correspondence with the elements of a) partition (of) ST_{out}.

3. For each γ, the function $OUTstr^{\gamma} \mathbin{⨾} \mathsf{fst}$ (where fst projects out the first of a pair) —which maps the output parameter positions of R^{γ} to the output streams given by $OUTstr^{\gamma}$— is injective.[7]

4. Let $t \in \langle t_{\mathsf{init}} \ldots t_{\mathsf{fin}} \rangle$. Let R^{γ} be one of the irreducible subrelations. Let $\lambda^{\gamma} \in \langle \lambda^{\gamma}_{\mathsf{init}} \ldots \lambda^{\gamma}_{\mathsf{fin}} \rangle$. Then

$$R^{\gamma}(\langle INstr^{\gamma}(1)(t+\lambda^{\gamma}-\theta(1)), INstr^{\gamma}(2)(t+\lambda^{\gamma}-\theta(2)), \ldots \rangle, \\ \langle OUTstr^{\gamma}(1)(t+\lambda^{\gamma}+\phi(1)), OUTstr^{\gamma}(2)(t+\lambda^{\gamma}+\phi(2)), \ldots \rangle) \tag{3}$$

holds, where $INstr^{\gamma}(1)$ is the input stream from which the values for the first input parameter of R^{γ} are drawn, $\theta(1)$ is the delay for that input parameter, and $INstr^{\gamma}(1)(t+\lambda^{\gamma}-\theta(1))$ is the actual value drawn from the stream. Similarly for the other input parameters of R^{γ} and for its output parameters. It is assumed that the domains of all the input and output streams are big enough so that all the values referred to in all instantiations of (3) are well defined.

In keeping with our earlier observation that irreducibility is not formally needed, we may erase the references to irreducibility in Def. 3.4. It is easy to see that if we use reducible relations in the description of an acyclic circuit behaviour according to Def. 3.4, then there will be an underlying description using smaller irreducible subrelations which is equivalent to it.

In the rest of the paper we will avoid the very heavy formal machinery of Def. 3.4, which has not only to formalise the behaviours of circuits and their irreducible relations, but in order to do so has also to formalise some of the syntactic machinery used in the first mentioned formalisation (specifically, this is done via the functions $INstr^{\gamma}$ and $OUTstr^{\gamma}$, which connect streams to the syntactic structure of irreducible subrelations). We drop the heavy formal machinery in favour of a lighter notation built by analogy with our preceding examples. Thus we will write an irreducible subrelation as:

$$AcyOpGenS^{\gamma}(\langle i_r(t_{\lambda^{\gamma}} - \theta_{r.1}), i_r(t_{\lambda^{\gamma}} - \theta_{r.2}), i_s(t_{\lambda^{\gamma}} - \theta_s), \ldots \rangle, \langle o_t(t_{\lambda^{\gamma}} + \phi_t), o_u(t_{\lambda^{\gamma}} + \phi_u), \ldots \rangle) \tag{4}$$

In (4), γ is (or represents) a generic element of the partition of the output signals of the circuit of which $AcyOpGenS^{\gamma}$ is one of the irreducible subrelations. Likewise, in keeping with conventions established in our earlier more informal account, we eschew the use of the *INstr* and *OUTstr* functions for doing the substitution of actual streams into parameters of the irreducible subrelation, and we write the substitution directly. A consequence of this is that the various delays (which are formalised via *INstr* and *OUTstr*) must be referred to without mentioning the parameter position information. So the θ and ϕ values in (4) are indexed via the stream rather than position, and because the same input stream can occur more than once in the input signature of $AcyOpGenS^{\gamma}$ (e.g. i_r in (4)), the θ values may need a sub-index to disambiguate the delays belonging to distinct occurrences. This possibility is illustrated in the first two parameters of (4), where the same input stream i_r occurs twice; once with delay $\theta_{r.1}$ and a second time with delay $\theta_{r.2}$.[8]

In this way, we see that in (4), λ^{γ} orients an occurrence of $AcyOpGenS^{\gamma}$ with respect to absolute time (this representation being permitted by the time invariance assumption); $\theta_{r.1}, \theta_{r.2}, \theta_s \ldots$ (all non-negative) parameterise the time

[7] This and the preceding item ensure that each output stream from ST_{out} occurs exactly once among all the outputs of all the irreducible relations $\{R^{\alpha}, R^{\beta} \ldots\}$.

[8] Of course, ϕ values do not need a sub-index because of points 2 and 3 of Def. 3.4.

indexes relative to $t+\lambda^\gamma$ at which input values are extracted from the input streams; and $\phi_t, \phi_u \ldots$ (all non-negative) parameterise the time indexes relative to $t+\lambda^\gamma$ at which output values are injected into the output streams. Each possible $\theta+\phi$ gives a potential data dependency delay within *AcyOpGenS*$^\gamma$.

Returning to the earlier *AcyOpS* example (of (1) and (2)), and in particular to the *AcyOpS*^{12} irreducible subrelation, if we refer to the λ^{12} values in the right part of Fig. 2, we see that there are three input values and two output values, hence six possible delays through the relevant part of the *AcyOp* circuit. These six delays are represented in the circuit in the left part of Fig. 2.

One aspect of the *AcyOpS* example, present in (1) and (2), but absent from Def. 3.4, is the implication in the last line of (1) and (2), insisting that the i_1 stream is single valued. When we are simply formulating the theory, such statements are redundant since streams are by definition functions, therefore single valued. However, in the rest of the paper, we are mainly concerned with backwards reasoning, effectively constructing acyclic circuit behaviours implicitly via the reasoning process, inferring values for streams along the way on the basis of earlier information. When the reasoning process branches into two different subrelation occurrences which may share some streams, incompatible values may be deduced in the separate branches. In this situation, the implications underpin the fact that consistency must be enforced, and it is thus useful to include them in the composition of subrelations.

Since large systems are composed out of smaller ones, we next consider composition mechanisms for the causal relations of synchronous digital circuits.

3.1. Skew-Parallel Composition:

A large circuit made up of two independent smaller circuits working in parallel, is described by asserting both component causal relations. Thus if *AcyOpS*$_1$ and *AcyOpS*$_2$ are as two such relations, with the same signatures as *AcyOpS* for convenience, but with additional subscripts 1 and 2 respectively, then their skew-parallel composition is given by:

$$\begin{aligned}
&\mathit{AcyOpS}_{1|2}(\langle i_{11}(t_{\lambda_1^0}), i_{11}(t_{\lambda_1^{12}}-1), i_{12}(t_{\lambda_1^{12}}-2), i_{12}(t_{\lambda_1^{12}}-3), i_{21}(t_{\lambda_2^0}), i_{21}(t_{\lambda_2^{12}}-1), i_{22}(t_{\lambda_2^{12}}-2), i_{22}(t_{\lambda_2^{12}}-3)\rangle,\\
&\qquad \langle o_{10}(t_{\lambda_1^0}), o_{11}(t_{\lambda_1^{12}}), o_{12}(t_{\lambda_1^{12}}+1), o_{20}(t_{\lambda_2^0}), o_{21}(t_{\lambda_2^{12}}), o_{22}(t_{\lambda_2^{12}}+1)\rangle) \equiv\\
&\mathit{AcyOpS}_1(\langle i_{11}(t_{\lambda_1^0}), i_{11}(t_{\lambda_1^{12}}-1), i_{12}(t_{\lambda_1^{12}}-2), i_{12}(t_{\lambda_1^{12}}-3)\rangle, \langle o_{10}(t_{\lambda_1^0}), o_{11}(t_{\lambda_1^{12}}), o_{12}(t_{\lambda_1^{12}}+1)\rangle) \wedge\\
&\mathit{AcyOpS}_2(\langle i_{21}(t_{\lambda_2^0}), i_{21}(t_{\lambda_2^{12}}-1), i_{22}(t_{\lambda_2^{12}}-2), i_{22}(t_{\lambda_2^{12}}-3)\rangle, \langle o_{20}(t_{\lambda_2^0}), o_{21}(t_{\lambda_2^{12}}), o_{22}(t_{\lambda_2^{12}}+1)\rangle)
\end{aligned} \tag{5}$$

Note that $\lambda_1^0, \lambda_1^{12}, \lambda_2^0, \lambda_2^{12}$, are all independent. That is why (5) is called the *skew*-parallel composition. Observe that the decomposed form of (5) will feature four irreducible subrelations and two constraints, arising as two copies of (1), as expected.

3.2. Skew-Sequential Composition:

Because of the nontrivial role played by the delays, the skew-sequential composition of causal relations is a more interesting proposition. Suppose again that *AcyOpS*$_1$ and *AcyOpS*$_2$ have the signatures of *AcyOpS*, and are both derived from circuits *AcyOp*$_1$ and *AcyOp*$_2$, both having the data dependency structure of *AcyOp*. Suppose we want to compose circuits *AcyOp*$_1$ and *AcyOp*$_2$ sequentially by identifying signals of *AcyOp*$_1$ and *AcyOp*$_2$ as follows:

$$o_{10} = i_{21} \qquad\qquad o_{11} = i_{22} \tag{6}$$

Note that to avoid ambiguity, we need to specify which signals are to be identified —the general case, in which it is not necessary that *all* the output signals of *AcyOp*$_1$ are connected to *all* the input signals of *AcyOp*$_2$— explains why it is called *skew*-sequential composition. Let us write the signal identifications in (6) as δ, giving us *AcyOp*$_{1;\delta 2}$. We suppress the '$_\delta$' subscript when the connecting signals can be understood from the context. In our example, o_{12} becomes an output signal of the composed circuit. Note that if there are *no* identified signals, skew-sequential composition reduces to skew-parallel composition.

The composed causal relation *AcyOpS*$_{1;\delta 2}$ can be calculated by composing the causal relations *AcyOpS*$_1$ and *AcyOpS*$_2$, allowing for the delays appropriately. Fig. 4 gives an idea of what this produces. We will simplify matters a little by just quoting the irreducible subrelations that contribute to the final outputs of the composition. The latter consist of the outputs of *AcyOp*$_2$ together with any outputs of irreducible subrelations of *AcyOp*$_1$ that remain free due to the skew nature of the composition. Obviously, any irreducible subrelations of *AcyOp*$_1$ not involved in producing

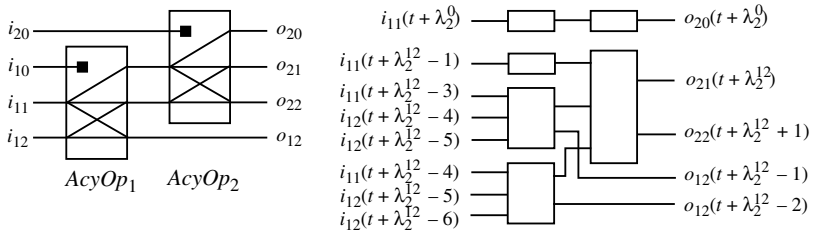

Fig. 4. The skew-sequential composition of circuits $AcyOp_1$ and $AcyOp_2$, and also of causal relations $AcyOpS_1$ and $AcyOpS_2$ via their irreducible subrelations. Thin rectangles represent occurrences of $AcyOpS^0$ while taller rectangles represent occurrences of $AcyOpS^{12}$.

outputs of $AcyOp_2$ will not impose unsatisfiable constraints on the composition, due to the input readiness assumption. In this way we obtain the following:

$$
\begin{aligned}
&AcyOpS_{1;_\delta 2}(\langle i_{11}(t_{\lambda_2^0}), i_{11}(t_{\lambda_2^{12}}-1), i_{11}(t_{\lambda_2^{12}}-3), i_{12}(t_{\lambda_2^{12}}-4), i_{12}(t_{\lambda_2^{12}}-5), i_{11}(t_{\lambda_2^{12}}-4), i_{12}(t_{\lambda_2^{12}}-5), i_{12}(t_{\lambda_2^{12}}-6)\rangle,\\
&\qquad \langle o_{20}(t_{\lambda_2^0}), o_{21}(t_{\lambda_2^{12}}), o_{22}(t_{\lambda_2^{12}}+1), o_{12}(t_{\lambda_2^{12}}-1), o_{12}(t_{\lambda_2^{12}}-2)\rangle) \equiv\\
&(\exists w,x,y,z \bullet\\
&\quad AcyOpS_1^0(i_{11}(t_{\lambda_2^0}), w) \wedge AcyOpS_2^0(w, o_{20}(t_{\lambda_2^0})) \wedge AcyOpS_1^0(i_{11}(t_{\lambda_2^{12}}-1), x) \wedge\\
&\quad AcyOpS_1^{12}(\langle i_{11}(t_{\lambda_2^{12}}-3), i_{12}(t_{\lambda_2^{12}}-4), i_{12}(t_{\lambda_2^{12}}-5)\rangle, \langle y, o_{12}(t_{\lambda_2^{12}}-1)\rangle) \wedge\\
&\quad AcyOpS_1^{12}(\langle i_{11}(t_{\lambda_2^{12}}-4), i_{12}(t_{\lambda_2^{12}}-5), i_{12}(t_{\lambda_2^{12}}-6)\rangle, \langle z, o_{12}(t_{\lambda_2^{12}}-2)\rangle) \wedge\\
&\quad AcyOpS_2^{12}(\langle x,y,z\rangle, \langle o_{21}(t_{\lambda_2^{12}}), o_{22}(t_{\lambda_2^{12}}+1)\rangle) \wedge\\
&\quad (\lambda_2^0 = \lambda_2^{12} - 1 \Rightarrow w = x)) \wedge\\
&(\lambda_2^0 = \lambda_2^{12} - 1 \Rightarrow i_{11}(t_{\lambda_2^0}) = i_{11}(t_{\lambda_2^{12}}-1)) \wedge\\
&(\lambda_2^0 = \lambda_2^{12} - 3 \Rightarrow i_{11}(t_{\lambda_2^0}) = i_{11}(t_{\lambda_2^{12}}-3)) \wedge\\
&(\lambda_2^0 = \lambda_2^{12} - 4 \Rightarrow i_{11}(t_{\lambda_2^0}) = i_{11}(t_{\lambda_2^{12}}-4))
\end{aligned}
\tag{7}
$$

In (7), in terms of the relevant delays, the existentially quantified intermediate variables w,x,y,z, need to take the values:

$$o_{10}(t_{\lambda_2^0}) = w = i_{21}(t_{\lambda_2^0}) \tag{8}$$

$$o_{10}(t_{\lambda_2^{12}}-1) = x = i_{21}(t_{\lambda_2^{12}}-1) \tag{9}$$

$$o_{11}(t_{\lambda_2^{12}}-2) = y = i_{22}(t_{\lambda_2^{12}}-2) \tag{10}$$

$$o_{11}(t_{\lambda_2^{12}}-3) = z = i_{22}(t_{\lambda_2^{12}}-3) \tag{11}$$

The way that (7) is arrived at should be clear from the above. One traces back the dependencies from the desired outputs of the irreducible subrelations of $AcyOpS_2$ to their needed inputs. This can involve various time indexes. One then takes the irreducible subrelations of $AcyOpS_1$ whose outputs are identified with these inputs, and instantiates them with the appropriate delays. In (7), we needed two occurrences each of $AcyOpS_1^0$ and $AcyOpS_1^{12}$, to handle the delays generated from the occurrences of $AcyOpS_2^0$ and $AcyOpS_2^{12}$ in the composition $AcyOpS_{1;_\delta 2}$. We note that as a consequence of the fixed delays within an irreducible subrelation, the free outputs $o_{12}(t_{\lambda_2^{12}}-1)$ and $o_{12}(t_{\lambda_2^{12}}-2)$ of the two occurrences of $AcyOpS_1^{12}$ have their delays constrained by the delays in $AcyOpS_2^{12}$.

Finally one adds as many equalities for occurrences of inputs and outputs of $AcyOpS_1$ as are needed to achieve consistency with hardware behaviour. In (7), this demands conditional equalities for occurrences of i_{11} (since i_{11} is the only input signal which occurred in more than one independent irreducible subrelation of $AcyOpS_1$), and for o_{10} (whose two occurrences, renamed to w and x within the existential quantification, acquired a constraint).

Fig. 4 illustrates the composition just calculated. On the left is the skew-sequential composition of circuits, while on the right, the skew-sequential composition of the causal relations is shown, decomposed into compositions of the

irreducible subrelations of $AcyOpS_1$ and $AcyOpS_2$. Fig. 4 makes it clear that the composition yields a circuit whose causal relation consists of two irreducible subrelations.

4. Retrenchment for Acyclic Synchronous Systems

Now we turn to retrenchment. Retrenchment was designed to capture evolution steps between system models described via general state transition systems. For this purpose we recall that a state transition system is one possessing a set of *states*, and at each moment in time, the system is *in* one of them; i.e. the *current state* is simply a choice of one of the states. For the kind of hardware systems we have in mind, and for the level of abstraction we work at in this paper, the current state will typically be defined by the contents of the system's registers and/or other non-transient components. General references on retrenchment include [BPJS07, BJP08, BJ]; latest results can be found at [Ret].

Suppose we have a(n abstract) system *Abs*, possessing states typically written as u, and operations Op_A described by transitions $Op_A(u,i,u',o)$ which take before-states u to after-states u', consuming inputs i and producing outputs o. Suppose similarly a (concrete) system *Conc*, with states v, and operations $Op_C(v,j,v',p)$ which take before-states v to after-states v', consuming inputs j and producing outputs p.

A retrenchment from *Abs* to *Conc* is specified by a collection of relations, and a collection of proof obligations (POs). Regarding the relations, there is firstly a *retrieve* relation $R(u,v)$, that normally defines once and for all how the state spaces of the *Abs* and *Conc* systems correspond. Beyond $R(u,v)$, it is assumed that particular pairs of operations in *Abs* and *Conc* correspond — we will use operation name identity in the two systems to express this correspondence, viz. Op_A corresponds to Op_C with subscripts A for *Abs* and C for *Conc* used in the obvious way as needed. For every corresponding pair of operations, there are three relations, W_{Op}, O_{Op}, C_{Op}, called the *within*, *output* and *concedes* relations respectively, whose role becomes apparent when we consider the POs of retrenchment.

Firstly among the POs there is the *initialisation* PO:

$$CInit(v') \Rightarrow (\exists u' \bullet AInit(u') \wedge R(u',v')) \tag{12}$$

This says that for each concrete initial state (described by $CInit(v')$) there is an abstract initial state (described by $AInit(u')$) such that the $R(u',v')$ holds between the after-states. Essentially, this 'lines up' the two systems at the beginning of two respective executions that we wish to compare. Secondly there is a *correctness* PO:

$$\begin{aligned} &R(u,v) \wedge W_{Op}(i,j,u,v) \wedge Op_C(v,j,v',p) \Rightarrow \\ &\quad (\exists u', o \bullet Op_A(u,i,u',o) \wedge ((R(u',v') \wedge O_{Op}(o,p;\ u',v',i,j,u,v)) \vee C_{Op}(u',v',o,p;\ i,j,u,v))) \end{aligned} \tag{13}$$

This says that if the retrieve relation $R(u,v)$ and the within relation $W_{Op}(i,j,u,v)$ hold in the before-states and inputs, and there is a concrete transition $Op_C(v,j,v',p)$ issuing from the concrete before-state and input, then there is an abstract after-state and output u',o for which we can find an abstract transition $Op_A(u,i,u',o)$ for which a certain kind of simulation relationship holds. This simulation states that *either* the retrieve relation $R(u',v')$ is re-established and the output relation $O_{Op}(o,p;\ u',v',i,j,u,v)$ holds — in which case the two systems are playing out a fairly standard sort of refinement-like relationship (in this pair of steps), *or*, the concedes relation $C_{Op}(u',v',o,p;\ i,j,u,v)$ holds instead — in which case *something else* is going on in the relationship between the two systems. which we describe using C_{Op}. Typically the concedes relation is used to capture the fact that some kind of exceptional or faulty behaviour is being displayed in the two system steps that are being thus compared. Obviously this makes retrenchment a natural tool with which to investigate all kinds of faulty phenomena; in our work we apply it to fault trees.[9] Observe that the presence of the within relation in the hypotheses of (13) allows the retrenchment relationship to focus on specific parts of the two systems' behaviours, if desired.

The various relations $R, W_{Op}, O_{Op}, C_{Op}$ of the retrenchment, (R specified once and for all for the two state spaces of *Abs* and *Conc*, and W_{Op}, O_{Op}, C_{Op}, featuring the variables that appear above and specified individually for each common operation, as per the notation) are called the retrenchment data of the retrenchment. The retrenchment data, together with the POs generated using them and the two systems *Abs* and *Conc*, specify a retrenchment from *Abs* to *Conc*. Fixing the two systems at issue, *Abs* and *Conc*, retrenchments are parameterised by the choice of retrenchment data, so that each different choice of retrenchment data yields a different retrenchment between them, making retrenchment a highly flexible medium for formally describing how the two systems are related. How one chooses the retrenchment data depends on what one's higher level objectives are. In one sense, the remainder of this paper is

[9] Note that the semicolons in O_{Op} and C_{Op} are purely cosmetic. They separate the variables that occur most often, from the others, which though permitted, are frequently not needed.

concerned with showing how a particular way of choosing retrenchment data proves particularly useful for the higher level objective of constructing fault trees in a mechanical way for clocked digital circuits.

The implicative nature of the correctness PO permits a 'don't care' interpretation when the hypotheses are not true. So it is important to ensure, in particular, that the hypotheses do not inadvertently exclude important cases of system behaviour — typically we have to check that W_{Op} is not too strong for our purposes.[10] Below we will see that our within relations turn out to be maximally permissive, so there is no danger from that angle.

As in PaperI, it is not the PO itself that we need most, but the associated simulation relation itself, the statement that all the conjuncts of both its antecedent and consequent are true:

$$\Sigma^1 \equiv R(u,v) \wedge W_{Op}(i,j,u,v) \wedge Op_A(u,i,u',o) \wedge Op_C(v,j,v',p) \wedge \\ ((R(u',v') \wedge O_{Op}(o,p;\ u',v',i,j,u,v)) \vee C_{Op}(u',v',o,p;\ i,j,u,v)) \tag{14}$$

4.1. Indexed Dataflow Retrenchment

We saw earlier that memory elements in tandem with instantaneous circuits were sufficient for our needs, We next extend and reformulate the retrenchment theory so that, as far as possible, the I/O transformer version will do duty for the timed case too.

4.1.1. The Delay-1 Case:

Since causal relations are equivalent to circuits with relational elements and stateful memory elements, and we base our approach on causal relations, we must eliminate the explicit state from our retrenchments. Fortunately this is not hard to do. It is enough to reduce retrenchments of delay-1 memory elements to the causal relation framework, and to let composition mechanisms do the rest.

To describe the behaviour of memory elements, operations of the following form are sufficient:

$$Op(u,i,u',o) \equiv (i\ X\ u') \wedge (o = u) \tag{15}$$

In (15) X is just a relation that describes how the input values are related to the values that will be output at the next clock tick. For a delay-1 pure memory element, X is just an identity, but (15) obviously allows for more general possibilities too. Given (15), we can take occurrences at two successive time indexes, separate independent variables, and derive a form:

$$Op(i(t),o(t+1)) \equiv i(t)\ X\ o(t+1) \tag{16}$$

As is evident, this connects directly to our time indexed formulation, and eliminates the explicit state from the description. Provided the details of initialisation are taken care of properly, the two formulations are essentially interchangeable.

Suppose we now assume (15) for both the abstract and concrete systems of a retrenchment. We make the following assumptions on the structure of the retrieve, within, output and concedes relations of the retrenchment:

$$R(u,v) \equiv \mathsf{true} \tag{17}$$

$$W_{Op}(i,j,u,v) \equiv i\ W\ j \tag{18}$$

$$O_{Op}(o,p;\ u',v',i,j,u,v) \equiv \langle i,j\rangle\ Y\ \langle u',v'\rangle \tag{19}$$

$$C_{Op}(u',v',o,p;\ i,j,u,v) \equiv \langle i,j\rangle\ Z\ \langle u',v'\rangle \tag{20}$$

where W,Y,Z are suitable relations, completely general for now. Note that the assumption on $R(u,v)$ delegates all responsibility for checking that the abstract and concrete systems are suitably aligned for comparison via the operation PO to the within relation; with the assumption on operations (15) on board, this is innocuous.

Assuming now that the relations W,Y,Z are time invariant, then from (13), we can derive a correctness PO in the form:

$$W_{Op}(i(t),j(t)) \wedge Op_C(j(t),p(t+1)) \Rightarrow \\ (\exists o(t+1) \bullet Op_A(i(t),o(t+1)) \wedge (O_{Op}(o(t+1),p(t+1),i(t),j(t)) \vee C_{Op}(o(t+1),p(t+1),i(t),j(t)))) \tag{21}$$

[10] Note that this is a context dependent statement. It is perfectly reasonable to exclude some important system behaviours from a retrenchment if one is certain that they are covered by some other means.

where:

$$W_{Op}(i(t),j(t)) \equiv i(t)\ W\ j(t) \tag{22}$$

$$O_{Op}(o(t+1),p(t+1),i(t),j(t)) \equiv \langle i(t),j(t)\rangle\ Y\ \langle o(t+1),p(t+1)\rangle \tag{23}$$

$$C_{Op}(o(t+1),p(t+1),i(t),j(t)) \equiv \langle i(t),j(t)\rangle\ Z\ \langle o(t+1),p(t+1)\rangle \tag{24}$$

The corresponding simulation relation becomes:

$$\begin{aligned}\Sigma^1 \equiv\ & W_{Op}(i(t),j(t)) \wedge Op_A(i(t),o(t+1)) \wedge Op_C(j(t),p(t+1)) \wedge \\ & (O_{Op}(o(t+1),p(t+1),i(t),j(t)) \vee C_{Op}(o(t+1),p(t+1),i(t),j(t)))\end{aligned} \tag{25}$$

We call this formulation of retrenchment the indexed dataflow formulation. In this paper, the context (and specifically the signatures of the various relations) will make it clear that we stick with the indexed dataflow formulation throughout.

Note that we have derived the indexed dataflow PO (21) from the conventional one (13). However, we could just as easily have taken the indexed dataflow version as fundamental, and in fact from now on, we will make no further reference to conventional retrenchment.

