DESIGNING CIRCUITS BY CALCULATION

by

Geraint Jones

Oxford University Computing Laboratory
Programming Research Group

Designing circuits by calculation

by

Geraint Jones

9 April 1990

Oxford University Computing Laboratory
Programming Research Group
11 Keble Road
Oxford OX1 3QD

Contents

Chapter 1

Introduction

The business of making integrated circuits is a peculiarly difficult engineering task largely because it spans such a wide range of levels of abstraction. Figure 1.1 suggests the sorts of names that might be used to describe these layers. A good design process must necessarily insulate each step of the design – as far as is possible – from the concerns that are most readily understood and tackled at other levels. As used here, the word *designing* simply means making progress down the spectrum from specification to implementation.

On the whole, we are not going to be concerned with details of implementation like gates and transistors. Neither are we going to be concerned with the business of capturing a precise specification, which is largely independent of the sort of thing being specified, and so is properly the subject of other courses. So, in these notes, the word *circuit* means no more that the sort of algorithm that might well be implemented by something like an integrated circuit or a collection of them. Designing circuits is the business of filling in the gap in the figure between specification capture and circuit fabrication.

That means, for example, that we will be interested in highly parallel algorithms

Vague specification	↓ capture
Non-executable specification	
Abstract algorithm	
Word-level algorithm	
Bit-level algorithm	
Gates	↓ fabrication
Transistors	
Rectangles	
Silicon	

Figure 1.1: the design process

with a more-or-less static structure; also that we are going to be concerned to minimize expensive sorts of communication, like broadcasting, and in emphasizing locality. We will not, however, be at all concerned with the details of any fabrication technology; nor in such concerns as how many transistors can be fabricated on a single chip. These things are again properly the concerns of other courses. Indeed it will often make sense to divert the later stages of a design, proceeding not through gates, transistors and silicon, but rather through (say) processes and channels, processors and inter-process links to an occam program that can be executed on an array of transputers.

It is quite common for the design process to be realized by a number of steps passing from one level to a lower level, where the designer 'invents' the implementation at the next level and then proves – after the fact – that this new design does indeed implement the specification at the preceding level. These *proof* steps would be upwards in the figure.

On the other hand, some steps of the design process are implemented by a purely mechanical transformation of specification into implementation: for example, the turning of a set of masks into a pattern on silicon. These can be thought of as *compilation* steps, and require no ingenuity or invention. Indeed, they are usually tedious and are best left to be done by a machine.

The use of the word *calculation* in these notes means stages in the design process which proceed forwards, without large inventive leaps and subsequent proofs of correctness, but without necessarily being purely mechanical.

1.1 A sense of correctness

In order to be able to speak of the correctness of an implementation, it is necessary to have a clear idea of what the specification requires. When discussing software, the specification of a program is usually something like a relationship which its output must have to its input. A correct implementation of a specification is one which when executed realises that relationship.

For example, in order to talk about correctness of an implementation of a programming language we need to give a semantics for the language. This would give a 'standard' meaning to any particular program. The compiler will translate that program into a corresponding piece of code. If the compiler is to be correct, executing that code has to have the effect described by the standard interpretation of the program. That is to say, the diagram in figure 1.2 commutes. If the diagram commutes for every program, then the compiler is correct. In this case the translation implemented by the compiler is a 'partial interpretation' of the programming language: given any program, it turns it into a code that can be used to reconstruct the meaning of the program.

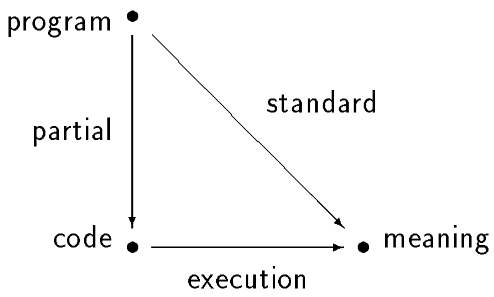

Figure 1.2: the soundness of a partial interpretation

Exactly the same argument explains what it means to implement correct circuits. We will start from a program in a declarative programming language. The standard semantics for this language is just what you would expect. Ultimately, we would want to construct a partial interpretation that translated these programs into systems of electronic components, and the 'execution' of these components is just what they do when supplied with power.

In fact, in order not to get involved in irrelevant details, the 'code' into which circuits are translated will just be pictures with boxes for the components and lines joining them for the connections. Both the translation of programs into pictures, and the translation of pictures into meanings will be left rather vague in these notes, although there seems no reason not to formalise either process – nor indeed any reason not to mechanise them.

These notes are about a framework in which designs can be changed by program transformation. Suppose that P and Q are two programs with the same meaning as pictures: it is usual to write $P = Q$ meaning that the expressions P and Q have the same values in the standard semantics for the language. This does not mean that P and Q are the same expression, and it is quite possible that the partial interpretation – the picture – of P is quite different from that of Q. However, it is guaranteed by the correctness of the partial interpretation that either picture will have the same effect. This is because the meaning of the picture of P is the standard meaning of P, which is equal to the standard meaning of Q, and that is the meaning of the picture of Q. Replacing P by Q is therefore a safe design step, as is replacing Q by P.

It would be possible to make a similar argument for an 'is no worse than' relation on standard meanings, $P \leq Q$, which would mean that anyone satisfied with a picture of P would be happy with a picture of Q so that replacing P by Q is allowed, but not vice versa.

To as great an extent as possible, these notes will be about formal manipulation of expressions: replacing, say P by Q. There will be many equations like $P = Q$, but as little as possible about what either P or Q means in any particular standard semantics. This means that the standard semantics can be replaced by any other standard semantics in which the equations remain true, and the formal development of a circuit will remain valid. The best example of this is the replacement of a static, combinational semantics by a time-sequence, sequential semantics in a way that means that much of the development of a sequential circuit can be done without even having to think about time.

One thing to be wary of is that in these notes $P \neq Q$ means 'P is not necessarily equal to Q', that you cannot show them to be equal on the basis of the other equalities that you know and furthermore that there may be a particular model in which they really are different. This fine point of detail is not relevant to the correctness of calculations, for the soundness of calculations depends on things being equal and not on their being distinct.

1.2 The rôle of pictures

Throughout these notes you will find pictures that are intended to motivate the discussion that goes with them. At least for some people, the pictures help with an intuitive understanding of what an equation means, or of why a proof proceeds in a certain way.

These 'abstract floorplans' share many of the important properties of the circuits with which a designer would be concerned. Principal amongst these is *locality*: whether or not components are connected to components that are near to them. Because the costs which dominate in circuit construction are those of communication, locality is very important. Similarly, the extent to which connections cross over each other shows up, and this is another cost in circuit construction.

Beware, however, of reasoning about the pictures. In particular, notice that quite often the pictures are just particular instances of the equations and proofs which they portray. There is a danger in trying to reason from the example in the picture that you might generalize a result which happens only to be true for the particular instance.

Chapter 2

Composition and inverse

The central idea is that of putting two circuits together to co-operate with each other. The composition of two circuits R and S will be written $R;S$ and you should have in mind the idea of a circuit in which connections are made from one side of R to the other side of S, rather as shown in figure 2.1. Sometimes, for definiteness the 'left-hand' edge of a circuit will be called its *domain* and the 'right-hand' edge its *range*. The domain of $R\,;S$ is that of R, its range is that of S, and the range of R is connected to the domain of S. The fact that these domain appears on the left of the picture and the range on the right is of course just a convention, but will be one to which these notes adhere much of the time.

On the whole it is will not matter too much how composition is interpreted, so long as you choose a meaning for it which obeys all the 'laws' that are used in a calculation. Principal amongst these is that composition is associative: $(R;S);T = R\,;(S\,;T)$, which is to say that the order in which the components of figure 2.2 are assembled cannot affect the meaning of the circuit. Associativity gives us an excuse for leaving out the parentheses in, for example, $R\,;S\,;T$.

There are a number of fairly natural seeming models for the notation. If you choose to represent circuits by functions taking their inputs on the right, and delivering outputs to the left, the operation that corresponds to ';' is (backward) function composition: $R\,;S = R \cdot S$, where $(f \cdot g)(x) = f(g(x))$. You can readily check that this is indeed an associative operation, since for any x

$$((f \cdot g) \cdot h)(x) = (f \cdot g)(h(x)) = f(g(h(x))) = f((g \cdot h)(x)) = (f \cdot (g \cdot h))(x)$$

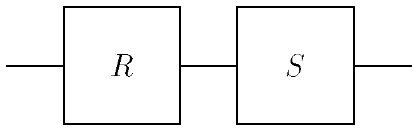

Figure 2.1: the composition $R\,;S$

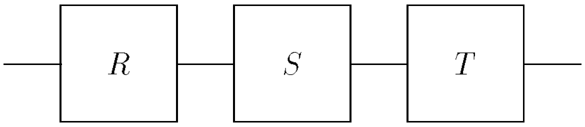

Figure 2.2: composition is associative: $(R\,;S)\,;T = R\,;S\,;T = R\,;(S\,;T)$

which is just what is meant by saying that $(f \cdot g) \cdot h = f \cdot (g \cdot h)$.

Another perfectly reasonable model is obtained by thinking of the component circuits as taking their inputs from the left and delivering outputs to the right. In this case, the operation that corresponds to ';' is forward function composition: $R;S = S \cdot R$. The associativity of this is immediate from associativity of backward composition.

In describing a particular circuit, it might be that there would be components that could not be explained as functions from the signals at one side to the signals on the other side. Another possible model is that of a relation between the signals on one side and the other. The composition in this case is the usual (forward) relational composition: $x\,(R\,;S)\,z = \exists y.\, x\,R\,y \;\&\; y\,S\,z$. Again, it is easy to show that this is associative.

Of course, you can translate any composition of functions into an equivalent composition of relations. In this sense, the relational model is more general than the other two, and it would be possible to do deal with much of the material in these notes referring only to the relational model. It is also a natural model to keep in mind. (For example, the naming of the domain and range of a circuit coincides with the domain and range of a relation in this model.) All sorts of other models are possible, including various subtly different ones like that where a circuit corresponds to a relation with its domain and range the wrong way around.

There will be some laws about specific constants (like $\mathcal{D}$) which are most easily understood in non-standard models, so it will pay to start as we mean to go on. You should try to avoid reasoning about the data – x, y, z in this example – and as far as possible, you should try to avoid reasoning about the specific component circuits – R, S, T here – concentrating instead on the combining forms, like composition, and the laws which they obey. The associative law – that $(R\,;S)\,;T = R\,;(S\,;T)$ for any R, S and T – is not about the data on which the circuit operates, nor is it about any particular circuit components R, S and T; rather it is a general statement about ';'.

2.1 Repeated composition

Where it makes sense to plug two components of the same kind together, we can talk about the n-times repeated composition of R, written R^n, and which is defined by $R^1 = R$, and $R^{n+1} = R\,;R^n$.

Since we know that composition is associative, we already know many things about repeated composition, for example $R^{n+1} = R^n\,;R$, and $R^{m+n} = R^m\,;R^n$, and so on. The proofs can be done by induction on n, but notice that the proof depends only on the associativity of composition, and there is no need to appeal to the meaning of R at all. If $n = 1$ then

$$R^{n+1} = R\,;R^n = R\,;R = R^n\,;R$$

and if $R^{n+1} = R^n\,;R$ then

$$R^{n+2} = R\,;R^{n+1} = R\,;(R^n\,;R) = (R\,;R^n)\,;R = R^{n+1}\,;R$$

and so by induction $R^{n+1} = R^n\,;R$ for all $n \geq 1$. The proof that $R^{m+n} = R^m\,;R^n$ is similar.

Suppose we assume for some R and S that $R;S = S;R$, then a simple induction shows that $R\,;S^n = S^n\,;R$. The case $n = 1$ is immediate from definitions, and

$$R\,;S^{n+1} = R\,;S\,;S^n = S\,;R\,;S^n = S\,;S^n\,;R = S^{n+1}\,;R$$

showing the result for all $n \geq 1$. A similar induction shows that on the same assumption $(R\,;S)^n = R^n\,;S^n$. The case $n = 1$ is immediate from definitions, and using the previous result

$$(R\,;S)^{n+1} = (R\,;S)^n\,;R\,;S = R^n\,;S^n\,;R\,;S = R^n\,;R\,;S^n\,;S = R^{n+1}\,;S^{n+1}$$

showing the result for all $n \geq 1$.

Now, notice that although $R\,;S^n = S^n\,;R$ and $(R\,;S)^n = R^n\,;S^n$ are only proved in case R and S commute (in case $R\,;S = S\,;R$), and so are statements about R and S; nevertheless the theorems 'If $R\,;S = S\,;R$ then $R\,;S^n = S^n\,;R$' and 'If $R\,;S = S\,;R$ then $(R\,;S)^n = R^n\,;S^n$' are – just like the associative law – statements about composition.

These statements about composition can be added to the collection of laws that we are accumulating, and which will eventually constitute a body of knowledge about the *forms* of all circuits.

2.2 Inverse

The inverse of a circuit is its left-to-right reflection: the connections on the left of R appear on the right of R^{-1} and vice versa. Since reflecting twice gets us back the original circuit, we require that for any circuit R the inverse of its inverse should be the same circuit: $(R^{-1})^{-1} = R$.

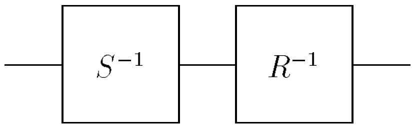

Figure 2.3: the inverse of a composition: $(R\,;S)^{-1} = (S^{-1})\,;(R^{-1})$

In the relational model, of course, inverse corresponds to relational converse: $x\,R^{-1}\,y = y\,R\,x$. Since not every function has an inverse, it does not make sense to talk about inverse in functional models, but you might construct a model using one-to-one functions.

Consider the relation which represents negation of Booleans: $\mathsf{T}\,\mathit{not}\,\mathsf{F}$ and $\mathsf{F}\,\mathit{not}\,\mathsf{T}$. Its inverse not^{-1} is of course the same relation. That means that $\mathit{not}^{-1} = \mathit{not}$, so you should be aware that 'reflecting the picture' of *not* leaves it unchanged. In particular, there is no idea that the inputs and outputs of a gate have been exchanged: the sense in which $\mathit{not}^{-1} = \mathit{not}$ abstracts from such concerns as which signals are inputs and which are outputs.

The next question to ask is how inverse interacts with composition. As shown in figure 2.3 you can reflect the picture of a composition by swapping the components and reflecting each of them, so keeping corresponding edges connected. That is why we require that $(R;S)^{-1} = (S^{-1});(R^{-1})$ for any R and S. (If you have a vivid imagination, you can imagine the R^{-1} and S^{-1} labels in the figure as backwards R and S. My word processor can manage **Я** but not the other!)

An immediate consequence is that $(R^n)^{-1} = (R^{-1})^n$, so as a notational convenience define $R^{-n} = (R^{-1})^n$. Be careful not to let this representational recklessness lead you astray – the induction that proved $R^{m+n} = R^m\,;R^n$ did so for positive n. Notice that since we know nothing in general about $R;R^{-1}$, it is not necessarily the case that $R^{m-n} = R^m\,;R^{-n}$. In this respect at least it can be argued that it was a mistake to christen the reflection of a circuit its 'inverse' with all the connotations that has. Unfortunately this seems to be one of the few bits of the notation and nomenclature that stubbornly resist the flux to which the rest of this work is prey.

2.3 Identity and types

Having defined R^n for positive and negative n, it is natural to seek an interpretation for R^0 which fits neatly into the remaining gap. The time has come to be honest about types. (Perhaps, since we are talking about 'circuits' and not relations, it might have been better to talk of '*interfaces*' rather than types.) Because it avoids talking about the data, this presentation may look rather strange – unless you have seen it before, in which case it will look like category theory, where instead of circuits and interfaces one talks of arrows and objects.

As far as will concern us here, a type is little more than a collection of values. In the world of relations, the things that look like types are equivalence relations: an equivalence relation determines a set of equivalence classes, and the set of equivalence classes determines the equivalence relation. To describe what it is to be a type, we have to encode in our notation the properties that make a relation be an equivalence relation.

An equivalence relation is one which is reflexive, symmetric and transitive: symmetry is just that $R = R^{-1}$; transitivity of a reflexive relation is just that $R = R^2$ (you can show that any transitive relation contains its square, but you need reflexivity to show containment of the square in the relation). From knowing just that $R = R^{-1}$ and $R = R^2$ it follows that $R\,;R^{-1} = R\,;R = R^2 = R$, which is a limited form of reflexivity: if there is some y for which $x\,R\,y$, then necessarily $x\,R\,x$. So any R for which $R = R^{-1} = R^2$ (and so $R = R^n$ for all non-zero n) is an equivalence on all the relevant values. Such an R we call a *type*. It will clearly do no harm to our intuition to define $R^0 = R$ so long as R is a type.

A type D is a type of the domain (not, notice, *the* type of the domain) of a circuit C if $C = D\,;C$, and similarly a type R is a type of its range if $C = C\,;R$. Sometimes this will be written as $C : D \sim R$, and we speak of $D \sim R$ as a type for C. When speaking of a circuit C, we ought really to quote a particular D and R, and speak of the circuit $C : D \sim R$. Composition makes much more sense if you only ever compose circuits with matching types, say $P : A \sim B$ and $Q : B \sim C$, in which case $(P\,;Q) : A \sim C$, as you can check from the definition.

In that case, it makes sense to talk of repeated composition only when you have in mind a *homogeneous* component $R : T \sim T$, in which case $R^n : T \sim T$ for all positive and negative n. For that reason it is intuitively safe to define R^0 to be T, because then certainly $R^{n+0} = R^n = R^n\,;T = R^n\,;R^0$ and similarly for R^{0+n} and for negative powers. Be careful with this definition: strictly speaking it does not make sense to talk about R^0 without making explicit which particular type you have in mind for R – remember that the type is not unique – and a pedant would insist on saying that it is $(R : T \sim T)^0$ that is defined to be T.

Of course the reason for being sloppy is the usual one: that it does not often matter all that much exactly what choice you make, and when it does the right choice is obvious. You should beware, however, of any intuition that leads you to expect R^0 to be the identity of composition. That would be an ι for which $\iota\,;Q = Q = Q\,;\iota$ for all Q.

2.4 Conjugation

A circuit and its inverse often appear together bracketing another component, as in figure 2.4. The R-conjugation of S (the conjugation of S by R) is defined by

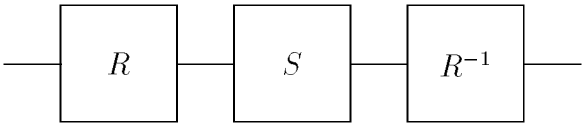

Figure 2.4: the conjugation $R \,/\, S$ of S by R

$R \,/\, S = R \,;\, S \,;\, R^{-1}$.

Conjugations can be composed in the obvious way, because $R \,/\, (S \,/\, T) = (R;S)/T$. A conjugation behaves rather like the S part, and you can think of R as a 'translation' (in almost any of the many different senses of that word) applied to the language in which you communicate with S. For example, $(R/S)^{-1} = R/(S^{-1})$, so the inverse of a conjugation of S is the same conjugation of the inverse of S. It is almost the case that the composition of two conjugations is the conjugation of the compositions. If $R^{-1} \,;\, R$ is a type of the range of S and the domain of T, then

$$(R \,/\, S) \,;\, (R \,/\, T) = R \,;\, S \,;\, (R^{-1} \,;\, R) \,;\, T \,;\, R^{-1} = R \,;\, (S \,;\, T) \,;\, R^{-1} = R \,/\, (S \,;\, T)$$

Such an R is called an *abstraction* to the type $R^{-1} \,;\, R$. This is another unfortunate historical accident of the notation: you might prefer to think not of R, but of R^{-1} which is a *representation* for the type $R^{-1} \,;\, R$.

The idea behind these names is that we start from some operation defined on some abstract objects, say negation of Booleans: T *not* F and F *not* T. Suppose that we choose to represent Booleans by bits: each bit value is given a meaning by the abstraction: say 0 *abs* F and 1 *abs* T. You can calculate that $abs^{-1} \,;\, abs$ is the identity relation on the Booleans. Then $abs \,/\, not$ is an implementation of the abstract negation operation, in that given a representation of a bit, it relates it to a representation of its negation. You can calculate that $0\,(abs \,/\, not)\,1$ and $1\,(abs \,/\, not)\,0$.

The condition that $abs^{-1} \,;\, abs$ be a type is just the requirement that if you take any abstract object of that type, then abs^{-1} will give at least one concrete representative of that object, and moreover that any representative given by abs^{-1} will be a representative of that same abstract object. That is to say, abs^{-1} gives a faithful representation of the type.

$$\begin{tikzcd}
(abs^{-1} \mathbin{;} abs) \bullet \arrow[r, "R"] & \bullet\, (abs^{-1} \mathbin{;} abs) \arrow[d, "abs^{-1}"] \\
\bullet \arrow[u, "abs"] \arrow[r, "abs \,/\, R"'] & \bullet
\end{tikzcd}$$

Figure 2.5: $abs \,/\, R$ implements $R : (abs^{-1} \mathbin{;} abs) \sim (abs^{-1} \mathbin{;} abs)$

Chapter 3

Lists and tuples

In order to talk about operations on collections of data, we need some structured types like lists and tuples. In fact we are going to blur the distinction that people usually make between lists and tuples: there will be no difference between a triple of things that happen to have a common type, and a list of things that have that type and which happens to be three elements long. This makes some things easier, and some things – like mechanical type deduction – tremendously difficult. In fact, to get anywhere with mechanical type deduction, you have to be much more careful about types than these notes will be.

Of course, we will not say anything (formally) about the lists and tuples themselves, rather about circuit constructors that make circuits that operate on lists and tuples.

3.1 Parallel composition

The fundamental operation that builds circuits that operate on lists is *parallel composition.* Informally, the parallel composition $[R, S]$ of R and S is a circuit

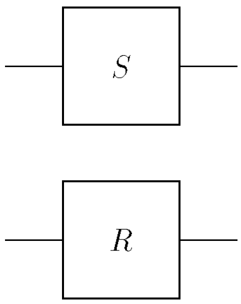

Figure 3.1: the parallel composition $[R, S]$

whose domain and range are pairs of signals – as shown in figure 3.1 – which has an R component operating on the first elements of the pairs, and an S component independent of it operating on the second elements. The intuition is that, at least for relations, $\langle x, y\rangle\ [R, S]\ \langle u, v\rangle$ exactly when both $x\ R\ u$ and $y\ S\ v$. Notice the following convention, to which the pictures in this note adhere: that the first component of a list of signals is on the left if you stand in the range of a circuit, looking towards the domain. Since the range is on the right of figure 3.1, and the domain on the left, the first component is at the bottom and the last at the top.

The independence of the components of a parallel composition is reflected in the way that composition and inverse should distribute over it, as $[R, S] ; [T, U] = [R ; T, S ; U]$ and $[R, S]^{-1} = [R^{-1}, S^{-1}]$. Beware the apparently misleading punctuation in $[R ; T, S ; U]$, which means $[(R ; T), (S ; U)]$.

3.2 Pairs and projections

In these notes at least, the parallel composition of a circuit with an identity arises so often that we abbreviate the forms $\mathsf{fst}\, R = [R, \iota]$, read 'first R', and $\mathsf{snd}\, R = [\iota, R]$ read 'second R'. As you can calculate from the definition, a first and a second commute with each other,

$$\mathsf{fst}\, R ; \mathsf{snd}\, S = [R, \iota] ; [\iota, S] = [R, S] = [\iota, S] ; [R, \iota] = \mathsf{snd}\, S ; \mathsf{fst}\, R$$

and each distributes over composition and inverse, for similarly $\mathsf{fst}(R ; S) = \mathsf{fst}\, R ; \mathsf{fst}\, S$ and $(\mathsf{fst}\, R)^{-1} = \mathsf{fst}\, R^{-1}$, and similarly for second.

More formally, the idea that the two component circuits of a parallel pair are independent can be written as the two equations $[R, S] ; \pi_1 = \mathsf{snd}\, S ; \pi_1 ; R$ and $[R, S] ; \pi_2 = \mathsf{fst}\, R ; \pi_2 ; S$ asserting the existence of projections that extract the two component signals from a pair. Again, if you want to think of the model in relations, π_1 projects a pair onto its first component, $\langle x, y\rangle\ \pi_1\ x$, and π_2 onto its second, $\langle x, y\rangle\ \pi_2\ y$.

The inverse of a projection, of course, injects its domain into a pair (with an arbitrary other component) so that ${\pi_1}^{-1} ; \pi_1 = \iota$ is the universal type, as is ${\pi_2}^{-1} ; \pi_2$. The projections give us a way of talking about things that behave like pairs, and so of talking about operations on more than one argument.

Beware that although $\mathsf{fst}\, R ; \pi_1 = \pi_1 ; R$, in general $[R, S] ; \pi_1 \neq \pi_1 ; R$ for general S. They would have been equal in a model based on total functions: they are for example equal in Richard Bird's calculus of total functions, 'squiggol' citeconstructive-fp. Without the equivalent of totality, they may be different: in the relational model, if S is the empty relation then so is $[R, S]$, and so is any composition in which it appears, but $\pi_1 ; R$ need not be empty. So it is the case that ${\pi_1}^{-1} / \mathsf{fst}\, R = R$, but ${\pi_1}^{-1} / [R, S] \neq R$.

3.3 Types for lists

Without going into any further formality, the notion of the parallel composition of two circuits extends naturally to the composition of any number of components. In particular, there is the parallel composition of one component alone, written $[R]$. The domain signal of this circuit is a singleton list, as is its range signal. Writing 1 for the type $[\iota]$ of singletons, $[R] : 1 \sim 1$.