4.1.2. The General Case:

In the general case, the retrenchment PO and its simulation relation will be statements connecting the causal relations of operations in the abstract and concrete systems. We return to the earlier example circuit *AcyOp* and its causal relation *AcyOpS*. A retrenchment for this will involve abstract and concrete circuits $AcyOp_A, AcyOp_C$, and their abstract and concrete causal relations $AcyOpS_A, AcyOpS_C$. We decompose $AcyOpS_A, AcyOpS_C$, into their irreducible subrelations. The abstract subrelations $AcyOpS_A^0, AcyOpS_A^{12}$, have signatures as in (1), and their concrete counterparts have signatures:

$$AcyOpS_C^0(j_1(t_{\lambda^0}),p_0(t_{\lambda^0})) \tag{26}$$

$$AcyOpS_C^{12}(\langle j_1(t_{\lambda^{12}}-1),j_2(t_{\lambda^{12}}-2),j_2(t_{\lambda^{12}}-3)\rangle,\langle p_1(t_{\lambda^{12}}),p_2(t_{\lambda^{12}}+1)\rangle) \tag{27}$$

$AcyOpS_C^0$ and $AcyOpS_C^{12}$ combine to make $AcyOpS_C$ in the manner of (1). The signatures above make it clear that we are working in a framework in which:

- The number of irreducible subrelations at abstract and concrete levels is the same. (28)
- For each irreducible subrelation, the identity and number of instances of abstract and concrete inputs and outputs correspond. (29)
- For each irreducible subrelation, for each pair of corresponding abstract and concrete inputs or outputs, the delay parameters are the same. (30)

This is not only convenient, but is also needed in Appendix B.

It is clear that (28)-(30) constitute a fairly tight constraint on the relationship between the signatures of abstract and concrete models so we pause to discuss the restrictions we have just introduced. It is of course easy to imagine failure modes that do not conform to these restrictions, but they would require a considerable departure from the retrenchment theory of this paper.

Thus if one imagines differing delays for corresponding operations at the two levels, prevented by (30), and concatenates occurrences of the operation at the two levels, the two levels potentially drift further and further out of sync. What then does a simulation (of the kind used in this paper) between the two levels actually mean? One can address such questions. They require the use of coarse grained retrenchments to cater for the spread in elapsed times at the two levels in a reasonable way, but this would take us far outside the scope of this paper, and is a topic that will be dealt with elsewhere.

Regarding the similarity in signatures at the two levels, addressed in (29), the situation is much more benign. A hardware circuit is hardly likely to spawn a fresh input or output channel during the course of failure for instance. Or, if it did, then it would signal something like the breakdown of the insulating medium surrounding its electronics, a failure mode that would require much more detailed modelling of the physical medium than we can handle with the techniques of this paper.

Regarding the similarity in number of irreducible subrelations though, addressed in (28), the situation is completely benign. We have noted already that while minimality in irreducible subrelations is highly desirable for the sake of

efficiency, the minimality is nonetheless never actually *needed*. Therefore, we are free to amalgamate the intrinsic irreducible subrelations at abstract and concrete levels until they are big enough that they *do* satisfy (28), and to use the resulting subrelations in the sequel.

Thus, while (28)-(30) do pose some restrictions, we do not regard these as severe, and the theory we develop is well suited for the analysis of purely functional failure modes, the ones most suited to automatic analysis.

We now consider a retrenchment from abstract to concrete, focusing on $AcyOpS_A^{12}$ and $AcyOpS_C^{12}$ from our running example. This is expressible in a PO of the form:

$$
\begin{array}{l}
W_{AcyOpS^{12}}(\langle i_1(t_{\lambda^{12}}-1), i_2(t_{\lambda^{12}}-2), i_2(t_{\lambda^{12}}-3)\rangle, \langle j_1(t_{\lambda^{12}}-1), j_2(t_{\lambda^{12}}-2), j_2(t_{\lambda^{12}}-3)\rangle) \wedge \\
AcyOpS_C^{12}(\langle j_1(t_{\lambda^{12}}-1), j_2(t_{\lambda^{12}}-2), j_2(t_{\lambda^{12}}-3)\rangle, \langle p_1(t_{\lambda^{12}}), p_2(t_{\lambda^{12}}+1)\rangle) \Rightarrow \\
\quad (\exists\ o_1(t_{\lambda^{12}}), o_2(t_{\lambda^{12}}+1) \bullet \\
\quad\quad AcyOpS_A^{12}(\langle i_1(t_{\lambda^{12}}-1), i_2(t_{\lambda^{12}}-2), i_2(t_{\lambda^{12}}-3)\rangle, \langle o_1(t_{\lambda^{12}}), o_2(t_{\lambda^{12}}+1)\rangle) \wedge \\
\quad\quad\quad (O_{AcyOpS^{12}}(\langle o_1(t_{\lambda^{12}}), o_2(t_{\lambda^{12}}+1)\rangle, \langle p_1(t_{\lambda^{12}}), p_2(t_{\lambda^{12}}+1)\rangle, \\
\quad\quad\quad\quad\quad \langle i_1(t_{\lambda^{12}}-1), i_2(t_{\lambda^{12}}-2), i_2(t_{\lambda^{12}}-3)\rangle, \langle j_1(t_{\lambda^{12}}-1), j_2(t_{\lambda^{12}}-2), j_2(t_{\lambda^{12}}-3)\rangle) \vee \\
\quad\quad\quad C_{AcyOpS^{12}}(\langle o_1(t_{\lambda^{12}}), o_2(t_{\lambda^{12}}+1)\rangle, \langle p_1(t_{\lambda^{12}}), p_2(t_{\lambda^{12}}+1)\rangle, \\
\quad\quad\quad\quad\quad \langle i_1(t_{\lambda^{12}}-1), i_2(t_{\lambda^{12}}-2), i_2(t_{\lambda^{12}}-3)\rangle, \langle j_1(t_{\lambda^{12}}-1), j_2(t_{\lambda^{12}}-2), j_2(t_{\lambda^{12}}-3)\rangle))))
\end{array}
\tag{31}
$$

In (31) (and in all similar within relations) we will additionally assume that $W_{AcyOpS^{12}}$ decomposes as:

$$
\begin{array}{l}
W_{AcyOpS^{12}}(\langle i_1(t_{\lambda^{12}}-1), i_2(t_{\lambda^{12}}-2), i_2(t_{\lambda^{12}}-3)\rangle, \langle j_1(t_{\lambda^{12}}-1), j_2(t_{\lambda^{12}}-2), j_2(t_{\lambda^{12}}-3)\rangle) \equiv \\
\quad W_{AcyOpS^{12.1}}(i_1(t_{\lambda^{12}}-1), j_1(t_{\lambda^{12}}-1)) \wedge \\
\quad W_{AcyOpS^{12.2}}(i_2(t_{\lambda^{12}}-2), j_2(t_{\lambda^{12}}-2)) \wedge \\
\quad W_{AcyOpS^{12.2}}(i_2(t_{\lambda^{12}}-3), j_2(t_{\lambda^{12}}-3))
\end{array}
\tag{32}
$$

where $W_{AcyOpS^{12.1}}, W_{AcyOpS^{12.2}}$ are within relations for the individual abstract/concrete input value pairs.[11]

The reasons for this run as follows. Firstly, we already mentioned above that our within relations were going to be maximally permissive. (N. B. *Why* we might want to make such a stipulation on the within relations will be explained in Section 5.1.) This means that each variable involved in describing the input should be permitted to take any value (within its type). This immediately leads to a within relation which is a product of subrelations, as in (32). Secondly, when we consider skew-parallel composition below, we want to plug in (perhaps only part of) the outputs of one circuit into (perhaps only part of) the inputs of a successor circuit. Nontrivial dependencies among 'involved' and 'non-involved' inputs in a composition of this kind can create severe theoretical problems. The product form avoids them entirely.

Suppose we now consider the retrenchment PO for the abstract and concrete causal relations $AcyOpS_A$ and $AcyOpS_C$. Then we arrive at a formula of a very similar shape:

$$
\begin{array}{l}
W_{AcyOpS}(\langle i_1(t_{\lambda^0}), i_1(t_{\lambda^{12}}-1), i_2(t_{\lambda^{12}}-2), i_2(t_{\lambda^{12}}-3)\rangle, \langle j_1(t_{\lambda^0}), j_1(t_{\lambda^{12}}-1), j_2(t_{\lambda^{12}}-2), j_2(t_{\lambda^{12}}-3)\rangle) \wedge \\
AcyOpS_C(\langle j_1(t_{\lambda^0}), j_1(t_{\lambda^{12}}-1), j_2(t_{\lambda^{12}}-2), j_2(t_{\lambda^{12}}-3)\rangle, \langle p_0(t_{\lambda^0}), p_1(t_{\lambda^{12}}), p_2(t_{\lambda^{12}}+1)\rangle) \Rightarrow \\
\quad (\exists\ \langle o_0(t_{\lambda^0}), o_1(t_{\lambda^{12}}), o_2(t_{\lambda^{12}}+1)\rangle \bullet \\
\quad\quad AcyOpS_A(\langle i_1(t_{\lambda^0}), i_1(t_{\lambda^{12}}-1), i_2(t_{\lambda^{12}}-2), i_2(t_{\lambda^{12}}-3)\rangle, \langle o_0(t_{\lambda^0}), o_1(t_{\lambda^{12}}), o_2(t_{\lambda^{12}}+1)\rangle) \wedge \\
\quad\quad\quad (O_{AcyOpS}(\langle o_0(t_{\lambda^0}), o_1(t_{\lambda^{12}}), o_2(t_{\lambda^{12}}+1)\rangle, \langle p_0(t_{\lambda^0}), p_1(t_{\lambda^{12}}), p_2(t_{\lambda^{12}}+1)\rangle, \\
\quad\quad\quad\quad\quad \langle i_1(t_{\lambda^0}), i_1(t_{\lambda^{12}}-1), i_2(t_{\lambda^{12}}-2), i_2(t_{\lambda^{12}}-3)\rangle, \langle j_1(t_{\lambda^0}), j_1(t_{\lambda^{12}}-1), j_2(t_{\lambda^{12}}-2), j_2(t_{\lambda^{12}}-3)\rangle) \vee \\
\quad\quad\quad C_{AcyOpS}(\langle o_0(t_{\lambda^0}), o_1(t_{\lambda^{12}}), o_2(t_{\lambda^{12}}+1)\rangle, \langle p_0(t_{\lambda^0}), p_1(t_{\lambda^{12}}), p_2(t_{\lambda^{12}}+1)\rangle,
\end{array}
$$

[11] Note that we have extended the general time invariance assumptions, so that $W_{AcyOpS^{12.1}}$ and $W_{AcyOpS^{12.2}}$ are themselves time invariant, and therefore $W_{AcyOpS^{12.2}}$ is applicable for i_2 and j_2 at both $t-2$ and $t-3$.

$$\langle i_1(t_{\lambda^0}), i_1(t_{\lambda^{12}}-1), i_2(t_{\lambda^{12}}-2), i_2(t_{\lambda^{12}}-3)\rangle, \langle j_1(t_{\lambda^0}), j_1(t_{\lambda^{12}}-1), j_2(t_{\lambda^{12}}-2), j_2(t_{\lambda^{12}}-3)\rangle))) \tag{33}$$

The question now arises as to the relationship between the retrenchment data in (31) and in (33). We return to this as we investigate retrenchment composition.

Of course, the simulation relations corresponding to (31) and (33) are obtained in the usual way, by removing the existential quantification and changing the implication to a conjunction.

4.2. Compositions of Retrenchments

In this section, we extend our results on compositions of causal relations *per se*, to compositions of retrenchment data concerning abstract and concrete versions of them. Not only do these results give a detailed handle on the structure of the retrenchment data when a system is built bottom-up, but also they provide a basis for later, when we develop feedback composition, to deal with systems featuring data dependency cycles.

To minimise the proliferation of cumbersome formulae, we reuse notations established earlier to the maximum. Thus both compositions will be based on our running example *AcyOpS*. To do compositions of retrenchments, we need four versions of *AcyOpS*, namely: $AcyOpS_{1,A}, AcyOpS_{2,A}, AcyOpS_{1,C}, AcyOpS_{2,C}$. Of these, $AcyOpS_{1,A}, AcyOpS_{2,A}$, have the variables that appear in (5) for $AcyOpS_1$ and $AcyOpS_2$ respectively; and $AcyOpS_{1,C}, AcyOpS_{2,C}$, have similar variables, but with all occurrences of input variables *i* replaced by *j*, and all occurrences of output variables *o* replaced by *p*.

We also need retrenchment data. Thus the retrenchment from $AcyOpS_{1,A}$ to $AcyOpS_{1,C}$ will have data: $W_{AcyOpS,1}$, $O_{AcyOpS,1}, C_{AcyOpS,1}$, and the retrenchment from $AcyOpS_{2,A}$ to $AcyOpS_{2,C}$ will have data: $W_{AcyOpS,2}, O_{AcyOpS,2}$, $C_{AcyOpS,2}$. The relation $W_{AcyOpS,1}$ has variables like W_{AcyOpS} in (33), but each variable has an additional '1' first subscript. Relations $O_{AcyOpS,1}$ and $C_{AcyOpS,1}$ also have variables like O_{AcyOpS} and C_{AcyOpS} in (33), but with additional '1' first subscripts. The story for $W_{AcyOpS,2}$, $O_{AcyOpS,2}, C_{AcyOpS,2}$ is similar, except that the additional first subscript is '2' rather than '1'. The additional '1' and '2' subscripts work as they do in (5). The time index parameters of the $AcyOpS_{1,A}$ to $AcyOpS_{1,C}$ retrenchment will be $\lambda_1^0, \lambda_1^{12}$, while those of the $AcyOpS_{2,A}$ to $AcyOpS_{2,C}$ retrenchment are $\lambda_2^0, \lambda_2^{12}$.

Both skew-sequential and skew-parallel composition of retrenchments can easily be shown to be sound, in the sense that if the retrenchment data and operations of each component satisfy the correctness PO individually, then the combined operations satisfy the correctness PO using the combined retrenchment data. We do not cover these soundness arguments here, referring to [BJP08] for related theory, from which the arguments needed here follow by rather trivial modifications (principally the relabelling of variables). The soundness arguments extend readily to the simulation relations: if the simulation relations for the components are true, then the simulation relations for the combinations are true too.

4.2.1. Skew-Parallel Composition:

Given the notations just described, the construction of the skew-parallel composition of two retrenchments, from Abs_1 to Conc_1, and Abs_2 to Conc_2 is straightforward. We use $|$ and $\vee$ for the parallel composition and union of independent relations (which correspond to $\wedge$ and $\vee$ for the predicates which denote those relations). It is convenient enough to treat the operations as a whole, as in (5).

The skew-parallel composition of the operations as a whole (i.e. not just one or other irreducible subrelation) is:

$$AcyOpS_{1|2,A} \equiv AcyOpS_{1,A} \,|\, AcyOpS_{2,A} \tag{34}$$

$$AcyOpS_{1|2,C} \equiv AcyOpS_{1,C} \,|\, AcyOpS_{2,C} \tag{35}$$

In this, $AcyOpS_{1|2,A}$ is just like (5), except that the identifiers $AcyOpS_{1|2,A}, AcyOpS_{1,A}, AcyOpS_{2,A}$ in (34) and (35) contain an extra '*A*' subscript. The story for $AcyOpS_{1|2,C}$ is similar except that the extra subscript is '*C*', and all *i* and *o* variables become *j* and *p* variables respectively. The parameters $\lambda_1^0, \lambda_1^{12}, \lambda_2^0, \lambda_2^{12}$, appearing in $AcyOpS_{1|2,A}, AcyOpS_{1|2,C}$ are all independent.

With the operations defined, the retrenchment data for the skew-parallel composition are given in terms of the component retrenchment data as follows:

$$W_{AcyOpS,1|2} \equiv W_{AcyOpS,1} \,|\, W_{AcyOpS,2} \tag{36}$$

$$O_{AcyOpS,1|2} \equiv O_{AcyOpS,1} \,|\, O_{AcyOpS,2} \tag{37}$$

$$C_{AcyOpS,1|2} \equiv O_{AcyOpS,1} \,|\, C_{AcyOpS,2} \vee C_{AcyOpS,1} \,|\, O_{AcyOpS,2} \vee C_{AcyOpS,1} \,|\, C_{AcyOpS,2} \tag{38}$$

What these formulae say is actually rather simple. The combined within relation is just the combination of the two withins, each acting on its own variables. The combined output relation is similar. The combined concedes relation expresses a disjunction of three possibilities: either the first system is well behaved and establishes the output relation while the second fails and establishes the concession, or *vice versa*, or both fail and establish their concessions.

We can now return to the question of the relationship between the retrenchment data in (31) and in (33). Notice that the assembly of a causal relation from its irreducible subrelations (1) is just like the skew-parallel composition of causal relations (5), except that we have additional constraints in (1) to cope with possibly overlapping sets of input variables.[12] It follows that the skew-parallel composition in (36)-(38) will suffice for generating the retrenchment data for the causal relations from the retrenchment data for the irreducible subrelations, provided we take the extra constraints into account. In fact, since the needed constraints already appear in the causal relations themselves, the composition in (5) can be used verbatim.

4.2.2. Skew-Sequential Composition:

Skew-sequential composition is similar, but with some additional complexity, foreseeable from Section 3.2.

As before we start with two retrenchments, from Abs_1 to Conc_1, and Abs_2 to Conc_2. Suppose that $AcyOpS_{1,A}$, $AcyOpS_{2,A}$,$AcyOpS_{1,C}$,$AcyOpS_{2,C}$ are derived from circuits $AcyOp_{1,A}$, $AcyOp_{2,A}$,$AcyOp_{1,C}$,$AcyOp_{2,C}$, all with three input signals and three output signals, each labelled as in Fig. 2 except with additional first subscripts 1 or 2 and called j and p in the concrete cases. We sequentially compose the circuits by identifying output signals of $AcyOp_{1,A}$ and $AcyOp_{1,C}$ with input signals of $AcyOp_{2,A}$ and $AcyOp_{2,C}$ respectively via an extension of equations (6):

$$\begin{aligned} o_{10} &= i_{21} & \qquad o_{11} &= i_{22} \\ p_{10} &= j_{21} & \qquad p_{11} &= j_{22} \end{aligned} \tag{39}$$

Let us write these signal identifications as δ, and the sequential composition of relations based on these identifications as ${}^{0}_{9\delta}$.

The skew-sequentially composed causal relations that arise are, on the abstract side (7), but where the relation names have an additional '$_A$' subscript, and on the concrete side something similar, but as usual with inputs called j and outputs called p and an additional '$_C$' subscript for the relation names.

We now turn to the composed retrenchment data for the composed operations. Unlike the skew-parallel case, in skew-sequential composition, the two components interact nontrivially via (39), so we can no longer describe the properties of the result using simple logical compositions of the overall component data, but must work at the level of individual variables. This results in some rather unwieldy formulae for the composed retrenchment data, but the apparent complexity is more illusory than real, and the structural principles behind the formulae can be easily enough explained.

The key insight is that the various relations are composed out of two instances of the first component's data and one instance of the second component's data. This is a direct reflection of the same phenomenon for the skew-sequential composition of the operations themselves, discernable from (7) and illustrated in Fig. 4.

We consider first the composed within relation. It turns out that this is by far the most complicated relation in the composed data. It also turns out that in the specific application of this theory to fault tree construction, only a trivial case, given by true is sufficient. Thus, in order to avoid a lengthy detour, we relegate the details of the composed within computation to Appendix A.

Next we turn to the composed output relation. This has a considerably simpler structure than the within relation, the only complication arising from the need for explicit manipulation of the various time delays in the first component, due to various time delays in the second.

$$\begin{aligned} &O_{AcyOpS^{1,1;}{}^{0}_{9\delta}{}^{2}}(\langle o_{20}(t_{\lambda_2^0}), o_{21}(t_{\lambda_2^{12}}), o_{22}(t_{\lambda_2^{12}}+1), o_{12}(t_{\lambda_1^{12}}-1), o_{12}(t_{\lambda_1^{12}}-2)\rangle, \langle \ldots p\mathrm{s} \ldots\rangle, \\ &\quad \langle i_{11}(t_{\lambda_2^0}), i_{11}(t_{\lambda_2^{12}}-1), i_{11}(t_{\lambda_2^{12}}-3), i_{12}(t_{\lambda_2^{12}}-4), i_{12}(t_{\lambda_2^{12}}-5), i_{11}(t_{\lambda_2^{12}}-4), i_{12}(t_{\lambda_2^{12}}-5), i_{12}(t_{\lambda_2^{12}}-6)\rangle, \\ &\quad \langle \ldots j\mathrm{s} \ldots\rangle) \equiv \\ &(\exists w_a, w_c, x_a, y_a, x_c, y_c, z_a, z_c \bullet \end{aligned}$$

[12] N. B. The output variables do not overlap by Def. 3.4.2,3.

$$
\begin{aligned}
&O_{AcyOpS,1}(\langle w_a, y_a, o_{12}(t_{\lambda_2^{12}}-1)\rangle, \langle w_c, y_c, p_{12}(t_{\lambda_2^{12}}-1)\rangle,\\
&\quad \langle i_{11}(t_{\lambda_2^{0}}), i_{11}(t_{\lambda_2^{12}}-3), i_{12}(t_{\lambda_2^{12}}-4), i_{12}(t_{\lambda_2^{12}}-5)\rangle, \langle j_{11}(t_{\lambda_2^{0}}), j_{11}(t_{\lambda_2^{12}}-3), j_{12}(t_{\lambda_2^{12}}-4), j_{12}(t_{\lambda_2^{12}}-5)\rangle) \wedge\\
&O_{AcyOpS,1}(\langle x_a, z_a, o_{12}(t_{\lambda_2^{12}}-2)\rangle, \langle x_c, z_c, p_{12}(t_{\lambda_2^{12}}-2)\rangle,\\
&\quad \langle i_{11}(t_{\lambda_2^{12}}-1), i_{11}(t_{\lambda_2^{12}}-4), i_{12}(t_{\lambda_2^{12}}-5), i_{12}(t_{\lambda_2^{12}}-6)\rangle, \langle j_{11}(t_{\lambda_2^{12}}-1), j_{11}(t_{\lambda_2^{12}}-4), j_{12}(t_{\lambda_2^{12}}-5), j_{12}(t_{\lambda_2^{12}}-6)\rangle) \wedge\\
&O_{AcyOpS,2}(\langle o_{20}(t_{\lambda_2^{0}}), o_{21}(t_{\lambda_2^{12}}), o_{22}(t_{\lambda_2^{12}}+1), o_{12}(t_{\lambda_1^{12}}-1), o_{12}(t_{\lambda_1^{12}}-2)\rangle,\\
&\quad \langle p_{20}(t_{\lambda_2^{0}}), p_{21}(t_{\lambda_2^{12}}), p_{22}(t_{\lambda_2^{12}}+1), p_{12}(t_{\lambda_1^{12}}-1), p_{12}(t_{\lambda_1^{12}}-2)\rangle, \langle w_a, x_a, y_a, z_a\rangle, \langle w_c, x_c, y_c, z_c\rangle)) \qquad (40)
\end{aligned}
$$

In the above, for brevity, $\langle \ldots ps \ldots \rangle$ and $\langle \ldots js \ldots \rangle$ abbreviate obvious $[p/o]$ and $[j/i]$ substitutions in the immediately preceding argument strings. We can write (40) more succinctly as:

$$O_{AcyOpS^1,1;\delta 2} \equiv (O_{AcyOpS,1}(w,y,1) \,|\, O_{AcyOpS,1}(x,z,2)) \,{}^{\circ}_{\delta}\, O_{AcyOpS,2} \qquad (41)$$

In (41), the argument strings $(w,y,1)$ and $(x,z,2)$ are constructed from the relative delays appearing in the first lines of the arguments of the first and second occurrences of $O_{AcyOpS,1}$ in (40) and are thus mnemonics for the more explicit formulae.

Finally we turn to the concedes relation. This we give in the highly abbreviated form of (41) to avoid verbosity. In (42), each O and C term has exactly the same signature as the corresponding O term has in (40) or (41), provided we adhere to the same convention for naming the bound variables.

$$
\begin{aligned}
&C_{AcyOpS^1,1;\delta 2} \equiv\\
&\quad (O_{AcyOpS,1}(w,y,1) \,|\, O_{AcyOpS,1}(x,z,2)) \,{}^{\circ}_{\delta}\, C_{AcyOpS,2} \vee\\
&\quad (O_{AcyOpS,1}(w,y,1) \,|\, C_{AcyOpS,1}(x,z,2)) \,{}^{\circ}_{\delta}\, O_{AcyOpS,2} \vee\\
&\quad (C_{AcyOpS,1}(w,y,1) \,|\, O_{AcyOpS,1}(x,z,2)) \,{}^{\circ}_{\delta}\, O_{AcyOpS,2} \vee\\
&\quad (C_{AcyOpS,1}(w,y,1) \,|\, C_{AcyOpS,1}(x,z,2)) \,{}^{\circ}_{\delta}\, O_{AcyOpS,2} \vee\\
&\quad (C_{AcyOpS,1}(w,y,1) \,|\, O_{AcyOpS,1}(x,z,2)) \,{}^{\circ}_{\delta}\, C_{AcyOpS,2} \vee\\
&\quad (O_{AcyOpS,1}(w,y,1) \,|\, C_{AcyOpS,1}(x,z,2)) \,{}^{\circ}_{\delta}\, C_{AcyOpS,2} \vee\\
&\quad (C_{AcyOpS,1}(w,y,1) \,|\, C_{AcyOpS,1}(x,z,2)) \,{}^{\circ}_{\delta}\, C_{AcyOpS,2} \qquad (42)
\end{aligned}
$$

As pointed out in PaperI for the corresponding instantaneous case, the structure of (40) and (42) conforms to a generic schema for combining retrenchments. Each pair of transitions that makes a retrenchment true can establish either the output or the concedes relation (provided the within relation holds). So when n retrenchments are combined (which eventually yields a conjunction of the facts established by each), a disjunction of 2^n terms arises via the distributive law. The disjunct in which only output relations occur, is deemed to be the output relation of the combination, while the remaining $2^n - 1$ terms are deemed to be the concedes relation of the combination. In (7), if we pair up $AcyOpS_2^0$ and $AcyOpS_2^{12}$ to make $AcyOpS_2$, and pair up occurrences of $AcyOpS_1^0$ and $AcyOpS_1^{12}$ twice to make two occurrences of $AcyOpS_1$, we see that we have a composition of three operations, hence the $n = 3$ case for the corresponding retrenchments. Therefore while (40) predictably has just one term, (42) consists of seven terms. In the skew-parallel composition treated earlier, we had the simpler $n = 2$ case.

4.2.3. Pipelined Circuits, Associativity:

Above, we treated a fairly general example of composition for circuits having varying delays along paths from inputs to outputs. In many practical cases though, acyclic circuits are built up in a structured way from primitive components having delays of at most 1, and having a largely pipelined structure. In such cases one can track the dependencies and delays in the overall circuit much more precisely than can be done in the generic unstructured case. If a circuit is strictly pipelined, i.e. there is a fixed number (say k) of register stages, through which all information flows from inputs to outputs, then the delays along all paths through the circuit are the same (i.e. k). In such a case the signature of the causal relation has the same shape as the circuit itself, i.e. the output variables of *AcyOpS* at time t correspond one-to-one with output signals of *AcyOp*, and input variables of *AcyOpS* at time $t-k$ correspond one-to-one with input signals of *AcyOp*. In this case especially, composition of causal relations follows composition of circuits very closely, and one can do away with the 'skew-' aspects of the theory, leading to correspondingly simpler formulae.

We also point out now that the various compositions of causal relations and retrenchment data that we have considered are associative, both individually, and when working together. This was discussed in PaperI, so we do not repeat

everything here, save to highlight that the truth of it rests primarily on the fact that these compositions are built on compositions of relations, which do enjoy the desired associativity properties.

5. Fault Tree Structure for Acyclic Synchronous Systems

In this section, we briefly recall the route that leads from fault descriptions to retrenchment simulation relations that are amenable to convenient, systematic analysis for fault tree construction, and we then illustrate that process on an example adapted from PaperI, but involving nontrivial timing issues.

5.1. Fault Injection and the Retrenchment Simulation Relation

We design retrenchments to capture fault injection as follows. Let us assume we are dealing with a simple strictly pipelined component called *Op*, having a single irreducible causal subrelation *OpS*, which features a single unit of delay:

$$OpS(i(t_\lambda - 1), o(t_\lambda)) \tag{43}$$

We recall that in the retrenchment, the abstract system OpS_A captures fault-free behaviour, and the concrete system OpS_C includes both fault-free and faulty behaviour. So we design the output relation O_{OpS} of the retrenchment for *OpS* to say that both systems behave ideally, except that the concrete system expresses this using concrete variables:[13]

$$O_{OpS}(o(t_\lambda), p(t_\lambda), i(t_\lambda - 1), j(t_\lambda - 1)) \equiv OpS_A(i(t_\lambda - 1), o(t_\lambda)) \wedge j(t_\lambda - 1) = i(t_\lambda - 1) \wedge p(t_\lambda) = o(t_\lambda) \tag{44}$$

In the concedes relation we state that while the abstract system behaves correctly, the concrete system behaves according to an error relation $Err_{C,OpS}$:

$$C_{OpS}(o(t_\lambda), p(t_\lambda), i(t_\lambda - 1), j(t_\lambda - 1)) \equiv OpS_A(i(t_\lambda - 1), o(t_\lambda)) \wedge Err_{C,OpS}(j(t_\lambda - 1), p(t_\lambda)) \tag{45}$$

The error relation satisfies two conditions. On the one hand, the error transitions are a subset of the concrete ones:

$$Err_{C,OpS}(j(t_\lambda - 1), p(t_\lambda)) \Rightarrow Op_C(j(t_\lambda - 1), p(t_\lambda)) \tag{46}$$

On the other, all concrete transitions which are not ideal transitions (expressed using concrete variables) are error transitions:

$$Op_C(j(t_\lambda - 1), p(t_\lambda)) \wedge \neg OpS_A(j(t_\lambda - 1), p(t_\lambda)) \Rightarrow Err_{C,OpS}(j(t_\lambda - 1), p(t_\lambda)) \tag{47}$$

Finally, the within relation is trivial, i.e. given by true, which we can justify as follows. Seeing that we have both nominal and faulty behaviour contained in the concedes relation, and that $Err_{C,OpS}$ is *a priori* unrestricted, means that, even if we assume that the inputs to *OpS* are the same, there is nothing that we can assume about the relationship of the outputs. If we now imagine a component of this kind *feeding its outputs* into the inputs of our *OpS*, then assuming a within relation any stronger than true risks potentially excluding cases of interest from the remit of the retrenchment via the 'don't care' interpretation of the retrenchment PO's implication. Thus we make the within relation maximally liberal, as pointed out above.

This design for the retrenchment makes the retrenchment's simulation relation (25) decompose into independent abstract and concrete parts, so we can discard the abstract part and concentrate on the concrete one. Adjusting the notation to remove abstract variables, the retrenchment data become:

$$W_{OpS}(j(t_\lambda - 1)) \equiv \mathsf{true} \tag{48}$$

$$O_{OpS}(p(t_\lambda), j(t_\lambda - 1)) \equiv OpS_A(j(t_\lambda - 1), p(t_\lambda)) \tag{49}$$

$$C_{OpS}(p(t_\lambda), j(t_\lambda - 1)) \equiv Err_{C,OpS}(j(t_\lambda - 1), p(t_\lambda)) \tag{50}$$

and the simulation relation absorbs the explicit statement of the transition and within relations, and merely decomposes the $(O \vee C)$ part into its nominal and faulty pieces:

$$\Sigma^1 \equiv O_{OpS}(p(t_\lambda), j(t_\lambda - 1)) \vee C_{OpS}(p(t_\lambda), j(t_\lambda - 1)) \tag{51}$$

[13] In line with PaperI, we assume concrete variables range over the same set of values as their abstract counterparts.

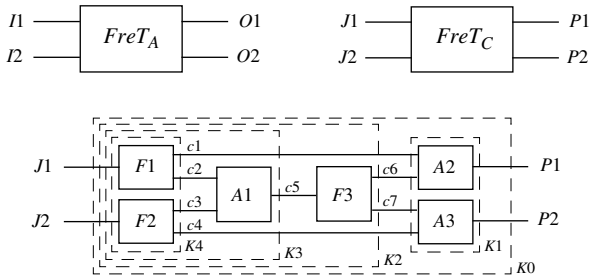

Fig. 5. A subsystem *FreT* and its internal structure.

5.2. An Example

We now consider an example circuit *FreT*, adapted from the example *Fred* of PaperI to include nontrivial timing phenomena. At the top of Fig. 5 we see both abstract and concrete black-box versions of *FreT*, with their input and output signals. Below, we see $FreT_C$ in detail, and our aim is to analyse top level events (TLEs) involving the external I/O of *FreT*, in terms of a fault tree of potential causes attributable to basic faults in the underlying components of *FreT*. In the detailed view of Fig. 5, data flows left to right through components $A1, A2, A3, F1, F2, F3$, interconnected by wires $c1$-$c7$. All signals $J1, J2, P1, P2, c1$-$c7$ are of a fixed finite number of bits.

To save space, for the rest of this section we suppress the λ parameters, since they will all prove to be the same.

Components $A1, A2, A3$ are adders. We assume that the adders do instantaneous cutoff addition without overflow, and, as in PaperI, we will assume that **adders never fail**. Therefore the causal relation $A2S_C$ is the same as $A2S_A$, and similarly for $A1, A3$:

$$A2S_C(\langle c1(t), c6(t)\rangle, P1(t)) \equiv P1(t) = \min(c1(t) + c6(t)), \mathsf{MAX}) \qquad (\mathsf{MAX} > 1) \tag{52}$$

Components $F1, F2, F3$ are delay-1 fanouts. So at a clock tick, they accept an input value, and when they are behaving in a fault-free manner, emit it on both of their outputs on the next tick. Thus the fault-free subrelation of $F3S_C$ is:

$$F3S_C(c5(t-1), \langle c6(t), c7(t)\rangle) \equiv c6(t) = c7(t) = c5(t-1) \tag{53}$$

Our fault *injection* strategy means that the action of faults is governed by fault variables. These are themselves time dependent (to permit the description of e.g. 'glitches'), but there is no *a priori* restriction on their allowed time dependence, so that every fault variable behaviour can be arbitrarily displaced forward or backward in time (within the time interval of interest, and allowing for end effects). In this sense, fault variables, and thus the relational descriptions of all components, are time independent.

Fanouts are assumed capable of failure. Their failure modes are *stuck_at_zero* faults on one or other of their outputs. For fanout $F3$ the fault variables are $F3.c6(t)$ and $F3.c7(t)$, and when true they signify that outputs $c6$ and $c7$ respectively are stuck at zero at time t. Also, **we assume that at most one of the *stuck_at_zero* faults is active at any time for any fanout**. So the full concrete causal relation for fanout $F3$ is given by:

$$\begin{aligned} &F3S_C(c5(t-1), \langle c6(t), c7(t)\rangle) \equiv \\ &\quad (F3.c6(t) \wedge c6(t) = 0 \vee \neg F3.c6(t) \wedge c6(t) = c5(t-1)) \wedge \\ &\quad (F3.c7(t) \wedge c7(t) = 0 \vee \neg F3.c7(t) \wedge c7(t) = c5(t-1)) \wedge \\ &\quad \neg(F3.c6(t) \wedge F3.c7(t)) \end{aligned} \tag{54}$$

and $F1, F2$ are similar. Now the retrenchment data for *FreT* become, firstly for adder $A2$:

$$W_{A2S}(\langle c1(t), c6(t)\rangle) \equiv \mathsf{true} \tag{55}$$

$$O_{A2S}(P1(t), \langle c1(t), c6(t)\rangle) \equiv P1(t) = c1(t) + c6(t) \tag{56}$$

$$C_{A2S}(P1(t), \langle c1(t), c6(t)\rangle) \equiv \mathsf{false} \tag{57}$$

and secondly for fanout $F3$:

$$W_{F3S}(c5(t-1)) \equiv \mathsf{true} \tag{58}$$

$$O_{F3S}(\langle c6(t), c7(t)\rangle, c5(t-1)) \equiv c6(t) = c7(t) = c5(t-1) \tag{59}$$

$$C_{F3S}(\langle c6(t), c7(t)\rangle, c5(t-1)) \equiv (F3.c6(t) \wedge c6(t) = 0 \wedge c7(t) = c5(t-1)) \oplus (F3.c7(t) \wedge c6(t) = c5(t-1) \wedge c7(t) = 0) \quad (60)$$

With these details of the individual components in place, we can use the composition laws presented in Section 4 to compute the causal relation of $FreTS_C$. In particular its signature can be seen to be:

$$FreTS_C(\langle J1(t-1), J1(t-2), J2(t-1), J2(t-2)\rangle, \langle P1(t), P2(t)\rangle) \quad (61)$$

as can be confirmed by tracing the various paths through Fig. 5.

Taking account of the internal details also enables us to deduce that $FreTS_C$ is itself irreducible, since the $\neg(F3.c6(t) \wedge F3.c7(t))$ constraint in $F3$ (and similar constraints for $F1$ and $F2$) properly correlate the $P1$ and $P2$ outputs (by precluding any output pair which can only be derived with both faults for some fanout active). This justifies our earlier presumption that a single λ would be needed in this example, and that we could therefore disregard the λs. Note by contrast that the abstract version, $FreTS_A$, being a functional relation, *is* reducible, splitting into two functions $FreTS_A^0$ and $FreTS_A^1$. So we have a case as discussed around (28)-(30), in which abstract irreducible subrelations have to be amalgamated to conform to the concrete one. The fact that we can perform the whole analysis below using the concrete system alone conceals the details of this. (N. B. These observations also hold for the causal relation of the instantaneous version of this circuit in PaperI, though it is not possible to deduce this just by inspecting the black box picture of the circuit.)

5.3. Fault Analysis for Acyclic Synchronous Systems

Finite acyclic circuits, such as the ones we are dealing with, possess a parsing which builds them up via skew-sequential and (skew-)parallel composition. Of the many possible such parsings, we choose the one in which the elements closest to the inputs are the most deeply nested. Such a structure is convenient for a top-down fault analysis starting at the outputs, and is connected with causal properties of the FT which is ultimately derived, as discussed in PaperI. For $FreT_C$, the structuring we use is illustrated by $K0$-$K4$ in Fig. 5.

We do fault analysis for a subsystem like *FreT* by taking a TLE, and deriving its causes by resolution with the retrenchment simulation relation (51). The parsing of the circuit yields a strategy for decomposing the complex retrenchment data into simpler pieces according to the composition laws, and the same structure for the retrenchment data for *FreT* can be used to guide the assembly of the basic causes of the TLE into a richly structured fault tree.

A TLE for *FreT* is a constraint on the values that the interface variables defining $FreT_C$ can take. We unify the TLE with the simulation relation Σ^1. Since $\Sigma^1_{FreT} \equiv O_{FreT} \vee C_{FreT}$, the behaviour of *FreT* splits into the fault-free behaviour, described by O_{FreT}, and the faulty behaviour described by C_{FreT}. We will assume the former is readily available whenever needed, and similarly for other O terms.

We now illustrate the unification process with a TLE which reads: $J1 = J2 = P1(t) = 1$. This expression, terse to allow compact presentation, specifies two things: firstly that output $P1$ is to be 1 at a specific point in time, t, with $P1$ being otherwise unconstrained — all this being indicated by the presence of t itself in the formula; secondly that inputs $J1, J2$ remain at 1 at all times (in particular, at all times that could possibly influence the value $P1(t)$, bearing in mind causality) — this being indicated by the absence of any time reference in the $J1 = J2$ subexpression. So the TLE could be restated in English as: 'P1 becomes true at a time-step t even though the values of J1 and J2 have remained 1 from start-up'. As in PaperI, our presentation expedites certain details, for clarity of exposition.

TLE^T: To start with, $K0 = K2 \fatsemi K1$, so that $C_{K0} = O_{K2} \fatsemi C_{K1} \vee C_{K2} \fatsemi O_{K1} \vee C_{K2} \fatsemi C_{K1}$. Since $K1$ is nearest to the outputs, we first decompose $K1$. Since $K1 = A2|A3$ and adders don't fail, C_{K1} is false, reducing C_{K0} to $C_{K2} \fatsemi O_{K1}$. Also $O_{K1} = O_{A2}|O_{A3}$. Now O_{A3} merely imposes existential constraints on $P2, c7, c4$, and the TLE does not constrain these values; so we can argue by input readiness that *some* satisfying values inevitably exist without detaining ourselves further. Meanwhile, O_{A2} demands that $c1(t) + c6(t) = 1$ holds. There are two ways to satisfy this: $c1(t) = 0 \wedge c6(t) = 1$ or $c1(t) = 1 \wedge c6(t) = 0$, giving a top level disjunction into **TLE.L**^{T} or **TLE.R**^{T} for $C_{K2} \fatsemi O_{K1}$.

TLE.L^T: Since $c1(t)$ and $c6(t)$ are outputs of $K2$, we next decompose $C_{K2} = C_{K3;F3} = O_{K3} \fatsemi C_{F3} \vee C_{K3} \fatsemi O_{F3} \vee C_{K3} \fatsemi C_{F3}$. Now introducing appropriate timing information for the bottom level components, $C_{F3}(\langle c6(t), c7(t)\rangle, c5(t-1)) = C_{F3,c6}(\langle c6(t), c7(t)\rangle, c5(t-1)) \oplus C_{F3,c7}(\langle c6(t), c7(t)\rangle, c5(t-1))$, we find that $C_{F3,c6}(\langle c6(t), c7(t)\rangle, c5(t-1))$ (which entails $c6$ *stuck_at_zero* at time t) is inconsistent with $c6(t) = 1$. Also $O_{K3}(\langle c1(t-1), c5(t-1), c4(t-1)\rangle, \langle J1(t-2), J2(t-2)\rangle)$ (specifically, the output of adder $A1(t-1)$ with inputs $J1 = J2 = 1$ and correctly working fanouts $F1(J1(t-2), \langle c1(t-1), c2(t-1)\rangle), F2(J2(t-2), \langle c3(t-1), c4(t-1)\rangle)$) forces $c5(t-1) = 2$, also inconsistent

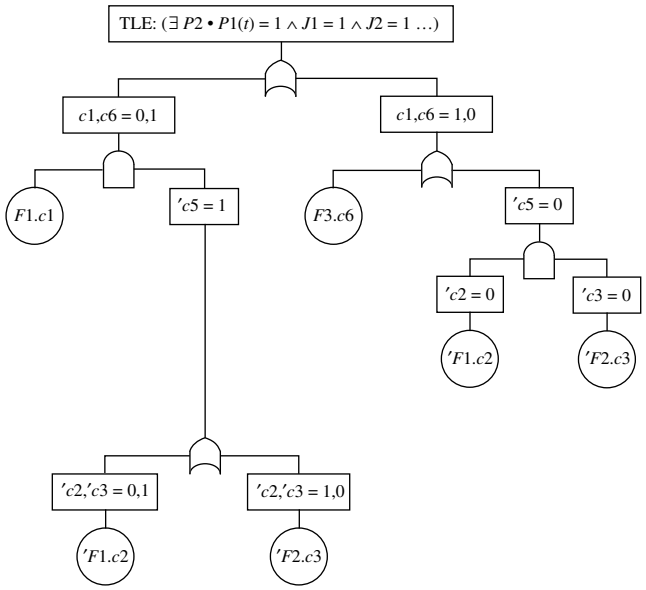

Fig. 6. A Resolution Tree for the TLE of *FreT*, pruning non-minimal cases and enclosing basic faults in round nodes.

with $c6(t) = 1$. So the term containing O_{K3} is dropped and C_{F3} reduces to $C_{F3,c7}$.[14] So $C_{K3;F3}$ reduces to $C_{K3}{}^{\circ}_{9}O_{F3} \vee C_{K3}{}^{\circ}_{9}C_{F3,c7}$. The distinction between these concerns only $c7$, whose precise value is immaterial, so only C_{K3} is of further interest. From $c6(t) = 1$, since fault variable $F3.c6(t)$ is false, we deduce $c5(t-1) = 1$. So we can now decompose $C_{K3} = C_{K4;A1}$, which reduces to just $C_{K4}{}^{\circ}_{9}O_{A1}$ since adders don't fail. Now having $c5(t-1) = 1$ as adder output, implies $c2(t-1) = 0 \wedge c3(t-1) = 1$ or $c2(t-1) = 1 \wedge c3(t-1) = 0$, giving a disjunction into $\mathbf{TLE.L.L}^T$ or $\mathbf{TLE.L.R}^T$ for $C_{K4}{}^{\circ}_{9}O_{A1}$.

TLE.L.L^T: Since $K4 = F1|F2$, we have $C_{K4} = O_{F1}|C_{F2} \vee C_{F1}|O_{F2} \vee C_{F1}|C_{F2}$. In the timed world, this yields a separate fact for each time index. In particular it implies the following two statements:

$$\begin{aligned}
&C_{K4}(\langle J1(t-1), J2(t-1)\rangle, \langle c1(t), c2(t), c3(t), c4(t)\rangle) = \\
&\quad O_{F1}(J1(t-1), \langle c1(t), c2(t)\rangle) \mid C_{F2}(J2(t-1), \langle c3(t), c4(t)\rangle) \vee \\
&\quad C_{F1}(J1(t-1), \langle c1(t), c2(t)\rangle) \mid O_{F2}(J2(t-1), \langle c3(t), c4(t)\rangle) \vee \\
&\quad C_{F1}(J1(t-1), \langle c1(t), c2(t)\rangle) \mid C_{F2}(J2(t-1), \langle c3(t), c4(t)\rangle)
\end{aligned} \tag{62}$$

and

$$\begin{aligned}
&C_{K4}(\langle J1(t-2), J2(t-2)\rangle, \langle c1(t-1), c2(t-1), c3(t-1), c4(t-1)\rangle) = \\
&\quad O_{F1}(J1(t-2), \langle c1(t-1), c2(t-1)\rangle) \mid C_{F2}(J2(t-2), \langle c3(t-1), c4(t-1)\rangle) \vee \\
&\quad C_{F1}(J1(t-2), \langle c1(t-1), c2(t-1)\rangle) \mid O_{F2}(J2(t-2), \langle c3(t-1), c4(t-1)\rangle) \vee \\
&\quad C_{F1}(J1(t-2), \langle c1(t-1), c2(t-1)\rangle) \mid C_{F2}(J2(t-2), \langle c3(t-1), c4(t-1)\rangle)
\end{aligned} \tag{63}$$

Also each occurrence of C_{F1}, C_{F2} is itself an exclusive or of two faults. Earlier, we derived $c1(t) = 0$, which with $J1(t-1) = 1$, implies that $C_{K4}(\langle J1(t-1), J2(t-1)\rangle, \langle c1(t), c2(t), c3(t), c4(t)\rangle)$ must be satisfied by means of the term $C_{F1,c1}(J1(t-1), \langle c1(t), c2(t)\rangle)|O_{F2}$ (O_{F2} rather than C_{F2}, for minimality). We also derived earlier that (for this branch) $c2(t-1) = 0$ holds, which with $J1(t-2) = 1$, implies that $C_{K4}(\langle J1(t-2), J2(t-2)\rangle, \langle c1(t-1), c2(t-1), c3(t-1), c4(t-1)\rangle)$ must be satisfied via the term $C_{F1}(J1(t-2), \langle c1(t-1), c2(t-1)\rangle) \mid O_{F2}$ (O_{F2} for minimality again). This yields a valid cause of the TLE.

etc. etc.

The analysis can be continued in this manner until all the branches of the resolution tree have been explored. The

[14] Note how the precomputation of the O terms is more complicated in the presence of time than in the instantaneous world of PaperI.

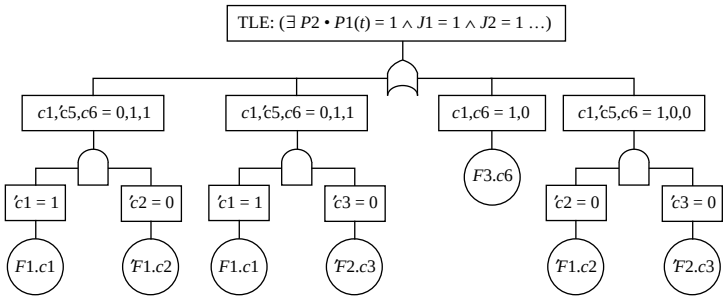

Fig. 7. A Minimised Fault Tree for the TLE of *FreT*.

procedure is very similar to that pursued for the instantaneous case in PaperI, except that the timing information must be taken into account at each stage, as it was above. As for the instantaneous case in PaperI, pursuing this process exhaustively, generates a tree with a good deal of redundancy in the sense of including many non-minimal causes of the TLE. An illustration of the resulting resolution tree can be seen in Fig. 6, where preprimed variables denote values at $t-1$, and where, for space reasons, we have pruned structure from the tree that would correspond to the non-minimal causes (in any case, this closely resembles similar structure in Fig. 6 of PaperI); we also enclosed basic faults in round nodes. A little extra cosmetic work turns Fig. 6 into a *bona fide* fault tree; in essence extra (intermediate) events have to be introduced into the tree so that basic fault nodes are isolated from logical connectives by an (intermediate) event node. This again follows the prescription in PaperI, and the additional structure can be discerned in Fig. 8.

In all these trees, for space reasons again, the labelling of the intermediate events is incomplete compared to the information about the various different current states that a genuine analysis tool would have to maintain. How much of the complete state information could or should be depicted in the pictorial output of such a tool is to some extent a matter of taste. This is further complicated by the fact that FTs according to [VSD$^+$02] must conform to structure and notions of causality that do not always emerge immediately in our resolution tree analysis.

For one example, aspects of structure, which require post-processing as discussed in PaperI, can result in the introduction of some redundancy, e.g. the introduced duplications of $'c2 = 0$ and $'c3 = 0$ in the lower branch of Fig. 8. For another, the top-down approach can give rise to issues of causality compared with a bottom-up one, in which facts can appear in the tree that are not 'caused' by their descendants, but are merely needed by virtue of the top-down resolution process, e.g. the $c1 = 1$ in the right branch of Figs. 6 and 8. Events such as these could be handled by using external (house) events, or alternatively, by introducing a further post-processing phase in which they could be eliminated in a bottom-up sweep through the tree, that would remove any fact that did not have dataflow from some descendant fact.

In general, fuller representation lends a greater degree of completeness to the resulting tree, at risk of more rapidly over-cluttering the output. Sparser representation lends a greater degree of economy (and potentially clarity), at risk of blurring cause-and-effect issues. Perhaps a good compromise would be a hyper-document, sparser on the surface, but with links at relevant nodes to fuller information should the reader desire it.

Taking the transformation of Fig. 6 into a suitable FT for granted, in Fig. 7 we transform the (now minimised) tree further into disjunctive normal form (corresponding to what would be output by the FSAP tool), in order to display most clearly a state of affairs having no counterpart in the corresponding minimised instantaneous FT in PaperI (i.e. Fig. 14 of PaperI). The leftmost disjunct of Fig. 7 shows the TLE being caused by two basic faults of $F1$ which are active at successive time indexes, but not simultaneously. This gets round the at-most-one-fault-active-at-a-time restriction on fanout failures, and gives rise to a glitch in the circuit at t. (The corresponding case in the instantaneous framework of PaperI was cut off there (Fig. 7 of PaperI) due to the absence of dynamics, leading ultimately to an instantaneous FT with fewer causes.) The example vividly illustrates the increased expressive power gained by adding timing to techniques which are essentially the same as used in the instantaneous case.

Of course if we asked for the glitch to persist for two time ticks, this branch would get cut off once more, since it would demand the forbidden simultaneity. We discuss this scenario in more detail in the next section.

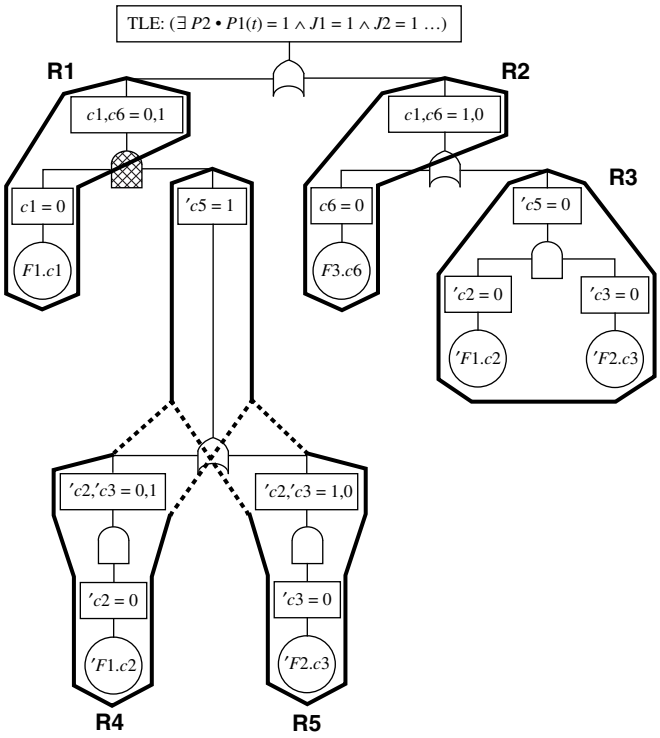

Fig. 8. A Fault Tree for the TLE of *FreT* indicating regions.