In a picture it is hard to imagine there being any difference between an element of some type and a list of signals that consists of just one signal of that given type. The formalization of this similarity is that a list of one thing can be used to represent that thing, and any thing can be used to represent a list consisting of that thing alone. The abstraction that relates signals to singleton signals is here written $[\text{-}] : \iota \sim 1$, which you can read as 'singleton' (or following Oege de Moor as 'gift-wrap'). It is a representation for any type (its inverse is an abstraction to any type), which is to say that $[\text{-}] ; [\text{-}]^{-1} = \iota$, and it is an abstraction to singletons, which is to say that $[\text{-}]^{-1} ; [\text{-}] = 1$.

Be careful to distinguish between $[\text{-}] : \iota \sim 1$ and $[\iota]$. Singleton promotes a circuit to operate on one-lists, $[R] = [\text{-}]^{-1} \,/\, R$, so $R ; [\text{-}] = [\text{-}] ; [R]$. The identity on singletons, $[\iota] = 1$, is the type of each interface of $[R]$, so $[R] = [\iota] \,/\, [R]$ and $[R] ; [\iota] = [R] = [\iota] ; [R]$.

In order to talk about longer lists, more constants ('plumbing circuits') are needed. The thing that corresponds to appending lists is the app circuit for which

$$\begin{aligned} &[R_0, R_1, \ldots, R_i, R_{i+1}, R_{i+2}, \ldots, R_n] \\ &\quad = \mathsf{app}^{-1} \,/\, [[R_0, R_1, \ldots, R_i], [R_{i+1}, R_{i+2}, \ldots, R_n]] \end{aligned}$$

for any n and $i \leq n$. In particular, $[[\text{-}], [\text{-}]] ; \mathsf{app} = 2$, so $[1, 1] ; \mathsf{app} = [[\text{-}]^{-1}, [\text{-}]^{-1}]$. In terms of relations on signals, app relates a pair of lists to the list obtained by concatenating them; but note that all the names in this equation are names of circuits, rather than signals.

The component app is just another circuit – a piece of plumbing. The unfortunate price paid for not having an infixed operator to represent appending, is that the associativity of appending is expressed by the rather inscrutable $(\mathsf{app}^{-1} ; \mathsf{fst}\,\mathsf{app}^{-1}) \,/\, [[R, S], T] = (\mathsf{app}^{-1} ; \mathsf{snd}\,\mathsf{app}^{-1}) \,/\, [R, [S, T]]$. Although $\mathsf{app}^{-1} ; \mathsf{app}$ is not the identity, it is a type: the type of all lists.

You may be more familiar with the idea of building up lists one component at a time: $\mathsf{apl} = \mathsf{fst}\,[\text{-}] ; \mathsf{app}$, read 'append left' or by some people 'cons', is a circuit which adds one signal to the left of a list; and $\mathsf{apr} = \mathsf{snd}\,[\text{-}] ; \mathsf{app}$, read 'append right' or 'snoc', adds a signal to the right-hand end of a list. For example, $[R, [S, T]] ; \mathsf{apl} = [R, S, T]$ so $[R, [S, T]] ; \mathsf{apl} ; \mathsf{apr}^{-1} = \mathsf{apl} ; \mathsf{apr}^{-1} ; [[R, S], T]$. Notice that $[R, [S, T]]$ and $[[R, S], T]$ are different circuits, each relating pairs, and each is different from $[R, S, T]$ which relates triples.

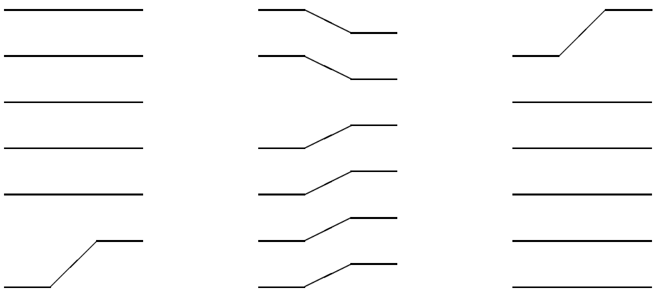

Figure 3.2: instances of apl, app, and apr^{-1}

With these constructors, the types of lists of any given length can be defined by an induction. Structures with one component have type $1 = [\iota]$. Those with $n+1$ components have one more component than those of length n, and can be written as any one of $n+1 = \mathsf{apl}^{-1} \,/\, \mathsf{snd}\, n = \mathsf{app}^{-1} \,/\, [1, n] = \mathsf{app}^{-1} \,/\, [n, 1] = \mathsf{apr}^{-1} \,/\, \mathsf{fst}\, n$. Neither $\mathsf{apl}\,;\mathsf{apl}^{-1}$ nor $\mathsf{apl}^{-1}\,;\mathsf{apl}$ is the identity, but each is a type: $\mathsf{apl}^{-1}\,;\mathsf{apl}$ is the type of lists of at least one element, and $\mathsf{apl}\,;\mathsf{apl}^{-1}$ that of pairs with a list in the second element; similarly for apr.

In order to complete the picture, the natural definition for $0 = [\,]$, the parallel composition of no component circuits, makes it behave like the identity relation on zero length lists. Beware the usual problem of the type of an empty structure: you might want to distinguish between various instances of $0 : [T] \sim [T]$. You might equally feel that the trouble caused by empty lists would justify banning them altogether.

If you are that way inclined, you can prove by induction over the natural numbers concerned that $n + m = \mathsf{app}^{-1} \,/\, [n, m]$. For example, the type 137 of structures with one hundred and thirty-seven elements of any type satisfies $137 = \mathsf{app}^{-1} \,/\, [95, 42]$. Because we are interested in circuits which make physical sense, there is no real interest in infinite lists, at least not in infinite parallel compositions. That means that every list will be of some finite width, and you can prove something about all lists by induction over the naturals.

3.4 Map

The parallel compositions that can be written with the notation above are all circuits of a given, fixed width. So, for example, $[R, R]$ takes pairs of signals in the domain and in the range. The 'map' construction generalizes this, so that $\mathsf{map}\, R$ is the parallel composition of any number of R circuits. That means that $1;\mathsf{map}\, R = [R] = \mathsf{map}\, R;1$ and $2;\mathsf{map}\, R = [R, R] = \mathsf{map}\, R;2$, and so on. Figure 3.3 shows an instance of $\mathsf{map}\, R$; this is an example of a picture from which it would be dangerous to generalize, because it has many properties not shared by, for example,

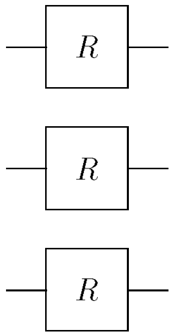

Figure 3.3: an instance of $\mathsf{map}\,R$

137 / $\mathsf{map}\,R$, or 1 / $\mathsf{map}\,R$.

By induction on the width of the instance, you can show that map distributes over sequential composition and inverse in the same way that parallel composition does. We will simply assert that $\mathsf{map}(R\,;S) = (\mathsf{map}\,R)\,;(\mathsf{map}\,S)$ and $\mathsf{map}(R^{-1}) = (\mathsf{map}\,R)^{-1}$. It follows that if T is a type, then so is $\mathsf{map}\,T$, in fact it is the type of all lists of type T and in particular $\mathsf{map}\,\iota$ is the type of lists.

There are a great number of things things that you might want to be told about map to be sure that we were all thinking of the same circuit structure, amongst them perhaps

$$
\begin{aligned}
1\,;\mathsf{map}\,R &= [R] \\
n\,;\mathsf{map}\,R &= \mathsf{map}\,R\,;n \\
[\,\text{-}\,]\,;\mathsf{map}\,R &= R\,;[\,\text{-}\,] \\
\mathsf{app}\,;\mathsf{map}\,R &= \mathsf{map}\,\mathsf{map}\,R\,;\mathsf{app} \\
&= [\mathsf{map}\,R, \mathsf{map}\,R]\,;\mathsf{app} \\
\mathsf{apl}\,;\mathsf{map}\,R &= [R, \mathsf{map}\,R]\,;\mathsf{apl} \\
\mathsf{apr}\,;\mathsf{map}\,R &= [\mathsf{map}\,R, R]\,;\mathsf{apr}
\end{aligned}
$$

but it is clear that some of these are consequences of the others. You can calculate, for example, that

$$
\begin{aligned}
\mathsf{apl}\,;\mathsf{map}\,R &= \mathsf{fst}\,[\,\text{-}\,]\,;\mathsf{app}\,;\mathsf{map}\,R \\
&= \mathsf{fst}\,[\,\text{-}\,]\,;[\mathsf{map}\,R, \mathsf{map}\,R]\,;\mathsf{app} \\
&= [[\,\text{-}\,]\,;\mathsf{map}\,R, \mathsf{map}\,R]\,;\mathsf{app} \\
&= [R\,;[\,\text{-}\,], \mathsf{map}\,R]\,;\mathsf{app} \\
&= [R, \mathsf{map}\,R]\,;\mathsf{fst}\,[\,\text{-}\,]\,;\mathsf{app} \\
&= [R, \mathsf{map}\,R]\,;\mathsf{apl}
\end{aligned}
$$

using the other equations and the definition of apl. In fact, you ought to be convinced by being told only that

$$\begin{aligned}[\text{-}] ; \mathsf{map}\, R &= R ; [\text{-}] \\ [m, n] ; \mathsf{app} ; \mathsf{map}\, R &= [\mathsf{map}\, R, \mathsf{map}\, R] ; [m, n] ; \mathsf{app} \\ (\mathsf{map}\, R)^{-1} &= \mathsf{map}(R^{-1})\end{aligned}$$

and perhaps some equations involving $0 ; \mathsf{map}\, R$: certainly $0 ; \mathsf{map}\, R = R ; 0$ and perhaps but not necessarily $0 ; \mathsf{map}\, R = 0$.

The above equations for all m and n are more than enough to define map, and it follows that there is a requirement to show that they do not force too many things to be equal. It never does any harm to throw in a few other equalities, so long as they are consistent with our idea that some things are different from each other. The simplest test for consistency is whether they are true in your favourite model. If they are, they are safe; if they are false in that model, either they are inconsistent or you need a better model. Choosing to omit, say $0 ; \mathsf{map}\, R = 0$, increases the number of possible models, at the expense of decreasing the number of things that can be shown to be true in all models. In these notes we will have $0 ; \mathsf{map}\, R = 0$.

3.5 Reverse

There is clearly a connection between left- and right-handed views of lists. Since any finite list could have been built up in either way, every time you say something about apl you are saying something about apr, and every time you say something about both there is a danger of saying something inconsistent about them.

To make it easier to talk about the mirror-similarity of apl and apr, we use a piece of plumbing $\mathsf{rev} : \mathsf{map}\, \iota \sim \mathsf{map}\, \iota$, read 'reverse', which flips a list over. In particular, $2 \,/\, \mathsf{rev}$ exchanges two signals and so satisfies $\mathsf{rev} ; [R, S] = [S, R] ; \mathsf{rev}$. For structures of other lengths it can be defined by $0 ; \mathsf{rev} = 0$ and $[\text{-}] ; \mathsf{rev} = [\text{-}]$ and $\mathsf{app} ; \mathsf{rev} = [\mathsf{rev}, \mathsf{rev}] ; \mathsf{rev} ; \mathsf{app}$.

Immediately from these definitions,

$$\begin{aligned}\mathsf{apl} ; \mathsf{rev} &= \mathsf{fst}\, [\text{-}] ; \mathsf{app} ; \mathsf{rev} \\ &= \mathsf{fst}\, [\text{-}] ; [\mathsf{rev}, \mathsf{rev}] ; \mathsf{rev} ; \mathsf{app} \\ &= [[\text{-}] ; \mathsf{rev}, \mathsf{rev}] ; \mathsf{rev} ; \mathsf{app} \\ &= [[\text{-}], \mathsf{rev}] ; \mathsf{rev} ; \mathsf{app} \\ &= \mathsf{rev} ; [\mathsf{rev}, [\text{-}]] ; \mathsf{app} \\ &= \mathsf{rev} ; \mathsf{fst}\, \mathsf{rev} ; \mathsf{snd}\, [\text{-}] ; \mathsf{app} \\ &= \mathsf{rev} ; \mathsf{fst}\, \mathsf{rev} ; \mathsf{apr}\end{aligned}$$

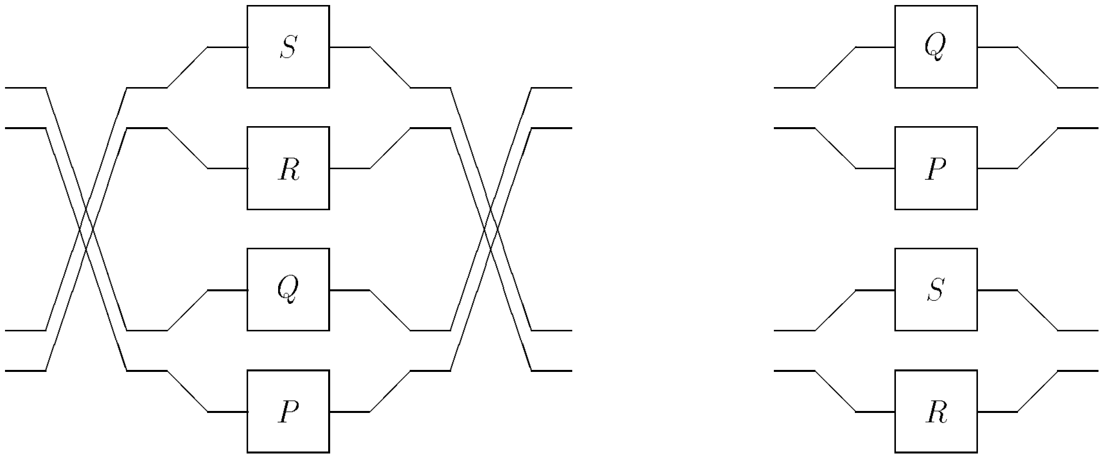

Figure 3.4: $\mathsf{rev} \,/\, [[P, Q], [R, S]]$ and the equivalent $[[R, S], [P, Q]]$

and similarly $\mathsf{apr} \,;\, \mathsf{rev} = \mathsf{rev} \,;\, \mathsf{snd}\,\mathsf{rev} \,;\, \mathsf{apl}$. By induction on the length of the list, $\mathsf{rev} = \mathsf{rev}^{-1}$ and $\mathsf{rev} \,;\, \mathsf{map}\, R = \mathsf{map}\, R \,;\, \mathsf{rev}$ and so on.

Because rev reverses a list of signals, and because the convention about drawing circuits puts the beginning of a list at the bottom of the picture, the end of the list at the top, you might expect that a picture of $\mathsf{rev} \,/\, R$ would be the same as that of R but flipped over about a horizontal axis. This is almost the case, but be careful: suppose R is a circuit operating on lists of lists, then the outermost list is reversed but the elements in that list are in the same order, so for example $\mathsf{rev} \,/\, [[P, Q], [R, S]] = [[R, S], [P, Q]]$ as can be seen in figure 3.4.

Reverse is the first example in these notes of something that you might need to describe, but which it would be unpleasant to have to implement in a finished circuit. Any non-trivial instance of reverse drawn as a planar picture has to contain crossing wires: $2 \,/\, \mathsf{rev}$ contains exactly one crossing, $n \,/\, \mathsf{rev}$ contains $n(n-1)/2$. Such crossings are presumed to cost something in any implementation technology. In general therefore, in the course of designing a circuit it should be an aim to eliminate instances of rev, and if that can not be done at least to reduce their width. Beware however that the cost of a cross-over is likely to vary with what the types of the crossing signals are, so minimizing the cost is not necessarily a matter of minimizing the number of crossings.

3.6 Triangle

A useful construction that is closely related to map, but which seems to be almost peculiar to the sorts of programs that arise in describing circuits, is the triangle. Instances of this are illustrated in figure 3.5. For any homogeneous $R : T \sim T$,

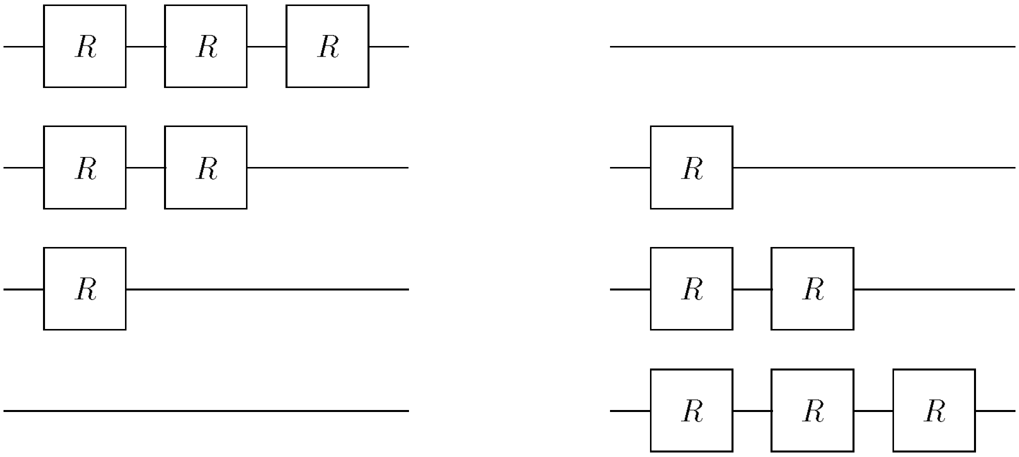

Figure 3.5: an instance of $\mathsf{tri}\,R$, and an instance of $\mathsf{rev}\ /\ \mathsf{tri}\,R$

the triangle $\mathsf{tri}\,R : \mathsf{map}\,T \sim \mathsf{map}\,T$ relates lists of signals, just as map would, but it relates the i-th components according to R^i. For example, if $(\times 2)$ is the circuit which doubles a number, $\mathsf{tri}(\times 2)$ relates a list of zeroes and ones to a list of those bits weighted by the powers of two which they would represent in a binary number with its least significant bit first. Similarly, $\mathsf{rev}\ /\ \mathsf{tri}(\times 2)$ would give them the weights appropriate to having the most significant bit first.

It will suffice to require that $\mathsf{apl}\,;\mathsf{tri}\,R = [R^0, \mathsf{map}\,R\,;\mathsf{tri}\,R]\,;\mathsf{apl}$, that $(\mathsf{tri}\,R)^{-1} = \mathsf{tri}(R^{-1})$, and that $0;\mathsf{tri}\,R = 0;\mathsf{map}\,R$. but of course there is a rich collection of laws that can be derived from these and the properties of other operators by induction over lists. Among the more useful ones are those that relate a triangles and maps of the same components such as

$$\begin{aligned} (\mathsf{tri}\,R)^n &= \mathsf{tri}(R^n) \\ \mathsf{tri}\,R\,;\mathsf{map}\,R &= \mathsf{map}\,R\,;\mathsf{tri}\,R \end{aligned}$$

and those about commuting circuits, for if $R\,;S = S\,;R$ you can show that

$$\mathsf{tri}\,R\,;\mathsf{tri}\,S = \mathsf{tri}(R\,;S)$$

and for types

$$\mathsf{tri}(R^0) = \mathsf{map}(R^0)$$

and moreover $\mathsf{tri}\,R : \mathsf{map}\,R^0 \sim \mathsf{map}\,R^0$.

Decomposing a triangle with apr instead of apl, or decomposing a reversed triangle with apl, you can show by induction on n that $\mathsf{fst}\,n\ ;\mathsf{apr}\ ;\mathsf{tri}\,R = [n\ ;\mathsf{tri}\,R, R^n]\ ;\mathsf{apr}$ and it turns out that this configuration appears often: an R^n will appear somewhere where by writing $\mathsf{apr}\ /\ \mathsf{tri}\,R$ or $\mathsf{apl}\ /\ (\mathsf{rev}\ /\ \mathsf{tri}\,R)$ you can avoid mentioning n altogether.

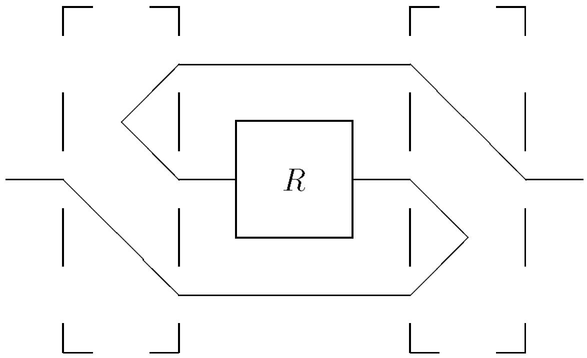

Figure 3.6: $\mathcal{L}\ ;[\iota, R, \iota]\ ; \mathcal{R}$

3.7 A curiosity

You may wonder why flipping a picture about a vertical axis is described by a function that operates on a circuit, taking R to R^{-1}, but flipping it about a horizontal axis was coded by introducing some plumbing and clamping the circuit between two bits of this plumbing. Surprisingly enough, inverse is not *really* necessary since R^{-1} can also be implemented by wiring around an instance of R as shown in figure 3.6.

Other wiring layouts are possible, of course, but the example in the figure shows an implementation of $R^{-1} = \mathcal{L}\ ;[\iota, R, \iota]\ ; \mathcal{R}$ where $\mathcal{R} = (\mathcal{L}\ ; \mathsf{rev})^{-1}$. In the case of the relational model, $\mathcal{L} : \iota \sim 3$ is the relation for which $x\ \mathcal{L}\ \langle x, y, y\rangle$ for all x and y. It is harder to see exactly how to axiomatize $\mathcal{L}$ without reference to the model, but it seems that however you do it you will need to use inverse!

Chapter 4

Rows and columns

There are fundamental reasons for being interested in circuits which can be laid out in a little more than two dimensions. Whatever technology is used, any very large system with many connections has to be approximately flat. In particular, circuits that are going to be printed on the surface of silicon have to be more or less planar, with there being a cost to be paid for each excursion out of the surface to permit wires to cross. This means that there are good reasons to be interested in two-dimensional meshes.

This chapter is almost entirely about circuits $R : 2 \sim 2$ with a pair of signals in the domain and a pair of signals in the range. In order to make pictures of networks of such circuits lie naturally on the page, they are drawn by following a new convention illustrated in figure 4.1. The first component of the domain lies to the left, the second component above the circuit; the first component of the range lies below the circuit, and the second component to its right. This keeps the order of the signals on the paper consistent with the convention given earlier for drawing circuits that operate on pairs. (Imagine a map of Britain with a line drawn from the Tees estuary in the North East to that of the Exe in the South West: the domain signals lie in the hill country to the West and North; the range signals in the more prosperous lowlands to the South and East.)

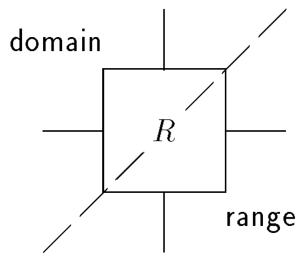

Figure 4.1: a Tees–Exe picture of $R : 2 \sim 2$

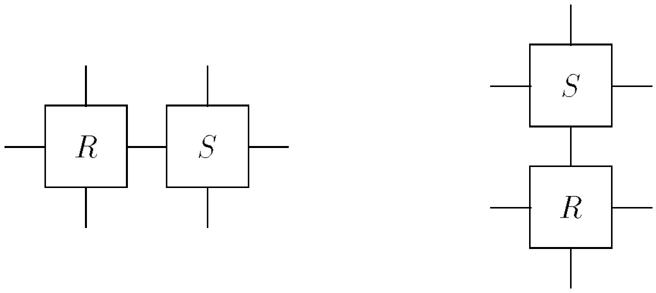

Figure 4.2: R beside S making $R \leftrightarrow S$, and R below S making $R \updownarrow S$

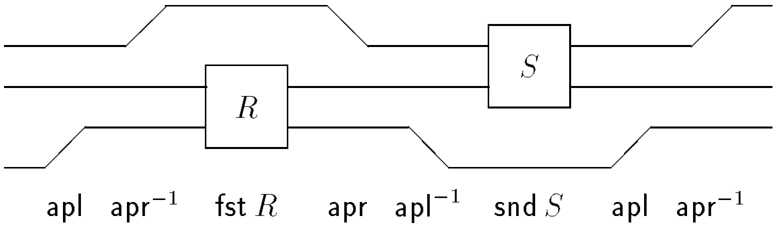

Figure 4.3: another circuit layout which implements $R \leftrightarrow S$

4.1 Beside and below

There are two ways of connecting these square tiles that seem natural: as shown in figure 4.2. In each case the wires are grouped so that both $R \leftrightarrow S$, read 'R beside S', and $R \updownarrow S$, read 'R below S', are also of type $2 \sim 2$; indeed $R \leftrightarrow S : \mathsf{snd}\,2 \sim \mathsf{fst}\,2$ and $R \updownarrow S : \mathsf{fst}\,2 \sim \mathsf{snd}\,2$.

It is clear that circuits made by beside and below are not new ones, in the sense that we already know enough wiring to be able to duplicate their function without any new primitives. Reading off the picture in figure 4.3, the definitions of beside and below would appear to be straightforward

$$\begin{aligned} R \leftrightarrow S &= \mathsf{apl} ; (\mathsf{apr}^{-1} \,/\, \mathsf{fst}\,R) ; (\mathsf{apl}^{-1} \,/\, \mathsf{snd}\,S) ; \mathsf{apr}^{-1} \\ R \updownarrow S &= (R^{-1} \leftrightarrow S^{-1})^{-1} \\ &= \mathsf{apr} ; (\mathsf{apl}^{-1} \,/\, \mathsf{snd}\,S) ; (\mathsf{apr}^{-1} \,/\, \mathsf{fst}\,R) ; \mathsf{apl}^{-1} \end{aligned}$$

Take careful note of the order of S and R in $R \updownarrow S$, for the apparent inversion is not just the outcome of a badly chosen convention: we might have chosen to use *above* instead of below, but in order to do so we would have reverse the order of the components of a beside to maintain the consistency of the intrinsic co-ordinates. If you stand on the Tees-Exe line of a picture and face the domain, you can read straight through the horizontal and vertical signals from left to right; similarly, if you face the range you can read the pair of signals from right to left. That makes these pictures appear just like the linear ones, from inside.

Unfortunately, this definition would not be very useful: this is an example – which caught me out when I was preparing these notes – of recklessly generalizing from a picture and getting more than one bargains for. One of the properties we expect of beside and below is that $(A \leftrightarrow B) \updownarrow (C \leftrightarrow D) = (A \updownarrow C) \leftrightarrow (B \updownarrow D)$. Richard Bird describes this property by saying that 'beside *abides with* below', a contraction of above-and-besides. Beware if you are comparing this material with his, that his vertical composition is 'above' and not 'below'; we use right-handed co-ordinates which are natural in pictures, he uses left-handed co-ordinates which are natural in matrices.

I spent a long time wondering how to prove this from the above definitions before it occurred to me that it was not true: that the abides property of beside and below depends on the component circuits being of type $2 \sim 2$. Better definitions would therefore be given by

$$\begin{aligned} R \leftrightarrow S &= \mathsf{apl} \,;\, (\mathsf{apr}^{-1} \,/\, \mathsf{fst}(2 \,/\, R)) \,;\, (\mathsf{apl}^{-1} \,/\, \mathsf{snd}(2 \,/\, S)) \,;\, \mathsf{apr}^{-1} \\ R \updownarrow S &= \mathsf{apr} \,;\, (\mathsf{apl}^{-1} \,/\, \mathsf{snd}(2 \,/\, S)) \,;\, (\mathsf{apr}^{-1} \,/\, \mathsf{fst}(2 \,/\, R)) \,;\, \mathsf{apl}^{-1} \end{aligned}$$

Particularly useful in the discussion of these circuits will be the circuits consisting of two identity components either beside or below each other. Let $rsh = \iota \leftrightarrow \iota$, for 'right shift', and $lsh = \iota \updownarrow \iota$, for 'left shift'; clearly $rsh = lsh^{-1}$ and $lsh = rsh^{-1}$.

$$\begin{aligned} \iota \leftrightarrow \iota &= 2 \leftrightarrow 2 \\ &= \mathsf{apl} \,;\, (\mathsf{apr}^{-1} \,/\, \mathsf{fst}\, 2) \,;\, (\mathsf{apl}^{-1} \,/\, \mathsf{snd}\, 2) \,;\, \mathsf{apr}^{-1} \\ &= \mathsf{apl} \,;\, 3 \,;\, 3 \,;\, \mathsf{apr}^{-1} \\ &= \mathsf{apl} \,;\, 3 \,;\, \mathsf{apr}^{-1} \end{aligned}$$

You can see that $[a, [b, c]] \,;\, rsh = rsh \,;\, [[a, b], c]$, whence the name, and similarly $lsh = \mathsf{apr} \,;\, 3 \,;\, \mathsf{apl}^{-1}$ and $[[a, b], c] \,;\, lsh = lsh \,;\, [a, [b, c]]$. Notice that $\mathsf{snd}\, 2 \,;\, \mathsf{apl} \,;\, \mathsf{apr}^{-1} = \mathsf{apl} \,;\, 3 \,;\, \mathsf{apr}^{-1} = \mathsf{apl} \,;\, \mathsf{apr}^{-1} \,;\, \mathsf{fst}\, 2$, these will prove useful later.

The definitions of beside and below can be recast in terms of left and right shifts, for

$$\begin{aligned} R \leftrightarrow S &= \mathsf{apl} \,;\, (\mathsf{apr}^{-1} \,/\, \mathsf{fst}(2 \,/\, R)) \,;\, (\mathsf{apl}^{-1} \,/\, \mathsf{snd}(2 \,/\, S)) \,;\, \mathsf{apr}^{-1} \\ &= \mathsf{apl} \,;\, ((\mathsf{apr}^{-1} \,;\, \mathsf{fst}\, 2) \,/\, \mathsf{fst}\, R) \,;\, ((\mathsf{apl}^{-1} \,;\, \mathsf{snd}\, 2) \,/\, \mathsf{snd}\, S) \,;\, \mathsf{apr}^{-1} \\ &= \mathsf{apl} \,;\, ((3 \,;\, \mathsf{apr}^{-1}) \,/\, \mathsf{fst}\, R) \,;\, ((3 \,;\, \mathsf{apl}^{-1}) \,/\, \mathsf{snd}\, S) \,;\, \mathsf{apr}^{-1} \\ &= (\mathsf{apl} \,;\, 3 \,;\, \mathsf{apr}^{-1}) \,;\, \mathsf{fst}\, R \,;\, (\mathsf{apr} \,;\, 3 \,;\, \mathsf{apl}^{-1}) \,;\, \mathsf{snd}\, S \,;\, (\mathsf{apl} \,;\, 3 \,;\, \mathsf{apr}^{-1}) \\ &= rsh \,;\, \mathsf{fst}\, R \,;\, lsh \,;\, \mathsf{snd}\, S \,;\, rsh \end{aligned}$$

and similarly $R \updownarrow S = lsh \,;\, \mathsf{snd}\, S \,;\, rsh \,;\, \mathsf{fst}\, R \,;\, lsh$.

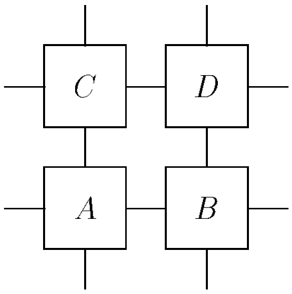

Figure 4.4: $(A \leftrightarrow B) \updownarrow (C \leftrightarrow D) = (A \updownarrow C) \leftrightarrow (B \updownarrow D)$

The proof that beside abides with below proceeds by showing that each of $(A \leftrightarrow B) \updownarrow (C \leftrightarrow D)$ and $(A \updownarrow C) \leftrightarrow (B \updownarrow D)$ is equal to the same symmetrical form; as follows

$$
\begin{array}{ll}
(A \leftrightarrow B) \updownarrow (C \leftrightarrow D) & \\
\quad = & \mathit{lsh} \,;\, \mathsf{snd}(C \leftrightarrow D) \,;\, \mathit{rsh} \,;\, \mathsf{fst}(A \leftrightarrow B) \,;\, \mathit{lsh} \\
\quad = & \mathit{lsh} \,;\, \mathsf{snd}\,(\mathit{rsh} \,;\, \mathsf{fst}\, C \,;\, \mathit{lsh} \,;\, \mathsf{snd}\, D \,;\, \mathit{rsh}) \,;\, \mathit{rsh} \,; \\
 & \quad \mathsf{fst}\,(\mathit{rsh} \,;\, \mathsf{fst}\, A \,;\, \mathit{lsh} \,;\, \mathsf{snd}\, B \,;\, \mathit{rsh}) \,;\, \mathit{lsh} \\
\quad = & \mathit{lsh} \,;\, \mathsf{snd}\, \mathit{rsh} \,;\, \mathsf{snd}\,\mathsf{fst}\, C \,;\, \mathsf{snd}\, \mathit{lsh} \,;\, \mathsf{snd}\,\mathsf{snd}\, D \,;\, \mathsf{snd}\, \mathit{rsh} \,;\, \mathit{rsh} \,; \\
 & \quad \mathsf{fst}\, \mathit{rsh} \,;\, \mathsf{fst}\,\mathsf{fst}\, A \,;\, \mathsf{fst}\, \mathit{lsh} \,;\, \mathsf{fst}\,\mathsf{snd}\, B \,;\, \mathsf{fst}\, \mathit{rsh} \,;\, \mathit{lsh}
\end{array}
$$

Now each of the component circuits occurs inside two applications of one of fst or snd, and these can be re-arranged into flatter forms because

$$\mathsf{snd}\,\mathsf{fst}\, C = [\iota, [C, \iota]] = \mathsf{apl} \,/\, [\iota, C, \iota]$$

and similarly $\mathsf{fst}\,\mathsf{snd}\, B = \mathsf{apr} \,/\, [\iota, B, \iota]$, and

$$\mathsf{fst}\,\mathsf{fst}\, A = [[A, \iota], \iota] = \mathsf{apr} \,/\, [A, \iota, \iota] = (\mathsf{apr} \,;\, \mathsf{apl}^{-1}) \,/\, [A, 2] = \mathit{lsh} \,/\, \mathsf{fst}\, A$$

and similarly $\mathsf{snd}\,\mathsf{snd}\, D = \mathit{rsh} \,/\, \mathsf{snd}\, D$.

Substituting these into the previous calculation, and remembering that lsh and rsh are inverses of each other

$$
\begin{array}{ll}
(A \leftrightarrow B) \updownarrow (C \leftrightarrow D) & \\
\quad = & (\mathit{lsh} \,;\, \mathsf{snd}\, \mathit{rsh} \,;\, \mathsf{apl}) \,;\, [\iota, C, \iota] \,;\, (\mathsf{apl}^{-1} \,;\, \mathsf{snd}\, \mathit{lsh} \,;\, \mathit{rsh}) \,;\, \mathsf{snd}\, D \,; \\
 & (\mathit{lsh} \,;\, \mathsf{snd}\, \mathit{rsh} \,;\, \mathit{rsh} \,;\, \mathsf{fst}\, \mathit{rsh} \,;\, \mathit{lsh}) \,; \\
 & \mathsf{fst}\, A \,;\, (\mathit{rsh} \,;\, \mathsf{fst}\, \mathit{lsh} \,;\, \mathsf{apr}) \,;\, [\iota, B, \iota] \,;\, (\mathsf{apr}^{-1} \,;\, \mathsf{fst}\, \mathit{rsh} \,;\, \mathit{lsh}) \\
\quad = & M \,;\, [\iota, C, \iota] \,;\, M^{-1} \,;\, \mathsf{snd}\, D \,;\, M \,;\, 3 \,;\, N^{-1} \,;\, \mathsf{fst}\, A \,;\, N \,;\, [\iota, B, \iota] \,;\, N^{-1}
\end{array}
$$

where $M = \mathit{lsh} \,;\, \mathsf{snd}\, \mathit{rsh} \,;\, \mathsf{apl}$ and $N = \mathit{rsh} \,;\, \mathsf{fst}\, \mathit{lsh} \,;\, \mathsf{apr}$.

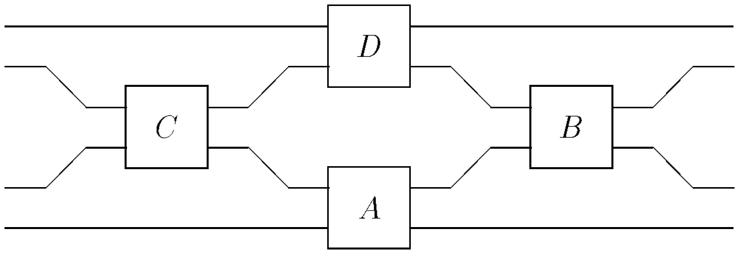

Figure 4.5: $(mid \mathbin{/} [\iota, C, \iota]) \mathbin{;} [A, D] \mathbin{;} (mid \mathbin{/} [\iota, B, \iota])$

In fact M and N are both equal to the unique $mid : [2,2] \sim [\iota, 2, \iota]$ for which $[[a,b],[c,d]] \mathbin{;} mid = mid \mathbin{;} [a,[b,c],d]$ and by far the easiest way to see this is to show that $M : [2,2] \sim [\iota, 2, \iota]$ and that for any a, b, c and d

$$\begin{array}{rcl} [[a,b],[c,d]] \mathbin{;} M & = & [[a,b],[c,d]] \mathbin{;} lsh \mathbin{;} \mathsf{snd}\ rsh \mathbin{;} \mathsf{apl} \\ & = & lsh \mathbin{;} [a,[b,[c,d]]\ \mathsf{snd}\ rsh \mathbin{;} \mathsf{apl} \\ & = & lsh \mathbin{;} \mathsf{snd}\ rsh \mathbin{;} [a,[[b,c]],d] \mathbin{;} \mathsf{apl} \\ & = & lsh \mathbin{;} \mathsf{snd}\ rsh \mathbin{;} \mathsf{apl} \mathbin{;} [a,[b,c],d] \\ & = & M \mathbin{;} [a,[b,c],d] \end{array}$$

and similarly for N. Moreover, since $mid \mathbin{;} 3 \mathbin{;} mid^{-1} = mid \mathbin{;} mid^{-1} = [2,2]$,

$$\begin{array}{l} (A \leftrightarrow B) \updownarrow (C \leftrightarrow D) \\ \quad = \quad mid \mathbin{;} [\iota, C, \iota] \mathbin{;} mid^{-1} \mathbin{;} \mathsf{snd}\ D \mathbin{;} [2,2] \mathbin{;} \mathsf{fst}\ A \mathbin{;} mid \mathbin{;} [\iota, B, \iota] \mathbin{;} mid^{-1} \\ \quad = \quad mid \mathbin{;} [\iota, C, \iota] \mathbin{;} mid^{-1} \mathbin{;} [2 \mathbin{;} A, D \mathbin{;} 2] \mathbin{;} mid \mathbin{;} [\iota, B, \iota] \mathbin{;} mid^{-1} \\ \quad = \quad (mid \mathbin{/} [\iota, C, \iota]) \mathbin{;} [A, D] \mathbin{;} (mid \mathbin{/} [\iota, B, \iota]) \end{array}$$

A circuit layout suggestive of this symmetrical form is shown in figure 4.5. You can now either argue from symmetry, or rather the duality of below and beside, that

$$\begin{array}{l} (A \updownarrow C) \leftrightarrow (B \updownarrow D) \\ \quad = \quad ((A^{-1} \leftrightarrow C^{-1}) \updownarrow (B^{-1} \leftrightarrow D^{-1}))^{-1} \\ \quad = \quad ((mid \mathbin{/} [\iota, B^{-1}, \iota]) \mathbin{;} [A^{-1}, D^{-1}] \mathbin{;} (mid \mathbin{/} [\iota, C^{-1}, \iota]))^{-1} \\ \quad = \quad (mid \mathbin{/} [\iota, C^{-1}, \iota])^{-1} \mathbin{;} [A^{-1}, D^{-1}]^{-1} \mathbin{;} (mid \mathbin{/} [\iota, B^{-1}, \iota])^{-1} \\ \quad = \quad (mid \mathbin{/} [\iota, C, \iota]) \mathbin{;} [A, D] \mathbin{;} (mid \mathbin{/} [\iota, B, \iota]) \end{array}$$

which completes the proof that beside abides with below.

The other things that one may want to know about beside and below also say that it does not matter how you choose to bracket a picture, such as those in

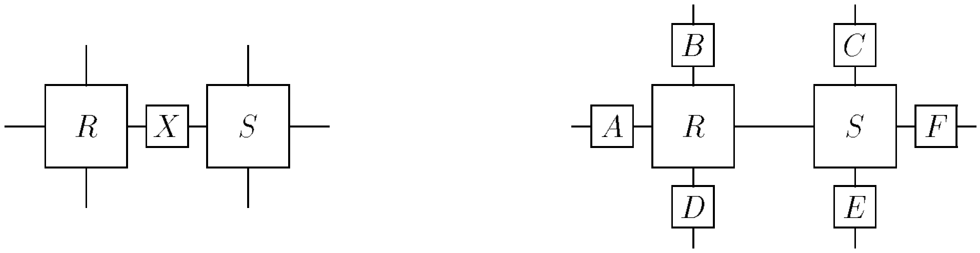

Figure 4.6: $(R ; \mathsf{snd}\, X) \leftrightarrow S = R \leftrightarrow (\mathsf{fst}\, X ; S)$ and $[A, [B, C]] ; (R \leftrightarrow S) ; [[D, E], F] = ([A, B] ; R ; \mathsf{fst}\, D) \leftrightarrow (\mathsf{snd}\, C ; S ; [E, F])$

figure 4.6. First it is the case that

$$
\begin{aligned}
(R ; \mathsf{snd}\, X) \leftrightarrow S &= rsh ; \mathsf{fst}(R ; \mathsf{snd}\, X) ; lsh ; \mathsf{snd}\, S ; rsh \\
&= rsh ; \mathsf{fst}\, R ; \mathsf{fst}\, \mathsf{snd}\, X ; lsh ; \mathsf{snd}\, S ; rsh \\
&= rsh ; \mathsf{fst}\, R ; lsh ; \mathsf{snd}\, \mathsf{fst}\, X ; \mathsf{snd}\, S ; rsh \\
&= rsh ; \mathsf{fst}\, R ; lsh ; \mathsf{snd}(\mathsf{fst}\, X ; S) ; rsh \\
&= R \leftrightarrow (\mathsf{fst}\, X ; S)
\end{aligned}
$$

and similarly by inverting both sides $R \updownarrow (S ; \mathsf{fst}\, X) = (\mathsf{snd}\, X ; R) \updownarrow S$. In particular, if $P ; Q = Y ; Z$ and $R = R ; \mathsf{snd}\, Y$ and $\mathsf{fst}\, Z ; S = S$ then $(R ; \mathsf{snd}\, P) \leftrightarrow (\mathsf{fst}\, Q ; S) = R \leftrightarrow S$. Secondly it is the case that

$$
\begin{aligned}
&[A, [B, C]] ; (R \leftrightarrow S) ; [[D, E], F] \\
&\quad = [A, [B, C]] ; rsh ; \mathsf{fst}\, R ; lsh ; \mathsf{snd}\, S ; rsh ; [[D, E], F] \\
&\quad = rsh ; [[A, B], C] ; \mathsf{fst}\, R ; lsh ; \mathsf{snd}\, S ; [D, [E, F]] ; rsh \\
&\quad = rsh ; \mathsf{fst}([A, B] ; R) ; \mathsf{snd}\, C ; lsh ; \mathsf{fst}\, D ; \mathsf{snd}(S ; [E, F]) ; rsh \\
&\quad = rsh ; \mathsf{fst}([A, B] ; R) ; \mathsf{fst}\, \mathsf{fst}\, D ; lsh ; \mathsf{snd}\, \mathsf{snd}\, C ; \mathsf{snd}(S ; [E, F]) ; rsh \\
&\quad = rsh ; \mathsf{fst}([A, B] ; R ; \mathsf{fst}\, D) ; lsh ; \mathsf{snd}(\mathsf{snd}\, C ; S ; [E, F]) ; rsh \\
&\quad = ([A, B] ; R ; \mathsf{fst}\, D) \leftrightarrow (\mathsf{snd}\, C ; S ; [E, F])
\end{aligned}
$$

and similarly $[[A, B], C] ; (R \updownarrow S) ; [D, [E, F]] = (\mathsf{fst}\, A ; R ; [D, E]) \updownarrow ([B, C] ; S ; \mathsf{snd}\, C)$.

Most of the things that one needs to prove about these circuits are consequences of these last observations, setting various of the variables to the identity: for example, suppose that $\mathsf{fst}\, X ; R = S ; \mathsf{snd}\, X$ and $\mathsf{fst}\, X ; S = R ; \mathsf{snd}\, X$, then

$$
\begin{aligned}
\mathsf{fst}\, X ; (R \leftrightarrow S) &= (\mathsf{fst}\, X ; R) \leftrightarrow S \\
&= (S ; \mathsf{snd}\, X) \leftrightarrow S \\
&= S \leftrightarrow (\mathsf{fst}\, X ; S) \\
&= S \leftrightarrow (R ; \mathsf{snd}\, X) \\
&= (S \leftrightarrow R) ; \mathsf{snd}\, X
\end{aligned}
$$

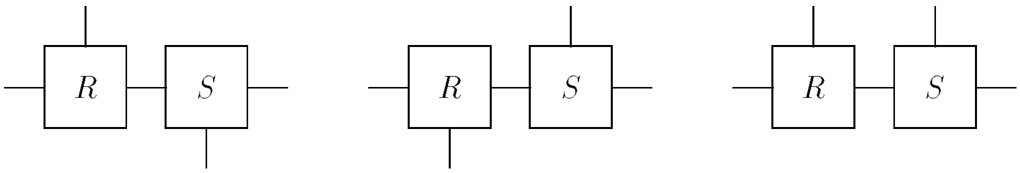

Figure 4.7: $2 \,/\, (R\,;S)$ and $\mathsf{snd}\,{\pi_2}^{-1}\,;((\pi_1\,;R) \leftrightarrow (S\,;{\pi_2}^{-1}))\,;\mathsf{fst}\,\pi_1$ and $((R\,;{\pi_2}^{-1}) \leftrightarrow (S\,;{\pi_2}^{-1}))\,;\pi_2$

4.2 Other orthogonally connected circuits

There are a number of other ways that circuits might be connected with the wires lying orthogonally to each other, and that might be interesting: for example it will be common to connect circuits with two inputs and one output in rows when doing things that are like adding up a row of numbers. There seem to be a vast number of possibilities corresponding to the combinations of connections that might be missing from the picture of beside and below. Three of these configurations are illustrated in figure 4.7.

Fortunately, however, these are all just instances of beside, for example $2 \,/\, (R\,; S) = \mathsf{snd}\,{\pi_1}^{-1}\,;((R\,;{\pi_2}^{-1}) \leftrightarrow (\pi_1\,;S))\,;\mathsf{fst}\,\pi_2$. This means that the existing results about beside carry over immediately to any new operator of this form that we might want. For example, suppose that $R \oplus S = \mathsf{snd}\,{\pi_2}^{-1}\,;((\pi_1\,;R) \leftrightarrow (S\,;{\pi_2}^{-1}))\,;\mathsf{fst}\,\pi_1$, which is one of the configurations shown in figure 4.7, then

$$
\begin{array}{ll}
\multicolumn{2}{l}{[A,B]\,;(R \oplus S)\,;[C,D]} \\
= & [A,B]\,;\mathsf{snd}\,{\pi_2}^{-1}\,;((\pi_1\,;R) \leftrightarrow (S\,;{\pi_2}^{-1}))\,;\mathsf{fst}\,\pi_1\,;[C,D] \\
= & \mathsf{snd}\,{\pi_2}^{-1}\,;[A,\mathsf{snd}\,B]\,;((\pi_1\,;R) \leftrightarrow (S\,;{\pi_2}^{-1}))\,;[\mathsf{fst}\,C,D]\,;\mathsf{fst}\,\pi_1 \\
= & \mathsf{snd}\,{\pi_2}^{-1}\,;((\mathsf{fst}\,A\,;\pi_1\,;R\,;\mathsf{fst}\,C) \leftrightarrow (\mathsf{snd}\,B\,;S\,;{\pi_2}^{-1}\,;\mathsf{snd}\,D))\,;\mathsf{fst}\,\pi_1 \\
= & \mathsf{snd}\,{\pi_2}^{-1}\,;((\pi_1\,;(A\,;R\,;\mathsf{fst}\,C)) \leftrightarrow ((\mathsf{snd}\,B\,;S\,;D)\,;{\pi_2}^{-1}))\,;\mathsf{fst}\,\pi_1 \\
= & (A\,;R\,;\mathsf{fst}\,C) \oplus (\mathsf{snd}\,B\,;S\,;D)
\end{array}
$$

It is the existence of the inverses of the projections that make this unification possible, and this is one of the excuses for not confining ourselves to functions, nor indeed to total functions.

Notice that the argument above carefully avoids attempting to prove the equality of, say

$$
\begin{array}{ll}
\multicolumn{2}{l}{[A,[X,B]]\,;((\pi_1\,;R) \leftrightarrow (S\,;{\pi_2}^{-1}))\,;[[C,Y],D]} \\
\neq & (\pi_1\,;(A\,;R\,;\mathsf{fst}\,C)) \leftrightarrow ((\mathsf{snd}\,B\,;S\,;D)\,;{\pi_2}^{-1})
\end{array}
$$

because the two sides are not equal for general X and Y.

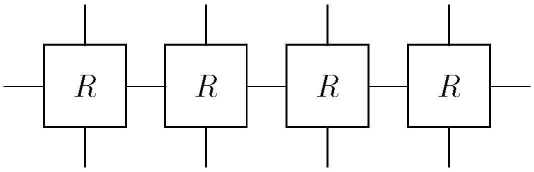

Figure 4.8: an instance of $\mathsf{row}\, R$

Figure 4.9: another instance of $\mathsf{row}\, R$

4.3 Rows

Having put two circuits next to each other with beside, the next step is of course to put another next to them and so on, as shown in figure 4.8. This is the same generalization that takes you from parallel composition to map, and the following requirements on row should look very like those for map

$$\begin{aligned}
\mathsf{snd}\,[\text{-}]\,;\mathsf{row}\, R &= R\,;\mathsf{fst}\,[\text{-}] \\
\mathsf{row}\, R\,;\mathsf{fst}\,[\text{-}] &= \mathsf{snd}\,[\text{-}]\,;R \\
\mathsf{snd}([m,n]\,;\mathsf{app})\,;\mathsf{row}\, R &= ((\mathsf{row}\, R\,;\mathsf{fst}\, m) \leftrightarrow (\mathsf{row}\, R\,;\mathsf{fst}\, n))\,;\mathsf{fst}\,\mathsf{app} \\
\mathsf{row}\, R\,;\mathsf{fst}(\mathsf{app}^{-1}\,;[m,n]) &= \mathsf{snd}\,\mathsf{app}^{-1}\,;((\mathsf{snd}\, m\,;\mathsf{row}\, R) \leftrightarrow (\mathsf{snd}\, n\,;\mathsf{row}\, R))
\end{aligned}$$

Notice that these four conditions cannot be abbreviated in the same way as those for map, because $(\mathsf{row}\, R)^{-1} \neq \mathsf{row}\, R^{-1}$.

It is a little trickier to see what the requirements are on the degenerate row, one with no components in it at all. The only meaning for a row of zero width, shown in figure 4.9, that is consistent with the decomposition of a row into two is that it must connect its left and right signals and have zero-length lists of signals at the top and bottom, so something like $\mathsf{snd}\, 0\,;\mathsf{row}\, R = \mathsf{row}\, R\,;\mathsf{fst}\, 0 = \mathsf{snd}\, 0\,;\pi_1\,;{\pi_2}^{-1}\,;\mathsf{fst}\, 0 = \mathsf{snd}\, 0\,;\mathsf{rev} = \mathsf{rev}\,;\mathsf{fst}\, 0$. However, by analogy with $R^0 \neq \iota$, it is more consistent to define $\mathsf{snd}\, 0\,;\mathsf{row}(R : \mathsf{fst}\, T \sim \mathsf{snd}\, T)\,;\mathsf{fst}\, 0 = [T, 0]\,;\mathsf{rev}\,;[0, T]$.