5.4. Regions and Durable TLEs

In Fig. 7 we simply amalgamated facts about (only) the minimal cases and then branched the tree in such a way as to derive a compact DNF form. Certain issues however, require a more state-aware analysis, and for this, we need to backtrack a little. In Fig. 8, the fruition of Fig. 6 into FT form, although we still restrict to those cases that turn out to be minimal in the end, the individual facts (or sets of facts) encountered as the analysis proceeds are each held in their own node. In particular, each node holds a fact or facts about a single time index only. Inspecting the tree, we see that while consistent sets of nodes (i.e. sets connected via conjunctions alone) are accumulating facts about a given time index, they are accumulating information about a specific occurrence of some particular system state. On the other hand, when consistent sets of nodes refer to different time indexes, they must be referring to different occurrences of states. So the nodes of Fig. 8 condense into *regions*, each region containing a maximal conjunctively connected set of facts, about some specific state or family of states (as defined by the facts that the region contains about that time index). Fig. 8 shows the regions of the more detailed FT in our running example (we have avoided enclosing the TLE node in any region, for clarity). We see that regions can overlap due to the precise placement of disjunctive nodes in the structure of the FT.

If two regions are connected by an AND node, then both have to be taken into account in deriving a cut set, due to the semantics of AND nodes in the FT (in Fig. 8 there is one such AND node, shown cross-hatched).

Disjoint regions can also refer to the same time index, due to details of the generated FT structure. Two such regions may be consistent or they may be inconsistent. While the former poses no problems, the latter requires some attention. If the path between two inconsistent regions contains only conjunctions, then the subtree containing both of them, up to the nearest ancestral disjunction, must be truncated. If the path between them contains at least one disjunction, the regions represent alternative valid states for that time index, and it is the ‘inclusiveness’ of the OR that needs to be eliminated.

Let us call a TLE that constrains the outputs at more than one time index, a durable TLE (DTLE) for brevity. Asking the glitch in the previous example to persist for two time indexes would be a good example, as the relevant TLE would contain $P1(t) = 1 \wedge P1(t+1) = 1$.

There are a number of approaches to the analysis of a DTLE. In one approach, one can simply treat all the output

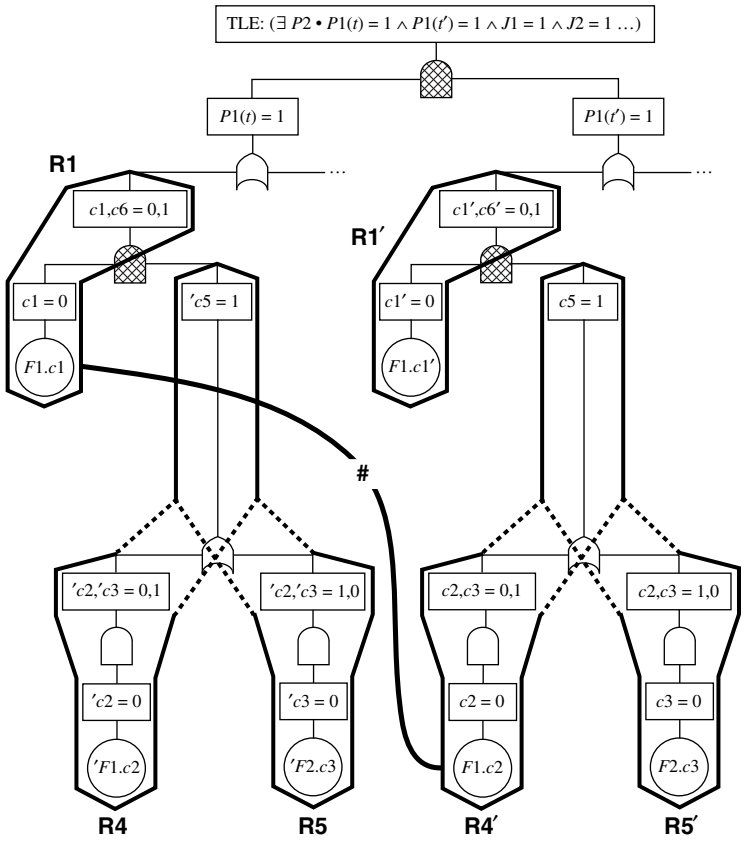

Fig. 9. Regions in a minimised Fault Tree for a TLE of *FreT* constraining two output time indexes, and containing a conflict between two disjoint regions referring to the same time index.

variables in the DTLE on the same footing, attributing no special treatment to occurrences of the same variable at different times — they are just distinct variables. This is the standard approach. In the case of our two step glitch specifically, this would simply lead to a fourfold disjunction immediately below the TLE node in the equivalent of Fig. 8. More generally, it would lead to a d^k-fold disjunction if the DTLE (conjunctively) mentioned k time indexes, each of which individually led to a d-fold disjunction. For further comments on this aspect, see Section 5.7.

An alternative approach treats time in a special way. The first step decomposes the output constraints into the various top level cases: $P1(t) = 1$ and $P1(t+1) = 1$ in the example. From there the analysis can proceed as previously down the two branches. Fig. 9 shows what happens for the glitch once the analysis has progressed far enough.[15] The regions **R.1** and **R.4′** each demand a different fault in $F1$ at time t. The usual constraint says that this cannot happen, so **R.1** and **R.4′** are inconsistent regions for time t. In Fig. 9 a # conflict link is shown between **R.1** and **R.4′**, to indicate that both cannot be true simultaneously in an acceptable cause of the DTLE.

One can adopt such conflict links as enhancements to the structure of FTs, to enable large examples to be represented more compactly. They can be eliminated, if necessary, by using the distributive law to lift all disjunctions on the path between **R.1** and **R.4′** until the path between them is disjunction-free, at which point the relevant subtree can be truncated. Of course the price to pay for this is the exponential blowup noted in the previous approach. At any rate, the more compact form is always beneficial for the internal processing of a tool, to help cope with large examples.

The fact that we could use binary conflict links # (exclusively) is a consequence of the mechanical reasoning system used for deriving the FT. Implicitly, we assume that everything reduces to enumerating the possible satisfying assignments, so that inconsistency always reduces to the assertion of a fact and of its negation. For further comments on this aspect, see Section 5.7 again.

[15] In Fig. 9, postprimed variables denote values at $t+1$, just as preprimed variables denote values at $t-1$.

5.5. Initial States and Cold Start Failures

In the instantaneous scenario of PaperI initial states played no role, since we were only concerned with consistency at a single instant. However, the transition systems that underlie our analysis of hardware circuits normally have sets of initial states, which we have thus far ignored. So our analysis has been aimed at *hot running failures*, i.e. failures that occur once the transients due to initialisation are in the distant past, enabling us to pursue the causal predecessors of individual system states as far back into the past as system structure warrants.

If we are close to system startup, then this pursuit into the past can be interrupted by encountering an initial state. In such a case, failures arising prior to that state in the hot running scenario, simply do not occur. As regards the FT (which probes more and more deeply into the past as it grows layer by layer), one or more of its subtrees has been truncated.

On a technical level, the preceding discussions allow us to organise the hot running and cold start analyses into a coherent whole by exploiting the fact that t remains a free parameter in the TLE. We assume that all TLE specifications refer to only a finite number of time indexes for outputs (corresponding to the idea that the TLE indeed describes an *event*), and that t refers to the earliest of them. We also assume that $t = 0$ is the time index of the initial state. Then we can set up a top level disjunction in which the TLE is satisfied either from $t = 0$ or from $t = 1$ or ... or from $t = \mathsf{ml} + 1$, where ml is the maximum length of any path through the circuit; let us call these the top level instantiations (TLIs). Then we proceed with the usual analysis down all the TLIs. Whenever we hit a region indexed by $t = 0$, we must, in addition, unify the constraints identified for $t = 0$ with the specification of the initial state(s). If the unification succeeds, we continue the analysis for that TLI (although we have no need to explore earlier than $t = 0$ of course). If it fails, we discard up to the nearest ancestral disjunction, including up to the whole TLI if need be. The $t = \mathsf{ml} + 1$ disjunct evidently corresponds to hot running since it never encounters $t = 0$.

Of course one would never do such an analysis, with its implied profligate duplication of work, exactly as described. One would do the hot running analysis (for a general t), and then check which of the regions constructed unified with the initial state(s). A successful unification would indicate a satisfiable TLI, which could then be created by relabeling a copy of the hot running tree, and truncating every subtree below a $t = 0$ region. In the case of our example, if we had an initialisation specified by $c2 = 0 \wedge c3 = 0 \wedge c5 = 0$, then referring to Fig. 8, region **R.3** would be eliminated as such, being replaced by a disjunct indicating that the TLE was satisfiable without the need for any fault variable to be set to true (which would be done using a GOAS node, in the terminology of PaperI, Section 5). Furthermore, regions **R.4** and **R.5**, which refer to the same time index as **R.3**, demand that $c5 = 1$ and that one of $c2 = 1$ or $c3 = 1$ holds, which is inconsistent with the initial state definition; therefore **R.4** and **R.5** would be eliminated, and **R.1** along with them. The FT (appropriately relabeled) would reduce to the disjunction of **R.2** and the GOAS node.

From a reliability engineering perspective, the hot running and cold start branches would be organized into an overall tree in which all the branches were mutually consistent. Branches that refer to cold start analyses would be qualified by using, e.g, inhibit gates and conditioning events, as explained in Section 2. (As an example, a conditioning event could be attached to the GOAS node previously mentioned to indicate the corresponding enabling initialization condition — alternatively, GOAS nodes might be modeled using external (house) events, as described in PaperI.) Evidently, transforming the fault tree along these lines in the most effective way requires some human expertise, and ultimately would rely on safety engineers having a good comprehension, not only of the system at hand, but also of the information provided by the automatic analysis.

Note that the initial-aware reinterpretation of the hot running FT can be done *a posteriori*, i.e. after the hot running FT has been generated. In particular, if it is desired to do minimisation (which will normally be the case due to the large number of non-minimal cases that are typically present), it can be done after the various minimisation tactics described in PaperI Sections 5 and 6 have been applied. The reason for this is that these tactics respect causality. We confirm this now, rule by rule.

Rule **M.1** recommends discarding subtrees which are not needed by the TLE. If some collection of facts does not affect the TLE at some point q in the past, then it will not affect the TLE at all moments subsequent to q, so truncating the FT *above* the point where this is detected (i.e. above q) is safe. Rule **M.2** recommends discarding subtrees at input-insensitive faults. Again, if some input does not affect the TLE at some point q in the past, then it will not affect the TLE at all moments after q, and the truncation is safe. Rule **M.3** recommends discarding locally subsumed expressions. Again the argument is temporal. If one can be sure *now* that something is redundant, then it will be redundant later too. And rule **M.4** recommends final subsumption checking at subsystem rather than full system level. Obviously final subsumption checking cannot interfere with operational details of any implementation prior to its invocation, so is safe here.

Furthermore in PaperI Section 7, Prop. 7.4 recommends two optimisations. The first one, relatively obvious, is that if the TLE is satisfiable with an empty cut set, then no fault configurations at all need be considered. The limiting case

of this in the present context is when the TLE is itself unifiable with an initial state (which we naturally presume to be fault-free). In other words O_S is satisfiable in an initial state, and is also unifiable with the TLE. The second one says that if one has a composition of two subsystems, in which some self-contained fault of one of them gives rise to a given TLE, then one need not consider fault configurations that include both components of the composition, since any cut set of the composition is bound to be non-minimal. This too is safe, for if one of the self-contained faults undergoes some truncation, so will its counterpart in the composition.

Note that all of the above depends on the FT respecting temporal order along its paths, so that later facts occur higher up the FT than earlier ones. This in turn can be achieved using the inputs-innermost parsing strategy discussed in PaperI. (The sole exception to this arises for non-needed facts, which can come to light *after* facts that occur later (because they are not needed), and they can thus get placed deeper in the FT; this itself allows for more convenient truncation.)

The preceding discussion allows us to deal with intialisation transients, *aka* cold start failures, but leaves one last question connected with initial states unanswered, namely whether in the hot running scenario, any particular cause of the TLE identified by our analysis is actually reachable from an initial state. This is equivalent to asking whether the earliest state encountered in deriving some cut set, is itself reachable from an initial state. Since retrenchment is based on simulating *forwards*, retrenchment *by itself* does not answer this question. The solution lies with symbolic model checking, and any practical implementation of our techniques will contain a supplementary check to confirm that any TLE cause derived, can indeed be reached from an initial state (and if not, then the relevant subtree can be truncated). We will discuss the relationship between model checking and the retrenchment based techniques in more detail elsewhere.

5.6. Formal Acyclic Synchronous Fault Analysis

The informal account of the preceding sections can be given a more precise treatment along the lines of Sections 5 and 7 of PaperI. We do not repeat these details here for one principal reason, namely that *provided* we regard the timing information as *part of the name* of a variable, we can apply the previous theory more or less verbatim.

'More or less' recognises that there are some differences of detail. To start with, in order to reuse the previous theory to the greatest degree, the structural description of a subsystem is no longer based on a syntactic decomposition of its *circuit diagram*, but must now be based on a syntactic decomposition of its *causal relation* — causal relations were *conceived* with the intention that timing details could be absorbed into the names of variables.

Earlier, we explored composition issues for causal relations and we saw that the theory was a relatively straightforward adaptation of the composition theory for conventional relations. The main difference that arises is the presence of implications, such as those in (2) and (7), that ensure that whenever the same data stream is used by two different (occurrences of) irreducible subrelations in a composition, if the time indexes of the two uses coincide, then the two uses must utilise the same data value. On the other hand, the remarks following (30) allow us to aggregate irreducible subrelations. So we can use such aggregations to arrive at a formulation in which whenever more than one irreducible subrelation shares the same data stream, the two are aggregated, and thereby the need for the relevant implication is eliminated.

Possible multiple uses of data streams within an irreducible subrelation formulation evidently complicates the recording of the data assignments that a genuine implementation of a resolution tree algorithm makes as it progresses. The reason is that, before an assignment can be committed to, a check has to be performed to ensure that it is not inconsistent with an already present assignment to the same variable for the same time index made in a different branch of the algorithm. A (sole) exception to the latter is when the two assignments occur in different OR branches, in which case the 'inclusive' interpretation of the OR must be eliminated. While this can impact genuine implementations, for the abstract algorithm of Sections 5 and 7 of PaperI, there is hardly any change, since vigilance over these details can be absorbed within the angelic nondeterminism that is used, and in an enriched notion of the '*m-root-realisable*' concept that appears in the $Expand(n)$ function.

In fact, if we assume that all fanout-like behaviour is captured in fanout components of circuits (i.e. no 'shared wires', as formalised in PaperI, Section 5, Def. 5.1.3.(ii)), and that no circuit component has its functionality split across more than one irreducible subrelation (as could happen for the fanouts in *FreT* above if the faults on the two outputs were not constrained to occur 'at most one at a time'), then the feared sharing of data streams cannot arise anyway.

If we have DTLEs, then the above observations extend into the time dimension. In reality, the behaviour of circuit elements covers *all* time indexes, which necessitates a composition of causal relations for all of them. Restricting DTLEs to constrain only a *finite* number of output time indexes, restricts this composition to a finite number of

copies. The composition is then similar to that in say (1), except that the *same* causal (irreducible sub)relation(s) is/are being composed together several times, at different (output) time indexes. Since, in general, the same input signal can contribute several input values to a given causal (irreducible sub)relation via several input time indexes (a consequence of there being paths of different lengths between the input and output signals), the families of input values for different copies in the composition may overlap, necessitating the kind of constraining implications present in (1). This done, the retrenchment compositions needed, will follow (by now) familiar lines, and the resulting formal theory is identical in character to what we have already seen.

Additionally, using causal relations rather than circuit diagrams does not alter the set of *paths* between input and output signals through a subsystem. Since fault tree analysis (in the sense of this paper) can be seen as a deductive process about the paths and sets of paths through a circuit, the parallels with the instantaneous case are unsurprising.

The DTLE example of the preceding section did not follow this route directly, but used a more pragmatic variation (as any implementation is free to do), the net result being equivalent. In reality every implementation would have to deviate from the formal ideal to some extent, since the angelic nondeterminism used in the formal theory is not a directly implementable algorithmic primitive.

Beyond the above, the only interesting extension of the preceding results concerns Prop. 5.25 of PaperI, on causality, which says that fault trees can be arranged so that more deeply nested nodes are dataflow precursors of more shallowly nested nodes. In the timed context, it may be the case that different path delays between the same two components, allied with the system parse structure, may result in a later timed fact being placed lower in the tree than an earlier timed one. In such a situation it is *always* the case that the two facts are independent, since the retrenchment approach is based on following causal chains (backwards), and a later fact can (obviously) never *cause* an earlier one. Therefore, in such a situation it is always possible to migrate the later timed fact higher up the resolution tree, until none of its ancestors predate it. Fig. 8 surreptitiously features an example of this, since the $F1.c1$ basic fault appears just below the derivation of fact $c1 = 0$, whereas its placement according to the parse structure of Fig. 5 would demand that it appears in AND subtrees paired with the $'F1.c2$ and $'F2.c3$ basic faults respectively.

5.7. Efficiency, Large Time Intervals and Symbolic Processing

All the preceding discussion was based on the explicit enumeration of the states pertinent to the analysis. The use of causal relations (rather than circuit diagrams) does not alter the fact that our systems are all still acyclic, and are still finite as regards the number of components, the number of I/O signals, the cardinality of any data type and the number of states. These properties enable us to derive easy *a priori* static bounds on the maximum delay through an acyclic circuit and hence statically bound the number of variables needed for the analysis.

Being able to statically bound the number of variables needed is one thing. Actually instantiating them all in an analysis tool is another. If the number of variables grows huge (e.g. if the TLE lasts for 1000 ticks), then a direct implementation can become prohibitive. This is a matter of efficiency, not of principle. The way round this obstacle is to use a more sophisticated representation internally for the needed data, and contemporary tools typical use symbolic processing for this.

An expression like $t \geq 3 \Rightarrow P(t) \leq 5$ refers to an infinity of individual values of timestamped P variables, despite needing only ten (nonspace) symbols. What can be achieved by the mechanical manipulation of such expressions is the concern of automated reasoning, a detailed discussion of which is outside the scope of our work. Suffice it to say that the state of the art in this field is such that the majority of sensible engineering concerns can be expressed and reasoned about in an effective way. This leads to the possibility of tools which can attack a much wider range of problems than can be tackled using naive enumeration. For example, DTLEs of long duration (e.g. the 1000 ticks mentioned above), or TLEs concerned with moded operation —requiring that an appreciable time period has elapsed since startup before a state of affairs becomes 'of interest', or that one thing happens for a while and then at some later time another thing happens— can all be tackled in principle.[16]

We discuss the issues related to a symbolic implementation of the retrenchment-based approach further in Section 6.5, for the more general case of circuits with feedback. In that context, we also compare our approach with what is currently available in FSAP [BV07], and we indicate ways forward to incorporate timing information in fault trees and practically realize some of the features of the retrenchment-based approach.

[16] N. B. The precise choice of symbolic reasoning framework can have tangible consequences. For example, consider a nonempty triangle in the plane, which can be viewed as the intersection of three half-planes A, B, C. The intersection of their *complements* $\overline{A}, \overline{B}, \overline{C}$ is empty (in logical terms unsatisfiable), even though any pair intersects in a nonempty cone. So a reasoner directly working with symbolic representations of half-planes would need ternary conflict, as well as the binary conflicts that were sufficient for our earlier example.

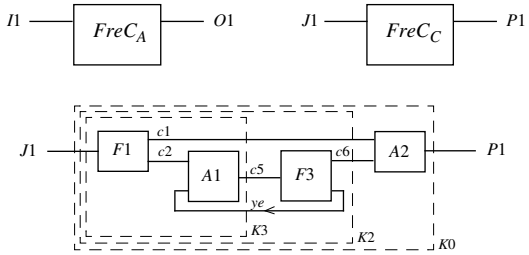

Fig. 10. A subsystem *FreC* with cyclic internal structure.

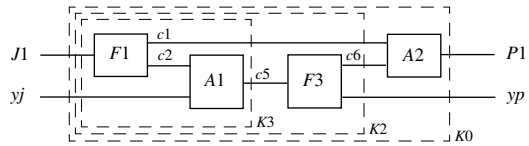

Fig. 11. The circuit for $AFreC_C$.

6. Fault Tree Structure for Synchronous Feedback Systems

The introduction of synchronous timing for finite acyclic circuits provoked, as it turned out, a relatively mild departure from the instantaneous case as far as fault tree construction was concerned. It was enough to incorporate the time indexes within the names of system variables, and to use the casual relations of the circuit (or their irreducible subrelations) instead of the instantaneous description. In this section we allow paths through the circuit to feed back, creating cycles, so that the same circuit element can be visited more than once along the same path during a backwards fault analysis. The formal underpinnings of the fault analysis for such cyclic structures rely on the more complex cyclic system structure and retrenchment theory given in Appendices B and C. Fortunately, for the algorithmic approach that makes the abstract mathematical ideas practical, there is not a great deal of change from the preceding, save for the need to be sensitive to potentially non-terminating behaviour in the analysis. We illustrate this in the examples that follow.

6.1. A Simple Fault-Free Cyclic Example

We start with a simple scenario. We modify our subsystem *FreT* by removing $A3$ and $F2$, and introducing a feedback signal (called *yd* in the abstract system and *ye* in the concrete system) from $F3$ to $A1$, resulting in subsystem *FreC*. See Fig. 10, which again shows the concrete subsystem in detail.

In the presence of feedback, we can no longer rely on a static syntactic description of the system for the analysis, but must unfold a recursive structure.

For the sake of a simple example, let us for the moment examine the behaviour of the ideal, abstract, version of *FreC*, i.e. $FreC_A$, in which the internal signals are respectively named $a1, a2, a5, a6, yd$ rather than the concrete $c1, c2, c5, c6, ye$. Noting as before, that fanouts $F1, F3$ each introduce a unit delay, and that adders work instantaneously, the behaviour of $FreC_A$ is given by solutions to the following set of coupled equations, obtained by simply writing down the relations for the functionality of each basic component:

$$O1(t) = a1(t) + a6(t) \tag{64}$$
$$a1(t) = I1(t-1) \tag{65}$$
$$a2(t) = I1(t-1) \tag{66}$$
$$yd(t) = a5(t-1) \tag{67}$$
$$a6(t) = a5(t-1) \tag{68}$$
$$a5(t) = a2(t) + yd(t) = a2(t) + a5(t-1) = \ldots \tag{69}$$

These equations can be seen to yield a simple standard feedback control system, and its I/O behaviour can be computed

by well known means. From (69), we extract the essential recursive piece:

$$a5(t-q) = a2(t-q) + a5(t-q-1) \qquad q \geq 0 \tag{70}$$

which gives a typical $a5$ value in terms of its predecessor. Taking this, and (64), (65), into account, and simply performing back substitutions naively, we obtain:

$$O1(t) = I1(t-1) + I1(t-2) + I1(t-3) + \ \ldots \ = \sum_{q=1}^{\infty} I1(t-q) \tag{71}$$

The infinite summation in (71) is a shorthand for describing an infinite set of possible finite behaviours of the real ideal $FreC_A$ system. In practice, $FreC_A$ is initialised at the start of its activity. This initialisation would include the initial values of the state elements in $F1, F3,$ as emitted on their output signals $a1, a2$ and $a6, y$ respectively. Since we have the recursive equations (69) and (70), referring to preceding values, one can interpret the initial value of $a6$ say, as corresponding (in the context of (71)) to a (fictitious) set of possible histories of preceding behaviours which culminated in the required initial value of $a6$ at initialisation time.

The snag is that, unlike in the acyclic case, there is no *a priori* bound on how far into the past we must look in order that a backwards analysis will take into consideration all the behaviours of interest for the system. If one assumes a standard intialisation of all state elements, say to values 0, and insists on a particular point in the past, there is no guarantee that every interesting behaviour of the system (i.e. one satisfying the prestated TLE) will be characterised by state elements having these standard values at the given starting point — we saw this to be true even in the acyclic case in Section 5.5, where the issue was addressed via the disjunction of TLIs over the *a priori* finite set of possibilities.

Moreover, if we do not impose an *a priori* bound in the past beyond which we do not search, then the analysis is not guaranteed to terminate. For example, let our TLE be characterised as $O1(t) = 1$ (without any other restrictions). Then if the signals $a5, a6, yd$ satisfy:

$$a5(t-q) = 1 \qquad\qquad a6(t-q) = 1 \qquad\qquad yd(t-q) = 1 \quad q \geq 0 \tag{72}$$

then the output $O1(t)$ will be 1 as required, and all inputs $I1(t-q)$ $(q \geq 0)$ can be set to 0. This scenario would correspond to a(n infinite) number of finite behaviours, each featuring a nonstandard initialisation of $a5, a6, yd$ to 1 at some point in the finite past. A backwards search that restricted its consideration to the *standard* initialisation $a6(t-q) = 0$ for some q, would never find any of them.

The fact that we have an infinite set of possibilities to consider in this simple example is a consequence of the fact that $A1$'s output remains stable when its $a2$ input is held at 0, so that the value held in $F3$ (and hence the output at $O1$) remain invariant as long as $I1$ continues to remain at 0. Such behaviours are not untypical in digital circuits.

Even when a backwards analysis reaches an initial state of whatever kind, that may not be the whole story. Initial states may themselves be reachable, i.e. there may be cycles involving initial states, as above. A given fault scenario may not have arisen since the most recent visit of the system to an initial state, but may have a number of causes distributed over different traversals of a cycle involving an initial state.