The interface rule for row, by analogy with that for beside, would seem to be that if $D\,;A = Y\,;Z$ where $R : \mathsf{fst}\, Z \sim \mathsf{snd}\, Y$, then

$$[A, \mathsf{map}\, B]\,;\mathsf{row}\, R\,;[\mathsf{map}\, C, D] = \mathsf{row}([A, B]\,;R\,;[C, D])$$

You can show this for all non-empty rows, by induction on the width of the row.

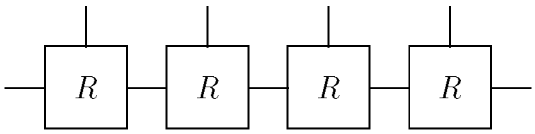

Figure 4.10: an instance of $\mathsf{rdl}\, R$

It turns out not to be consistent with $\mathsf{snd}\, 0 \,;\, \mathsf{row}\, R \,;\, \mathsf{fst}\, 0 = [T, 0] \,;\, \mathsf{rev} \,;\, [0, T]$ unless the hypothesis is strengthened to include $A \,;\, D = T$.

The inductive consequence of the interface rule about a component lying between two circuits beside each other is that $\mathsf{fst}\, X \,;\, \mathsf{row}(R \,;\, \mathsf{snd}\, X) = \mathsf{row}(\mathsf{fst}\, X \,;\, R) \,;\, \mathsf{snd}\, X$ again for non-empty rows. For it to be true of empty rows it is necessary that $X \,;\, T = T \,;\, X$ where $\mathsf{snd}\, 0 \,;\, \mathsf{row}\, R \,;\, \mathsf{fst}\, 0 = [T, 0] \,;\, \mathsf{rev} \,;\, [0, T]$, although this is quite a reasonable requirement since it will usually be the case that $X : T \sim T$.

There is a row-like repeated form that deserves note, namely $\mathsf{rdl}\, R$ read 'reduce (from the) left'. This is illustrated in figure 4.10 and defined by $\mathsf{rdl}\, R = \mathsf{row}(R \,;\, {\pi_2}^{-1}) \,;\, \pi_2$. Its properties are immediate from the corresponding properties of row.

It should be suggesting operations like summation to you: for example, if $\langle x, y\rangle\; add\; (x + y)$ then $\langle a, \langle p, q, r\rangle\rangle\; (\mathsf{rdl}\; add)\; (((a + p) + q) + r)$. The order of the bracketing – from the left – is important for non-associative operators, for example $\mathsf{rdl}\,\mathsf{apr} = \mathsf{map}\, \iota$ and $\mathsf{rdl}(\mathsf{rev} \,;\, \mathsf{apl}) = \mathsf{rev}$.

4.4 Columns

Of course if beside has below as its dual, there is going to be a corresponding dual for row, which is column, $\mathsf{col}\, R = (\mathsf{row}\, R^{-1})^{-1}$, an instance of which is illustrated in figure 4.11. and its properties are the duals of the properties of row.

$$
\begin{array}{rcl}
\mathsf{fst}\,[\text{-}] \,;\, \mathsf{col}\, R & = & R \,;\, \mathsf{snd}\,[\text{-}] \\
\mathsf{col}\, R \,;\, \mathsf{snd}\,[\text{-}] & = & \mathsf{fst}\,[\text{-}] \,;\, R \\
\mathsf{fst}([m, n] \,;\, \mathsf{app}) \,;\, \mathsf{col}\, R & = & ((\mathsf{col}\, R \,;\, \mathsf{snd}\, m) \updownarrow (\mathsf{col}\, R \,;\, \mathsf{snd}\, n)) \,;\, \mathsf{snd}\,\mathsf{app} \\
\mathsf{col}\, R \,;\, \mathsf{snd}(\mathsf{app}^{-1} \,;\, [m, n]) & = & \mathsf{fst}\,\mathsf{app}^{-1} \,;\, ((\mathsf{fst}\, m \,;\, \mathsf{col}\, R) \updownarrow (\mathsf{fst}\, n \,;\, \mathsf{col}\, R))
\end{array}
$$

and so on.

The column-shaped thing that corresponds to reduce from the left is of course reduce from the right, defined by $\mathsf{rdr}\, R = \mathsf{col}(R \,;\, {\pi_1}^{-1}) \,;\, \pi_1$. It might seem that it would have been more logical to christen this something like 'reduce downwards', but again its name derives from the bracketing in the formula which describes the reduction of a function; for example, if $\langle x, y\rangle\; add\; (x + y)$ then $\langle\langle p, q, r\rangle, a\rangle\; (\mathsf{rdr}\; add)\; (p + (q + (r + a)))$.

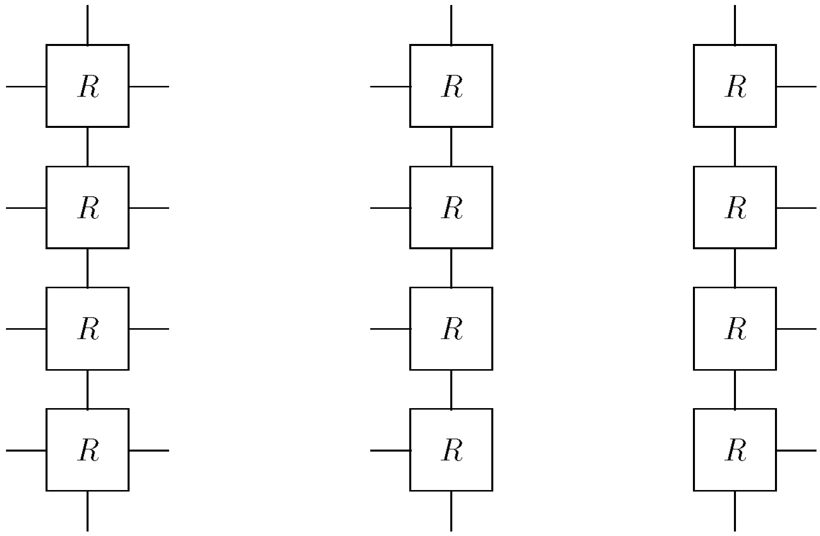

Figure 4.11: an instance of $\mathsf{col}\,R$, one of $\mathsf{rdr}\,R$, and one of $(\mathsf{rdl}(R^{-1}))^{-1}$

Beware that reduce right is not the inverse-dual of reduce left! The easiest way to see the difference is to compare the examples of $\mathsf{rdr}\,R$ and $(\mathsf{rdl}(R^{-1}))^{-1}$ that are shown in figure 4.11. Of course $(\mathsf{rdr}(R^{-1}))^{-1}$ is yet a fourth different reduce-shaped circuit.

4.5 Grids

Since for any $R : 2{\sim}2$ both $\mathsf{row}\,R : 2{\sim}2$ and $\mathsf{col}\,R : 2{\sim}2$, it is natural to consider the properties of $\mathsf{row}\,\mathsf{col}\,R$ and $\mathsf{col}\,\mathsf{row}\,R$. These will both look like the example shown in figure 4.12, and in fact you can show by induction that $\mathsf{row}\,\mathsf{col}\,R = \mathsf{col}\,\mathsf{row}\,R$.

The proof is an induction that eventually reduces to the abides property of the below and beside operators. There is a little subtlety in the exact choice of induction hypothesis, but the inductive step proceeds as follows. Writing $\mathsf{row}_p\,R$ for $\mathsf{snd}\,p\,;\mathsf{row}\,R\,;\mathsf{fst}\,p$ and $\mathsf{col}_m\,R$ for $\mathsf{fst}\,m\,;\mathsf{col}\,R\,;\mathsf{snd}\,m$,

$$
\begin{aligned}
&[[m,n]\,;\mathsf{app},[p,q]\,;\mathsf{app}]\,;\mathsf{row}\,\mathsf{col}\,R\\
&\qquad = \{\text{ decomposition of row }\}\\
&\qquad\quad \mathsf{fst}([m,n]\,;\mathsf{app})\,;(\mathsf{row}_p\,\mathsf{col}_{m+n}\,R \leftrightarrow \mathsf{row}_q\,\mathsf{col}_{m+n}\,R)\,;\mathsf{fst}([p,q]\,;\mathsf{app})\\
&\qquad = \{\text{ induction hypothesis (1) }\}\\
&\qquad\quad \mathsf{fst}([m,n]\,;\mathsf{app})\,;(\mathsf{col}_{m+n}\,\mathsf{row}_p\,R \leftrightarrow \mathsf{col}_{m+n}\,\mathsf{row}_q\,R)\,;\mathsf{fst}([p,q]\,;\mathsf{app})\\
&\qquad = \{\text{ interface property of beside }\}\\
&\qquad\quad ((\mathsf{fst}([m,n]\,;\mathsf{app})\,;\mathsf{col}_{m+n}\,\mathsf{row}_p\,R) \leftrightarrow \mathsf{col}_{m+n}\,\mathsf{row}_q\,R)\,;\mathsf{fst}([p,q]\,;\mathsf{app})
\end{aligned}
$$

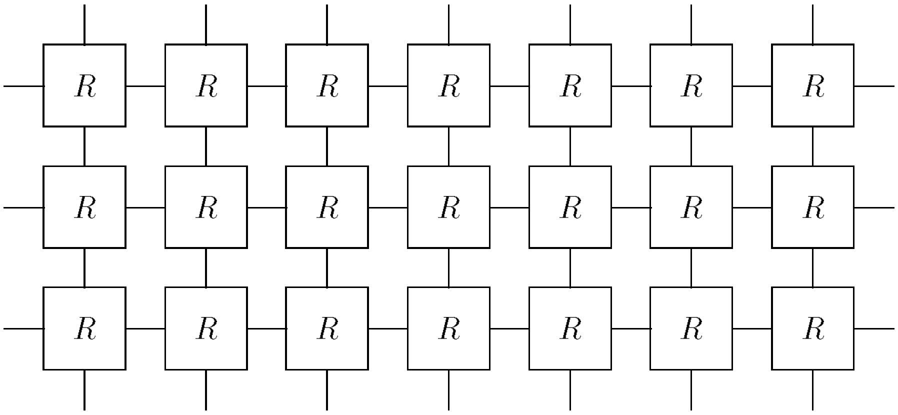

Figure 4.12: an instance of $\mathsf{row}\,\mathsf{col}\,R = \mathsf{col}\,\mathsf{row}\,R$

$$
\begin{aligned}
&= \{\text{ property of column }\}\\
&\quad (((\mathsf{col}_m\,\mathsf{row}_p\,R \updownarrow \mathsf{col}_n\,\mathsf{row}_p\,R)\,;\mathsf{snd}([m,n]\,;\mathsf{app})) \leftrightarrow \mathsf{col}_{m+n}\,\mathsf{row}_q\,R)\,;\\
&\quad \mathsf{fst}([p,q]\,;\mathsf{app})\\
&= \{\text{ interface property of beside and the same property of column }\}\\
&\quad ((\mathsf{col}_m\,\mathsf{row}_p\,R \updownarrow \mathsf{col}_n\,\mathsf{row}_p\,R) \leftrightarrow\\
&\quad ((\mathsf{col}_m\,\mathsf{row}_q\,R \updownarrow \mathsf{col}_n\,\mathsf{row}_q\,R)\,;\mathsf{snd}([m,n]\,;\mathsf{app})))\,;\\
&\quad \mathsf{fst}([p,q]\,;\mathsf{app})\\
&= \{\text{ interface property of beside }\}\\
&\quad ((\mathsf{col}_m\,\mathsf{row}_p\,R \updownarrow \mathsf{col}_n\,\mathsf{row}_p\,R) \leftrightarrow (\mathsf{col}_m\,\mathsf{row}_q\,R \updownarrow \mathsf{col}_n\,\mathsf{row}_q\,R))\,;\\
&\quad [[p,q]\,;\mathsf{app},[m,n]\,;\mathsf{app}]\\
&= \{\text{ abiding property of below and beside, and induction hypothesis (2) }\}\\
&\quad ((\mathsf{row}_p\,\mathsf{col}_m\,R \leftrightarrow \mathsf{row}_p\,\mathsf{col}_n\,R) \updownarrow (\mathsf{row}_q\,\mathsf{col}_m\,R \leftrightarrow \mathsf{row}_q\,\mathsf{col}_n\,R))\,;\\
&\quad [[p,q]\,;\mathsf{app},[m,n]\,;\mathsf{app}]\\
&= \{\text{ by an exactly dual argument }\}\\
&\quad [[m,n]\,;\mathsf{app},[p,q]\,;\mathsf{app}]\,;\mathsf{col}\,\mathsf{row}\,R
\end{aligned}
$$

Note carefully the nature of the inductive hypothesis:

- at point (1) assume that $\mathsf{row}_p\,\mathsf{col}_{m+n}\,R = \mathsf{col}_{m+n}\,\mathsf{row}_p$ for all $p < p+q$,
- at point (2) assume $\mathsf{row}_p\,\mathsf{col}_m\,R = \mathsf{col}_m\,\mathsf{row}_p\,R$ for all $p < p+q$ and $m < m+n$,
- and at the point which corresponds to point (1) in the dual argument assume $\mathsf{row}_{p+q}\,\mathsf{col}_m\,R = \mathsf{col}_m\,\mathsf{row}_{p+q}\,R$ for all $m < m+n$.

So each assumption is either for a smaller row, or a smaller column, or both, and in any case for neither a larger row nor a larger column; therefore a sound induction follows from showing the result both for a degenerate row and a degenerate column.

The proof for a degenerate column is that

$$\begin{aligned}
&\mathsf{fst}\,[\,\text{-}\,]\,;\,\mathsf{row}\,\mathsf{col}\,R \\
&\quad = \mathsf{fst}\,[\,\text{-}\,]\,;\,\mathsf{row}(\mathsf{fst}\,1\,;\,\mathsf{col}\,R\,;\,\mathsf{snd}\,1)\,;\,\mathsf{snd}\,1 \\
&\quad = \mathsf{fst}\,1\,;\,\mathsf{row}(\mathsf{fst}\,[\,\text{-}\,]\,;\,\mathsf{col}\,R\,;\,\mathsf{snd}\,[\,\text{-}\,]^{-1})\,;\,\mathsf{snd}\,[\,\text{-}\,] \\
&\quad = \mathsf{row}\,R\,;\,\mathsf{snd}\,[\,\text{-}\,] \\
&\quad = \mathsf{fst}\,[\,\text{-}\,]\,;\,\mathsf{col}\,\mathsf{row}\,R
\end{aligned}$$

and that for a degenerate column is of course an exact dual, as can be inferred from $(\mathsf{row}\,\mathsf{col}\,R)^{-1} = \mathsf{col}\,\mathsf{row}\,R^{-1}$.

4.6 Horner's rule

Horner's rule is the name usually given to a method of evaluating polynomials without needing to raise powers or do unnecessary multiplications; it is encapsulated in the equality of $a_0 + xa_1 + \cdots + x^{n-2}a_{n-2} + x^{n-1}a_{n-1} + x^n a_n = a_0 + x(a_1 + \cdots + x(a_{n-2} + x(a_{n-1} + xa_n))\cdots)$. Naïve evaluation of the left-hand side would appear to require a quadratic number of multiplications, that of the right-hand side only a linear number. The equality is a consequence of a property that is usually expressed as the distribution of multiplication over addition. Specifically it follows from $x \times a + x \times b = x \times (a + b)$.

The rule applies to any structure with two operations that distribute in this way, and in particular it has a counterpart in circuits: if $[R, R]\,;\,S = S\,;\,R$ then $(\mathsf{apr}\,/\,\mathsf{tri}\,R)\,;\,\mathsf{rdr}\,S = \mathsf{rdr}(\mathsf{tri}\,R\,;\,S)$. The hypothesis is illustrated in figure 4.13 and an example of the consequence for a particular size appears in figure 4.14.

The most familiar instance of Horner's rule will convert a list of bits into the number that they represent in binary notation. Suppose R is the circuit $(\times 2)$ that doubles a number, then recall that $\mathsf{tri}(\times 2)$ relates a list of zeroes and ones to a list of those bits weighted by the powers of two which they would represent in a binary number with its least significant bit first. Suppose that S is the circuit add that relates a pair of numbers to their sum, then $\mathsf{apr}^{-1}\,;\,\mathsf{rdr}\,add$ relates a list of numbers to their sum; so $N = \mathsf{tri}(\times 2)\,;\,\mathsf{apr}^{-1}\,;\,\mathsf{rdr}\,add$ relates a list of bits to the number that they represent as a binary number, least significant bit first. (It also relates other lists of numbers to the same value, but that does not matter for the moment.)

The hypothesis of Horner's rule is satisfied by these two circuits, because the circuit on one side is $[(\times 2), (\times 2)]\,;\,add$ which relates a pair of numbers to the sum of twice each of them, and that on the other is $\mathsf{fst}(\times 2)^0\,;\,add\,;\,(\times 2) = add\,;$

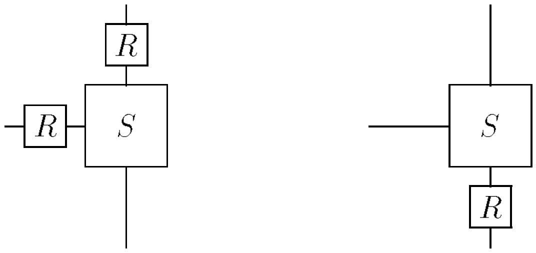

Figure 4.13: the hypothesis for Horner's rule, that $[R, R]\,;S = S\,;R$

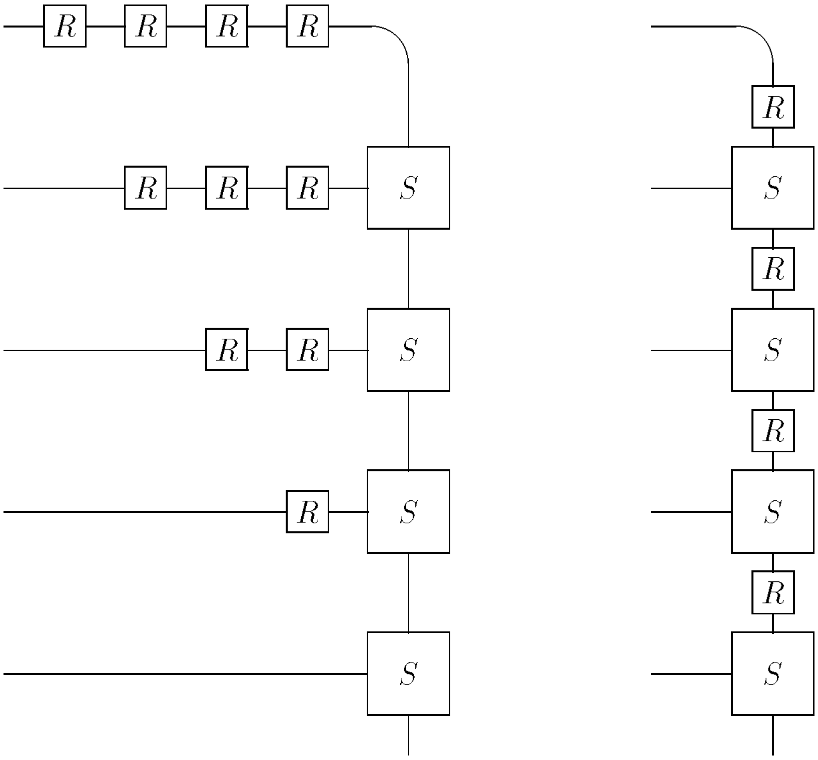

Figure 4.14: an instance of the consequence of Horner's rule, that $\mathsf{tri}\,R;\mathsf{apr}^{-1};\mathsf{rdr}\,S = \mathsf{apr}^{-1}\,;\mathsf{rdr}(\mathsf{tri}\,R\,;S)$

$(\times 2)$ which relates a pair of numbers to twice their sum. These are the same circuit because $2x + 2y = 2(x + y)$, so $[(\times 2), (\times 2)] \,;\, add = \mathsf{fst}(\times 2)^0 \,;\, add \,;\, (\times 2)$. Our formulation of Horner's rule therefore allows the conclusion that $N = \mathsf{apr}^{-1} \,;\, \mathsf{rdr}(\mathsf{tri}(\times 2); add) = \mathsf{apr}^{-1}; \mathsf{rdr}\, step$ where $step = \mathsf{snd}(\times 2); add$, which is to say that a bit-vector can be converted into the number that it represents by a right-reduction of *step* components, each of which doubles one input and adds it to the other.

The proof of Horner's rule is an induction on the size of the reduction. In order to simplify the notation, we will prove a slightly different statement, namely that if $[R, R] \,;\, S' = \mathsf{fst}\, R^0 \,;\, S' \,;\, \mathsf{fst}\, R$ then $\mathsf{apr} \,/\, \mathsf{tri}\, R \,;\, \mathsf{col}\, S' = \mathsf{col}([R^0, R] \,;\, S')$. This is adequate, because if $S' = S \,;\, {\pi_1}^{-1}$, then from the hypothesis of the rule

$$\begin{aligned} [R, R] \,;\, S' &= \mathsf{fst}\, R^0 \,;\, [R, R] \,;\, S \,;\, {\pi_1}^{-1} \\ &= \mathsf{fst}\, R^0 \,;\, S \,;\, R \,;\, {\pi_1}^{-1} \\ &= \mathsf{fst}\, R^0 \,;\, S \,;\, {\pi_1}^{-1} \,;\, \mathsf{fst}\, R \\ &= \mathsf{fst}\, R^0 \,;\, S' \,;\, \mathsf{fst}\, R \end{aligned}$$

and once the result about col is proved, from the definition of rdr

$$\begin{aligned} \mathsf{apr} \,/\, \mathsf{tri}\, R \,;\, \mathsf{rdr}\, S &= \mathsf{apr} \,/\, \mathsf{tri}\, R \,;\, \mathsf{col}\, S' \,;\, \pi_1 \\ &= \mathsf{col}([R^0, R] \,;\, S') \,;\, \pi_1 \\ &= \mathsf{rdr}([R^0, R] \,;\, S) \end{aligned}$$

which is the form of the statement to be proved.

The induction step in the proof goes as follows

$$\begin{aligned}
&\mathsf{fst\,apl} \,;\, \mathsf{apr} \,/\, \mathsf{tri}\, R \,;\, \mathsf{col}\, S' \\
&\quad = \{\, \mathsf{fst\,apl} \,;\, \mathsf{apr} = lsh \,;\, \mathsf{snd\,apr} \,;\, \mathsf{apl} \,\} \\
&\qquad lsh \,;\, \mathsf{snd\,apr} \,;\, \mathsf{apl} \,;\, \mathsf{tri}\, R \,;\, \mathsf{apr}^{-1} \,;\, \mathsf{col}\, S' \\
&\quad = \{\, \text{definition of triangle} \,\} \\
&\qquad lsh \,;\, \mathsf{snd\,apr} \,;\, [R^0, \mathsf{map}\, R \,;\, \mathsf{tri}\, R] \,;\, \mathsf{apl} \,;\, \mathsf{apr}^{-1} \,;\, \mathsf{col}\, S' \\
&\quad = \{\, \text{homogeneity of map and triangle} \,\} \\
&\qquad lsh \,;\, \mathsf{snd\,apr} \,;\, [R^0, \mathsf{map}\, R \,;\, \mathsf{tri}\, R] \,;\, \mathsf{snd\,apr}^{-1} \,;\, \mathsf{snd\,apr} \,;\, \mathsf{apl} \,;\, \mathsf{apr}^{-1} \,;\, \mathsf{col}\, S' \\
&\quad = \{\, \mathsf{snd\,apr} \,;\, \mathsf{apl} \,;\, \mathsf{apr}^{-1} = rsh \,;\, \mathsf{fst\,apl} \,\} \\
&\qquad lsh \,;\, \mathsf{snd\,apr} \,;\, [R^0, \mathsf{map}\, R \,;\, \mathsf{tri}\, R] \,;\, \mathsf{snd\,apr}^{-1} \,;\, rsh \,;\, \mathsf{fst\,apl} \,;\, \mathsf{col}\, S' \\
&\quad = \{\, \text{decomposition of column} \,\} \\
&\qquad lsh \,;\, \mathsf{snd\,apr} \,;\, [R^0, \mathsf{map}\, R \,;\, \mathsf{tri}\, R \,;\, \mathsf{apr}^{-1}] \,;\, rsh \,;\, (S' \updownarrow \mathsf{col}\, S') \,;\, \mathsf{snd\,apl} \\
&\quad = \{\, \text{interface of below} \,\} \\
&\qquad lsh \,;\, \mathsf{snd\,apr} \,;\, rsh \,;\, ((\mathsf{fst}\, R^0 \,;\, S') \updownarrow (\mathsf{map}\, R \,;\, \mathsf{tri}\, R \,;\, \mathsf{apr}^{-1} \,;\, \mathsf{col}\, S')) \,;\, \mathsf{snd\,apl}
\end{aligned}$$

At this point the induction hypothesis can be invoked, because the remaining instance of $\mathsf{col}\, S'$ is strictly smaller, so

$$
\begin{aligned}
&\mathsf{map}\, R \,;\, \mathsf{tri}\, R \,;\, \mathsf{apr}^{-1} \,;\, \mathsf{col}\, S' \\
&\quad = \{\text{ induction hypothesis }\} \\
&\qquad \mathsf{map}\, R \,;\, \mathsf{apr}^{-1} \,;\, \mathsf{col}([R^0, R] \,;\, S') \\
&\quad = \{\text{ property of map }\} \\
&\qquad \mathsf{apr}^{-1} \,;\, [\mathsf{map}\, R, R] \,;\, \mathsf{col}([R^0, R] \,;\, S') \\
&\quad = \{\text{ interface for column }\} \\
&\qquad \mathsf{apr}^{-1} \,;\, \mathsf{snd}\, R \,;\, \mathsf{col}([R, R] \,;\, S') \\
&\quad = \{\text{ hypothesis of Horner's rule }\} \\
&\qquad \mathsf{apr}^{-1} \,;\, \mathsf{snd}\, R \,;\, \mathsf{col}(\mathsf{fst}\, R^0 \,;\, S' \,;\, \mathsf{fst}\, R) \\
&\quad = \{\text{ interface for column }\} \\
&\qquad \mathsf{apr}^{-1} \,;\, \mathsf{col}([R^0, R] \,;\, S') \,;\, \mathsf{fst}\, R
\end{aligned}
$$

then substituting back into the original calculation

$$
\begin{aligned}
&\mathsf{fst}\,\mathsf{apl} \,;\, \mathsf{apr} \,/\, \mathsf{tri}\, R \,;\, \mathsf{col}\, S' \\
&\quad = \; lsh \,;\, \mathsf{snd}\,\mathsf{apr} \,;\, rsh \,;\, ((\mathsf{fst}\, R^0 \,;\, S') \updownarrow (\mathsf{apr}^{-1} \,;\, \mathsf{col}([R^0, R] \,;\, S') \,;\, \mathsf{fst}\, R)) \,;\, \mathsf{snd}\,\mathsf{apl} \\
&\quad = \{\text{ interface for below }\} \\
&\qquad lsh \,;\, \mathsf{snd}\,\mathsf{apr} \,;\, rsh \,;\, (([R^0, R] \,;\, S') \updownarrow (\mathsf{apr}^{-1} \,;\, \mathsf{col}([R^0, R] \,;\, S'))) \,;\, \mathsf{snd}\,\mathsf{apl} \\
&\quad = \{\text{ interface for below }\} \\
&\qquad lsh \,;\, \mathsf{snd}\,\mathsf{apr} \,;\, \mathsf{snd}\,\mathsf{apr}^{-1} \,;\, rsh \,;\, (([R^0, R] \,;\, S') \updownarrow \mathsf{col}([R^0, R] \,;\, S')) \,;\, \mathsf{snd}\,\mathsf{apl} \\
&\quad = \{\; lsh \,;\, \mathsf{snd}(\mathsf{apr} \,;\, \mathsf{apr}^{-1}) \,;\, rsh = \mathsf{fst}(\mathsf{apl} \,;\, \mathsf{apl}^{-1}) \;\} \\
&\qquad \mathsf{fst}\,\mathsf{apl} \,;\, \mathsf{fst}\,\mathsf{apl}^{-1} \,;\, (([R^0, R] \,;\, S') \updownarrow \mathsf{col}([R^0, R] \,;\, S')) \,;\, \mathsf{snd}\,\mathsf{apl} \\
&\quad = \{\text{ decomposition of column }\} \\
&\qquad \mathsf{fst}\,\mathsf{apl} \,;\, \mathsf{col}([R^0, R] \,;\, S')
\end{aligned}
$$

To start the induction, the base case is that

$$
\begin{aligned}
\mathsf{fst}\, 0 \,;\, \mathsf{apr} \,/\, \mathsf{tri}\, R \,;\, \mathsf{col}\, S' \;&=\; \mathsf{apr} \,;\, 1 \,;\, \mathsf{tri}\, R \,;\, \mathsf{apr}^{-1} \,;\, \mathsf{col}\, S' \\
&=\; \mathsf{apr} \,;\, [R^0] \,;\, \mathsf{apr}^{-1} \,;\, \mathsf{col}\, S' \\
&=\; [0, R^0] \,;\, \mathsf{col}\, S' \\
&=\; \mathsf{fst}\, 0 \,;\, \mathsf{col}([R^0, R^0] \,;\, S') \\
&=\; \mathsf{fst}\, 0 \,;\, \mathsf{col}([R^0, R] \,;\, S')
\end{aligned}
$$

and so by induction on the length of the column the result is proved.

There is a statement of Horner's rule for left reduction: that if $[R, R] ; S = S ; R$ then $\mathsf{apl} \,/\, (\mathsf{rev} \,/\, \mathsf{tri}\, R) ; \mathsf{rdl}\, S = \mathsf{rdr}([R, R^0] ; S)$. Similarly there are statements for the inverses of both reductions. There are also a variety of similar results for row and column, and for grids. Their proofs by induction would be tedious and lengthy, but the whole point of this work is that such proofs once having been done, their consequences can be applied in the course of a design without the necessity of recourse to induction.

In fact there is really only one proof by induction; the skewing theorem which will turn up later in §7, and of which all these and some others are a consequence.

Chapter 5

Sequential circuits

So far we have intended to give the impression that all the circuits that we are talking about are combinational: that they behave as described in a steady state, in which they are presented with an unchanging input and deliver an unchanging output which depends only on that input. Of course no circuit is ever used in this way: at the very least there was a time before it was presented with any input; however the combinational description faithfully models the behaviour of a circuit that was presented with its input so long ago that it has settled down.

If an entire system is allowed to settle after each change in its input, the whole of the system is tied up for a settling time that depends on the longest propagation path, so it usual to place 'latches' – sometimes called 'registers' – at the outputs of the modules that make up a system. Each latch stores the output from its module in the previous steady state of that module, allowing that module to proceed with the next calculation while the modules that follow it are still settling into the previous steady state. In figure 5.1, suppose that the F is presented with a succession x_i of inputs. The output from the latch $\mathcal{D}$ at any time will be the value that was previously presented to it by F, so G can be completing the calculation of $G(F(x_i))$ while F is working on the calculation of $F(x_{i+1})$. When the circuit has settled the latch is 'clocked' – it is told to discard its current output and to propagate $F(x_{i+1})$ instead – by a signal not shown in the diagram. This technique of overlapping a succession of combinational calculations is called 'pipelining'. Since it has the effect of shortening the longest time to settle, it allows a greater number of calculations

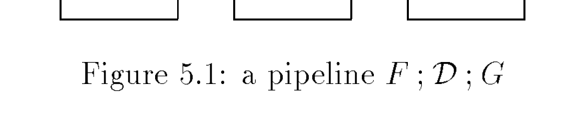

Figure 5.1: a pipeline $F \,;\mathcal{D}\,; G$

to be completed by a circuit in a given time. Notice, however, that there is no longer a single combinational description of what the whole circuit is doing.

It is usual to think of the latches in such a circuit as being the repository of the state of the circuit. The calculation being implemented by the circuit at any given time depends not only on the circuit and the inputs to it, but also on the values being held in latches. By means of these the calculation at any given time can be influenced by past calculations. In a pipeline like that in figure 5.1 the calculation, and so the output, depends only on a finite number of past inputs, in this case on only one; but in general there may be cycles in the circuit with the input to a latch depending on some function of its output. In that case the state held in the latch is an accumulation of information about the whole past history of calculations performed by the circuit, and the output can depend on an arbitrary large number of past inputs. There is no way of giving a convenient and convincing account of such circuits in terms of a succession of independent combinational calculations.

The process of designing a sequential circuit appears to be significantly harder than that of designing a combinational one: even in the simple case of a pipelining there is an additional concern, that of making sure that the inputs that arrive together at a component are those for the same calculation; and more generally it is not possible to separate the concern of getting the time right from that of getting the value right. Fortunately, there is a large range of circuits for which it is possible to reason that a design which would work as a combinational circuit would work more or less the same way as a sequential circuit. The purpose of this chapter is to formalize that statement.

5.1 Time sequences

The sequential circuits with which we are concerned are those about which it is sound to reason in exactly the same way as those in previous chapters. What that means is that the algebra is going to be the same as it was in previous chapters, but the interpretation of the symbols will have to be different: the variables like R, S and so on will represent sequential circuits; and the combining forms will be correspondingly different. This is going to make another model of the algebra, and one that is radically different from the more-or-less similar functional and relational ones that we have used so far.

Think of a circuit as relating not values, but time-sequences of values. A time sequence is just a function from an index which represents the time at which a signal is observed to the value that the signal has at that time. Suppose for the moment that we use the natural numbers to represent successive times at which a signal is changed: a signal representing an integer, one of type int, is a function $\mathbb{N} \to \mathbb{Z}$, and one representing a pair of integers, one of type $[int, int]$, is a function

$\mathbb{N} \to (\mathbb{Z} \times \mathbb{Z})$, that is a pair-valued function, and *not* a pair of functions. So for example to say that $(\times 2) : \mathit{int} \sim \mathit{int}$ is to say that in this model it represents a relation between a function of type $\mathbb{N} \to \mathbb{Z}$ and another of the same type, and to say that $\mathit{add} : \mathit{int} \sim [\mathit{int}, \mathit{int}]$ is to say that it represents a relation between a function of type $\mathbb{N} \to \mathbb{Z}$ and one of type $\mathbb{N} \to (\mathbb{Z} \times \mathbb{Z})$.

Which relation should $(\times 2)$ be in this model? It should relate one integer to another which is twice as large, and it should do this at each time; that is to say

$$x\,(\times 2)\,y \iff \forall i.\, x_i \times 2 = y_i$$

and similarly

$$x\ \mathit{add}\ y \iff \forall i.\, \exists p, q.\, x_i = \langle p, q\rangle \ \&\ p + q = y_i$$

Now $x_i \times 2 = y_i$ is what was meant by $x_i\,(\times 2)\,y_i$ in the old model; and similarly $x_i\ \mathit{add}\ y_i$ means in the old model that $\exists p, q.\, x_i = \langle p, q\rangle \ \&\ p + q = y_i$.

Let $\mathcal{S}[\![R]\!]$ be the relation that R represents in the sequential model, and $\mathcal{C}[\![R]\!]$ be the relation that it represented in the combinational model, then

$$\begin{array}{rcl} x\,\mathcal{S}[\![\times 2]\!]\,y & \iff & \forall i.\, x_i\,\mathcal{C}[\![\times 2]\!]\,y_i \\ x\,\mathcal{S}[\![\mathit{add}]\!]\,y & \iff & \forall i.\, x_i\,\mathcal{C}[\![\mathit{add}]\!]\,y_i \end{array}$$

This is a pleasantly simple pattern: it says that a sequential *add* does at each instant what a combinational *add* would have done. That is clearly what we want of any circuit R which has no internal state, that $x\,\mathcal{S}[\![R]\!]\,y \iff \forall i.\, x_i\,\mathcal{C}[\![R]\!]\,y_i$, and we say that any R for which this is true is *stateless*.

The time sequences can also be thought of as being infinite lists. Thinking of x and y as lists, $\forall i.\, x_i\,\mathcal{C}[\![R]\!]\,y_i$ is what we would previously have written as $x\,\mathsf{map}(\mathcal{C}[\![R]\!])\,y$, so a stateless circuit is an R for which $\mathcal{S}[\![R]\!] = \mathsf{map}(\mathcal{C}[\![R]\!])$. Beware however that everywhere else in these notes, $\mathsf{map}\,X$ relates *finite* lists of the same length, and the time sequences that we are dealing with may be infinite. The simplicity of $\mathcal{S}[\![R]\!] = \mathsf{map}(\mathcal{C}[\![R]\!])$ is directly attributable to the decision that a circuit with several inputs, which if it were combinational would have a single list as input, shall as a sequential circuit have a single time-sequence of lists as input, rather than a list of time-sequences.

In fact the suggestion so far is that time should be modelled by $\mathbb{N}$, which means that there is an infinitely long future, but there is a distinguished 'first' instant. In some applications you might want to consider modelling time by some finite interval of $\mathbb{N}$, so that there was also a last instant. Sequences over finite intervals of $\mathbb{N}$ are of course just finite lists. More importantly, in these notes we will eventually concentrate on modelling time by the whole set of integers, $\mathbb{Z}$, so that there is an arbitrarily long past as well as an arbitrarily long future. You can also think of sequences over $\mathbb{Z}$ as being lists, but they have to be doubly infinite ones, something like $\langle \ldots, x_{-2}, x_{-1}, x_0, x_1, x_2, \ldots \rangle$. Notice that although they have no beginning and no end you can still identify the i-th element of one of them, and in particular you can align them so that the noughth elements (the ones representing the 'current' values of the signals) correspond

$$\begin{array}{ccc} \langle \ldots, x_{-2}, x_{-1}, & x_0, & x_1, x_2, \ldots \rangle \\ \langle \ldots, y_{-2}, y_{-1}, & y_0, & y_1, y_2, \ldots \rangle \end{array}$$

and that this alignment is unique.

5.2 Composition, parallel composition and so on

Having decided what the interpretation of some circuits will be in the sequential model, what should the interpretation of composition be? At least for stateless circuits it is already determined, for if R and S are stateless we certainly want $R\,;S$ to be stateless, so

$$\begin{array}{rcl} \mathcal{S}[\![R\,;S]\!] & = & \mathsf{map}\,\mathcal{C}[\![R\,;S]\!] \\ & = & \mathsf{map}(\mathcal{C}[\![R]\!]\,;\mathcal{C}[\![S]\!]) \\ & = & \mathsf{map}\,\mathcal{C}[\![R]\!]\,;\mathsf{map}\,\mathcal{C}[\![S]\!] \\ & = & \mathcal{S}[\![R]\!]\,;\mathcal{S}[\![S]\!] \end{array}$$

and in general $\mathcal{S}[\![R\,;S]\!]$ is defined to be $\mathcal{S}[\![R]\!]\,;\mathcal{S}[\![S]\!]$. (The ';' outside the brackets is of course relational composition. Had we been this careful to distinguish the notation from its interpretation in §2, we could have introduced the interpretation of composition in the combinational model by defining $\mathcal{C}[\![R\,;S]\!] = \mathcal{C}[\![R]\!]\,;\mathcal{C}[\![S]\!]$ in the same way.) Similarly, the inverse of a sequential circuit is interpreted as the relational converse of its interpretation: $\mathcal{S}[\![R^{-1}]\!] = \mathcal{S}[\![R]\!]^{-1}$, and so also for repeated composition.

In order to talk about the interpretation of parallel composition, we need a new constant, zip, which relates a pair of sequences to the obvious sequence of pairs to which it corresponds:

$$\langle x, y \rangle \;\mathsf{zip}\; z \iff \forall i.\, \langle x_i, y_i \rangle = z_i$$

It satisfies $[\mathsf{map}\,R, \mathsf{map}\,S]\,;\mathsf{zip} = \mathsf{zip}\,;\mathsf{map}[R, S]$ as you can show by calculating from the definitions. That means that the argument that if R and S are stateless then so should be $[R, S]$ leads to the conclusion that at least for these circuits

$$\begin{array}{rcl} \mathcal{S}[\![[R, S]]\!] & = & \mathsf{map}\,\mathcal{C}[\![[R, S]]\!] \\ & = & \mathsf{map}[\mathcal{C}[\![R]\!], \mathcal{C}[\![S]\!]] \\ & = & \mathsf{zip}^{-1} \,/\, [\mathsf{map}\,\mathcal{C}[\![R]\!], \mathsf{map}\,\mathcal{C}[\![S]\!]] \\ & = & \mathsf{zip}^{-1} \,/\, [\mathcal{S}[\![R]\!], \mathcal{S}[\![S]\!]] \end{array}$$

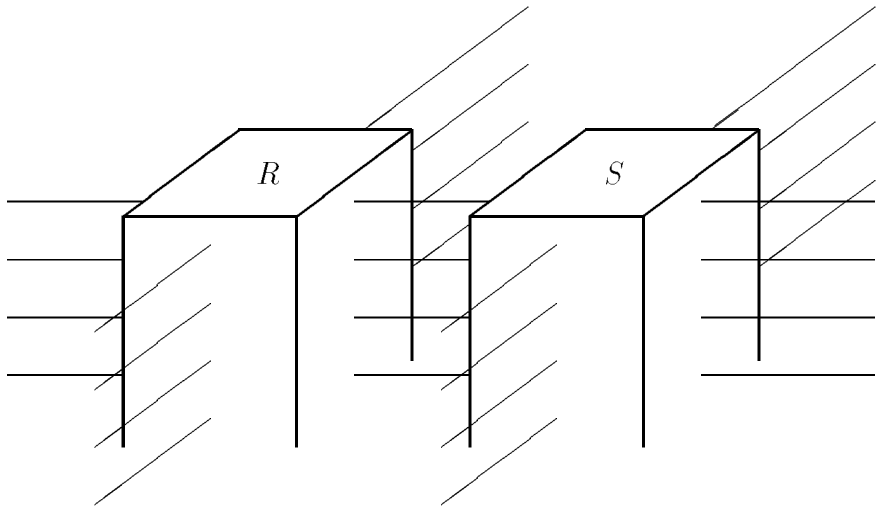

Figure 5.2: a sequential interpretation of $R \leftrightarrow S$

so define $\mathcal{S}[\![[R,S]]\!] = \mathsf{zip}^{-1} \,/\, [\mathcal{S}[\![R]\!], \mathcal{S}[\![S]\!]]$ for general R and S. The marginal extra complexity of this definition over that of composition is the price to be paid for the choice of time-sequences of pairs in preference to pairs of time-sequences.

The interpretation of map follows from the properties in §3.4 which we require be maintained. In fact $\mathcal{S}[\![\mathsf{map}\, R]\!] = \mathsf{trn}^{-1} \,/\, \mathsf{map}\, \mathcal{S}[\![R]\!]$ where trn, read 'transpose', is the relation for which $x \,\mathsf{trn}\, y$ if and only if $\forall i, j.\, x_{i,j} = y_{j,i}$. In particular $\mathsf{zip} = 2 \,;\, \mathsf{trn} = \mathsf{trn} \,;\, \mathsf{map}\, 2$ from which can be seen that this interpretation of $\mathcal{S}[\![\mathsf{map}\, R]\!]$ is consistent with that for $\mathcal{S}[\![[R,S]]\!]$.

(You could argue that since $\mathsf{trn} = \mathsf{trn}^{-1}$, there is no reason to use trn^{-1} in the interpretation of map, but in these notes wherever there is a possibility that the list being transposed is infinite, think when we speak of trn of relating a finite list of potentially infinite lists to a potentially infinite list of finite lists. There will not arise a need to transpose an infinite list of infinite lists. It is just a small hint to the reader, and ensures that the use of trn and trn^{-1} is consistent with the use of zip and zip^{-1}.)

The interpretation of triangle follows immediately from those of map and composition, and $\mathcal{S}[\![\mathsf{tri}\, R]\!] = \mathsf{trn}^{-1} \,/\, \mathsf{tri}\, \mathcal{S}[\![R]\!]$. Similarly, those of the other combining forms, for example

$$\begin{aligned}
\mathcal{S}[\![R \leftrightarrow S]\!] &= \mathcal{S}[\![rsh \,;\, \mathsf{fst}\, R \,;\, lsh \,;\, \mathsf{snd}\, S \,;\, rsh]\!] \\
&= \mathsf{map}\, rsh \,;\, \mathcal{S}[\![\mathsf{fst}\, R]\!] \,;\, \mathsf{map}\, lsh \,;\, \mathcal{S}[\![\mathsf{snd}\, S]\!] \,;\, \mathsf{map}\, rsh \\
&= \mathsf{map}\, rsh \,;\, \mathsf{zip}^{-1} \,/\, \mathsf{fst}\, \mathcal{S}[\![R]\!] \,;\, \mathsf{map}\, lsh \,;\, \mathsf{zip}^{-1} \,/\, \mathsf{snd}\, \mathcal{S}[\![S]\!] \,;\, \mathsf{map}\, rsh
\end{aligned}$$

The effect of this is to be seen in figure 5.2 in which the time axis is downwards, and the plan view of which would look exactly like the picture of $R \leftrightarrow S$ in figure 4.2.

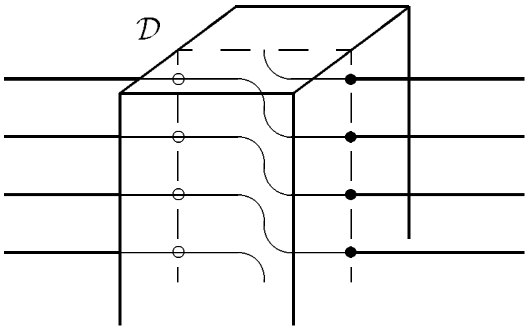

Figure 5.3: an interpretation of $\mathcal{D}$

5.3 Delay

So far, all the sequential circuits we can talk about have been stateless. In order to talk about circuits with state it will be enough for the present to introduce one new primitive: a *delay*, written $\mathcal{D}$. Its meaning in the sequential model is the relation

$$x\,\mathcal{D}\,y \iff \forall i.\, x_i = y_{i+i}$$

which says that delay relates two signals if the range signal is at all times the same as the domain signal had been exactly one time step earlier.

The inverse of a delay, an *anti-delay* written of course $\mathcal{D}^{-1}$, is also a useful concept. It represents the relation $x\,\mathcal{D}^{-1}\,y \iff \forall i.\, x_i = y_{i-1}$ which says that it relates two signals if the range signal is at all times the same as the domain signal would be exactly one time step later.

If these relations are to be implemented by circuits taking inputs from their domains and returning outputs in their ranges, then $\mathcal{D}$ is just a latch: the output after each clock is the same as the input immediately preceding it. The corresponding interpretation of $\mathcal{D}^{-1}$ as a circuit is altogether less implementable: it would have to predict *before* each clock what its input was going to be *after* that clock.

Conversely, if the relations are to be implemented by circuits taking inputs from their ranges and returning outputs in their domains, $\mathcal{D}^{-1}$ is a latch, and it is $\mathcal{D}$ that becomes unimplementable. What this means is that if an expression including $\mathcal{D}$ or $\mathcal{D}^{-1}$ were to be implemented by translating it into a circuit there would be these additional constraints on the choice of direction for each of the signal flows, that all $\mathcal{D}$ and $\mathcal{D}^{-1}$ in the expression be implemented the right way around.

So long as the time-sequences in the model are functions from all of the integers, positive and negative, so that there is no first time and no last time, you can show that $\mathcal{D}\,;\mathcal{D}^{-1} = \iota = \mathcal{D}^{-1}\,;\mathcal{D}$. There is a perfect symmetry between $\mathcal{D}$ and $\mathcal{D}^{-1}$ as presented here: you should not think of either of them as being more realistic, or more implementable than the other. Indeed, one of them is just the other one seen the other way around.

The statement that $\mathcal{D} ; \mathcal{D}^{-1} = \iota$ bears careful attention: the ι on the right hand side is of course the identity of composition, but it is the identity of composition in the sequential model. Specifically, this equation says that $\mathcal{S}[\![\mathcal{D} ; \mathcal{D}^{-1}]\!] = \mathcal{S}[\![\iota]\!] = \mathsf{map}\,\iota$, but then every value is a time-sequence in the sequential model, so there is nothing in that model that is any more like the identity of composition than $\mathsf{map}\,\iota$.

Even though $\mathcal{D}$ makes no sense in the combinational model, reasoning about circuits that contain $\mathcal{D}$ can be done in exactly the same way as about combinational circuits. Any calculation with a variable R in it, a calculation that does not depend on the particular properties of R, can be instantiated by putting $\mathcal{D}$ for R. For example Horner's rule that if $[R, R] ; S = S ; R$ then $(\mathsf{apr} / \mathsf{tri}\, R) ; \mathsf{rdr}\, S = \mathsf{rdr}(\mathsf{tri}\, R ; S)$ is a property of the combining forms like composition and reduce right, and not of R or S. Consequently it is the case that if $[\mathcal{D}, \mathcal{D}] ; S = S ; \mathcal{D}$ then $(\mathsf{apr} / \mathsf{tri}\, \mathcal{D}) ; \mathsf{rdr}\, S = \mathsf{rdr}(\mathsf{tri}\, \mathcal{D} ; S) = \mathsf{rdr}(\mathsf{snd}\, \mathcal{D} ; S)$.

A circuit like $\mathsf{rdr}(\mathsf{snd}\, \mathcal{D} ; S)$ is said to be pipelined, because there is a latch between each of the stages of the reduction. You can read this instance of Horner's rule as follows: to implement a $\mathsf{rdr}\, S$ that is pipelined, implement $\mathsf{rdr}(\mathsf{snd}\, \mathcal{D} ; S)$, and adjust the timing of the domain signals according to $(\mathsf{apr} / \mathsf{tri}\, \mathcal{D})$. If the circuit takes input from the domain, that adjustment consists of supplying each component of the input one time step earlier than the one to its left. Such a specification is called a *data skew*, and an expression like $(\mathsf{apr} / \mathsf{tri}\, \mathcal{D})$ can either represent a skewing circuit which is implemented and performs the required adjustment, or else represent a specification for the interface to the circuit which is implemented.

Of course there are things that are true about $\mathcal{D}$ which do depend on its being a particular circuit, for example that $\mathsf{tri}\, \mathcal{D} ; \mathsf{tri}\, \mathcal{D}^{-1} = \mathsf{map}\,\iota$, but you should always keep in mind that it is also a perfectly normal component of the kind that we have been reasoning about all along. It is just this simplicity that makes the approach presented here so useful.

One of the remarkable properties of $\mathcal{D}$ is quite how polymorphic it is: $\mathcal{D}$ can be used to represent a unit delay applied to signals of any type, and the tightest type-specification that can be given is that $\mathcal{D} : \iota \sim \iota$. Moreover $n \,/\, \mathcal{D} = n \,/\, \mathsf{map}\, \mathcal{D}$, so that for example $\mathsf{apl} ; \mathcal{D} = \mathsf{apl} ; \mathsf{map}\, \mathcal{D} = [\mathcal{D}, \mathsf{map}\, \mathcal{D}] ; \mathsf{apl}$. This means that even though we may be talking about circuit components that will be realised as latches we do not *need* to be concerned with exactly what type of signals they handle, just with the delaying properties. The effect is a clean separation of two concerns: that of making sure that the correct values are handled, and that of ensuring that they arrive at the right times.