A comprehensive analysis technique must take into account situations such as those just discussed, and must include suitable strategies for handling them. The thing to be aware of, is that the properties of a given search technique may, in effect, conjoin additional constraints to the TLE which were not originally present. Depending on one's point of view, this either amounts to solving a different problem than originally conceived, or (implicitly) alters the semantics of the TLE by narrowing the scope of possible satisfying solutions. There is, of course, no harm at all in this, provided one is sure that that the discarded solutions have no significant engineering importance.

6.2. A Faulty Feedback Scenario for $FreC_C$

We now apply the preceding insights to the fault-prone concrete $FreC_C$ model of Fig. 10, in which the adders are ideal, as usual, but the fanouts are prone to the errors captured in (54). Suppose we sever the back-link ye in $FreC_C$, generating new output yp and new input yj, and getting the circuit $AFreC_C$. This is illustrated in Fig. 11.

An examination of Fig. 11 shows that there is a single path from yp back to yj; via $c5$ with a delay of one time unit. There are also two paths from $P1$ back to $J1$; one via $c1$ with a delay of one time unit, and the other via $F3$ and $c2$ with a delay of two time units. The same delays cover paths from $P1$ to yj and from yp to $J1$. So we can deduce that the signature of the causal relation $AFreCS_C$ will be:

$$AFreCS_C(\langle J1(t-1), J1(t-2), yj(t-1)\rangle, \langle P1(t), yp(t)\rangle) \tag{73}$$

We can see a direct similarity between (73), featuring two $J1$ inputs, and the leftmost fault configuration in Fig. 6, featuring two $F1$ faults acting at different moments — although *FreT* and *FreC* are not the same, the noted similarities do arise from the paths they have in common.

Since $AFreCS_C$ has a unit delay from $yp(t)$ back to $yj(t-1)$, and features the $J1$ input at times $J1(t-1), J1(t-2)$, in general, in a backwards analysis, successive iterations of $AFreCS_C$ may give rise to overlapping constraints, as noted above.

Let us now fix a TLE: $P1(t) = 1, J1 = 1$. As previously, this indicates that $P1$ is set to 1 at t and is unconstrained at other times, whereas input $J1$ is 1 permanently. We embark on a backwards fault analysis similar to preceding ones but with some significant differences.

Previously, we could discuss the black box picture all at once, because the relations that comprised it were finite in every aspect. Now, the black box picture concerns infinitary operations and infinitary relationships between them. In order to discuss these properly, we have to reduce their analysis to finite pieces, the iterations of *AFreCS*, as one has to do with any looping structure. Thus the change in perspective from black box picture to basic component picture, which underpins the analysis and which could be achieved in one step for acyclic circuits, now becomes a two step process. The first concerns the understanding of the cyclic black box picture in terms of the iterates of the corresponding acyclic black box picture. The second concerns the understanding of the acyclic black box picture in terms of the underlying components.

For practical purposes, the style of the fault analysis remains very similar to earlier in the paper. We simply trace back through $FreC_C$ as needed, deriving appropriate facts as we go, a process that is largely insensitive to the difference between $FreC_C$ and iterates of $AFreCS_C$. The main difference between the finitary and the infinitary cases relates to the *duration* of the analysis process — which could be unbounded for the infinitary process because of the possibility of visiting the same component of $FreC_C$ an unbounded number of times (albeit with ever decreasing time indexes).

As a result of the potentially unbounded duration, we have to stay alert for branches of the analysis that stretch indefinitely far into the past. If we were dealing purely with a finite state cyclic system without I/O, then the finite state space would ensure that its behaviour would eventually become periodic, thereby making sure that the analysis of its behaviour became periodic too. This would guarantee a resolution tree or FT that eventually became self-similar below a certain point, and this would enable us to represent the infinite tree finitely using back-links. Unfortunately our systems can have I/O, and this spoils the picture. Imagine an input signal that, like the decimal digits of π (or any other irrational number) never repeated. Then the resolution tree or FT would never become truly self-similar (even though limited parts of it would resemble one another sporadically, like the digit stream of π), and we could not represent such a tree by finite means.

On the other hand, since the information in such non-repeating input signals is unboundedly high, they fall outside the remit of engineering interest —engineering systems *never* need infinite amounts of information for their description— so the situation is not so bad. By imposing some straightforward constraints on TLEs and inputs, we exclude these unruly (and impractical) situations, at the same time guaranteeing the existence of a finite representation for the resulting resolution trees or FTs. So, as well as the finiteness stipulations discussed earlier, we now demand that:

- TLE specifications refer to only a finite number of time indexes for outputs (corresponding to the idea that the TLE indeed describes an *event*). (74)
- Input constraints eventually into the past (at worst) repeat with finite period. (75)

With these restrictions, the system as a whole will, going into the past, at worst eventually settle down to cyclic behaviour, with an overall cycle time not worse than the least common multiple of all individual cycle times in the system, since the transients attributable to the TLE itself will eventually die away into the sufficiently distant past. Fortunately, constant behaviour into the distant past, a common state of affairs in engineering situations, is the most benign cyclic behaviour of all.

The above restrictions lead to a practical analysis technique that is little different from the acyclic version. Its simplicity, in turn relies on the soundness of the retrenchment theory in Appendix C, which ensures that accounts at different levels of abstraction can be bound into a consistent methodology.

Here then, is an outline of the analysis of the TLE mentioned above, structured round the disjunctive choices that emerge. For brevity (and clarity), we suppress mentioning the retrenchment data that the steps of the derivation belong to here; we return to this below.

$\mathbf{TLE}^{FB}$: From $P1(t) = 1$, since our instantaneous adders cannot fail, we have either: case **$\mathbf{TLE}^{FB}$.L** in which $c1(t) = 0 \wedge c6(t) = 1$; or case **$\mathbf{TLE}^{FB}$.R** in which $c1(t) = 1 \wedge c6(t) = 0$.

TLE^{FB}.L: From $c1(t) = 0$ we deduce $F1.c1(t)$ is true. From $c6(t) = 1$ we deduce $c5(t-1) = 1$; whence we have either: case **TLE^{FB}.L.L** where $c2(t-1) = 1 \wedge ye(t-1) = 0$; or case **TLE^{FB}.L.R$^{\infty}$** where $c2(t-1) = 0 \wedge ye(t-1) = 1$.

TLE^{FB}.L.L: From $c2(t-1) = 1 \wedge ye(t-1) = 0$ we deduce that $F1.c2(t-1)$ is false and $J1(t-1) = 1$, which is consistent with the TLE. From $ye(t-1) = 0$ we have either: case **TLE^{FB}.L.L.L** where $F3.ye(t-1)$ is true; or case **TLE^{FB}.L.L.R$^{\infty}$** where $F3.ye(t-1)$ is false, and $c5(t-2) = 0$.

TLE^{FB}.L.L.L: $F3.ye(t-1)$ is true yields a valid cause of the TLE.

TLE^{FB}.L.L.R$^{\infty}$: With $c5(t-2) = 0$, we have a subtree identical to that from **TLE^{FB}.R.R$^{\infty}$** below, with timing index decremented by 1. (See below.)

TLE^{FB}.L.R$^{\infty}$: From $c2(t-1) = 0$ we deduce that $F1.c2(t-1)$ is true. From $ye(t-1) = 1$ we deduce that $c5(t-2) = 1$ holds, from which we have either: case **TLE^{FB}.L.R$^{\infty}$.L** where $c2(t-2) = 1 \wedge ye(t-2) = 0$; or case **TLE^{FB}.L.R$^{\infty}$.R** where $c2(t-2) = 0 \wedge ye(t-2) = 1$.

TLE^{FB}.L.R$^{\infty}$.L: From $c2(t-2) = 1$ we deduce that $F1.c2(t-2)$ is false and $J1(t-3) = 1$, consistent with the TLE. From $ye(t-2) = 0$ we have either: case **TLE^{FB}.L.R$^{\infty}$.L.L** where $F3.ye(t-2)$ is false and $c5(t-3) = 0$ holds; or case **TLE^{FB}.L.R$^{\infty}$.L.R** where $F3.ye(t-2)$ is true.

TLE^{FB}.L.R$^{\infty}$.L.L: With $c5(t-3) = 0$, the subtree from here is as for case **TLE^{FB}.R.R$^{\infty}$** below, with timing index decremented by 2. (See below.)

TLE^{FB}.L.R$^{\infty}$.L.R: $F3.ye(t-2)$ is true yields a valid cause of the TLE.

TLE^{FB}.L.R$^{\infty}$.R: With $c2(t-2) = 0$, we have a copy of case **TLE^{FB}.L.R$^{\infty}$** with timing index decremented by 1. (See below.) The remaining argument is by now familiar.

TLE^{FB}.R: From $c1(t) = 1$ we deduce $F1.c1(t)$ is false and $J1(t-1) = 1$, which is consistent with the TLE. From $c6(t) = 0$ we have either: case **TLE^{FB}.R.L** where $F3.c6(t)$ is true; or case **TLE^{FB}.R.R$^{\infty}$** where $F3.c6(t)$ is false, and $c5(t-1) = 0$.

TLE^{FB}.R.L: $F3.c6(t)$ is true yields a valid cause of the TLE.

TLE^{FB}.R.R$^{\infty}$: From $c5(t-1) = 0$, since adders don't fail, we get $c2(t-1) = 0 \wedge ye(t-1) = 0$. Now $c2(t-1) = 0$ implies that $F1.c2(t-1)$ is true. Also $ye(t-1) = 0$ yields either: case **TLE^{FB}.R.R$^{\infty}$.L** where $F3.ye(t-1)$ is true; or case **TLE^{FB}.R.R$^{\infty}$.R** where $F3.ye(t-1)$ is false and $c5(t-2) = 0$.

TLE^{FB}.R.R$^{\infty}$.L: $F3.ye(t-1)$ is true yields a valid cause of the TLE.

TLE^{FB}.R.R$^{\infty}$.R: With $c5(t-2) = 0$ we have a copy of case **TLE^{FB}.R.R$^{\infty}$** with timing index decremented by 1. (See below.) This leads to an unbounded replication of the subtree from case **TLE^{FB}.R.R$^{\infty}$** with diminishing indexes, unless it is terminated by an initialisation.

A FT derived from the above case analysis is shown in Fig. 12. As previously, preprimes denote values at times past, the number of preprimes equaling the displacement into the past.

The FT is closely related to the resolution tree that comes before it. In fact, if we prune some non-minimal branches (to be described shortly), the two are practically identical. The labels under selected events in Fig. 12 correspond to nodes of the development of the RT that match similarly labelled paragraphs in the analysis just above. As in the discussion of the same issue for the case of acyclic circuits, there is some flexibility regarding which/how many of the derived facts ought to be represented in any pictorial representation of the tree, especially when it is to be regarded as a *bona fide* fault tree according to [VSD^{+}02]. All the points made earlier regarding this, apply here *mutatis mutandis*.

Regarding the non-minimal pruned branches, we note that these arise in the context of a concession for a composition of components X and Y (either (skew-)parallel or (skew-)sequential), which is of the form $C \equiv C_X O_Y \vee O_X C_Y \vee C_X C_Y$. The non-minimal cases come from the possibility of satisfying the $C_X C_Y$ disjunct, in circumstances when one or other of $C_X O_Y$ or $O_X C_Y$ is already satisfiable — moreover where the assignment to variables for satisfying $C_X C_Y$ matches that for satisfying $C_X O_Y$ or $O_X C_Y$ on all variables pertinent to the C_X or C_Y term respectively, that they have in common.

We can look for opportunities of this kind in the minimised tree of Fig. 12. Firstly, we observe from Fig. 11 that all the compositions in our circuit are sequential, so all feature interface variables. Therefore, all compositions exhibit the compatibility of variable assignment issues just mentioned. Because of the simplicity of our circuit, all disjunctions in Fig. 12 are of ways to get a satisfying assignment as one traverses some component (backwards). Therefore they are all assignments to the interface variables of some sequential composition. To obtain all the non-minimal cases, it is therefore sufficient to inspect all the disjunctions in Fig. 12, to see which of them disjoin consistent variable assignments and which do not. Those that do not, describe mutually exclusive possibilities: the relevant $C_X C_Y$ disjunct is not satisfiable. For those that do disjoin consistent variable assignments however, the relevant $C_X C_Y$ disjunct *is* satisfiable, and in those cases we can add a third disjunct to the tree which is just the conjunction of the two existing

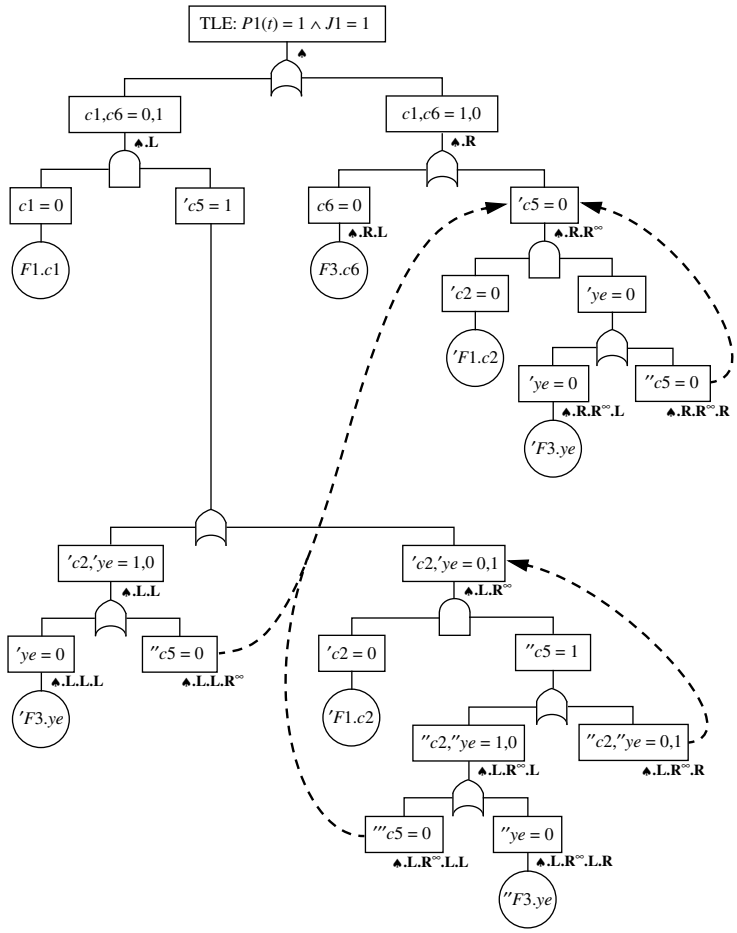

Fig. 12. A Fault Tree for $FreC_C$ and TLE: $P1(t) = 1, J1 = 1$. This is closely related to a minimised precursor Resolution Tree, which differs only in the representation of basic fault nodes and their immediate ancestor events. The RT structure is discernable with the help of the labelled events, where the labels correspond to the cases discussed in the text, except that the prefix $\mathbf{TLE}^{FB}$ is represented by ♠ for brevity.

disjuncts. Of course, all this is complicated by the need to be vigilant in case the disjuncts under consideration are connected via any overlapping feedback loops, with the resulting possibility of the need to ensure consistent variable assignments at various moments going further and further back in time, but this is not a problem in our case, as we will see in a moment.

Let us examine the disjunctions in Fig. 12 on this basis. The disjunction of **♠.L** and **♠.R** assigns mutually inconsistent values to $c1, c6$ i.e. $0,1$ and $1,0$ respectively. So this OR is exclusive, and any cut set must commit to one or other of the possibilities. Likewise, the disjunction of **♠.L.L** and **♠.L.R**, which assigns mutually inconsistent values to $'c2,'ye$. Likewise, the disjunction of **♠.L.R**$^\infty$**.L** and **♠.L.R**$^\infty$**.R**, which concerns $''c2,''ye$. The other disjunctions assign mutually consistent values to variables on their two branches. There is the disjunction of **♠.R.L** and **♠.R.R**$^\infty$, the conjunction of which has $F3.c6(t)$ stuck at zero, regardless of the fact that $F3$ received zero at the previous clock tick anyway. The remaining disjunctions are of **♠.R.R**$^\infty$**.L** and **♠.R.R**$^\infty$**.R**; of **♠.L.L.L** and **♠.L.L.R**$^\infty$; of **♠.L.R**$^\infty$**.L.L** and **♠.L.R**$^\infty$**.L.R**. All of these concern (variously timestamped versions of) $'ye = 0$ and $''c5 = 0$, the conjunction of which, this time reduces to $F3.ye(t-1)$ stuck at zero, regardless of the fact that $F3$ received zero at the previous clock tick. Finally, we observe that in all these last three cases, one of the disjuncts refers to a finite subtree without back-links, and the other to a subtree which, although it does unroll indefinitely far into the past due to a back-link, it does so starting from a time index earlier than any index in the other disjunct, so there is no possibility of the inadvertently clashing variable assignments cautioned against above.

At four places in the FT analysis above we encountered a situation where we claimed that the resolution tree would

simply grow indefinitely (generating similarly shaped fragments labelled with time indexes towards the increasingly remote past), if the derivation process were allowed to continue uninterrupted, which we handled by terminating the process and installing a back-link. The justification for this rests on dataflow arguments which we consider in detail in the next section.

6.3. Regions, Back-Links, Initial States

To justify the back-links of Fig. 12 we re-examine the preceding discussion within the more detailed region-sensitive formulation used in Section 5.4. The considerations needed to deal with initial states and cold start fails will then emerge naturally.

Putting aside for the moment, the fact that the analysis might continue forever, we can proceed much as in Section 5.4. The fact that the analysis *might indeed* not terminate, means that a genuine implementation would need to interleave the various phases that could earlier be done sequentially. Thus, the resolution tree (RT) creation process would commence (applying the on-the-fly minimisation tactics discussed earlier as it goes, a *non post hoc* approach to minimisation being particularly important in a non-terminating scenario), and once the RT had developed to a reasonable depth, a suitable portion of the existing part of it could be transformed into (a suitable portion of) a FT, and once enough of the FT existed, suitable portions of it could be grouped into regions. The three activities would then continue in an interleaved manner, deeper and deeper.

What enables this whole process to be brought to a conclusion is the set of restrictions we have imposed on our TLEs. Since TLEs impose constraints on the free outputs at only a finite number of time indexes (let us call this the TLE output window), and impose *particular* constraints on the free inputs at only a finite number of time indexes (let us call this the TLE input window, which exists because the input constraints eventually into the past repeat with finite period), once the analysis has gone far enough back into the past (by traversing the cycles in the circuit a sufficient number of times), the only free output values *directly* accessible (i.e. accessible without re-traversing cycles forwards again) will fall outside the TLE output window, so can be freely set to whatever values input readiness of components asks of them, without endangering the satisfiability of the TLE. Moreover, once the analysis has also gone further into the past than the TLE input window, we can conclude that the behaviour of the system will repeatedly offer the same set of choices periodically into the past, and so the resulting FT will have a regular structure, encodable via back-links.

Let us apply this to our example. The regions of the FT in Fig. 12 are shown in Fig. 13. Let us pretend momentarily that the back-links in Figs. 12 and 13 are not there, so that Figs. 12 and 13 represent merely a portion of the infinitary analysis. Then looking at the leftmost $''c5 = 0$ node and referring to Figs. 10 or 11, we see that that value of $c5$ at that time index, can only influence the free output $P1$ directly at $t-1$, which is irrelevant for satisfying the TLE. Looking into the past, we see that the possible causes of $''c5 = 0$ depend ultimately only on the input $J1$, which is held constant. Thus any earlier occurrence of the value 0 on $c5$, e.g. $'''c5 = 0$, will have a set of possible explanations identical to that of $''c5 = 0$, except displaced in time, justifying a back-link from e.g. $'''c5 = 0$ to $''c5 = 0$. However we also notice that there is a later occurrence of the value 0 on $c5$, namely $'c5 = 0$, which happens to feed directly into the TLE. So, since the input constraints are identical here too, we can redirect *any* back-link from an early occurrence of 0 on $c5$ to the $'c5 = 0$ node.

Once it is recognised that the behaviour is ultimately cyclic into the past, one can arrange the back-links at one's convenience. Figs. 12 and 13 are in fact suboptimal in the sense that the back-links were chosen to align with the disjunctive structure of the tree derivation in Section 6.2. A discussion just like the one above, reveals that the subtree descending from the $''c5 = 1$ node is a replica displaced in time of the one descending from $'c5 = 1$. So a back-link from $''c5 = 1$ to $'c5 = 1$ would be justified, and would have resulted in a smaller tree. But both $''c5 = 1$ and $'c5 = 1$ occur in a conjunctive context, so we avoided instituting this optimisation in order to not complicate the tree derivation.

In general, good places to look for opportunities to install back-links in the developing FT are the roots of regions, since the subtrees under them will refer to a maximal consistent set of facts about a state (and every region will have a unique root, since it is a connected subgraph of a directed tree).

Of course, not all branches of the FT analysis extend infinitely far into the past. A branch can stop when it hits a free input value, or when it hits a component output which is independent of any of the component's inputs. In addition, stipulating a value outside of the codomain of the component that is supposed to produce it, will, in general, cause backtracking and truncation. The ultimately periodic nature of our constraints ensures that such situations will typically be replicated into the past.

A special case of terminating branches arises in the context of initial state analysis. This can be performed almost identically to the way described in Section 5.5, once the FT is in finite form. The only difference is that the looping nature of the FTs for cyclic circuits makes them reach infinitely far into the past as regards the hot running scenario,

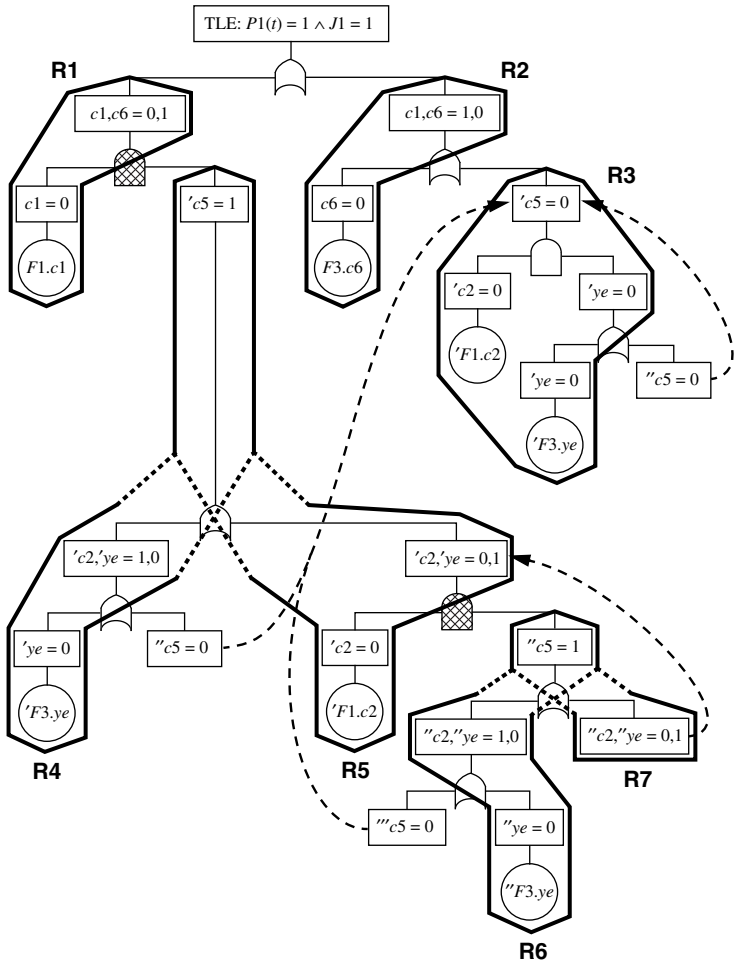

Fig. 13. The regions in the Fault Tree for the TLE of *FreC*.

whereas in the acyclic case, the finite reach of the FT resulted in a 'don't care' attitude before the earliest moment that could influence the TLE.

6.4. Formal Considerations in the Feedback Scenario

We now discuss the repercussions of introducing feedback into the formal analysis of PaperI, whose extension to timed acyclic circuits we discussed in Section 5.6. The introduction of feedback alters things considerably, since the unwinding of a cyclic system, while acyclic, is no longer finite. So the techniques of PaperI, which rely on induction (which needs to be based on a well-founded set), are no longer applicable, even if one takes for granted the increased complexity caused by the absorption of timing issues into variable names and the replacement of circuit structure by the structure of the associated causal relations, as in Section 5.6. Before we address this, let us comment on what was achieved in the analysis of PaperI.

In PaperI we described a formal, but fairly abstract resolution tree (RT) algorithm, and proved it sound and complete. The RT it produced could then be post-processed into a FT. The algorithm(s) all worked top-down (following the backwards oriented structure of circuit descriptions) and the proofs of soundness and completeness worked bottom-up (by induction). The reconciliation between the top-down and bottom-up views confirmed that our formulation gave a good account of the *requirements* of such an analysis, namely that the algorithm indeed generated a FT for all the cut sets of interest, which could in turn be specified in a manner quite independent of the algorithm itself.

In the feedback case, the unwound cyclic circuit is an infinitary object, possessing a top (the free outputs at time index t_λ say) but no bottom (since the unwinding, and hence the analysis, proceeds backwards into the indefinite past), precluding the use of bottom-up inductive techniques. Thus we lose one of the main merits of the acyclic case, namely the ability to reconcile the widely disparate top-down and bottom-up views of the same situation.

In the presence of infinitary objects termination arguments disappear. This applies as much to attempts to build 'proofs' as to the construction of algorithms. Since a proof is a finite logically sound argument, and a proof about an infinitary structure must cover the infinity of pieces of that structure, a proof cannot cover each piece individually. However, in general, we do not construct proofs about individual structures, but finitary *meta*-proofs about sets of individual structures which are generated in a predictable manner. Applied to the infinitary structures we have to deal with, we end up considering co-inductive proofs [Rut96, JR97].

These (have to) proceed top-down. Each stage of the process they describe corresponds to the unfolding of one level of the infinitary structure, and the reasoning associated with that. What would this correspond to in the context of our constructions? By analogy with PaperI, we would be proving that the RT-with-back-links (and ultimately FT-with-back-links), which were generated top-down, corresponded to the possible cut sets of a given TLE, these also being enumerated top-down. In the end, the two processes would be sufficiently similar that little would be gained by giving all the details. For that reason we do not do it.

The similarity goes beyond the fact that both processes are top-down. A reasoning step in a co-inductive proof posits the needed properties of the unwinding step monolithically, i.e. as a collection of facts about the newly revealed structure. In fact this is extremely close to our use of angelic nondeterminism in the description of the abstract formal algorithm in PaperI, which would also be used in a formalisation of the FT-with-back-links algorithm here.

The similarity between the algorithmic and proof-driven approaches is further enhanced by observing that unlike the examples in [Rut96, JR97], our structures are not freely generated, since we must take care that distinct paths that lead to the same delay for the same stream identifier do not lead to incompatible data assignments. In the co-inductive approach, this consistency does not emerge as a natural part of the proof process, but must be 'taken care of from outside'. How would one 'take care of it from outside'? Well, the only possibility would be to run the algorithm, ensuring it output only consistent results, and then to offer these results to the co-inductive proof.[17] (This said, we recall the remarks of Section 5.6, in which we pointed out how the prospect of shared data streams could be overcome relatively easily.)

How would one ensure that the algorithm output only consistent results? The only ultimately reliable basis for this comes from the finiteness of our circuits and data types, which implies that the state space of our systems is finite and can (in principle) be enumerated. So the worst that our systems can do is to loop when given a periodic input. This guarantees that all our approaches in the end are sound and complete.

6.5. Efficiency, Symbolic Processing, and Comparison with FSAP

In the acyclic case, we argued that the set of variables to keep under control was always finite but might become extremely large. In the present situation, the set of variables to manage can become infinite in principle, due to paths that might potentially continue into the infinite past. One tool we deployed to subdue the problem was to impose restrictions on our TLEs and inputs, in order that the behaviour of this infinite family of variables would ultimately repeat, and therefore that knowledge of a finite subset of them yielded knowledge about all. This is of course a form of symbolic reasoning. We therefore see that the cyclic case generates an even greater demand for the powers of symbolic reasoning than the acyclic case. Moreover, symbolic reasoning and some form of abstraction become unavoidable when it comes to a practical implementation, to keep the complexity of the analysis and of the generated results under control. In the rest of this section, we provide a short comparison with what is available in the FSAP tool [BV07], and we hint to possible solutions to incorporate timing information in fault trees, in a way which is both practicable and meaningful for safety engineers. A thorough discussion on this topic is, however, beyond the scope of this paper, and deserves to be reported elsewhere in full detail.

Fault tree generation in FSAP is based on symbolic model checking techniques [CGP00], among them techniques based on Binary Decision Diagrams [Bry92] (BDDs for short), which provide an efficient means for carrying out manipulations on set of states and transition systems. FSAP provides a number of BDD-based routines to perform reachability analyses [BCT07], including backwards reachability i.e. starting from a representation of the set of states

[17] This is somewhat analogous to the fact that for co-induction in general, the sought for final co-algebra is not guaranteed to exist, but must be supplied *ad hoc*. In 'benign' cases such as ours, there are generic constructions, but these would once again be essentially equivalent to our algorithms here.

satisfying the top level event. As already discussed in PaperI, FSAP focuses on the problem of minimisation, hence the generated fault trees have less structure than those produced using retrenchment, both in the combinational case and in the case of clocked and feedback circuits. A further difference, already hinted at in PaperI, is given by the need to avoid the use of angelic non-determinism, and to introduce a strategy for the way the search space is visited. In particular, FSAP uses breadth-first search, although elements of depth-first search may be introduced to make the search more efficient, see [BCT07]. When it comes to timing and feedback loops, it is important to note that the routines generated by FSAP are *time insensitive*.[18] In fact, when computing preimages, they record sets of states that that can occur one time step earlier, but no attempt is made to record this fact for reasons of efficiency. As a consequence, states coinciding with previously generated states are identified as being the same modulo a shift in time. This construction deals silently with looping behaviour due to cycles (it corresponds to carrying out a symbolic fixpoint computation for loop resolution), yielding completeness, albeit viewed through the abstraction of forgetting timing data. Moreover, there is a difference in the granularity at which a given system is decomposed. Namely, FSAP routines would decompose the system, depending on time delays which need not correspond to the hierarchical structure of the system being exploited by the retrenchment-based routines. Finally, it is important to remark that in FSAP there is a tight coupling between initialisation and minimisation. As a consequence, without detailed guidance, FSAP's approach to cold start failures is always eager, in that a cold start scenario will always preclude deeper investigation of possible hot running scenarios that need to 'predate' it, which is not the case for the retrenchment-based approach.

The preceding observations on time insensitiveness and granularity open the door to mixed/hybrid strategies in the BDD world, involving both a controlled introduction of timing information, and a different management of time. Since system structure and decomposition in both approaches are defined via logic primitives, there is scope for introducing more detail into the handling of timing information in the BDD world, even if holding all the details might be prohibitive on time and space grounds. As a minor augmentation of the FSAP strategy one could, for instance, contemplate adding indexing to each fault variable, to record the order in which these got set as a cut set was developed. This information could then be exploited to produce results more meaningful for safety analysts. A notable way to incorporate similar information in a fault tree is to record the relative order of events within a cut set. Namely, it is possible to record whether there must be timing constraints forcing a particular event to happen before or after another one (e.g. because of a causality relation or a functional dependency). This analysis, called *ordering analysis* in [BV07], has been incorporated recently within FSAP routines. These are now capable of generating dynamic fault trees, in which priority AND gates are used to express the ordering constraints [BCK+10]. Ordering analysis is just one example in which timing information can be —and has been— incorporated into fault trees. Another notable example is, for instance, the possibility of recording information on the duration of faults, in order to better capture the effect of sporadic and transient faults at system level. A further example is the modelling and analysis of systems including (cold, hot or warm) spare elements [DBB92]. Such extensions are still an open question in the symbolic realm, and deserve further investigation.

Although a full discussion on this topic is beyond the scope of this paper, our claim is that the resolution trees produced by the retrenchment-based approach can be used as a reference model, yielding a formal account of the different practical possibilities, which can then be instantiated in a specific implementation. Clearly, a practical solution has to re-introduce the information about system evolution in a controlled way. Storing and manipulating sets of traces can be achieved by introducing additional variables in the BDD package. However, a naive solution would very easily lead to an exponential blow-up both in terms of time and memory requirements. As a possible solution, one might concentrate on a subset of variables that are of interest in the generation of the fault tree, e.g. the fault variables, and possibly the interface variables. Moreover, we expect that designing a suitable mix of breadth first and depth first search approaches could have a significant impact on the performance. Overall, we believe that there is much fertile ground to be explored between the current FSAP strategy and the maximally detailed retrenchment-led one. Contemplating how the idealised features of the retrenchment-based approach can be realised, brings it closer to the world of FSAP, while contemplating how additional information can be incorporated into FSAP, brings it closer to the retrenchment world. Ultimately what is practicable is governed by complexity considerations.

7. Related Work

In addition to the review of existing material for fault tree analysis in the instantaneous case, presented in PaperI, in this section we discuss related work in the case of dynamic and cyclic systems.

[18] An exception to this is that a limited amount of timing information is recorded, to partially overcome the time insensitivity in the presence of sporadic failure modes.

As remarked in Section 2, traditionally the analysis of timed system relies on the safety engineer's expertise and the use of *ad hoc* strategies for modelling dynamic aspects, such as normal events, inhibit gates with conditioning events or qualification of basic events with exposure intervals. An alternative strategy is proposed in [CM02]. In that work, a methodology is presented, that extends classical fault trees with time requirements representing different scenarios. A fault tree in such a context assumes the form of an 'umbrella' fault tree, which uses house events to switch parts of the fault tree on or off, depending on timing requirements. Moreover, time-dependent probabilistic models are introduced in order to allow for quantitative evaluation. A major difference compared with our work is that the approach focuses on fault tree evaluation at a finite number of discrete time points (defined by means of a finite matrix), whereas our work does not have such a limitation.

Concerning the analysis of systems with feedback, we mention [Vau07]. That work proposes a method of solving (that is, of computing the MCSs of) cyclic fault trees, by using resolution on a set of recursive equations that define the fault tree. The proposed method amounts to iterative substitution of values corresponding to loop gates within the original equations, until convergence is reached. Convergence is guaranteed by the fact that there is only a finite set of possible MCSs, and redundant cut sets can be pruned. By contrast, the work given in the present paper uses infinite time, for which convergence in the above sense is not guaranteed — convergence is achieved by the introduction of back-links and the loop detection mechanism, which is not necessary in [Vau07]. Moreover, [Vau07] postulates that cyclic gates have value false in the far past, whereas our analysis does not make any such assumption.

Finally, [Wal05] describes a modular, compositional method, called Failure Propagation and Transformation Calculus (FPTC) to automatically derive the failure properties of a system, starting from a description of the architecture and the failure behaviour of single components. This work is inspired by, and tries to address some drawbacks of, the Fault Propagation and Transformation Notation (FPTN) [FMNP94], and is also evidently related to [Pap00] — both of these works have been discussed in PaperI. It describes a calculus to model component faults, both propagational and transformational ones, and to compose fault expressions in order to derive the behaviour of the complete system. The routines used for the derivation are equipped with a loop resolution mechanism, that uses a fixpoint algorithm to analyse the flows of failures along any failure propagation path. The loop resolution algorithm is shown to be terminating and convergent — the final result is the same, regardless of the order of application of transformation rules. As for [Pap00], this work relies on the identification and accurate modeling of all possible failure behaviours and their transformation, and is focused on the automatic composition of such behaviours. Moreover, it is intended for architectural analysis — our approach on the other hand, is focused on more 'low-level' modeling, be it relational (as in the retrenchment-based framework), or automata-like (as in symbolic model checking). Such models may in principle range from architectural to implementation-level models, and in general they need not be compositional (the details of the analysis of non-compositional models are sorted out by the routines for automatic fault tree generation). Finally, their fixpoint algorithm for loop resolution works in a similar fashion to ours, although ours is based on more 'low-level' entities than the tokensets of [Wal05].

8. Conclusions and Future Work

In this paper and PaperI we have presented a formal account of fault tree generation based on retrenchment. We have shown how the retrenchment framework is able to capture several aspects of fault tree generation, namely: fault injection, system model evolution, the mechanical construction of a fault tree based on structural information, and fault tree minimisation. In the present paper we showed how the instantaneous approach can be generalised to deal with dynamic and cyclic systems. The approach we advanced yields a reference framework for the mechanical generation of FTs which are better structured than those yielded by competing frameworks, since the approach is based on system structure. It is also capable of yielding soundness and completeness results of a precise kind, which improves on having to rely on safety engineers' vigilance to ensure adequate coverage, especially in complex timed and cyclic situations.

From an engineering perspective, as in the case of the instantaneous circuits of PaperI, we think that the fault trees generated by our routines in the timed and cyclic cases can be useful for standard safety analysis practice, when interpreted and post-processed by a safety engineer that has a good comprehension of the system at hand. In particular, in the timed and cyclic cases, post-processing requires analysing the time-related behaviours, and assigning them a meaningful semantical interpretation. The semantical interpretation can be facilitated if timing information is presented in a comprehensible form that abstracts away, in a controlled way, the full details generated by the retrenchment-based approach.

Once interpreted, the behaviours can be translated into suitable time-related intermediate events, possibly using inhibit gates with conditioning events, as discussed in Section 2. In the case of systems with feedback, our extended notion of fault trees provides a good starting point for analysing the regularities of the model (represented by the

looping structures) and for assigning them the correct interpretation. For instance, a loop could be post-interpreted and abstracted with a state-of-system fault. A major advantage of our approach resides in the fact that automated derivation of fault trees in the cyclic case is guaranteed to explore every branch of the search space, thus avoiding the risk of missing parts of the fault tree when cycles are required to be manually broken by safety engineers.

As regards complexity of the analyses, we already discussed in Section 6.5 ways to present the users with more comprehensible information, and suggested possible ways to reduce to gap between the ideal retrenchment world and a practical implementation. Moreover, besides presenting the users with abstractions of the generated results, one could contemplate restricting the boundary of the analysis or reducing its level of resolution, as in the case of the combinational circuits presented in PaperI.

Concerning the range of phenomena that our framework can account for, in the timed and cyclic cases, in this paper we have shown that the retrenchment-based approach can deal with a wide range of situations, and is able to manage timing effects such as glitches and transient effects. From a safety engineering perspective, a notable example of such a situation is a fault that induces signal delay propagation. Other interesting situations that arise in safety engineering contexts are FDIR (Fault Detection, Identification and Recovery) mechanisms, and dynamic reconfiguration in fault-tolerant systems. Accounting for such situations requires proper system modeling and adequate fault models. For instance, one would like to evaluate the effectiveness and timeliness of an automatic reconfiguration mechanism in presence of faults, possibly in a real-time environment. While this kind of phenomenon, can in principle be taken care of by the retrenchment-based framework presented so far, one has to carefully consider the impact of circuit delays (e.g. due to faults and system reconfiguration) on the abstract and concrete models that the retrenchment is trying to relate. While the effects of some faults may be limited in time and/or in space, and induce small variations on the behaviour of the overall system, in general fault propagation may cause substantial effects at system level — in terms of evolution over time and number of impacted subsystems. For instance, the effect of local delays may be amplified when signals propagate through the complete system. In such cases, the retrenchment data may end up in trying to relate concrete and abstract models that are seemingly unconnected in time when viewed from the perspective of the real world, leading to essentially meaningless conclusions. The way to overcome such problems is by the use of a more coarse-grained version of retrenchment, which, in relating *execution paths* of varying lengths via the retrenchment data, can absorb the discrepancies in timing generated in the concrete and abstract models during the kinds of phenomena alluded to, thereby returning the retrenchment-led derivations to the reasonable fold. However the use of such techniques lies beyond the scope of the present paper.

In these papers we dealt only with the *structure* of the FT. Above this lies the quantitative domain in which probabilities are assigned to the various possibilities, for safety engineering purposes. At this level, the choice of what to incorporate in the FT (e.g. whether to take into account initial states or not), serves to *condition* the relevant probabilities. Although outside the scope of this paper, these matters can be dealt with by standard probabilistic techniques.

As future work, we plan to address the incorporation of the techniques described in these papers into the FSAP platform [BV07, FSA] (an undertaking which has started already). This will completely automate the generation of structured fault trees starting from a specification of the system model and the requirement to be analyzed. As hinted at in Section 6.5, in the case of clocked and feedback circuits, it is essential that timing information is introduced in a controlled way. (An example is some recent work [BCK$^+$10], which addresses the generation of dynamic fault trees that model ordering constraints between basic events.) An interesting research direction we intend to explore is the recording of information on the duration of faults — particularly in the case of sporadic and transient faults. Moreover, we would like to address the analysis of systems including (cold, hot or warm) spare units — we envisage that such an extension will require proper annotation of the system model. Finally, we would like to investigate the possibility of adapting our techniques to the computation of truncated cut sets (see e.g. [RD97, Cep05]). The comparison and interplay between the approach developed here on the one hand, and the intrinsic FSAP model checking based approach on the other, is itself a fascinating area, and one that opens up opportunities for performance tradeoffs of many kinds. All of this will be reported on more extensively elsewhere.

One final comment. The retrenchment based algorithms described in these papers work in a purely data driven way. With minimal adaptation to do with the labelling of states, this makes them eminently suitable not only for the clocked systems we have focused on here, but for the fault analysis of asynchronous hardware too. Further details lie beyond the scope of these papers. The huge number of states that asynchronous working generates, can pose problems for model checking techniques if they approach such systems naively. However, considerable ground can be recovered through exploiting symmetries arising via temporal independence [CGP00]. Finally, it would be interesting to consider the application of our techniques for the safety analysis of software [LH83].

References

[BB10] R. Banach and M. Bozzano. The Mechanical Generation of Fault Trees for Reactive Systems via Retrenchment I: Combinational Circuits, 2010. submitted.

[Bc01] M. Broy and G. Ştefănescu. The Algebra of Stream Processing Functions. *Theoretical Computer Science*, 258:99–129, 2001.

[BCC+03] M. Bozzano, A. Cavallo, M. Cifaldi, L. Valacca, and A. Villafiorita. Improving Safety Assessment of Complex Systems: An Industrial Case Study. *International Symposium of Formal Methods Europe (FME 2003), Pisa, Italy, LNCS*, 2805:208–222, September 2003.

[BCK+10] M. Bozzano, A. Cimatti, J.-P. Katoen, V.Y. Nguyen, T. Noll, and M. Roveri. Safety, dependability, and performance analysis of extended AADL models. *The Computer Journal*, doi: 10.1093/com, March 2010.

[BCT07] M. Bozzano, A. Cimatti, and F. Tapparo. Symbolic Fault Tree Analysis for Reactive Systems. In *Proc. Symposium on Automated Technology for Verification and Analysis (ATVA 2007)*, pages 162–176, 2007.

[BJ] R. Banach and C. Jeske. Retrenchment and Refinement Interworking: the Tower Theorems. Submitted. See [Ret].

[BJP08] R. Banach, C. Jeske, and M. Poppleton. Composition Mechanisms for Retrenchment. *J. Log. Alg. Prog.*, 75:209–229, 2008.

[BPJS07] R. Banach, M. Poppleton, C. Jeske, and S. Stepney. Engineering and Theoretical Underpinnings of Retrenchment. *Sci. Comp. Prog.*, 67:301–329, 2007.

[Bry92] R. E. Bryant. Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[BV03a] M. Bozzano and A. Villafiorita. Integrating Fault Tree Analysis with Event Ordering Information. *Proc. ESREL 2003*, pages 247–254, 2003.

[BV+03b] M. Bozzano, A. Villafiorita, et al. ESACS: An Integrated Methodology for Design and Safety Analysis of Complex Systems. *Proc. ESREL 2003*, pages 237–245, 2003.

[BV07] M. Bozzano and A. Villafiorita. The FSAP/NuSMV-SA Safety Analysis Platform. *International Journal on Software Tools for Technology Transfer*, 9(1):5–24, 2007.

[Cep05] M. Cepin. Analysis of truncation limit in probabilistic safety assessment. *Reliability Engineering and System Safety*, 87(3):395–403, 2005.

[CGP00] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2000.

[CM02] M. Cepin and B. Mavko. A dynamic fault tree. *Reliability Engineering and System Safety*, 75(1):83–91, 2002.

[DBB92] J. Dugan, S. Bavuso, and M. Boyd. Dynamic fault tree models for fault tolerant computer systems. *IEEE Transactions on Reliability*, 41(3):363–377, 1992.

[FMNP94] P. Fenelon, J. A. McDermid, M. Nicholson, and D. J. Pumfrey. Towards Integrated Safety Analysis and Design. *Applied Computing Review*, 2(1):21–32, 1994.

[FSA] The FSAP/NuSMV-SA platform. `http://sra.itc.it/tools/FSAP`.

[JR97] B. Jacobs and J.J.M.M. Rutten. A Tutorial on (Co)Algebras and (Co)Induction. *E.A.T.C.S. Bulletin*, 62:222–259, 1997.

[LH83] N.G. Leveson and P.R. Harvey. Software Fault Tree Analysis. *Journal of Systems and Software*, 3(2):173–181, 1983.

[Pap00] Y. Papadopoulos. *Safety-Directed System Monitoring Using Safety Cases*. PhD thesis, Department of Computer Science, University of York, 2000. Tech. Rep. YCST-2000-08.

[RD97] A. Rauzy and Y. Dutuit. Exact and Truncated Computations of Prime Implicants of Coherent and Non-Coherent Fault Trees within Aralia. *Reliability Engineering and System Safety*, 58(2):127–144, 1997.

[Ret] Retrenchment Homepage. `http://www.cs.man.ac.uk/retrenchment`.

[Rut96] J. J. M. M. Rutten. Universal coalgebra: A theory of systems. Technical Report CS-R9652, Centrum voor Wiskunde en Informatica, 1996.

[Vau07] J.K. Vaurio. A recursive method for breaking complex logic loops in Boolean system models. *Reliability Engineering and System Safety*, 92(10):1473–1475, 2007.

[VGRH81] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. Fault Tree Handbook. Technical Report NUREG-0492, Systems and Reliability Research Office of Nuclear Regulatory Research U.S. Nuclear Regulatory Commission, 1981.

[VSD+02] W.E. Vesely, M. Stamatelatos, J. Dugan, J. Fragola, J. Minarick III, and J. Railsback. Fault Tree Handbook with Aerospace Applications. Technical report, NASA, 2002.

[Wal05] M. Wallace. Modular Architectural Representation and Analysis of Fault Propagation and Transformation. *ENTCS*, 141(3):53–71, 2005.

A. Within Relations for Skew-Sequential Compositions of Indexed Dataflow Retrenchments

As noted in Section 4, the composed within relation is the most complicated relation in the composed data for skew-sequentially composed indexed dataflow retrenchments. It consists of two parts. The first is given in (76) which gives the formula for the composed within relation itself. The second is given in (77) and is a proviso which must hold before (76) is valid.

In (76), we see $W_{AcyOpS,1;_\delta 2}$ decomposed in the first equivalence into two contributions from $W_{AcyOpS^0,1}$ relating to the two $AcyOpS_1^0$ contributions in $AcyOpS_{1;_\delta 2}$, and two contributions from $W_{AcyOpS^{12},1}$ relating to the two $AcyOpS_1^{12}$ contributions in $AcyOpS_{1;_\delta 2}$ (cf. (7)). The second equivalence decomposes these further into within relations for individual abstract and concrete input pairs in the manner of (32).

$$W_{AcyOpS,1;_\delta 2}($$
$$\langle i_{11}(t_{\lambda_2^0}), i_{11}(t_{\lambda_2^{12}}-1), i_{11}(t_{\lambda_2^{12}}-3), i_{12}(t_{\lambda_2^{12}}-4), i_{12}(t_{\lambda_2^{12}}-5), i_{11}(t_{\lambda_2^{12}}-4), i_{12}(t_{\lambda_2^{12}}-5), i_{12}(t_{\lambda_2^{12}}-6)\rangle,$$

$$
\begin{aligned}
&\langle j_{11}(t_{\lambda_2^0}), j_{11}(t_{\lambda_2^{12}}-1), j_{11}(t_{\lambda_2^{12}}-3), j_{12}(t_{\lambda_2^{12}}-4), j_{12}(t_{\lambda_2^{12}}-5), j_{11}(t_{\lambda_2^{12}}-4), j_{12}(t_{\lambda_2^{12}}-5), j_{12}(t_{\lambda_2^{12}}-6)\rangle) \equiv \\
&W_{AcyOpS^0,1}(i_{11}(t_{\lambda_2^0}), j_{11}(t_{\lambda_2^0})) \wedge W_{AcyOpS^0,1}(i_{11}(t_{\lambda_2^{12}}-1), j_{11}(t_{\lambda_2^{12}}-1)) \wedge \\
&W_{AcyOpS^{12},1}(\langle i_{11}(t_{\lambda_2^{12}}-3), i_{12}(t_{\lambda_2^{12}}-4), i_{12}(t_{\lambda_2^{12}}-5)\rangle, \langle j_{11}(t_{\lambda_2^{12}}-3), j_{12}(t_{\lambda_2^{12}}-4), j_{12}(t_{\lambda_2^{12}}-5)\rangle) \wedge \\
&W_{AcyOpS^{12},1}(\langle i_{11}(t_{\lambda_2^{12}}-4), i_{12}(t_{\lambda_2^{12}}-5), i_{12}(t_{\lambda_2^{12}}-6)\rangle, \langle j_{11}(t_{\lambda_2^{12}}-4), j_{12}(t_{\lambda_2^{12}}-5), j_{12}(t_{\lambda_2^{12}}-6)\rangle) \equiv \\
&\quad W_{AcyOpS^0,1}(i_{11}(t_{\lambda_2^0}), j_{11}(t_{\lambda_2^0})) \wedge W_{AcyOpS^0,1}(i_{11}(t_{\lambda_2^{12}}-1), j_{11}(t_{\lambda_2^{12}}-1)) \wedge \\
&\quad W_{AcyOpS^{12.1},1}(i_{11}(t_{\lambda_2^{12}}-3), j_{11}(t_{\lambda_2^{12}}-3)) \wedge W_{AcyOpS^{12.2},1}(i_{12}(t_{\lambda_2^{12}}-4), j_{12}(t_{\lambda_2^{12}}-4)) \wedge \\
&\quad W_{AcyOpS^{12.2},1}(i_{12}(t_{\lambda_2^{12}}-5), j_{12}(t_{\lambda_2^{12}}-5)) \wedge W_{AcyOpS^{12.1},1}(i_{11}(t_{\lambda_2^{12}}-4), j_{11}(t_{\lambda_2^{12}}-4)) \wedge \\
&\quad W_{AcyOpS^{12.2},1}(i_{12}(t_{\lambda_2^{12}}-5), j_{12}(t_{\lambda_2^{12}}-5)) \wedge W_{AcyOpS^{12.2},1}(i_{12}(t_{\lambda_2^{12}}-6), j_{12}(t_{\lambda_2^{12}}-6)) \qquad (76)
\end{aligned}
$$

Next comes the proviso, which in (77) stipulates that no matter whether the outcome of the first layer of the skew-sequential composition is realised via O or C (and since two occurrences of $AcyOpS_1$ are needed to cater for the first layer of the composition, two $O \vee C$ disjunctions appear in (77)), every such outcome is contained in the within relation for the second layer, i.e. $W_{AcyOpS,2}$. The proviso is written in *hypotheses* $\vdash$ *conclusion* style, where the *hypotheses* contain the equalities that glue output values from $AcyOpS_1$ to input values to $AcyOpS_2$ in the manner dictated by ${}^{\circ}_{9\delta}$, in order to show the timing indexes at the intermediate stage in a more explicit way than would be achieved by using an existential quantification.