It may help to associate delays and anti-delays with latches, but it is not necessary to implement a sequential circuit with latches. Any mechanism will do that ensures that corresponding items from time-sequences are brought together. For example the synchronization might be implemented by local handshaking, even though this means that events 'on the same clock' need not happen at the same time if they are not at the same place in the circuit.

5.4 Timelessness

For any stateless circuit R you can show that $R\,;\mathcal{D} = \mathcal{D}\,; R$, for $\mathcal{S}[\![R\,;\mathcal{D}]\!] = \mathsf{map}\,R\,;\mathcal{S}[\![\mathcal{D}]\!]$ relates x and y when $\forall i.\, x_i\,R\,y_{i+1}$, as does $\mathcal{S}[\![\mathcal{D}]\!]\,;\mathsf{map}\,R = \mathcal{S}[\![\mathcal{D}\,; R]\!]$.

In these notes any circuit R for which $R;\mathcal{D} = \mathcal{D};R$ is said to be *timeless*. Since $\mathcal{D}\,;\mathcal{D}^{-1} = \iota$ another way of giving the same condition is that $R = \mathcal{D}\,/\,R$. What this says is that a timeless circuit is one that implements the same relation now as it did in the past, and similarly that it will in the future. Notice that timelessness is *not* the same as statelessness: $\mathcal{D}$ is most definitely not stateless, indeed it the essence of statefulness! Nevertheless since $\mathcal{D} = \mathcal{D}\,;\mathcal{D}\,;\mathcal{D}^{-1} = \mathcal{D}\,/\,\mathcal{D}$ it is immediate from the definition that $\mathcal{D}$ is timeless.

Just as a stateless circuit is one that does not behave differently at any time according to the history of its inputs, so a timeless circuit is one that does not behave differently on account of the absolute time. A stateless circuit is necessarily timeless. An example of a timeful circuit would be one that generated a signal indicating time zero, say Z where $x\,Z\,y$ if and only if $y_0 = \mathsf{T}$ and $y_i = \mathsf{F}$ otherwise. Since this circuit 'knows' the exact time $\mathcal{D}^n\,/\,Z \neq Z$ for each $n \neq 0$. A different sort of example would be E for which $x\,E\,y$ if and only if $\forall i.\, x_{2i} = y_{2i}$. This circuit is not anchored at a single instant because $\mathcal{D}^2\,/\,E = E$, but nevertheless E is not timeless since $\mathcal{D}\,/\,E \neq E$.

One of the reasons that timelessness is interesting is that exactly like statelessness it is preserved by all of the circuit constructors that we have seen so far. That is to say that, for example if R and S are timeless then so is $R;S = (\mathcal{D}/R);(\mathcal{D}/S) = \mathcal{D}\,/\,(R;S)$, and so is $\mathsf{map}\,R = \mathsf{map}(\mathcal{D}\,/\,R) = (\mathsf{map}\,\mathcal{D})\,/\,(\mathsf{map}\,R) = \mathcal{D}\,/\,\mathsf{map}\,R$. What this means is that if all of the primitive circuits in an expression are timeless, then so is the circuit which the expression represents.

Knowing that a circuit is timeless is often going to be useful, because the precondition of the instance of Horner's rule applied to pipelining right reductions, that $S = [\mathcal{D},\mathcal{D}]\,; S\,;\mathcal{D} = \mathcal{D}\,/\,(2\,; S)$ is just that $S : 2 \sim \iota$ is timeless. Since the type $2 \sim \iota$ is implied by the use of S in the consequence, this instance of the rule can simply be stated as: if S is timeless then $(\mathsf{apr}\,/\,\mathsf{tri}\,\mathcal{D})\,;\mathsf{rdr}\,S = \mathsf{rdr}(\mathsf{snd}\,\mathcal{D}\,; S)$.

5.5 Lifting and oblivious operations

The transition from a model based on values to a model based on time-sequences, from combinational circuits to sequential circuits, from forms that combine combinational circuits to forms that combine sequential ones, and doing each of these in a consistent way, is called *lifting*. So for example if you lift an integer you get a constant integer-valued function; if you lift the relation *add* from pairs of integers to integers you get $\mathsf{map}\, add$ which is a relation between pair-of-integer valued functions and integer-valued functions; and if you lift an operation like composition of relations you get composition of relations-between-time-sequences.

Lifting is interesting because it preserves the truth of assertions: lifting both sides of an equation preserves the equality; lifting an expression and its type preserves the truth of the assertion that the expression has that type. In fact lifting is often done silently: for example although one of De Morgan's laws says that $(\neg p) \vee (\neg q) = \neg(p \wedge q)$ for any truth-values p and q, one would normally not be formal about lifting the operators in asserting that $(\neg P) \vee (\neg Q) = \neg(P \wedge Q)$ for predicates P and Q. Strictly speaking one should say that $(\neg P(x)) \vee (\neg Q(x)) = \neg(P(x) \wedge Q(x))$.

Since the interpretation of ';' in the sequential model is indeed the lifting of its interpretation in the combinational model, it follows immediately that anything we know about an expression in which the only constants are stateless circuits and ';', we also know to be true in the sequential model. Unfortunately this is almost as far as lifting can get us on its own, because for example $\mathcal{S}[\![[R, S]]\!]$ is not obtained by lifting parallel composition to time sequences. The lifting of parallel composition would give $[\mathcal{S}[\![R]\!], \mathcal{S}[\![S]\!]]$ which relates two pairs of time-sequences. There is, however, a systematic relationship between parallel composition in the two models, and between the other polymorphic combining forms in the two models.

The intuition is much easier than the formalism: essentially if some operation is to be polymorphic that limits the extent to which it can analyse the signals that it handles. If parallel composition is to take an $R : A \sim B$ and a $S : C \sim D$ and return a $[A, C] \sim [B, D]$ for *all* types A, B, C and D, then the only thing that the sequential interpretation of parallel composition is able to do is to take apart the sequences of pairs into pairs of sequences and pass them to $[\mathcal{S}[\![R]\!], \mathcal{S}[\![S]\!]]$. So we defined $\mathcal{S}[\![[R, S]]\!] = \mathsf{zip}^{-1} \,;\, [\mathcal{S}[\![R]\!], \mathcal{S}[\![S]\!]] \,;\, \mathsf{zip}$. The zip^{-1} is determined by the domain type being a pair, and the zip by the range type being a pair. The zip is the natural way of translating a pair of time-sequences into a sequence of pairs.

Similarly, trn is the translation of a list of time-sequences into a time-sequence of lists, so we defined $\mathcal{S}[\![\mathsf{map}\, R]\!] = \mathsf{trn}^{-1} \,;\, \mathsf{map}\, \mathcal{S}[\![R]\!] \,;\, \mathsf{trn}$ and $\mathcal{S}[\![\mathsf{tri}\, R]\!] = \mathsf{trn}^{-1} \,;\, \mathsf{tri}\, \mathcal{S}[\![R]\!] \,;\, \mathsf{trn}$.

Again to take an example with no parameter, $\mathsf{apl} : \mathsf{snd}\, \mathsf{map}\, \iota \sim \mathsf{map}\, \iota$ relates a pair the second component of which is a list to a list. The translation for a pair the second component of which is a list is $\mathsf{snd}\, \mathsf{trn} \,;\, \mathsf{zip}$, and that for a list is trn, so we should define $\mathcal{S}[\![\mathsf{apl}]\!] = (\mathsf{snd}\, \mathsf{trn} \,;\, \mathsf{zip})^{-1} \,;\, \mathsf{apl} \,;\, \mathsf{trn}$ and indeed this does turn out to be equal to $\mathsf{map}\, \mathsf{apl}$ which was the definition given earlier.

The translation for any type is going to be a bijection: in the terms of the earlier discussion, zip represents pairs of sequences as sequences of pairs, and zip^{-1} represents sequences of pairs as pairs of sequences. The translation for a tuple type is given by $T_{[A,B]} = [T_A, T_B]\,;\mathsf{zip}$ and similarly for larger tuples, and more generally for a list type by $T_{\mathsf{map}\,A} = \mathsf{map}\,T_A\,;\mathsf{trn}$, and the translation for a single signal is $T_\iota = \iota$.

An *oblivious* circuit $R : A \sim B$ is one for which $\mathcal{S}[\![R]\!] = T_A^{-1};\mathcal{C}[\![R]\!];T_B$. (Strictly speaking that is a property of A and B as well as of R.) An oblivious combining form $f : (A\sim B)\rightarrow(C\sim D)$ is one for which $\mathcal{S}[\![f(R)]\!] = T_C^{-1};\mathcal{C}[\![f]\!](T_A;\mathcal{S}[\![R]\!];T_B^{-1});T_D$, and similarly for forms with more arguments.

Well-typed applications of oblivious combining forms to oblivious circuits yield oblivious circuits, since the T are each bijective, so an expression in which the constants are all oblivious defines an oblivious form in its free variables. Moreover since the types of each side of an equation are going to be the same, if an equation holds in the combinational model and contains only oblivious constants, it holds in the sequential model.

All the polymorphic circuits and combining forms that have appeared in these notes so far are oblivious. Oblivious circuits are necessarily stateless, and so timeless; oblivious forms preserve statelessness and so timelessness. Oblivious circuits and forms are necessarily polymorphic: it does not make sense to ask of a circuit like $add : [int, int] \sim int$ whether it is oblivious, because $\mathcal{C}[\![add]\!]$ is not a relation between a pair of time-sequences and a time-sequence, so $\mathsf{zip}^{-1}\,;\mathcal{C}[\![add]\!]$ is just ill-typed.

5.6 Counter-examples

Much of this chapter has asserted that something which is right in the combinational model carries over into the sequential model. In fact the notation has been carefully chosen so that reasoning in the simpler model has a natural parallel reasoning in the more complicated one. There is a danger in such circumstances of the intuition going too far, and of one's assuming that anything that works in the one world works in the other. The proper preventive for this affliction is a simple counter-example.

Consider the conditional relation, defined by

$$x\,(Q \nleftarrow C \nrightarrow R)\,y \iff (x\,C\,\mathsf{T}\ \&\ x\,Q\,y) \vee (x\,C\,\mathsf{F}\ \&\ x\,R\,y)$$

which you can read as 'Q if C and otherwise R'. Think of a circuit something like that in figure 5.4 in which you are meant to imagine that C determines position of a switch, according to the value of the domain signal; and that the switch selects between the range signals of Q and R when given the same domain signal. This is a lifting of the everyday if-then-else, written in Hoare's infix notation, and you

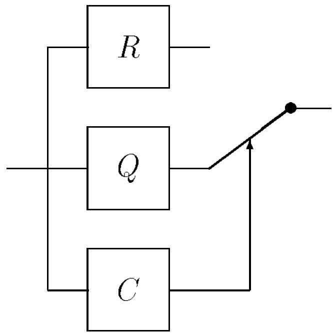

Figure 5.4: $Q \nless C \ngtr R$

might expect the following to be true:

$$\begin{array}{rcl} P\,;(Q \nless C \ngtr R) & = & (P\,;Q) \nless (P\,;C) \ngtr (R\,;S) \\ (Q \nless C \ngtr R)\,;S & = & (Q\,;S) \nless C \ngtr (R\,;S) \end{array}$$

Lifting guarantees that these are true in case C, P, Q, R and S are functions from domain to range, and the conditional is given its combinational interpretation.

Beyond that, all sorts of things go wrong. Neither equation is true in general for relations, because they depend on making copies of a circuit and getting the same output from it. Consider the relation fork defined by $x\,\mathsf{fork}\,\langle x, x\rangle$. In general $R;\mathsf{fork} \neq \mathsf{fork};[R, R]$ because if $x\,R\,y$ and $x\,R\,z$ with $y \neq z$, then $x(\mathsf{fork}\,;[R, R])\langle y, z\rangle$ whereas all pairs in the range of $R\,;\mathsf{fork}$ contain two equal values.

Something that is true is that if $R^{-1}\,;R = \iota$ then $R\,;\mathsf{fork} = \mathsf{fork}\,;[R, R]$, and similarly $\mathsf{fork};\mathsf{fst}\,R^{-1} = R^{-1};\mathsf{fork};\mathsf{snd}\,R$ and $\mathsf{fork};\mathsf{snd}\,R^{-1} = R^{-1};\mathsf{fork};\mathsf{fst}\,R$. More generally, if $R^{-1};R = T$ is a type, and $R;T = R$ then $R;\mathsf{fork};[T, T] = \mathsf{fork};[R, R]$. More symmetrically, and as a consequence of that last observation, if additionally $R : U \sim T$ then $R\,;(T\,;\mathsf{fork}\,;[T, T]) = (U\,;\mathsf{fork}\,;[U, U])\,;[R, R]$. The same sorts of considerations apply to $\mathsf{join} = \mathsf{fork}^{-1}$ as to fork.

Suppose we call an R *deterministic* if $R^{-1}\,;R = \iota$, then applying these observations to conditional gives that if P is deterministic then $P\,;(Q \nless C \ngtr R) = (P;Q) \nless (P;C) \ngtr (R;S)$ and if either C or S^{-1} is deterministic then $(Q \nless C \ngtr R);S = (Q\,;S) \nless C \ngtr (R\,;S)$. At least, that is to say that these equations are true for the combinational interpretation of conditional.

What should a sequential conditional do? It seems natural to require that the switch be set at each instant according to the range value of C at that time, so that the range sequence of $Q \nless C \ngtr R$ should be an interleaving of some values from the range sequence of Q and some from that of R.

$$x\,(Q \nless C \ngtr R)\,y \iff \forall i.\,(x_i\,C\,\mathsf{T}\ \&\ x_i\,Q\,y_i) \vee (x_i\,C\,\mathsf{F}\ \&\ x_i\,R\,y_i)$$

With this interpretation it remains the case that if P is deterministic $P\,;(Q \nleftarrow C \nrightarrow R) = (P;Q) \nleftarrow (P;C) \nrightarrow (R;S)$, but just requiring that components be deterministic is no help with $(Q \nleftarrow C \nrightarrow R)\,;S \neq (Q\,;S) \nleftarrow C \nrightarrow (R\,;S)$.

The problem this time is that if S can have internal state, the two instances of it in $Q\,;S$ and $R;S$ will have different domain histories, and can accumulate different states. Moreover these states may each be different from that of the S on the left-hand side. Suppose Q and R are (different) constant signal generators, which is to say that $Q\,;\mathcal{D} = Q$ and $R;\mathcal{D} = R$, and Z is the circuit mentioned earlier which generates a single T in the range signal at time zero. Then $(Q \nleftarrow Z \nrightarrow R)\,;\mathcal{D}$ has a range signal which is different at time one, but $(Q;\mathcal{D}) \nleftarrow Z \nrightarrow (R;\mathcal{D}) = Q \nleftarrow Z \nrightarrow R$ has a range signal which is different at time zero, so $(Q \nleftarrow Z \nrightarrow R);\mathcal{D} \neq (Q;\mathcal{D}) \nleftarrow Z \nrightarrow (R;\mathcal{D})$.

To show the equality you need both that either C or S^{-1} be deterministic, and also that S be stateless, and in that case $(Q \nleftarrow C \nrightarrow R)\,;S = (Q\,;S) \nleftarrow C \nrightarrow (R\,;S)$.

The complication of reasoning about conditional shows just what the benefits are of doing as much of the reasoning about sequential circuits as can be, without having to worry about time.

5.7 Slowing

So far we have ways of talking only about circuits in which all of the sequential components operate at the same rate. Sometimes it proves necessary to reason about circuits in which some components operate at different rates from others. In particular there are various common techniques which require some parts of a circuit to be clocked at twice the rate of other parts, or some such small multiple.

In order to talk about such circuits, we use a new primitive relation *pair* defined by $x \; pair \; y \iff \forall i.\, y_i = \langle x_{2i}, x_{2i+1} \rangle$. Define a slow version of R by $\mathsf{slow}\, R = pair \,/\, \mathsf{map}\, R$. (Strictly speaking, define $\mathcal{S}[\![\mathsf{slow}\, R]\!] = pair \,/\, \mathcal{S}[\![\mathsf{map}\, R]\!]$, but *pair* is stateless and conjugation is oblivious so it would be the same thing as defining $\mathcal{S}[\![\mathsf{slow}\, R]\!] = \mathcal{S}[\![pair/\mathsf{map}\, R]\!]$. This is the reason we are generally sloppy about such details.) The effect is that $\mathsf{slow}\, R$ separates the time-sequences in its domain and range each into two interleaved sequences, and performs the calculation described by R on each of the pairs of corresponding subsequences.

It is a matter of proof that if R is an expression involving only the oblivious combining forms and $\mathcal{D}$ and stateless components, then $\mathsf{slow}\, R$ is the same as the circuit obtained by replacing all occurrences of $\mathcal{D}$ in R by $\mathcal{D}^2$. For example $\mathsf{slow}\, \mathcal{D} = \mathcal{D}^2$, and $\mathsf{slow}(F\,;\mathcal{D}\,;G\,;\mathcal{D}) = F\,;\mathcal{D}^2\,;G\,;\mathcal{D}^2$. The proof is by structural induction on R.

This says that you can implement $\mathsf{slow}\, R$ either by following the recipe in the definition – making two copies of R and supplying them with alternate inputs to derive alternate outputs – or by doubling up each latch in R – effectively sharing

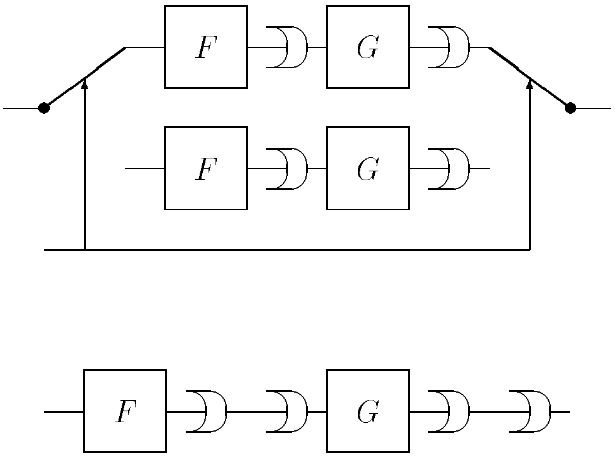

Figure 5.5: $\mathsf{slow}(F\,;\mathcal{D}\,;G\,;\mathcal{D})$ and $F\,;\mathcal{D}^2\,;G\,;\mathcal{D}^2$

the rest of the hardware in R between two calculations, with the state of each calculation being held in the extra latch while the other calculation is being done.

You may be concerned that $\mathsf{slow}\,R$ appears to be wrongly named, because it describes a piece of hardware that gets clocked twice as often as the circuit which surrounds it. On the other hand it is a circuit which takes twice as many clocks to produce the same result as would R. There are arguments both for and against this choice, and there seems no convincing reason for choosing to call this 'slow R' rather than 'fast R', beyond precedent which is followed here.

5.8 Cut-set retiming

Retiming is a procedure for disposing latches through a circuit, with a view to improving properties like the length of the longest unbroken propagation path. The name and the formalisation are usually attributed to Leiserson [Leise81] although the technique is well-known and presentations of it are to be found throughout the literature. They often look something like that in S. Y. Kung's tome [Kung88] which gives two rules.

- All delays may be scaled – replaced by multiple delays – provided the input and output rates are scaled accordingly.

- Given any cut-set – a set of wires in the circuit which if cut would divide it into two disjoint components – the edges of the cut-set can be divided into inbound- and outbound-edges according to which way the signal passes between the two components of the circuit. A latch may be moved from one component to the other along each inbound edge, provided a latch is

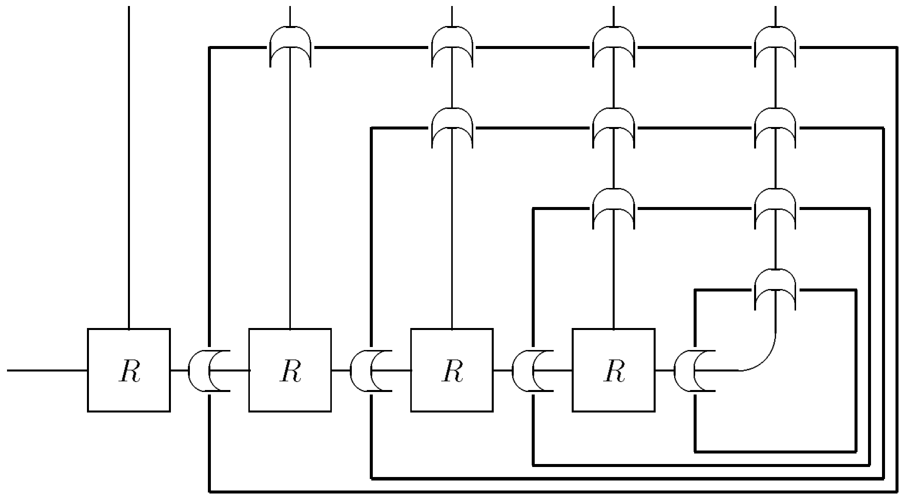

Figure 5.6: contours illustrating $(\mathsf{apr}\ /\ \mathsf{tri}\,\mathcal{D}^{-1})\ ;\ \mathsf{rdr}(\mathsf{snd}\,\mathcal{D}\ ;\ R)$

> also moved in the other direction along each outbound edge. This leaves the behaviour of the system 'generally unchanged', although the timing of input and output must be adjusted if there are inputs and outputs on both sides of the cut.

The quotation is not exact, but I have tried to be fair to the style of Kung's presentation.

We can look on the first rule in another way as saying that $\mathsf{slow}\ R(\mathcal{D}) = R(\mathcal{D}^2)$. The substitution of $\mathcal{D}^2$ for $\mathcal{D}$ is what is meant by the scaling of delays, and the slow describes precisely what is meant by scaling the input and output rates accordingly.

The second rule is also something that we know. In order to avoid the side-condition about adjusting the timing of the input and output, suppose we stipulate that all input and output must be one one side of the cut. This amounts to saying that the cut must be a closed contour, and the input and output must be outside it. Then if the component of the circuit within the contour is R, moving a latch across the contour along each edge in the cut-set amounts to replacing R by $\mathcal{D}\ /\ R$ or by $\mathcal{D}^{-1}\ /\ R$. The circuit will be unchanged if R is timeless. The relative simplicity of this formulation of the rule derives in part from our being happy to have both delays and anti-delays in a description at the same time.

Figure 5.6 shows a family of contours drawn on a reduction from the right, illustrating the route from $\mathsf{rdr}\,R$ to the equivalent $(\mathsf{apr}\ /\ \mathsf{tri}\,\mathcal{D}^{-1})\ ;\ \mathsf{rdr}(\mathsf{snd}\,\mathcal{D}\ ;\ R)$ which is a familiar instance of Horner's rule. There is no reason for the contours not to overlap: it just happens that they do in this figure. Neither is it necessary for them to form a regular family the way those in this figure do, because each contour is an independent step in the process of retiming the circuit by cutting.

Figure 5.7: $\mu R\,;\mu S$ and $\mu(R \leftrightarrow S)$

Of course those contours that correspond to an instance of one of our laws about a regular circuit will form a regular family that matches the regularity of the circuit.

5.9 State machines

One of the common ways of talking about sequential circuits is to talk of an automaton, or state-machine, which is described by a pair of functions. At each instant one gives the output from the machine, and the other gives a new state; each is a function the input at that instant, and of the previous state. There are slight variations on this theme, but they all amount to the same thing. The work here on sequential circuits began from an attempt to formalise the design of such state-machines.

Suppose the output function is O and the next-state function is S, then the automaton is implemented by $\mu(\mathsf{fork}\,;[S,O])$ where

$$x\,(\mathcal{S}[\![\mu R]\!])\,y \iff \exists p,q.\,\langle x,p\rangle\,(\mathsf{zip}\,/\,\mathcal{S}[\![R]\!])\,\langle q,y\rangle\;\&\;q\,\mathcal{D}\,p$$

Read μR as 'fix R' or 'mu R'; it is (almost) the reason that this notation used to be called μFP, being like Backus' FP notation, but with this extra μ operator. Indeed if you have μ as a primitive concept, $\mathcal{D}$ need not be give since it is definable by $\mathcal{D} = \mu\iota = \mu 2$.

Figure 5.7 suggests a layout for μR which makes it obvious how to go about proving some of its properties, amongst them

$$\begin{aligned} A\,;\mu R\,;B &= \mu(\mathsf{fst}\,A\,;R\,;\mathsf{snd}\,B) \\ \mu(\mathsf{snd}\,X\,;R) &= \mu(R\,;\mathsf{fst}\,X) \end{aligned}$$

Note that while these equations are unconditional it is necessary to be careful with, say, $\mu(\mathsf{fork}\,;[S\,;X,O]) = \mu(\mathsf{fork}\,;[S,O]\,;\mathsf{fst}\,X) = \mu(\mathsf{snd}\,X\,;\mathsf{fork}\,;[S,O]) \neq \mu(\mathsf{fork}\,;[\mathsf{snd}\,X\,;S,\mathsf{snd}\,X\,;O])$.

The following equations suggest ways of setting about factorising a large state-machine into several smaller ones.

$$\begin{aligned} \mu R\,;\mu S &= \mu(R \leftrightarrow S) \\ \mu\mu R &= \mu(rsh\,;R\,;lsh) \end{aligned}$$

This factorisation is a technique which has advantages for implementation both because small parcels of state can be distributed about a circuit, tending to reduce wire lengths, and the transition functions become smaller as functions of a smaller state, being perhaps implemented by look-up in a read-only memory.

5.10 Other times

Recall the statement of Horner's rule for retiming, that $(\mathsf{apr} \mathbin{/} \mathsf{tri}\,\mathcal{D})\ ;\ \mathsf{rdr}\,S = \mathsf{rdr}(\mathsf{snd}\,\mathcal{D}\ ;\ S)$ for stateless S. Assume for simplicity that inputs are in the domain and outputs in the range. As suggested earlier you can read this equation to say that 'a circuit $\mathsf{rdr}(\mathsf{snd}\,\mathcal{D} ; S)$ can be used to implement $\mathsf{rdr}\,S$ by supplying the input earlier and skewed according to $\mathsf{apr} \mathbin{/} \mathsf{tri}\,\mathcal{D}$.' In fact this reading is really

$$\begin{aligned} \mathsf{rdr}\,S &= (\mathsf{apr} \mathbin{/} \mathsf{tri}\,\mathcal{D}^{-1})\ ;(\mathsf{apr} \mathbin{/} \mathsf{tri}\,\mathcal{D})\ ;\ \mathsf{rdr}\,S \\ &= (\mathsf{apr} \mathbin{/} \mathsf{tri}\,\mathcal{D}^{-1})\ ;\ \mathsf{rdr}(\mathsf{snd}\,\mathcal{D}\ ;\ S) \end{aligned}$$

which relies on knowing that $\mathcal{D}^{-1}\ ;\ \mathcal{D} = \iota$.

This equality holds in the model where time sequences are doubly infinite, but not in case time is modelled by $\mathbb{N}$. If there is a distinguished first time then

$$\begin{aligned} x\,(\mathcal{D}^{-1}\ ;\ \mathcal{D})\,z &\iff \exists y.\,\forall i.\,x_{i+1} = y_i \,\&\, \forall i.\,y_i = z_{i+1} \\ &\iff \forall i.\,x_{i+1} = z_{i+1} \\ &\underset{\not\Leftarrow}{\Rightarrow} \forall i.\,x_i = z_i \end{aligned}$$

and the implication is strict because $x\,(\mathcal{D}^{-1}\ ;\ \mathcal{D})\,z$ does not say anything about x_0 and z_0, so $\mathcal{D}^{-1}\ ;\ \mathcal{D} \neq \iota$.

Since we did not need to be able to cancel $\mathcal{D}^{-1}\ ;\ \mathcal{D}$ in the course of its proof, $(\mathsf{apr} \mathbin{/} \mathsf{tri}\,\mathcal{D})\ ;\ \mathsf{rdr}\,S = \mathsf{rdr}(\mathsf{snd}\,\mathcal{D}\ ;\ S)$ remains true in this model. On the other hand, $\mathsf{rdr}\,S \neq (\mathsf{apr} \mathbin{/} \mathsf{tri}\,\mathcal{D}^{-1})\ ;\ \mathsf{rdr}(\mathsf{snd}\,\mathcal{D}\ ;\ S)$ so there is not going to be a pipelined circuit which you can substitute for $\mathsf{rdr}\,S$ in the course of a calculation. However it remains safe to reason that

$$\begin{aligned} \mathsf{fst}\,n\ ;\ \mathcal{D}^n\ ;\ \mathsf{rdr}\,S &= \mathsf{fst}\,n\ ;(\mathsf{apr} \mathbin{/} ((\mathsf{rev} \mathbin{/} \mathsf{tri}\,\mathcal{D})\ ;\ \mathsf{tri}\,\mathcal{D}))\ ;\ \mathsf{rdr}\,S \\ &= \mathsf{fst}\,n\ ;(\mathsf{apr} \mathbin{/} (\mathsf{rev} \mathbin{/} \mathsf{tri}\,\mathcal{D}))\ ;(\mathsf{apr} \mathbin{/} \mathsf{tri}\,\mathcal{D})\ ;\ \mathsf{rdr}\,S \\ &= \mathsf{fst}(n\ ;\ \mathsf{rev} \mathbin{/} \mathsf{tri}\,\mathcal{D}\ ;\ \mathsf{map}\,\mathcal{D})\ ;\ \mathsf{rdr}(\mathsf{snd}\,\mathcal{D}\ ;\ S) \end{aligned}$$

so there is a pipelined circuit which is the same as $\mathsf{rdr}\,S$ up to composition with a power of $\mathcal{D}$. The minimum such power – in this case n where n is the width of the reduction – is called the *latency* of the circuit, or more strictly the additional latency imposed by the pipelining.

This last calculation gives a clue as to why you might want to distinguish between $\mathcal{D}^{-1} ; \mathcal{D}$ and the identity. In a model where there is a time before which there was no time the interpretation of $\mathcal{D}$ is like a latch with an initial state that you cannot control. The reason $\mathsf{rdr}\, S \neq (\mathsf{apr} / \mathsf{tri}\, \mathcal{D}^{-1}) ; \mathsf{rdr}(\mathsf{snd}\, \mathcal{D} ; S)$ is that whatever interpretation you might want to do about implementing $\mathsf{apr} \,/\, \mathsf{tri}\, \mathcal{D}^{-1}$, the output from $\mathsf{rdr}(\mathsf{snd}\, \mathcal{D} ; S)$ has come from a pipelined circuit containing some number of registers whose initial state was not known. Suppose S is stateless, then the input to $\mathsf{rdr}\, S$ determines its output at time zero; whereas in general you cannot tell what the output of $\mathsf{rdr}(\mathsf{snd}\, \mathcal{D} ; S)$ will be until you have filled the pipeline.

Things that remain true without the assumption that $\mathcal{D}^{-1} ; \mathcal{D} = \iota$ do not depend on knowing the initial states of latches. If $\mathcal{D}^{-1} ; \mathcal{D} = \iota$ is used in a proof of some equality, it might be that the equality does not guarantee the behaviour of the circuits for the first few clocks after the power is first applied. If the proof of equality can be done without this assumption, the equality guarantees the same latency on both sides of the equation, so you might want to go about a development in this way in order to calculate the latency of a circuit.

A dual, but less useful, argument holds for the last moment of time in models that impose a deadline after which there will be no more time.

Chapter 6

A systolic correlator

This chapter treats a design as an example of the style which we are advocating. The requirement is to calculate

$$c(t) = \sum_{i=0}^{N-1} r_i(t) \times d(t-i)$$

at each time t given time sequences r_i and d. The circuit [Kung80, McCann82, McCabe82] is one which has appeared many times, including appearances in a number of earlier presentations of this material [Sheer83, Luk87, Sheer88, Luk88a], so ought to be well understood by now!

The first thing to do is to get the specification cast into a more familiar form. First of all, we separate the time and space dimensions by introducing a new sequence of values, d'_i, for which

$$c(t) = \sum_{i=0}^{N-1} r_i(t) \times d'_i(t) \ \& \ d'_i(t) = d(t-i)$$

the point being that the part with the arithmetic in it looks as though it can be implemented by a stateless circuit calculating

$$c = \sum_{i=0}^{N-1} r_i \times d'_i$$

and the other part looks like a shift-register.

Translating the stateless part of the specification, suppose we have components *mul* and *acc* for which $\langle x, y\rangle \ mul \ (x \times y)$ and $x \ acc \ (\sum_i x_i)$, then

$$\langle r, d'\rangle \ (\mathsf{zip} \ ; \mathsf{map} \ mul \ ; \ acc) \ c$$

There is a quiet design decision made in selecting an order for r and d', and perhaps in dividing the inputs and outputs between domain and range. The former will not matter much, but the latter is regrettably harder to modify without starting again. There is also some freedom of action in implementing acc, for example by either $\mathsf{apl}\,;\mathsf{rdl}\,add$ or $\mathsf{apr}^{-1}\,;\mathsf{rdr}\,add$. We will return to this after considering how to implement the shift register.

If $d'_i(t) = d(t-i)$ then from the definition of $\mathcal{D}$, it is clear that $d'_i\,\mathcal{D}^{-i}\,d$ for each i. Another way of saying this is that $d'\,(\mathsf{tri}\,\mathcal{D}^{-1}\,;J)\,d$ where J is a many-way join, which we can again implement in at least two different ways for $J = \mathsf{apl}^{-1};\mathsf{rdl}\,\mathsf{join} = \mathsf{apr}^{-1}\,;\mathsf{rdr}\,\mathsf{join}$. We have already met $\mathsf{tri}\,R\,;\mathsf{apr}^{-1}\,;\mathsf{rdr}\,S$ so for the moment let us pursue that possibility, for by Horner's rule

$$\mathsf{tri}\,\mathcal{D}^{-1}\,;\mathsf{apr}^{-1}\,;\mathsf{rdr}\,\mathsf{join} = \mathsf{apr}^{-1}\,;\mathsf{rdr}(\mathsf{snd}\,\mathcal{D}^{-1}\,;\mathsf{join})$$

It follows that $d\,(\mathsf{apr}^{-1}\,;\mathsf{rdr}(\mathsf{snd}\,\mathcal{D}^{-1}\,;\mathsf{join}))^{-1}\,d'$ and that means that we can implement the whole correlator as a composition of this shift-register part and the stateless part,

$$\langle r, d\rangle\,(\mathsf{snd}(\mathsf{apr}^{-1}\,;\mathsf{rdr}(\mathsf{snd}\,\mathcal{D}^{-1}\,;\mathsf{join}))^{-1}\,;\mathsf{zip}\,;\mathsf{map}\,mul\,;acc)\,c$$

This description still contains the rather odd-looking $\mathcal{D}^{-1}$ and join, but both appear inside an inversion which we are only prevented from rewriting by our not having a name for the circuit form that takes R to $(\mathsf{rdr}\,R^{-1})^{-1}$.

Going back to the definition of reduction, $\mathsf{rdr}\,R = \mathsf{col}(R\,;{\pi_1}^{-1})\,;\pi_1$, and since ${\pi_1}^{-1}\,;(\mathsf{snd}\,\mathcal{D}^{-1}\,;\mathsf{join})$ is the identity relation the apr^{-1} can be absorbed into the reduction, $\mathsf{apr}^{-1};\mathsf{rdr}(\mathsf{snd}\,\mathcal{D}^{-1};\mathsf{join})$ is a smaller relation than ${\pi_1}^{-1};\mathsf{rdr}(\mathsf{snd}\,\mathcal{D}^{-1};\mathsf{join})$. This effectively permits r to be a list of length zero.

$$\begin{aligned}
CORR &= \mathsf{snd}({\pi_1}^{-1}\,;\mathsf{rdr}(\mathsf{snd}\,\mathcal{D}^{-1}\,;\mathsf{join}))^{-1}\,;\mathsf{zip}\,;\mathsf{map}\,mul\,;acc \\
&= \mathsf{snd}({\pi_1}^{-1}\,;\mathsf{col}(\mathsf{snd}\,\mathcal{D}^{-1}\,;\mathsf{join}\,;{\pi_1}^{-1})\,;\pi_1)^{-1}\,;\mathsf{zip}\,;\mathsf{map}\,mul\,;acc \\
&= \mathsf{snd}({\pi_1}^{-1}\,;\mathsf{row}(\pi_1\,;\mathsf{fork}\,;\mathsf{snd}\,\mathcal{D})\,;\pi_1)\,;\mathsf{zip}\,;\mathsf{map}\,mul\,;acc
\end{aligned}$$

In a sense that is a complete design of a correlator, for we can implement all the components as functions from the domain to the range: the rather odd-looking ${\pi_1}^{-1}$ is no problem because it connects to a $\mathsf{row}(\pi_1\,;R)$ so it does not matter how we choose to invent the other component of the pair.

The rest of the development is going to be a matter of optimizing this initial implementation, and translating the implementation from one that operates on numbers to one that operates on binary representations of numbers. Perhaps the most obvious inefficiency is the instance of zip, which since it interleaves two lists of numbers would involve a great number of wire-crossings, and a large layout area.

6.1 Zipping rows and columns

One of the common optimisations in circuit design is to interleave a number of similar regular grid-shaped calculations, such as arise in dealing with binary

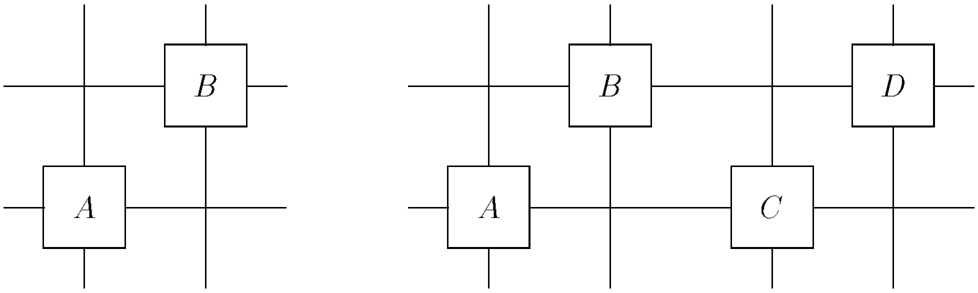

Figure 6.1: $\mathsf{zip} \mathbin{/} [A, B]$ and $(\mathsf{zip} \mathbin{/} [A, B]) \leftrightarrow (\mathsf{zip} \mathbin{/} [C, D])$

representations of numbers, to shorten the wires that connect them, for example bringing bits of the same weight in different number together if they are to be added. This is the transformation that takes a number of arithmetic circuits and interleaves them to make the data-path of a processor.

In our formalism, this transformation is captured by the observation that $(\mathsf{zip} \mathbin{/} [A, B]) \leftrightarrow (\mathsf{zip} \mathbin{/} [C, D]) = \mathsf{snd\,zip}^{-1} \,;\, (\mathsf{zip} \mathbin{/} [A \leftrightarrow C, B \leftrightarrow D]) \,;\, \mathsf{fst\,zip}$ and of course the dual result for below. It generalizes by induction to

$$
\begin{aligned}
\mathsf{row}(\mathsf{zip} \mathbin{/} [R, S]) &= \mathsf{snd\,zip}^{-1} \,;\, (\mathsf{zip} \mathbin{/} [\mathsf{row}\,R, \mathsf{row}\,S]) \,;\, \mathsf{fst\,zip}\\
\mathsf{col}(\mathsf{zip} \mathbin{/} [R, S]) &= \mathsf{fst\,zip}^{-1} \,;\, (\mathsf{zip} \mathbin{/} [\mathsf{col}\,R, \mathsf{col}\,S]) \,;\, \mathsf{snd\,zip}\\
\mathsf{row}(\mathsf{zip} \mathbin{/} \mathsf{map}\,R) &= \mathsf{snd\,trn}^{-1} \,;\, (\mathsf{zip} \mathbin{/} \mathsf{map\,row}\,R) \,;\, \mathsf{fst\,trn}\\
\mathsf{col}(\mathsf{zip} \mathbin{/} \mathsf{map}\,R) &= \mathsf{fst\,trn}^{-1} \,;\, (\mathsf{zip} \mathbin{/} \mathsf{map\,col}\,R) \,;\, \mathsf{snd\,trn}
\end{aligned}
$$

and by application of two of the above to such observations as

$$
\begin{aligned}
\mathsf{row\,col}(\mathsf{zip} \mathbin{/} \mathsf{map}\,R) &= \mathsf{row}(\mathsf{fst\,trn}^{-1} \,;\, (\mathsf{zip} \mathbin{/} \mathsf{map\,col}\,R) \,;\, \mathsf{snd\,trn})\\
&= \mathsf{fst\,trn}^{-1} \,;\, \mathsf{row}(\mathsf{zip} \mathbin{/} \mathsf{map\,col}\,R) \,;\, \mathsf{snd\,trn}\\
&= [\mathsf{trn}^{-1}, \mathsf{trn}^{-1}] \,;\, (\mathsf{zip} \mathbin{/} \mathsf{map\,row\,col}\,R) \,;\, [\mathsf{trn}, \mathsf{trn}]
\end{aligned}
$$

The relevance of these results is, in the first instance, in eliminating the zip from the initial description of the correlator.

6.2 Implementing the shift register

The components next to the zip are first cast into the form of two rows and these are then interleaved.

$$
\begin{aligned}
&\mathsf{snd}({\pi_1}^{-1} \,;\, \mathsf{row}(\pi_1 \,;\, \mathsf{fork} \,;\, \mathsf{snd}\,\mathcal{D}) \,;\, \pi_1) \,;\, \mathsf{zip}\\
&\quad = \{\, \text{type of } \mathsf{zip} \,\}\\
&\qquad [\mathsf{map}\,\iota, {\pi_1}^{-1} \,;\, \mathsf{row}(\pi_1 \,;\, \mathsf{fork} \,;\, \mathsf{snd}\,\mathcal{D}) \,;\, \pi_1] \,;\, \mathsf{zip}
\end{aligned}
$$

$$
\begin{aligned}
&= \{ \text{implementing } \mathsf{map}\,\iota \text{ as a row} \} \\
&\quad [\pi_2{}^{-1} ; \mathsf{row}(\pi_2 ; \pi_1{}^{-1}) ; \pi_1, \pi_1{}^{-1} ; \mathsf{row}(\pi_1 ; \mathsf{fork} ; \mathsf{snd}\,\mathcal{D}) ; \pi_1] ; \mathsf{zip} \\
&= \{ \text{rearranging terms} \} \\
&\quad [\pi_2{}^{-1}, \pi_1{}^{-1}] ; [\mathsf{row}(\pi_2 ; \pi_1{}^{-1}), \mathsf{row}(\pi_1 ; \mathsf{fork} ; \mathsf{snd}\,\mathcal{D})] ; [\pi_1, \pi_1] ; \mathsf{zip} \\
&= \{ [\pi_1, \pi_1] ; \mathsf{zip} = \mathsf{zip}^{-1} ; \mathsf{fst\,zip} ; \pi_1 \} \\
&\quad [\pi_2{}^{-1}, \pi_1{}^{-1}] ; [\mathsf{row}(\pi_2 ; \pi_1{}^{-1}), \mathsf{row}(\pi_1 ; \mathsf{fork} ; \mathsf{snd}\,\mathcal{D})] ; \mathsf{zip}^{-1} ; \mathsf{fst\,zip} ; \pi_1 \\
&= \{ \text{interleaving} \} \\
&\quad [\pi_2{}^{-1}, \pi_1{}^{-1}] ; \mathsf{zip}^{-1} ; \mathsf{snd\,zip} ; \mathsf{row}\,R ; \pi_1 \\
&\quad \text{where } R = \mathsf{zip} \,/\, [\pi_2 ; \pi_1{}^{-1}, \pi_1 ; \mathsf{fork} ; \mathsf{snd}\,\mathcal{D}] \\
&= \{ [\pi_2{}^{-1}, \pi_1{}^{-1}] ; \mathsf{zip}^{-1} = \mathsf{rev} ; [\pi_2{}^{-1}, \pi_1{}^{-1}] \} \\
&\quad \mathsf{rev} ; [\pi_2{}^{-1}, \pi_1{}^{-1}] ; \mathsf{snd\,zip} ; \mathsf{row}\,R ; \pi_1 \\
&= \{ \pi_1{}^{-1} ; \mathsf{zip} = \mathsf{map}\,\pi_1{}^{-1} \} \\
&\quad \mathsf{rev} ; [\pi_2{}^{-1}, \mathsf{map}\,\pi_1{}^{-1}] ; \mathsf{row}\,R ; \pi_1
\end{aligned}
$$

The appearance of the reverse on the left suggests that we made an inefficient choice for the order of the inputs, but we can change our minds and implement instead $\mathsf{rev} ; CORR$ to eliminate the awkward term.

Now for the interleaved components of which we have a row: the description we have for this cell is one that consists of two separate components and enough plumbing to make it appear that they are interleaved. A more efficient implementation can be obtained by a bit of simplification, using the knowledge that the left-most input to the row is provided through $\mathsf{fst}\,\pi_2{}^{-1}$ and the top input through $\mathsf{snd\,map}\,\pi_1{}^{-1}$.

$$
\begin{aligned}
[\pi_2{}^{-1}, \pi_1{}^{-1}] ; R &= [\pi_2{}^{-1}, \pi_1{}^{-1}] ; (\mathsf{zip} \,/\, [\pi_2 ; \pi_1{}^{-1}, \pi_1 ; \mathsf{fork} ; \mathsf{snd}\,\mathcal{D}]) \\
&= [\pi_2{}^{-1}, \pi_1{}^{-1}] ; \mathsf{zip} ; [\pi_2 ; \pi_1{}^{-1}, \pi_1 ; \mathsf{fork} ; \mathsf{snd}\,\mathcal{D}] ; \mathsf{zip}^{-1} \\
&= \mathsf{rev} ; [\pi_2{}^{-1}, \pi_1{}^{-1}] ; [\pi_2 ; \pi_1{}^{-1}, \pi_1 ; \mathsf{fork} ; \mathsf{snd}\,\mathcal{D}] ; \mathsf{zip}^{-1} \\
&= \mathsf{rev} ; [\pi_2{}^{-1} ; \pi_2 ; \pi_1{}^{-1}, \pi_1{}^{-1} ; \pi_1 ; \mathsf{fork} ; \mathsf{snd}\,\mathcal{D}] ; \mathsf{zip}^{-1} \\
&= \mathsf{rev} ; [\pi_1{}^{-1}, \mathsf{fork}] ; \mathsf{snd\,snd}\,\mathcal{D} ; \mathsf{zip}^{-1} \\
&= \mathsf{rev} ; [\pi_1{}^{-1}, \mathsf{fork}] ; \mathsf{zip}^{-1} ; \mathsf{snd\,snd}\,\mathcal{D} \\
&= \mathsf{fork} ; [\mathsf{rev}, \pi_1 ; \pi_2{}^{-1}] ; \mathsf{snd\,snd}\,\mathcal{D} \\
&= dist ; \mathsf{snd}\,\mathcal{D} ; [\mathsf{rev}, \pi_2{}^{-1}] \\
&\quad \text{where } dist = \mathsf{fork} ; \mathsf{snd}\,\pi_1
\end{aligned}
$$

so substituting for R in a $\mathsf{row}\,R$ in its context

$$
\begin{aligned}
[\pi_2{}^{-1}, \mathsf{map}\,\pi_1{}^{-1}] ; \mathsf{row}\,R ; \pi_1 &= \mathsf{row}(dist ; \mathsf{snd}\,\mathcal{D}) ; [\mathsf{map\,rev}, \pi_2{}^{-1}] ; \pi_1 \\
&= \mathsf{row}(dist ; \mathsf{snd}\,\mathcal{D}) ; \pi_1 ; \mathsf{map\,rev}
\end{aligned}
$$

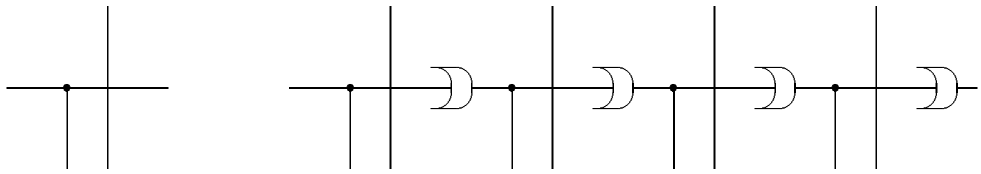

Figure 6.2: $dist = \mathsf{fork} \,;\, \mathsf{snd}\, \pi_1$ and $\mathsf{row}(dist \,;\, \mathsf{snd}\, \mathcal{D})$

and putting together what we have so far

$$
\begin{aligned}
CORR' &= \mathsf{rev} \,;\, CORR \\
&= \mathsf{row}(dist \,;\, \mathsf{snd}\, \mathcal{D}) \,;\, \pi_1 \,;\, \mathsf{map}\, \mathsf{rev} \,;\, \mathsf{map}\, mul \,;\, acc \\
&= \{ \text{commutativity of multiplication} \} \\
&\quad\; \mathsf{row}(dist \,;\, \mathsf{snd}\, \mathcal{D}) \,;\, \pi_1 \,;\, \mathsf{map}\, mul \,;\, acc
\end{aligned}
$$

implements a relation for which $\langle d, r\rangle\ CORR'\ c$.

Since we know that acc can be implemented by rows or columns of components which add a pair of numbers, it looks as though we might be able to implement $CORR'$ as a pair of rows, one below the other. The point of doing this is that $(\mathsf{row}\, R) \updownarrow (\mathsf{row}\, S) = \mathsf{row}(R \updownarrow S)$ and we can hope to make further simplifications in the component $R \updownarrow S$.