$$
\begin{aligned}
&o_{10}(t_{\lambda_2^0}) = i_{21}(t_{\lambda_2^0}) \ , \ p_{10}(t_{\lambda_2^0}) = j_{21}(t_{\lambda_2^0}) \ , \\
&o_{10}(t_{\lambda_2^{12}}-1) = i_{21}(t_{\lambda_2^{12}}-1) \ , \ p_{10}(t_{\lambda_2^{12}}-1) = j_{21}(t_{\lambda_2^{12}}-1) \ , \\
&o_{11}(t_{\lambda_2^{12}}-2) = i_{22}(t_{\lambda_2^{12}}-2) \ , \ p_{11}(t_{\lambda_2^{12}}-2) = j_{22}(t_{\lambda_2^{12}}-2) \ , \\
&o_{11}(t_{\lambda_2^{12}}-3) = i_{22}(t_{\lambda_2^{12}}-3) \ , \ p_{11}(t_{\lambda_2^{12}}-3) = j_{22}(t_{\lambda_2^{12}}-3) \\
\vdash \\
&(O_{AcyOpS,1}(\langle o_{10}(t_{\lambda_2^0}), o_{11}(t_{\lambda_2^{12}}-2), o_{12}(t_{\lambda_2^{12}}-1)\rangle, \langle p_{10}(t_{\lambda_2^0}), p_{11}(t_{\lambda_2^{12}}-2), p_{12}(t_{\lambda_2^{12}}-1)\rangle, \\
&\quad \langle i_{11}(t_{\lambda_2^0}), i_{11}(t_{\lambda_2^{12}}-3), i_{12}(t_{\lambda_2^{12}}-4), i_{12}(t_{\lambda_2^{12}}-5)\rangle, \langle j_{11}(t_{\lambda_2^0}), j_{11}(t_{\lambda_2^{12}}-3), j_{12}(t_{\lambda_2^{12}}-4), j_{12}(t_{\lambda_2^{12}}-5)\rangle) \vee \\
&\ C_{AcyOpS,1}(\langle o_{10}(t_{\lambda_2^0}), o_{11}(t_{\lambda_2^{12}}-2), o_{12}(t_{\lambda_2^{12}}-1)\rangle, \langle p_{10}(t_{\lambda_2^0}), p_{11}(t_{\lambda_2^{12}}-2), p_{12}(t_{\lambda_2^{12}}-1)\rangle, \\
&\quad \langle i_{11}(t_{\lambda_2^0}), i_{11}(t_{\lambda_2^{12}}-3), i_{12}(t_{\lambda_2^{12}}-4), i_{12}(t_{\lambda_2^{12}}-5)\rangle, \langle j_{11}(t_{\lambda_2^0}), j_{11}(t_{\lambda_2^{12}}-3), j_{12}(t_{\lambda_2^{12}}-4), j_{12}(t_{\lambda_2^{12}}-5)\rangle)) \wedge \\
&(O_{AcyOpS,1}(\langle o_{10}(t_{\lambda_2^{12}}-1), o_{11}(t_{\lambda_2^{12}}-3), o_{12}(t_{\lambda_2^{12}}-2)\rangle, \langle p_{10}(t_{\lambda_2^{12}}-1), p_{11}(t_{\lambda_2^{12}}-3), p_{12}(t_{\lambda_2^{12}}-2)\rangle, \\
&\quad \langle i_{11}(t_{\lambda_2^{12}}-1), i_{11}(t_{\lambda_2^{12}}-4), i_{12}(t_{\lambda_2^{12}}-5), i_{12}(t_{\lambda_2^{12}}-6)\rangle, \langle i_{11}(t_{\lambda_2^{12}}-1), i_{11}(t_{\lambda_2^{12}}-4), i_{12}(t_{\lambda_2^{12}}-5), i_{12}(t_{\lambda_2^{12}}-6)\rangle) \vee \\
&\ C_{AcyOpS,1}(\langle o_{10}(t_{\lambda_2^{12}}-1), o_{11}(t_{\lambda_2^{12}}-3), o_{12}(t_{\lambda_2^{12}}-2)\rangle, \langle p_{10}(t_{\lambda_2^{12}}-1), p_{11}(t_{\lambda_2^{12}}-3), p_{12}(t_{\lambda_2^{12}}-2)\rangle, \\
&\quad \langle i_{11}(t_{\lambda_2^{12}}-1), i_{11}(t_{\lambda_2^{12}}-4), i_{12}(t_{\lambda_2^{12}}-5), i_{12}(t_{\lambda_2^{12}}-6)\rangle, \langle i_{11}(t_{\lambda_2^{12}}-1), i_{11}(t_{\lambda_2^{12}}-4), i_{12}(t_{\lambda_2^{12}}-5), i_{12}(t_{\lambda_2^{12}}-6)\rangle)) \\
&\quad \Rightarrow \\
&W_{AcyOpS,2}(\langle i_{21}(t_{\lambda_2^0}), i_{21}(t_{\lambda_2^0}-1), i_{22}(t_{\lambda_2^0}-2), i_{22}(t_{\lambda_2^0}-3)\rangle, \langle j_{21}(t_{\lambda_2^0}), j_{21}(t_{\lambda_2^0}-1), j_{22}(t_{\lambda_2^0}-2), j_{22}(t_{\lambda_2^0}-3)\rangle) \qquad (77)
\end{aligned}
$$

Note that, as happens in (76), in general there can be input values of $AcyOp_2$ that are not connected to output values of $AcyOp_1$ via ${}^{\circ}_{9\delta}$. In such cases, the occurrence of $W_{AcyOp,2}$ in (77) must be decomposed into the within relations for individual abstract and concrete input pairs as in (32), and only those input pairs connected via ${}^{\circ}_{9\delta}$ would occur in (77). Any remaining input pairs would have to be unconstrained, or alternatively, one would have to rely on the environment to assert the necessary constraints, which is what (76) does. Note that the 'maximally liberal' strategy for within relations in this paper noted earlier, further justified in Section 5.1, is precisely what is needed to make this approach unproblematic.

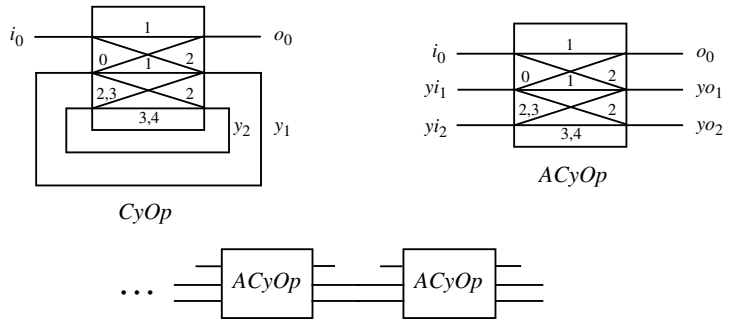

Fig. 14. The *CyOp* and *ACyOp* circuits, and a skew-sequential composition of an unbounded number of copies of *ACyOp*.

B. Synchronous Feedback Systems

In this Section we examine the theory of synchronous feedback systems, embodied in finite clocked cyclic circuits, and we see how to extend the theory of Section 3 to the new context.

Every finite cyclic circuit can be viewed as arising from a suitable finite acyclic circuit, by identifying some of the external output signals with an equal number of external input signals.[19] This leads, unsurprisingly, to a characterisation of the behaviour of the cyclic circuit as a fixed point of a recursive definition.

So let *CyOp* (short for *CyclicOperation*) be a cyclic circuit, and let *ACyOp* be the corresponding acyclic circuit, derived by cutting back-links.[20] In order to reuse as much existing material as possible, *CyOp* will have an input signal i_0 and an output signal o_0 and two back-links, y_1, y_2. When the back-links are cut, *ACyOp* has input signals i_0, yi_1, yi_2, and output signals o_0, yo_1, yo_2. We illustrate all this in Fig. 14, where the internal lines show the potential data dependencies in the two circuits, and the small numbers labelling them denote the delays (in terms of clock ticks) along these dependencies. Fig. 14 also represents *CyOp* as an unbounded number of copies of *ACyOp* skew-sequentially composed together.

Comparing with Fig. 2, which shows the *AcyOp* used in Sections 3 and 4, we see that in *ACyOp* there is a renaming of i_1, i_2, o_1, o_2, to yi_1, yi_2, yo_1, yo_2, and also that we have introduced some additional data dependency paths: one of length 2 from i_0 to yo_1 contributing to $ACyOpS^{12}$, and another of length 1 from i_0 to o_0 contributing to $ACyOpS^0$. (Note that because the set of input-to-output dependencies for an irreducible subrelation is given by adding an input delay to an output delay in all possible ways, as evident from the remarks immediately following (4), as a consequence of the new dependency path of length 2 from i_0 to yo_1 contributing to $ACyOpS^{12}$, there is also an additional data dependency of length 3 from i_0 to yo_2. We do not show this in Fig. 14 to avoid clutter.) Suppose though, that apart from this, the behaviour of *ACyOp* is as for *AcyOp* in Section 3. In particular, this enables us to write down its causal relation immediately:

$$\begin{aligned}
&ACyOpS(\langle i_0(t_{\lambda^0}-1), i_0(t_{\lambda^{12}}-2), yi_1(t_{\lambda^0}), yi_1(t_{\lambda^{12}}-1), yi_2(t_{\lambda^{12}}-2), yi_2(t_{\lambda^{12}}-3)\rangle,\\
&\qquad \langle o_0(t_{\lambda^0}), yo_1(t_{\lambda^{12}}), yo_2(t_{\lambda^{12}}+1)\rangle) \equiv\\
&\quad ACyOpS^0(\langle i_0(t_{\lambda^0}-1), yi_1(t_{\lambda^0})\rangle, o_0(t_{\lambda^0})) \wedge\\
&\quad ACyOpS^{12}(\langle i_0(t_{\lambda^{12}}-2), yi_1(t_{\lambda^{12}}-1), yi_2(t_{\lambda^{12}}-2), yi_2(t_{\lambda^{12}}-3)\rangle, \langle yo_1(t_{\lambda^{12}}), yo_2(t_{\lambda^{12}}+1)\rangle) \wedge\\
&\quad (\lambda^0 - 1 = \lambda^{12} - 2 \Rightarrow i_0(t_{\lambda^0}-1) = i_0(t_{\lambda^{12}}-2)) \wedge\\
&\quad (\lambda^0 = \lambda^{12} - 1 \Rightarrow yi_1(t_{\lambda^0}) = yi_1(t_{\lambda^{12}}-1))
\end{aligned} \tag{78}$$

The goal of this section is to calculate *CyOpS*, the analogous causal relation for *CyOp*, oriented for backwards analysis, on the basis that *ACyOpS* is known. We proceed as follows.

Let $\mathbf{y}$ be the set of names for the back-link signals in *CyOp*, with typical back-link $y \in \mathbf{y}$. (So in Fig. 14, $\mathbf{y} =$

[19] One obtains the required acyclic circuit by simply cutting all the back-links in the cyclic circuit.

[20] Note that our earlier *AcyOp* and the present *ACyOp* are both acyclic circuits. The slight change in typography is intended to hint at the connection with the cyclic case here.

$\{y_1, y_2\}$.) Let **yo** be the set of back-link-generated output streams in *AcyOpS*, with typical back-link-generated output stream $yo_k \in$ **yo**. (So in Fig. 14, **yo** $= \{yo_1, yo_2\}$.) Let **yi** be the set of back-link-generated input streams in *AcyOpS*, with typical back-link-generated input stream $yi_k \in$ **yi**. (So in Fig. 14, **yi** $= \{yi_1, yi_2\}$.)

Writing :: for concatenation of sequences, let **q** be the set of sequences of triples given by:

- **If** (in the representation of (4)), $o_w(t_{\lambda^\gamma} + \phi_w)$ and $i_x(t_{\lambda^\gamma} - \theta_{x.k})$ occur in the signature of some irreducible subrelation $ACyOpS^\gamma$ of *ACyOpS*, with $w \in \gamma$, and where o_w is an external output stream and i_x is either an external or back-link-generated input stream

  Then $\langle\ \langle o_w, \phi_w + \theta_{x.k}, i_x\rangle\ \rangle \in$ **q**
- **If** $q :: \langle o_w, d, yi_x\rangle \in$ **q** and (in the representation of (4)), $yo_x(t_{\lambda^\gamma} + \phi_x)$ and $i_z(t_{\lambda^\gamma} - \theta_{z.k})$ occur in the signature of some irreducible subrelation $ACyOpS^\gamma$ of *ACyOpS*, with $w \in \gamma$, and where o_w is an external or back-link-generated output stream, and yi_x/yo_x are a back-link-generated input/output stream pair, and i_z is either an external or back-link-generated input stream

  Then $q :: \langle o_w, d, yi_x\rangle :: \langle yo_x, \phi_x + \theta_{z.k}, i_z\rangle \in$ **q** (79)

The matching of the back-link y_x in back-link-generated input stream yi_x at the end of one triple with the corresponding back-link-generated output stream yo_x in the next triple in the second clause of (79), makes it clear that the set of sequences in **q** can be interpreted as a forest $\mathcal{F}$, the delay forest, with external outputs o_w as roots, and with branches corresponding to paths cycling round inside *CyOp*. For each $q \in$ **q**, let $dl(q)$, the accumulated delay along q, be given by:

- $dl(\langle\ \langle o_w, d, yi_x\rangle\ \rangle) = d$
- $dl(q :: \langle yo_x, d, i_z\rangle) = dl(q) + d$ (80)

Also, for $q \in$ **q**, let us write $q \equiv \langle o_x \ldots i_z\rangle$, to indicate that q starts with a triple whose first element is o_x, and ends with a triple whose last element is i_z.

Let us now define yd_1, yd_2 as the streams of data values on the back-links of our running example *CyOp*, so that $yd_k(t-q)$ is the data value on the k'th back-link, at a time q delay units before t, and let us consider how a backwards analysis of *CyOp* would go. In a backwards analysis of *CyOp* from time t, we know that:

$$ACyOpS^0(\langle i_0(t-d_{0,0}), yi_1(t-d_{1,0})\rangle, o_0(t)) \tag{81}$$

holds, where in general (in the representation of (4)), $d_{r.k,s} = \phi_s + \theta_{r.k}$ is the delay along a path from input i_r (with input delay parameter $\theta_{r.k}$), to output o_s (with output delay parameter ϕ_s) in *ACyOp*. (Thus $d_{0,0} = 1$ and $d_{1,0} = 0$ in the running example of Fig. 14.)

Similarly, because input yi_1 of $ACyOpS^0$ is not only an input of $ACyOpS^0$ but also corresponds to a severed back-link y_1 of *CyOp*, and therefore to an output yo_1 of $ACyOpS^{12}$, we must have that $ACyOpS^{12}$ is true for a collection of time indexes such that the output yo_1 of $ACyOpS^{12}$ appears just when it is needed for consumption by $ACyOpS^0$. In other words:

$$\begin{aligned} ACyOpS^{12}(&\langle i_0(t-d_{1,0}-d_{0,1}), yi_1(t-d_{1,0}-d_{1,1}), yi_2(t-d_{1,0}-d_{2.1,1}), yi_2(t-d_{1,0}-d_{2.2,1})\rangle,\\ &\langle yo_1(t-d_{1,0}), yo_2(t-d_{1,0}+\phi_2)\rangle) \end{aligned} \tag{82}$$

holds. When we substitute the numerical values for the delays in $ACyOpS^{12}$, namely $d_{0,1} = 2, d_{1,1} = 1, d_{2.1,1} = 2, d_{2.2,1} = 3$, (all relative to $t - d_{1,0}$) for the first four parameters, and $t - d_{1,0} + \phi_2 = t - d_{1,0} + 1$ for the last one (since the output of yo_2 is is not required till one unit later), this becomes:

$$\begin{aligned} ACyOpS^{12}(&\langle i_0(t-d_{1,0}-2), yi_1(t-d_{1,0}-1), yi_2(t-d_{1,0}-2), yi_2(t-d_{1,0}-3)\rangle,\\ &\langle yo_1(t-d_{1,0}), yo_2(t-d_{1,0}+1)\rangle) \end{aligned} \tag{83}$$

Also, in (83), for any t, we must have $yo_1(t) = yi_1(t) = yd_1(t)$, since the data values on yo_1, yi_1, are drawn from the same data steam yd_1 (and similarly for yd_2).

Speaking generically, we must demand an arbitrarily large number of such constraints. For every path back through *CyOp* from the free output o_0, visiting back-links $y_{k_1}, y_{k_2}, y_{k_3} \ldots$, there is a corresponding path back from the free output o_0 of $ACyOpS^0$, through a corresponding sequence of skew-sequentially composed copies of $ACyOpS^{k_1}, ACyOpS^{k_2}$, $ACyOpS^{k_3} \ldots$, such that not only does the relevant $ACyOpS^{k_x}$ hold for the relevant data values at the relevant moments as captured by the signature of $ACyOpS^{k_x}$, but whenever two such constraints generate values for the same stream at the same time index, the two values must be the same.

As it happens, there is only one irreducible subrelation involved in the feedback loops present in our running example of Fig. 14, namely $AcyOpS^{12}$. So, unlike more general cases which will be more cluttered notationally, we can give a reasonably compact and specific form for the causal relation of *CyOp*, i.e. *CyOpS*, in terms of the corresponding acyclic causal relation *ACyOpS*, as follows:

$$
\begin{array}{l}
CyOpS^0(i_0,o_0) \equiv \\
\quad (\exists yd_1,yd_2 \bullet ACyOpS^0(\langle i_0(t-d_{0,0}),yd_1(t-d_{1,0})\rangle,o_0(t)) \wedge \\
\qquad (\forall q \in \mathbf{q} \bullet q \equiv \langle o_0 \ldots yi_z\rangle \wedge yo_z \in \{yo_1,yo_2\} \Rightarrow \\
\qquad\quad ACyOpS^{12}(\langle i_0(t-dl(q)-d_{0,z}),yd_1(t-dl(q)-d_{1,z}),yd_2(t-dl(q)-d_{2.1,z}),yd_2(t-dl(q)-d_{2.2,z})\rangle, \\
\qquad\qquad \langle yd_1(t-dl(q)-\delta_{z,2}),yd_2(t-dl(q)+\delta_{z,1})\rangle))))
\end{array}
\tag{84}
$$

In (84), we have to assert the $ACyOpS^0$ explicitly since it is not covered by the quantifications. Note that the input delays $d_{0,z},d_{1,z},d_{2.1,z},d_{2.2,z}$ in $ACyOpS^{12}$ differ by 1 depending on whether z is 1 or 2. Thus when $z=1$ they are the constants quoted above, whereas if z is 2, they are all greater by 1 because of the output delay $\phi_2=1$. Moreover, the reference time index at which a quantified occurrence of $ACyOpS^{12}$ is attached to the preceding sequence q depends on whether the attachment is via yo_1 or yo_2. If it is via yo_1, then the accumulated delay $t-dl(q)$ refers to the time index at which yo_1 carries the value $yd_1(t-dl(q))$; this is 1 less than the time index at which yo_2 delivered its value $yd_2(t-dl(q)+1)$, because of the output delay $\phi_2=1$. If it is via yo_2, then the accumulated delay $t-dl(q)$ refers to the time index at which yo_2 carries the value $yd_2(t-dl(q))$; which this time is 1 more than the time index at which yo_1 delivered its value $yd_1(t-dl(q)-1)$, because of the same output delay. Both possibilities are covered by the 'delta functions' in (84), where $\delta_{z,k}=1$ if $z=k$ and is 0 otherwise.

We see from (84) how a general case goes. If we are interested in the possible causes of some external outputs of an arbitrary cyclic circuit, we have to trace back through all branches of $\mathcal{F}$, and assert the appropriate irreducible subrelation output at the time index corresponding to the leaf of the branch in question; this determines the time indexes of the irreducible subrelation's inputs and of any other outputs it may have. As a technical detail, $\mathcal{F}$ does not contain any sequences of length 0, so the top level irreducible subrelation(s) must be asserted explicitly. The fact that the external input and output streams i_0,o_0 are free variables, together with the outer level existential quantification over the back-link data streams yd_1,yd_2, ensure that whenever two different constraints generated in this manner involve the same stream at the same time index, they demand the same data value, regardless of which branches of $\mathcal{F}$ they arose from. These technical devices play a role here, analogous to that played by the explicit implications in (1), (7) and (78). Thus we see that (84) is self-referential, but not in a naively recursive manner.

Also observe that the specification in (84) need not be realisable. For instance, it can describe an instantaneous inverter whose output is wired straight back into its input, a situation with no solutions, since it demands that $\mathsf{true}=\mathsf{false}$. There are also less clear-cut cases, such as two such inverters in series, with the second inverter's output connected to the first inverter's input. Formulated according to (84), this has two valid solutions: inverter 1 inverts true to false, and inverter 2 inverts false to true; or *vice versa*. Whether such behaviour, not stabilised by the presence of any memory elements, is in fact realisable in practice or not, depends critically on the device physics of any putative implementation, and, as such, is beyond the scope of the analysis of this paper.

C. Retrenchment for Synchronous Feedback Systems

Let us now regard the preceding material as the abstract level of a retrenchment. More precisely, we add a subscript '$_A$' to identifiers $CyOp,AcyOp,ACyOpS,ACyOpS^0,\ldots\mathbf{q},\mathcal{F},\ldots$ occurring in Appendix B. The concrete level will be given by a circuit $CyOp_C$, similar to $CyOp_A$, with back-link-severed acyclic form $ACyOp_C$. The back-link-generated inputs and outputs will be named yj_1,yj_2,yp_1,yp_2, so that $ACyOp_C$ will have causal relation:

$$
\begin{array}{l}
ACyOpS_C(\langle j_0(t_{\lambda^0}-1),j_0(t_{\lambda^{12}}-2),yj_1(t_{\lambda^0}),yj_1(t_{\lambda^{12}}-1),yj_2(t_{\lambda^{12}}-2),yj_2(t_{\lambda^{12}}-3)\rangle, \\
\qquad \langle p_0(t_{\lambda^0}),yp_1(t_{\lambda^{12}}),yp_2(t_{\lambda^{12}}+1)\rangle) \equiv \\
\quad ACyOpS^0_C(\langle j_0(t_{\lambda^0}-1),yj_1(t_{\lambda^0})\rangle,p_0(t_{\lambda^0})) \wedge \\
\quad ACyOpS^{12}_C(\langle j_0(t_{\lambda^{12}}-2),yj_1(t_{\lambda^{12}}-1),yj_2(t_{\lambda^{12}}-2),yj_2(t_{\lambda^{12}}-3)\rangle,\langle yp_1(t_{\lambda^{12}}),yp_2(t_{\lambda^{12}}+1)\rangle) \wedge \\
\quad (\lambda^0-1=\lambda^{12}-2 \Rightarrow j_0(t_{\lambda^0}-1)=j_0(t_{\lambda^{12}}-2)) \wedge
\end{array}
$$

$$(\lambda^0 = \lambda^{12} - 1 \Rightarrow yj_1(t_{\lambda^0}) = yj_1(t_{\lambda^{12}} - 1)) \tag{85}$$

Note that the concrete delay forest $\mathcal{F}_C$ only differs from the abstract one $\mathcal{F}_A$ in that concrete rather than abstract inputs and outputs occur — the delay structure is exactly the same. This leads to a definition of $CyOp_C$ in terms of $ACyOpS_C$ analogous to (84), but with suitable '$_C$' subscripts etc.:

$$\begin{array}{l}
CyOpS^0_C(j_0,p_0) \equiv \\
\quad (\exists ye_1,ye_2 \bullet ACyOpS^0_C(\langle j_0(t-d_{0,0}),ye_1(t-d_{1,0})\rangle,p_0(t)) \wedge \\
\quad\quad (\forall q_C \in \mathbf{q}_C \bullet q_C \equiv \langle p_0 \ldots yj_z\rangle \wedge yp_z \in \{yp_1,yp_2\} \Rightarrow \\
\quad\quad\quad ACyOpS^{12}_C(\langle j_0(t-dl(q_C)-d_{0,z}),ye_1(t-dl(q_C)-d_{1,z}),ye_2(t-dl(q_C)-d_{2.1,z}),ye_2(t-dl(q_C)-d_{2.2,z})\rangle, \\
\quad\quad\quad\quad \langle ye_1(t-dl(q_C)-\delta_{z,2}),ye_2(t-dl(q_C)+\delta_{z,1})\rangle))))
\end{array} \tag{86}$$

We will assume retrenchment POs between the irreducible subrelations of $ACyOpS_A$ and $ACyOpS_C$ given by:

$$\begin{array}{l}
W_{ACyOpS^0}(\langle i_0(t_{\lambda^0}-1),yi_1(t_{\lambda^0})\rangle,\langle j_0(t_{\lambda^0}-1),yj_1(t_{\lambda^0})\rangle) \wedge ACyOpS^0_C(\langle j_0(t_{\lambda^0}-1),yj_1(t_{\lambda^0})\rangle,p_0(t_{\lambda^0})) \Rightarrow \\
\quad (\exists\ o_0(t_{\lambda^0}) \bullet ACyOpS^0_A(\langle i_0(t_{\lambda^0}-1),yi_1(t_{\lambda^0})\rangle,o_0(t_{\lambda^0})) \wedge \\
\quad\quad (O_{ACyOpS^0}(o_0(t_{\lambda^0}),p_0(t_{\lambda^0}),\langle i_0(t_{\lambda^0}-1),yi_1(t_{\lambda^0})\rangle,\langle j_0(t_{\lambda^0}-1),yj_1(t_{\lambda^0})\rangle) \vee \\
\quad\quad C_{ACyOpS^0}(o_0(t_{\lambda^0}),p_0(t_{\lambda^0}),\langle i_0(t_{\lambda^0}-1),yi_1(t_{\lambda^0})\rangle,\langle j_0(t_{\lambda^0}-1),yj_1(t_{\lambda^0})\rangle))
\end{array} \tag{87}$$

$$\begin{array}{l}
W_{ACyOpS^{12}}(\langle i_0(t_{\lambda^{12}}-2),yi_1(t_{\lambda^{12}}-1),yi_2(t_{\lambda^{12}}-2),yi_2(t_{\lambda^{12}}-3)\rangle, \\
\qquad \langle j_0(t_{\lambda^{12}}-2),yj_1(t_{\lambda^{12}}-1),yj_2(t_{\lambda^{12}}-2),yj_2(t_{\lambda^{12}}-3)\rangle) \wedge \\
ACyOpS^{12}_C(\langle j_0(t_{\lambda^{12}}-2),yj_1(t_{\lambda^{12}}-1),yj_2(t_{\lambda^{12}}-2),yj_2(t_{\lambda^{12}}-3)\rangle,\langle yp_1(t_{\lambda^{12}}),yp_2(t_{\lambda^{12}}+1)\rangle) \Rightarrow \\
\quad (\exists\ yo_1(t_{\lambda^{12}}),yo_2(t_{\lambda^{12}}+1) \bullet \\
\quad\quad ACyOpS^{12}_A(\langle i_0(t_{\lambda^{12}}-2),yi_1(t_{\lambda^{12}}-1),yi_2(t_{\lambda^{12}}-2),yi_2(t_{\lambda^{12}}-3)\rangle,\langle yo_1(t_{\lambda^{12}}),yo_2(t_{\lambda^{12}}+1)\rangle) \wedge \\
\quad\quad (O_{ACyOpS^{12}}(\langle yo_1(t_{\lambda^{12}}),yo_2(t_{\lambda^{12}}+1)\rangle,\langle yp_1(t_{\lambda^{12}}),yp_2(t_{\lambda^{12}}+1)\rangle, \\
\quad\quad\quad\quad \langle i_0(t_{\lambda^{12}}-2),yi_1(t_{\lambda^{12}}-1),yi_2(t_{\lambda^{12}}-2),yi_2(t_{\lambda^{12}}-3)\rangle, \\
\quad\quad\quad\quad \langle j_0(t_{\lambda^{12}}-2),yj_1(t_{\lambda^{12}}-1),yj_2(t_{\lambda^{12}}-2),yj_2(t_{\lambda^{12}}-3)\rangle) \vee \\
\quad\quad C_{ACyOpS^{12}}(\langle yo_1(t_{\lambda^{12}}),yo_2(t_{\lambda^{12}}+1)\rangle,\langle yp_1(t_{\lambda^{12}}),yp_2(t_{\lambda^{12}}+1)\rangle, \\
\quad\quad\quad\quad \langle i_0(t_{\lambda^{12}}-2),yi_1(t_{\lambda^{12}}-1),yi_2(t_{\lambda^{12}}-2),yi_2(t_{\lambda^{12}}-3)\rangle, \\
\quad\quad\quad\quad \langle j_0(t_{\lambda^{12}}-2),yj_1(t_{\lambda^{12}}-1),yj_2(t_{\lambda^{12}}-2),yj_2(t_{\lambda^{12}}-3)\rangle))))
\end{array} \tag{88}$$

The acyclic retrenchments (87)-(88) may be combined to give something similar to (33). Of more interest now however, is the combination needed to give a cyclic retrenchment, one from $CyOpS^0_A$ to $CyOpS^0_C$. For this we define retrenchment data as follows. We start with the within relation:

$$\begin{array}{l}
W_{CyOp^0}(i_0,j_0) \equiv \\
\quad (\forall ye_1,ye_2 \bullet \exists yd_1,yd_2 \bullet \\
\quad\quad W_{AcyOpS^{0.0}}(i_0(t-d_{0,0}),j_0(t-d_{0,0})) \wedge W_{AcyOpS^{0.1}}(yd_1(t-d_{1,0}),ye_1(t-d_{1,0})) \wedge \\
\quad\quad (\forall q_A \in \mathbf{q}_A, q_C \in \mathbf{q}_C \bullet q_A \equiv \langle yo_1 \ldots yi_z\rangle \cong \langle zo_1 \ldots zi_z\rangle \equiv q_C \Rightarrow \\
\quad\quad\quad W_{AcyOpS^{12.0}}(i_0(t-dl(q_A)-d_{0,z}),j_0(t-dl(q_C)-d_{0,z})) \wedge \\
\quad\quad\quad W_{AcyOpS^{12.1}}(yd_1(t-dl(q_A)-d_{1,z}),ye_1(t-dl(q_C)-d_{1,z})) \wedge \\
\quad\quad\quad W_{AcyOpS^{12.2}}(yd_2(t-dl(q_A)-d_{2.1,z}),ye_2(t-dl(q_C)-d_{2.1,z})) \wedge \\
\quad\quad\quad W_{AcyOpS^{12.2}}(yd_2(t-dl(q_A)-d_{2.2,z}),ye_2(t-dl(q_C)-d_{2.2,z}))))
\end{array} \tag{89}$$

Note that the relation $\cong$ in (89) denotes that the two paths q_A and q_C through $\mathcal{F}_A$ and $\mathcal{F}_C$ are congruent, i.e.: they are the same length; at each position of the two sequences the abstract and concrete triples relate corresponding outputs

and inputs; and (as assumed above), the delays at each position are the same. The delays themselves depend on which output of $AcyOpS^{12}$ the paths q_A and q_C arrive at. Also, (89) features the within relations for individual input pairs, $W_{AcyOpS^{-,-}}$, rather than those for the irreducible subrelations appearing in (87)-(88), according to the convention established in (32).

Note also that this cyclic within relation demands that the acyclic within constraints are demanded an arbitrary number of times into the past. In view of the fact that skew-sequential composition (76)-(77) reduces the composed within to the within of the earlier operation, this might seem surprising. It might appear that it should be enough to demand the within constraint once, early enough, or even to take the limit of demanding the within constraint infinitely far into the past. Unfortunately, for the former, there is no one moment in the past that will necessarily suffice for arbitrary analyses (since these might penetrate arbitrarily deep into the past). For the latter, the conjectured limit may or may not exist (depending on topological considerations — though since any fault in a real system will have a cause in the finite past, any non-existence of the limit is, in itself, no practical obstacle). Demanding the acyclic within constraint arbitrarily often into the finite past thus makes the cyclic within constraint analogous to a fairness constraint.

Next comes the output relation:

$$\begin{array}{l} O_{CyOp^0}(o_0,p_0,i_0,j_0) \equiv \\ \quad (\exists yd_1,yd_2,ye_1,ye_2 \bullet \\ \qquad O_{AcyOpS^0}(o_0(t),p_0(t),\langle i_0(t-d_{0,0}),yd_1(t-d_{1,0})\rangle,\langle j_0(t-d_{0,0}),ye_1(t-d_{1,0})\rangle) \wedge \\ \qquad (\forall q_A \in \mathbf{q}_A, q_C \in \mathbf{q}_C \bullet q_A \equiv \langle yo_1 \ldots yi_z\rangle \cong \langle zo_1 \ldots zi_z\rangle \equiv q_C \Rightarrow \\ \qquad\quad O_{AcyOpS^{12}}(\langle yd_1(t-dl(q_A)-\delta_{z,2}),yd_2(t-dl(q_A)+\delta_{z,1})\rangle,\langle ye_1(t-dl(q_C)-\delta_{z,2}),ye_2(t-dl(q_C)+\delta_{z,1})\rangle, \\ \qquad\qquad\quad \langle i_0(t-dl(q_A)-d_{0,z}),yd_1(t-dl(q_A)-d_{1,z}),yd_2(t-dl(q_A)-d_{2.1,z}),yd_2(t-dl(q_A)-d_{2.2,z})\rangle, \\ \qquad\qquad\quad \langle j_0(t-dl(q_C)-d_{0,z}),ye_1(t-dl(q_C)-d_{1,z}),ye_2(t-dl(q_C)-d_{2.1,z}),ye_2(t-dl(q_C)-d_{2.2,z})\rangle)))) \end{array} \tag{90}$$

The output relation simply demands that the output relation for the acyclic version holds at all moments in the past that a backwards analysis could reach. This is the infinitary analogue of (40) for skew-sequential composition.

Finally the concedes relation, in which all occurrences of C_{AcyOpS^0} and O_{AcyOpS^0} have the same signature as O_{AcyOpS^0} in (90), and all occurrences of $C_{AcyOpS^{12}}$ and $O_{AcyOpS^{12}}$ have the same signature as $O_{AcyOpS^{12}}$ in (90):

$$\begin{array}{l} C_{CyOp^0}(o_0,p_0,i_0,j_0) \equiv \\ \quad (\exists yd_1,yd_2,ye_1,ye_2 \bullet \\ \qquad (C_{AcyOpS^0} \wedge (\forall q_A \in \mathbf{q}_A, q_C \in \mathbf{q}_C \bullet q_A \equiv \langle yo_1 \ldots yi_z\rangle \cong \langle zo_1 \ldots zi_z\rangle \equiv q_C \Rightarrow O_{AcyOpS^{12}} \vee C_{AcyOpS^{12}})) \vee \\ \qquad (O_{AcyOpS^0} \wedge (\exists p_A \in \mathbf{q}_A, p_C \in \mathbf{q}_C \bullet p_A \cong p_C \wedge \\ \qquad\quad (\forall q_A \in \mathbf{q}_A, q_C \in \mathbf{q}_C \bullet q_A \equiv \langle yo_1 \ldots yi_z\rangle \cong \langle zo_1 \ldots zi_z\rangle \equiv q_C \Rightarrow \\ \qquad\qquad (p_A = q_A \cong q_C = p_C \Rightarrow C_{AcyOpS^{12}}) \wedge (p_A \neq q_A \cong q_C \neq p_C \Rightarrow O_{AcyOpS^{12}} \vee C_{AcyOpS^{12}}))))) \end{array} \tag{91}$$

For the concedes relation, the analogy with the finite sequential composition case that we seek, is that for some iteration of the acyclic system unwound arbitrarily far into the past, the concession assuredly holds. So, for the top level irreducible subrelation $AcyOpS^0$, either C holds or O holds, giving the top level disjunction in (91). If C holds for $AcyOpS^0$, then the branches below can satisfy $C \vee O$ without further constraint. If O holds for $AcyOpS^0$, then lower down, there must be some branch (identified via the existentially quantified p_A, p_C) for which we are able to assert C, and all other branches can satisfy $C \vee O$ without additional constraint.

We regard the feedback composition of retrenchments as a generalisation of skew-sequential composition. Therefore, whenever the after-values of one $AcyOpS$ step feed into the before-values of a following one, a constraint analogous to (77) must hold. We write this as:

$$\begin{array}{l} (\langle o_0,0,yi_1\rangle = q_A \cong q_C = \langle p_0,0,zj_1\rangle \Rightarrow (\forall \ \ldots \ O_{AcyOpS^{12}} \vee C_{AcyOpS^{12}} \Rightarrow W_{AcyOpS^{0.1}})) \wedge \\ (\forall q_A \in \mathbf{q}_A, q_C \in \mathbf{q}_C \bullet (\langle o_0,0,yi_1\rangle \neq q_A \cong q_C \neq \langle p_0,0,zj_1\rangle \Rightarrow \\ \quad (\forall \ \ldots \ O_{AcyOpS^{12}} \vee C_{AcyOpS^{12}} \Rightarrow W_{AcyOpS^{12.1}} \wedge W_{AcyOpS^{12.2}})) \end{array} \tag{92}$$

The first conjunct of (92) says that the outputs of the last $AcyOpS^{12}$ in the computation fall inside the within relation for the yi_1/yj_1 input pair of the $AcyOpS^0$s leading to the free o_0/p_0 outputs. The '$\forall \ \ldots$' quantifies over all data variables

occurring in $O_{AcyOpS^{12}}, C_{AcyOpS^{12}}, W_{AcyOpS^{0.1}}$ that are not common to the two subformulae $O_{AcyOpS^{12}} \vee C_{AcyOpS^{12}}$ and $W_{AcyOpS^{0.1}}$. Similarly, the second conjunct says that whenever a congruent pair of paths q_A and q_C identify a place where the outputs of one $AcyOpS^{12}$ step feed into the inputs of a subsequent $AcyOpS^{12}$ step, the output values on yo_1/yp_1 fall inside the within relation $W_{AcyOpS^{12.1}}$ and those on yo_2/yp_2 fall inside $W_{AcyOpS^{12.2}}$. Again, the '$\forall \ldots$' quantifies over all data variables occurring in (the identified occurrences of) $O_{AcyOpS^{12}} \vee C_{AcyOpS^{12}}$ and $W_{AcyOpS^{12.1}} \wedge W_{AcyOpS^{12.2}}$ which do not occur in both. As for the formulae appearing earlier, the specific shape of our running example simplifies the form of (92) compared with the more general case.

One can now write down an innocuous looking retrenchment PO involving these quantities:

$$W_{CyOp^0}(i_0,j_0) \wedge CyOpS^0_C(j_0,p_0) \Rightarrow (\exists o_0 \bullet CyOpS^0_A(i_0,o_0) \wedge (O_{CyOp^0}(o_0,p_0,i_0,j_0) \vee C_{CyOp^0}(o_0,p_0,i_0,j_0))) \quad (93)$$

and here is the corresponding simulation relation:

$$\Sigma \equiv W_{CyOp^0}(i_0,j_0) \wedge CyOpS^0_C(j_0,p_0) \wedge CyOpS^0_A(i_0,o_0) \wedge (O_{CyOp^0}(o_0,p_0,i_0,j_0) \vee C_{CyOp^0}(o_0,p_0,i_0,j_0)) \quad (94)$$

Since it deals with infinitary quantities, the soundness of (93) is not completely trivial, and it is examined more closely in the next section. Note that we need to confirm soundness to ensure that the posited within, output and concedes relations do indeed define a retrenchment, so that (an appropriate extension of) the soundness and completeness results for fault analysis via retrenchment do in fact hold for the feedback case (see below).

C.1. Soundness of the Infinitary PO

A retrenchment (for a given pair of abstract and concrete systems, or class of such system pairs) is sound iff the retrenchment POs are provable. For compositions (of operations and their corresponding retrenchments) this amounts to showing that the provability of the composed POs follows from that of the provability of the POs for the ingredients of the composition. For *finitary* compositions of operations and their corresponding retrenchments, the soundness issues are relatively straightforward and are dealt with by techniques covered adequately in [BJP08]. (The general idea is that the hypotheses of the PO assert the composed concrete operation, so one decomposes this, and starting at the earliest relevant moment, exploits the PO for the individual components, reasoning forwards, and extracting appropriate witness abstract operation steps and abstract after-states. These are then combined to give a composed abstract operation, and the component retrenchment data are similarly combined to yield composed retrenchment data in the required form.) Unfortunately this approach does not work for (93) since there is no 'earliest moment' to start from in an analysis stretching back indefinitely far into the past. Instead, to attack the soundness issue for the infinitary compositions discussed above, we must proceed as follows.

The hypotheses of (93) assert that we have $CyOpS^0_C(j_0,p_0)$ and $W_{CyOp^0}(i_0,j_0)$. The structure of $CyOpS^0_C(j_0,p_0)$ in (86) asserts the free input and output streams j_0 and p_0, and the existentially quantified back-link data streams zd_1, zd_2. The structure of W_{CyOp^0} ensures that for any given ye_1, ye_2, appropriate yd_1, yd_2 can be found, that together with the free input stream i_0, make the within relations for individual pairs of free or quantified values valid.

Now $CyOpS^0_C$ consists of many instances of the acyclic irreducible subrelations $AcyOpS^\gamma_C$ acting at time indexes determined by the delay forest $\mathcal{F}_C$. If we take any particular $AcyOpS^\gamma_C$ instance inside $CyOpS^0_C$, together with the individual within relations for its input values, we get the assumptions for an instance of the finitary acyclic $AcyOpS^\gamma$ retrenchment PO. The conclusions of this PO instance yield an $AcyOpS^\gamma_A$ step, which gives a value for any free output stream, e.g. o_0, and values for the back-link-generated outputs, e.g. yd_1, yd_2 of the step. While an o_0 value is not constrained further (except that we know that it must satisfy $O_{AcyOpS^\gamma} \vee C_{AcyOpS^\gamma}$), the challenge is to ensure that a consistent pair of yd_1, yd_2 streams can be formed from the individual yd_1, yd_2 values at particular time indexes produced and consumed by the various (thus far unconstrained) $AcyOpS^\gamma_A$ instances, so that they can form the required $CyOpS^0_A$ cyclic computation. We develop the argument as follows, starting with a couple of technical definitions.

Definition C.1 (Cousin, Sibling, Good Set, Age). Let $ACyOpS^\chi$ be an irreducible subrelation, and let $\mathbf{q}^\chi_p$ be the paths $q \equiv \langle p \ldots j \rangle$ in the delay forest $\mathcal{F}_C$ derived from $CyOp_C$, with p a fixed output of $ACyOpS^\chi_C$. For such a q, we say that q reaches j after $dl(q)$ from p.

- If $q_1, q_2 \in \mathbf{q}^\chi_p$, and both reach j after d from p, we say that q_1 and q_2 are cousins.
- If $q \in \mathbf{q}^\chi_p$ and $q_1 \equiv q :: \langle p_k, d_x, j_x \rangle$ and $q_2 \equiv q :: \langle p_k, d_y, j_y \rangle$ are extensions of q (so that d_x is the delay from input j_x

to p_k through some irreducible subrelation $ACyOpS^{\gamma}_{C}$, and d_y is the delay from input j_y to p_k through $ACyOpS^{\gamma}_{C}$), we say that q_1 and q_2 are siblings (of each other).

- The sibling and cousin relations are reflexive.
- A set D of paths in $\mathbf{q}^{\chi}_{p}$ is good, iff:
  1. D is finite, ordered by prefix, and prefix closed.
  2. If $q \in D$ then all siblings and cousins of q are also in D.
- The age of a good set D is $\min(\{dl(q) \mid q \text{ is maximal in } D\})$.

Thus the leaf nodes of two cousin paths in a good set D are roots of two identical subtrees; and there is a bijection between their descendants in these two subtrees which itself identifies cousins. The cousin relation is obviously symmetric and transitive as well as reflexive, so it generates an equivalence relation on D in which each equivalence class consists of mutual cousins.

Definition C.2. $\mathcal{D}$ is the set of equivalence classes of a good set D, ordered by the descendant relation inherited from D, and inheriting the sibling relation from D. It is clear that the elements of $\mathcal{D}$ are canonically represented by pairs $(StrId, Del)$ such that there is a backward path from p that reaches the stream identified by *StrId*, after delay *Del*. We call such a (canonical representation of) $\mathcal{D}$ a good set of (stream-identifier, delay) pairs.

The elements of a good set $\mathcal{D}$ identify occurrences of inputs and outputs of irreducible subrelations in the unwinding of a cyclic circuit into an unbounded skew-sequential composition of its acyclic counterpart (see Fig. 14). The 'good' property ensures that if any single input of an occurrence of an irreducible subrelation in the unwinding is identified, then *all* inputs are so identified. Moreover, good sets of arbitrarily large ages obviously exist, got by simply extending paths in D indefinitely far into the past, and in doing so, including all the other paths required by the cousin and sibling relations (and then factoring out by the cousin relation).

If we have retrenchments between abstract and concrete versions of the irreducible subrelations of *CyOp*, we can build a simulation of a finite piece of computation of the concrete $CyOp_C$, described by a good set $\mathcal{D}$ of paths, by following procedure $SimCyOp_C$ given in Fig. 15. The procedure uses a colouring of the pairs in $\mathcal{D}$ using four colours: red, blue, green, black. Roughly speaking, red refers to a piece of the simulation that depends on at least one as yet uncompleted earlier portion; blue refers to completed earlier portions that are ready to be extended; green to a place at which there is a free choice to be made; black to a place at which there is no more work to do.

We indicate briefly why procedure $SimCyOp_C$ works. It starts with all maximal elements of $\mathcal{D}$ green, the rest red. So any irreducible subrelation with concrete inputs at these maximal (green) elements does not depend on any prior steps, and can be simulated. Choosing one such subrelation, abstract inputs are chosen for it (step 4), and it can then be simulated using the retrenchment PO. The elements of $\mathcal{D}$ that have had abstract values chosen or produced for them are then coloured blue, indicating that an abstract value exists — unless, for inputs, the input is not needed for any other simulation step, or for outputs, the end of the simulation has been reached, in which case they can be coloured black to remove them from further consideration. Since we are dealing with a finite object $\mathcal{D}$, and are removing finite pieces from it at each stage, it is clear that $SimCyOp_C$ terminates. It is further clear that removal of the black elements from $\mathcal{D}_{\overline{B}}$ preserves the properties of $\mathcal{D}$ that are needed to make procedure $SimCyOp_C$ sound.

When all delays along cyclic paths through the cyclic circuit are non-zero, the preceding yields a finite simulation of a portion of a possibly infinitary computation, since all nodes of $\mathcal{D}$ represent distinct data streams at distinct time indexes. However if any cyclic path has delay zero, then procedure $SimCyOp_C$ is unsound, since a choice of an input (for some irreducible subrelation) might also amount to a choice of an output, and this choice need not satisfy the retrenchment PO for $ACyOp_C$. If it does not, then this simulation attempt must be abandoned. We have not included the relevant case analysis in $SimCyOp_C$, both for clarity, and because such situations are normally excluded from genuine hardware designs (cf. the last paragraph of Appendix B).

Now, given a valid simulation as above, let us represent the abstract data that it produces, namely values for abstract data streams at specific time indexes, as a finite sequence, *stvalprs*, of *sets of* $(StrId, value)$ *pairs* going towards the past, such that at each index of *stvalprs*, the set contains exactly the $(StrId, value)$ pairs that were assigned at that time index during procedure $SimCyOp_C$ (whether by choice to satisfy a within relation or by computation to satisfy the PO).

Now consider the collection of representing sequences that can be produced in the above manner by considering all possible simulations of finite pieces of $CyOp_C$ for all possible $\mathcal{D}$, making any available choices in all possible ways. These can be arranged into a forest by sharing maximal common prefixes (corresponding to identically set data values for abstract data streams for some final portion(s) of the simulation(s)). Since $ACyOpS_C$ has a finite number of inputs

Procedure *SimCyOp*$_C$
Input: A computation of *CyOp*$_C$, a good set $\mathcal{D}$, retrenchments for the irreducible subrelations of *CyOp*.
Output: A finite computation of *CyOp*$_A$, simulating the portion of the *CyOp*$_C$ computation given by $\mathcal{D}$.
Begin
1 Colour all maximal elements of $\mathcal{D}$ green. Colour all non-maximal elements of $\mathcal{D}$ red. Let $\mathcal{D}_{\overline{\mathrm{B}}}$ be the non-black elements of $\mathcal{D}$.
2 **While** $\mathcal{D}_{\overline{\mathrm{B}}}$ is nonempty
3 **Do** Choose an irreducible subrelation $ACyOpS^{\gamma}_{C}$ each of whose inputs $(x,d_x)\ldots(y,d_y)$ is green or blue, and whose outputs are red.
4 **ForAll** green pairs in $(x,d_x)\ldots(y,d_y)$
5 **Do** Choose an abstract input value for $ACyOpS^{\gamma}_{A}$ that satisfies the relevant pairwise within relation for the given inputs of $ACyOpS^{\gamma}_{C}$ and $ACyOpS^{\gamma}_{A}$.
6 **EndForAll**
7 Using the abstract input values just chosen for green pairs, existing abstract input values for blue pairs, and the retrenchment PO for $ACyOpS^{\gamma}$, construct a step of $ACyOpS^{\gamma}_{A}$, that simulates the step of $ACyOpS^{\gamma}_{C}$ inside *CyOp*$_C$ issuing from $(x,d_x)\ldots(y,d_y)$.
8 **ForAll** $s \in (x,d_x)\ldots(y,d_y)$
9 **Do If** s is an input of an irreducible subrelation $ACyOpS^{\delta}_{C} \neq ACyOpS^{\gamma}_{C}$ which has a red output
10 **Then** colour s blue **Else** colour s black
11 **EndIf**
12 **EndForAll**
13 **ForAll** t at which outputs of $ACyOpS^{\gamma}_{C}$ from inputs at $(x,d_x)\ldots(y,d_y)$ are produced
14 **Do If** t has a red ancestor
15 **Then** colour t blue **Else** colour t black
16 **EndIf**
17 **EndForAll**
18 **EndWhile**
End

Fig. 15. Simulating a computation of *CyOp*$_C$ on a good set $\mathcal{D}$.

and outputs, and all data types are finite, the forest is finitely branching. Since $\mathcal{D}$ of arbitrarily large ages exist, the forest is infinite. We can therefore apply König's Lemma to conclude that it has an infinite branch. This infinite branch represents a computation of *CyOp*$_A$, which simulates *CyOp*$_C$, and by construction satisfies the PO (93).

Observe that in the preceding, the construction of each finite simulation runs forward in time, like any normal simulation. However the forest construction stretches from time index 0 towards the past. This dissonance is why König's Lemma is needed to complete the argument.

N. B. The preceding assumed exactly one output variable p in exactly one irreducible subrelation $ACyOpS^{\chi}$ of *CyOp*$_C$, as in our running example $CyOpS^0$. More general cases have slightly more complicated data, but the essence of the argument, i.e. the construction of an infinite but finitely branching forest, and the application of König's Lemma, remains the same.

C.2. Compositions for Synchronous Feedback Systems and their Retrenchments

Thus far we have worked hard to establish the soundness of the causal relation representation for cyclic systems and for retrenchments between such specifications, in terms of the corresponding acyclic counterparts whose properties we explored at length above. The most important such properties are the compositionality properties, which we now address in the cyclic case.

In one sense, there is little to say. Since we have reduced cyclic systems to acyclic ones, the theory for acyclic systems should carry over. That is certainly the intention. However, the signature of the acyclic unwinding of a cyclic causal relation is generally infinite, since, although we focus on a finite collection of output values, the backwards analysis reveals that the output values can depend on an infinite number of input values. We comment on the consequences of this.

Consider the causal relation of the skew-sequential composition of two cyclic circuits, in which some output signals of the first circuit feed in to some input signals of the second circuit. Because of the feedback in both circuits, an infinite number of copies of both will be needed in general. The skew-sequential composition will thus acquire the character of (84), in which the infinity of copies will be constrained by the requirement that there exist well defined streams for the following: the back-links of the second circuit; the back-links of the first circuit; and the interface signals that feed values from the first to the second. We omit citing the rather complicated explicit formulae. For skew-parallel composition, the situation is slightly simpler, in that there are no interface signals to consider.

Turning to retrenchments between compositions, the situations of skew-sequential and skew-parallel compositions are broadly similar, bearing in mind the additional combinatorial effects of composing retrenchment data. Thus, deriving the composed output relation is the simplest case, being a straightforward extrapolation of circuit composition

(analogous to the way that (90) is an evident extrapolation of (84)). The composed concession has the character of (91), i.e. for every delay path through the unwinding of the relevant composition, there has to be a term that asserts the appropriate concession, and otherwise, either the concession or output relation holds. Finally, the within relation has the character of (89), asserting pairwise within relations for all relevant input value pairs — and the corresponding compatibility conditions, as in (92), must hold. Given the complexity of the formulae that would express all of this, the reader will forgive us for not writing out all the details.

Of at least as much interest as the formulae themselves is the issue of their practicable usability. In the acyclic cases, one can envisage using the corresponding formulae directly for e.g. FT construction. In the cyclic case, such direct use is obviously impossible since one cannot fully construct the infinite objects in finite time.