6.3 Implementing the accumulator

On the face of it, there is no choice here since we want to implement acc as a row, it will have to be something like a left reduction of add, say $acc = \mathsf{apl}^{-1} \,;\, \mathsf{rdl}\, add$. This would be a reasonable implementation, but since it does not allow empty rows it would not be natural in the context of a potentially zero-width shift register. Define instead

$$
\begin{aligned}
acc &= {\pi_2}^{-1} \,;\, \mathsf{fst}\, zero \,;\, \mathsf{rdl}\, add \\
&\quad\; \text{where } x\ zero\ y \iff y = 0 \\
&= {\pi_2}^{-1} \,;\, \mathsf{fst}\, zero \,;\, \mathsf{row}(add \,;\, {\pi_2}^{-1}) \,;\, \pi_2
\end{aligned}
$$

and if we do indeed proceed in this direction

$$
\begin{aligned}
& CORR' \\
&\quad = \mathsf{row}(dist \,;\, \mathsf{snd}\, \mathcal{D}) \,;\, \pi_1 \,;\, \mathsf{map}\, mul \,;\, {\pi_2}^{-1} \,;\, \mathsf{fst}\, zero \,;\, \mathsf{row}(add \,;\, {\pi_2}^{-1}) \,;\, \pi_2 \\
&\quad = \mathsf{row}(dist \,;\, \mathsf{snd}\, \mathcal{D}) \,;\, \pi_1 \,;\, {\pi_2}^{-1} \,;\, \mathsf{fst}\, zero \,;\, \mathsf{row}(\mathsf{snd}\, mul \,;\, add \,;\, {\pi_2}^{-1}) \,;\, \pi_2 \\
&\quad = \mathsf{fst}\, {\pi_2}^{-1} \,; \\
&\quad\qquad (\mathsf{fst}\, zero \,;\, \mathsf{row}(\mathsf{snd}\, mul \,;\, add \,;\, {\pi_2}^{-1}) \updownarrow (\mathsf{row}(dist \,;\, \mathsf{snd}\, \mathcal{D})) \,; \\
&\quad\qquad\quad \mathsf{snd}\, \pi_1 \,;\, \pi_2
\end{aligned}
$$

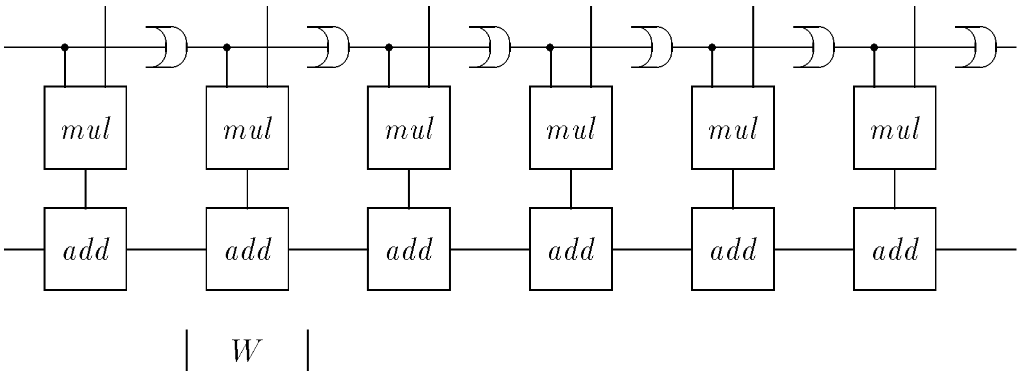

Figure 6.3: an instance of $\mathsf{rdl}(W\,;\pi_2\,;\mathsf{snd}\,\mathcal{D})$ where $W = (\mathsf{snd}\;mul\,;add\,;{\pi_2}^{-1}) \updownarrow dist$

$$
\begin{aligned}
&= \mathsf{fst}({\pi_2}^{-1}\,;\mathsf{fst}\;zero)\,;\mathsf{row}((\mathsf{snd}\;mul\,;add\,;{\pi_2}^{-1}) \updownarrow (dist\,;\mathsf{snd}\,\mathcal{D}))\,;\pi_2\,;\pi_1\\
&= \mathsf{fst}({\pi_2}^{-1}\,;\mathsf{fst}\;zero)\,;\mathsf{row}(W\,;\mathsf{snd}\,\mathsf{snd}\,\mathcal{D})\,;\pi_2\,;\pi_1\\
&\ \text{where } W = (\mathsf{snd}\;mul\,;add\,;{\pi_2}^{-1}) \updownarrow dist\\
&= \mathsf{fst}({\pi_2}^{-1}\,;\mathsf{fst}\;zero)\,;\mathsf{rdl}(W\,;\pi_2\,;\mathsf{snd}\,\mathcal{D})\,;\pi_1
\end{aligned}
$$

This completes the rearrangement of the correlator into a very regular circuit with few different sorts of components, and only local connections between those components. In fact there is only one sort of component: W which is a cell which performs the only 'word-level' operation that we need. By 'word' here we mean just a number.

6.4 Making the circuit systolic

Unfortunately although the wires explicit in the structure of the row are all short – which is what we mean by saying that the connections are all local – there are some potentially long propagation paths in the circuit. The one which stands out in figure 6.3 is the lower path along the row, that through the accumulator. We would prefer to avoid these long paths, on the assumption that they will take a long time to settle into a known state. Our strategy is to trade latency for throughput by pipelining the long paths.

An application of a skewing theorem analogous to Horner's rule can be used to pipeline this path: since $(\mathsf{rev}\ /\ \mathsf{tri}\,\mathcal{D})\,;\mathsf{apl}^{-1}\,;\mathsf{rdl}\,R = \mathsf{apl}^{-1}\,;\mathsf{rdl}(\mathsf{fst}\,\mathcal{D}\,;R)$

$$
\begin{aligned}
&CORR'\\
&\quad= \mathsf{fst}({\pi_2}^{-1}\,;\mathsf{fst}\;zero)\,;\mathsf{rdl}(W\,;\pi_2\,;\mathsf{snd}\,\mathcal{D})\,;\pi_1\\
&\quad= \mathsf{fst}({\pi_2}^{-1}\,;\mathsf{fst}\;zero)\,;(\mathsf{apl}\,;\mathsf{rev})\ /\ \mathsf{tri}\,\mathcal{D}^{-1}\,;\\
&\qquad\ \mathsf{rdl}(\mathsf{fst}\,\mathcal{D}\,;W\,;\pi_2\,;\mathsf{snd}\,\mathcal{D})\,;\pi_1
\end{aligned}
$$

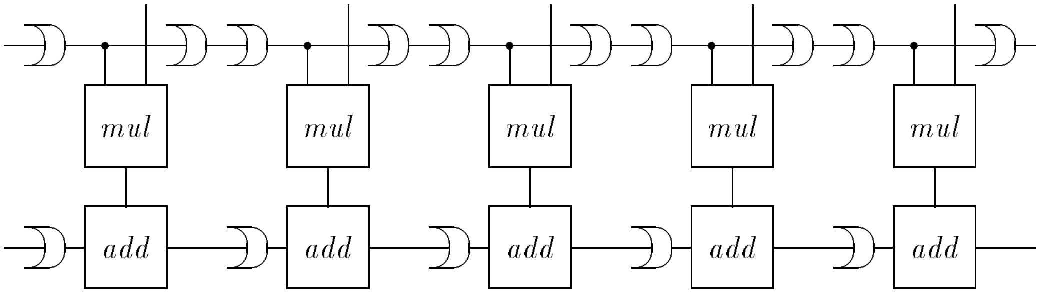

Figure 6.4: an instance of $\mathsf{rdl}(\mathsf{fst}\,\mathcal{D} \,;\, W \,;\, \pi_2 \,;\, \mathsf{snd}\,\mathcal{D})$

$$\begin{aligned} &= \mathsf{fst}({\pi_2}^{-1} \,;\, \mathsf{fst}\,zero \,;\, \mathcal{D}^{-N}) \,;\, \mathsf{snd}(\mathsf{rev} \,/\, \mathsf{tri}\,\mathcal{D}^{-1}) \,; \\ &\quad \mathsf{rdl}(\mathsf{fst}\,\mathcal{D} \,;\, W \,;\, \pi_2 \,;\, \mathsf{snd}\,\mathcal{D}) \,;\, \pi_1 \\ &= [\mathcal{D}^{-N} \,;\, {\pi_2}^{-1} \,;\, \mathsf{fst}\,zero, \mathsf{rev} \,/\, \mathsf{tri}\,\mathcal{D}^{-1}] \,; \\ &\quad \mathsf{rdl}(\mathsf{fst}\,\mathcal{D} \,;\, W \,;\, \pi_2 \,;\, \mathsf{snd}\,\mathcal{D}) \,;\, \pi_1 \end{aligned}$$

so if we adjust the timing of the inputs and implement instead

$$\begin{aligned} &(\mathsf{apl} \,;\, \mathsf{rev}) \,/\, \mathsf{tri}\,\mathcal{D} \,;\, CORR' \\ &\quad = \mathsf{fst}({\pi_2}^{-1} \,;\, \mathsf{fst}\,zero) \,;\, \mathsf{rdl}(\mathsf{fst}\,\mathcal{D} \,;\, W \,;\, \pi_2 \,;\, \mathsf{snd}\,\mathcal{D}) \,;\, \pi_1 \end{aligned}$$

an example of which is shown in figure 6.4, there are no combinational propagation paths longer than one cell. This sort of design is called systolic, in this case word-level systolic, perhaps because the clocking of the latches pumps data around the circuit. As we use it here the word 'systolic' means just that there are no combinational paths (at a given level of detail).

The problem with this implementation of the correlator is that, contrary to appearances in our diagrams, latches tend to be much bigger and so more expensive than the other components in a circuit. The design in figure 6.4 has in each cell three latches two of which are big enough to store a value of d, and one of which is big enough to store a value of the accumulated sum.

Looking for an alternative strategy we might try skewing in the other direction

$$\begin{aligned} &(\mathsf{apl} \,;\, \mathsf{rev}) \,/\, \mathsf{tri}\,\mathcal{D}^{-1} \,;\, CORR' \\ &\quad = \mathsf{fst}({\pi_2}^{-1} \,;\, \mathsf{fst}\,zero) \,;\, \mathsf{rdl}(\mathsf{fst}\,\mathcal{D}^{-1} \,;\, W \,;\, \pi_2 \,;\, \mathsf{snd}\,\mathcal{D}) \,;\, \pi_1 \end{aligned}$$

which introduces anti-delays into the accumulator. In order to be able to implement these by latches we would need to reverse the data-flow in the accumulator. This is a matter of a change of representation, for if *negate* is the relation for which $x\ negate\ y$ iff $x = -y$,

$$\begin{aligned} zero \,;\, \mathsf{rdl}\,add \quad &= \{\, zero = zero \,;\, negate \,\} \\ &\quad zero \,;\, negate \,;\, \mathsf{rdl}\,add \end{aligned}$$

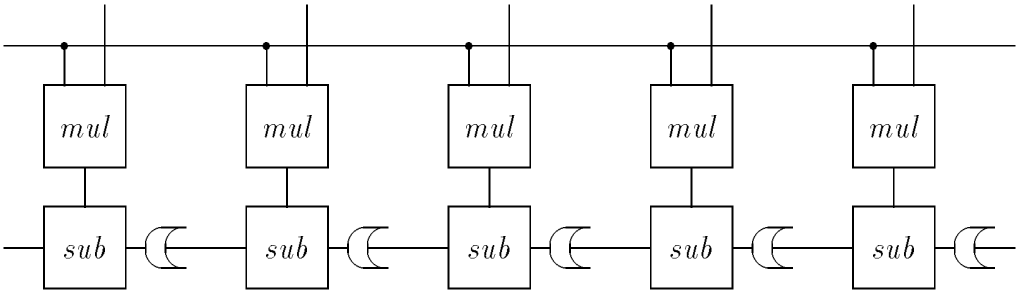

Figure 6.5: an instance of $\mathsf{rdl}(W' ; \pi_2 ; \mathsf{fst}\,\mathcal{D}^{-1})$ where $W' = (\mathsf{snd}\ mul ; sub ; {\pi_2}^{-1}) \updownarrow dist$

$$
\begin{aligned}
&= \{\ \mathsf{fst}\ negate\ ;\ add = \mathsf{snd}\ negate\ ;\ add\ ;\ negate\ \} \\
&\quad zero\ ;\ \mathsf{rdl}(\mathsf{snd}\ negate\ ;\ add)\ ;\ negate \\
&= \quad zero\ ;\ \mathsf{rdl}\ sub\ ;\ negate \\
&\quad \text{where } sub = \mathsf{snd}\ negate\ ;\ add
\end{aligned}
$$

and similarly, if $shift_c$ is the relation for which $x\ shift_c\ (x-c)$ then since $\mathsf{fst}\ shift_c\ ;$ $add = add\ ;\ shift_c$ it follows that $\mathsf{fst}\ shift_c\ ;\ \mathsf{rdl}\ sub = \mathsf{rdl}\ sub\ ;\ shift_c$ and

$$
\begin{aligned}
\langle 0, x\rangle\ (\mathsf{rdl}\ sub\ ;\ negate)\ c \quad &\Longleftrightarrow \quad \langle c, x\rangle\ (\mathsf{rdl}\ sub\ ;\ negate)\ 0 \\
&\Longleftrightarrow \quad \langle c, x\rangle\ (\mathsf{rdl}\ sub\ ;\ negate\ ;\ zero^{-1})\ y \\
&\Longleftrightarrow \quad \langle c, x\rangle\ (\mathsf{rdl}\ sub\ ;\ zero^{-1})\ y
\end{aligned}
$$

The idea of this is that if we implement sub by an adder, we can implement a new correlator circuit, $CORR''$, for which $\langle\langle c, d\rangle, r\rangle\ CORR''\ y$ for any y

$$
\begin{aligned}
&(\mathsf{apl}\ ;\ \mathsf{rev})\ /\ \mathsf{tri}\,\mathcal{D}^{-1}\ ;\ CORR'' \\
&\quad = \quad \mathsf{rdl}(\mathsf{fst}\,\mathcal{D}^{-1}\ ;\ W'\ ;\ \pi_2\ ;\ \mathsf{snd}\,\mathcal{D})\ ;\ \pi_1\ ;\ zero^{-1} \\
&\quad \qquad \text{where } W' = (\mathsf{snd}\ mul\ ;\ sub\ ;\ {\pi_2}^{-1}) \updownarrow dist \\
&\quad = \quad \mathsf{fst}\,\mathcal{D}^{-1}\ ;\ \mathsf{rdl}(W'\ ;\ \pi_2\ ;\ \mathsf{snd}\,\mathcal{D}\ ;\ \mathcal{D}^{-1})\ ;\ \mathcal{D}\ ;\ \pi_1\ ;\ zero^{-1} \\
&\quad = \quad \mathsf{fst}\,\mathcal{D}^{-1}\ ;\ \mathsf{rdl}(W'\ ;\ \pi_2\ ;\ \mathsf{fst}\,\mathcal{D}^{-1})\ ;\ \pi_1\ ;\ zero^{-1}
\end{aligned}
$$

so

$$
\begin{aligned}
&\mathsf{fst}\,\mathcal{D}\ ;\ (\mathsf{apl}\ ;\ \mathsf{rev})\ /\ \mathsf{tri}\,\mathcal{D}^{-1}\ ;\ CORR'' \\
&\quad = \quad \mathsf{rdl}(W'\ ;\ \pi_2\ ;\ \mathsf{fst}\,\mathcal{D}^{-1})\ ;\ \pi_1\ ;\ zero^{-1}
\end{aligned}
$$

Notice that the sub component is going to be an adder with its output on the left, the first component of the domain. The problem with this implementation, illustrated in figure 6.5, is that the 'unskewing' has cancelled the effect of the latches in the shift-register, and there is now a long combinational propagation path along the top of the circuit.

This failure has nothing to do with our decision to reverse the direction of the accumulator rather than that of the shift-register, which would in retrospect have been easier. In either case one or other of the paths through the row will be entirely combinational. The cure for this problem is to slow the circuit before the skewing, so that the anti-delay introduced by the skew cancels only half of the doubled delay in the shift-register, for

$$\begin{aligned}
&(\mathsf{apl} \;;\; \mathsf{rev}) \;/\; \mathsf{tri}\,\mathcal{D}^{-1} \;;\; \mathsf{slow}\; CORR'' \\
&\quad = \; (\mathsf{apl} \;;\; \mathsf{rev}) \;/\; \mathsf{tri}\,\mathcal{D}^{-1} \;;\; \mathsf{rdl}(W' \;;\; \pi_2 \;;\; \mathsf{snd}\,\mathcal{D}^2) \;;\; \pi_1 \;;\; zero^{-1} \\
&\quad = \; \mathsf{rdl}(\mathsf{fst}\,\mathcal{D}^{-1} \;;\; W' \;;\; \pi_2 \;;\; \mathsf{snd}\,\mathcal{D}^2) \;;\; \pi_1 \;;\; zero^{-1} \\
&\quad = \; \mathsf{fst}\,\mathcal{D}^{-1} \;;\; \mathsf{rdl}(W' \;;\; \pi_2 \;;\; [\mathcal{D}^{-1}, \mathcal{D}]) \;;\; \pi_1 \;;\; zero^{-1}
\end{aligned}$$

so

$$\begin{aligned}
&\mathsf{fst}\,\mathcal{D} \;;\; (\mathsf{apl} \;;\; \mathsf{rev}) \;/\; \mathsf{tri}\,\mathcal{D}^{-1} \;;\; \mathsf{slow}\; CORR'' \\
&\quad = \; \mathsf{rdl}(W' \;;\; \pi_2 \;;\; [\mathcal{D}^{-1}, \mathcal{D}]) \;;\; \pi_1 \;;\; zero^{-1}
\end{aligned}$$

There is something a bit strange about the description of the input timing requirements, which we can simplify by using the timelessness of everything and the fact that $CORR'' \;;\; \mathcal{D}^i = CORR''$, for

$$\begin{aligned}
&\mathsf{fst}\,\mathcal{D} \;;\; (\mathsf{apl} \;;\; \mathsf{rev}) \;/\; \mathsf{tri}\,\mathcal{D}^{-1} \;;\; \mathsf{slow}\; CORR'' \\
&\quad = \; [\mathcal{D}^{-(N-1)}, \mathsf{rev} \;/\; \mathsf{tri}\,\mathcal{D}^{-1}] \;;\; CORR'' \\
&\quad = \; \mathsf{snd}\,\mathsf{tri}\,\mathcal{D} \;;\; \mathcal{D}^{-(N-1)} \;;\; CORR'' \\
&\quad = \; \mathsf{snd}\,\mathsf{tri}\,\mathcal{D} \;;\; CORR'' \;;\; \mathcal{D}^{-(N-1)} \\
&\quad = \; \mathsf{snd}\,\mathsf{tri}\,\mathcal{D} \;;\; CORR''
\end{aligned}$$

so

$$\begin{aligned}
&\mathsf{snd}\,\mathsf{tri}\,\mathcal{D} \;;\; \mathsf{slow}\; CORR'' \\
&\quad = \; \mathsf{rdl}(W' \;;\; \pi_2 \;;\; [\mathcal{D}^{-1}, \mathcal{D}]) \;;\; \pi_1 \;;\; zero^{-1}
\end{aligned}$$

which is the circuit illustrated in figure 6.6. The latches on the extreme right are of course redundant and could be omitted at the cost of making the circuit less regular, or moved to the other end of the row since

$$\begin{aligned}
&[[\mathcal{D}^{-1}, \mathcal{D}], \mathsf{tri}\,\mathcal{D}] \;;\; \mathsf{slow}\; CORR'' \\
&\quad = \; \mathsf{fst}[\mathcal{D}^{-1}, \mathcal{D}] \;;\; \mathsf{rdl}(W' \;;\; \pi_2 \;;\; [\mathcal{D}^{-1}, \mathcal{D}]) \;;\; \pi_1 \;;\; zero^{-1} \\
&\quad = \; \mathsf{rdl}(\mathsf{fst}[\mathcal{D}^{-1}, \mathcal{D}] \;;\; W' \;;\; \pi_2) \;;\; [\mathcal{D}^{-1}, \mathcal{D}] \;;\; \pi_1 \;;\; zero^{-1} \\
&\quad = \; \mathsf{rdl}(\mathsf{fst}[\mathcal{D}^{-1}, \mathcal{D}] \;;\; W' \;;\; \pi_2) \;;\; \pi_1 \;;\; zero^{-1}
\end{aligned}$$

Read as a specification of the circuit on the right of the equals sign, in the equation

$$\mathsf{snd}\,\mathsf{tri}\,\mathcal{D} \;;\; \mathsf{slow}\; CORR'' \;=\; \mathsf{rdl}(W' \;;\; \pi_2 \;;\; [\mathcal{D}^{-1}, \mathcal{D}]) \;;\; \pi_1 \;;\; zero^{-1}$$

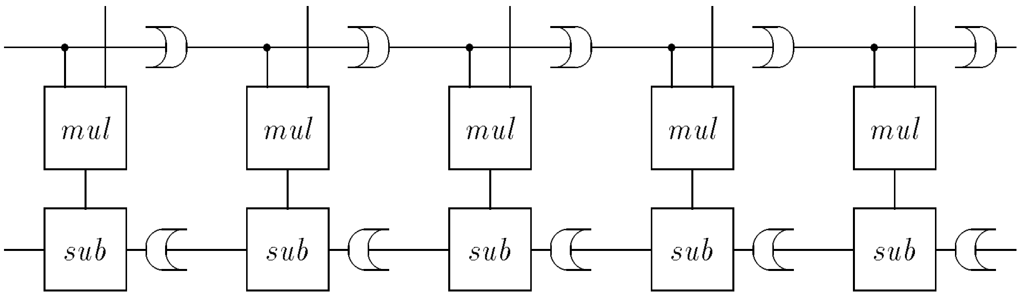

Figure 6.6: and instance of $\mathsf{rdl}(W' ; \pi_2 ; [\mathcal{D}^{-1}, \mathcal{D}])$

the left-hand side says that the data should be provided every clock cycle, with the weight r_i delayed by i cycles of the full-speed clock.

That completes the development of two word-level systolic correlators, one of which uses $3N$ latches, the other of which uses only N at the expense of a doubled clock speed.

6.5 Refining to a bit level implementation

If one of these correlators is to be implemented by a VLSI circuit we have to decide on an implementation of the arithmetic functions for which there exist realisations consisting of such things as transistors and pieces of metal. The next stage of the development is to refine the operations to ones that combine and produce bit-vectors; and there we will stop. The ultimate refinement of bit-level operations to electronics is less interesting to the programmer, and altogether more technology-specific than anything in these notes.

Earlier presentations of this circuit such as reference [Sheer88] have just made this refinement step in one leap, appealing to 'well-known' implementations of the arithmetic functions. However since we can express representation and abstraction relations in the same notation as the circuit, we have the opportunity to bridge the gap between word- and bit-level. This is work still in progress, and I have as yet no slick presentation of this refinement so this section does no more than outline the shape that the refinement would have.

The strategy for refining from word- to bit-level is to describe an abstraction relation that relates the bit-vectors that are to be operated on by the circuit to the numbers that they represent, to compose this relation with the word-level description of the correlator, and then to simplify the resulting expression. We will represent the numbers in the simplest binary representation, bit parallel with least significant bit first. If *bit* is the identity relation on bits, that is if it is the type of bits, and $bin = \mathsf{map}\ bit ; \mathsf{tri}(\times 2) ; acc$, then since $bin^{-1} ; bin = nat$ is the identity on natural numbers, bin is an abstraction from bit vectors to the natural numbers. Moreover, if $bin_n = n \,;\, bin$ then $bin_n \,;\, bin_n^{-1} = n \,;\, \mathsf{map}\, bit$ and $bin_n^{-1} \,;\, bin_n = nat_n$ where nat_n is the type of those numbers in the range $0 < i \leq 2^n$ representable in n bits, a sub-type of nat. So bin_n abstracts n-bit bit-vectors to small enough numbers, and bin_n^{-1} represents any small enough number as an n-bit bit-vector.

Suppose we are not going to want to give negative inputs to the correlator, then we will be happy with an implementation of

$$
\begin{aligned}
&[[\iota, nat], \mathsf{map}\, nat] \,;\, \mathsf{snd\,tri}\,\mathcal{D} \,;\, \mathsf{slow}\, CORR'' \\
&\quad = \quad [[\iota, nat], \mathsf{map}\, nat] \,;\, \mathsf{rdl}(W' \,;\, \pi_2 \,;\, [\mathcal{D}^{-1}, \mathcal{D}]) \,;\, \pi_1 \,;\, zero^{-1} \\
&\quad = \quad \mathsf{fst\,snd}\, nat \,;\, \mathsf{rdl}(\mathsf{snd}\, nat \,;\, W' \,;\, \pi_2 \,;\, [\mathcal{D}^{-1}, \mathcal{D}]) \,;\, \pi_1 \,;\, zero^{-1} \\
&\quad = \{\, \mathsf{fst\,snd}\, nat \,;\, \mathsf{snd}\, nat \,;\, W' = \mathsf{snd}\, nat \,;\, W' \,;\, \mathsf{snd\,snd}\, nat \,\} \\
&\qquad\quad \mathsf{rdl}(W' \,;\, \pi_2 \,;\, [\mathcal{D}^{-1}, \mathcal{D}]) \,;\, \mathsf{snd}\, nat \,;\, \pi_1 \,;\, zero^{-1} \\
&\quad = \{\, [[nat, nat], nat] \,;\, W' = \mathsf{snd}\, nat \,;\, W' \,;\, \mathsf{snd}[nat, nat] \,\} \\
&\qquad\quad \mathsf{rdl}(W'' \,;\, \pi_2 \,;\, [\mathcal{D}^{-1}, \mathcal{D}]) \,;\, \pi_1 \,;\, zero^{-1} \\
&\qquad\quad \text{where } W'' = [[nat, nat], nat] \,;\, W' \,;\, \mathsf{snd}[nat, nat] \\
&\quad = \quad [[nat, nat], \mathsf{map}\, nat] \,;\, \mathsf{snd\,tri}\,\mathcal{D} \,;\, \mathsf{slow}\, CORR''
\end{aligned}
$$

which is to say that the word-level cell W'' need only ever deal with non-negative numbers, and the result will also be non-negative.

The next thing to do is to decide on some appropriate widths for the bit-vectors. Since each of the component parts of W'' is going to become a column of cells, and since we will want to interleave these columns, it will be be simplest if we can arrange for everything to have the same width. This is a great simplification, but one that you need not make if you are determined to produce the smallest possible implementation of the circuit.

Consider for example the word-adder sub. By putting a bound on the size of the numbers that pass through each cell we can choose a width n which is big enough to represent each of the inputs, as well as the output. In the course of calculating the refinement of W'' we would get to

$$
\begin{aligned}
&\mathsf{snd}(nat_k \,;\, bin_n) \,;\, sub \,;\, (nat_l \,;\, bin_n)^{-1} \\
&\quad = \quad [nat_m, nat_k \,;\, bin_n] \,;\, sub \,;\, (nat_l \,;\, bin_n)^{-1} \\
&\qquad\quad \text{where } m > k \;\&\; m > l \\
&\quad = \quad [nat_m \,;\, bin_n^{-1}, nat_k] \,;\, [bin_n, bin_n] \,;\, sub \,;\, bin_n^{-1} \,;\, nat_l^{-1} \\
&\quad = \quad [nat_m\, bin_n^{-1}, nat_k] \,;\, [n \,;\, \mathsf{map}\, bit, n \,;\, \mathsf{map}\, bit] \,;\\
&\qquad\qquad [\mathsf{tri}(\times 2) \,;\, acc, \mathsf{tri}(\times 2) \,;\, acc] \,;\, sub \,;\, acc^{-1} \,;\, \mathsf{tri}(\times 2)^{-1} \,;\\
&\qquad\qquad\quad n \,;\, \mathsf{map}\, bit \,;\, nat_l^{-1}
\end{aligned}
$$

all provided n is at least as big as m. Reference [Jones90a] reasons that relations like the middle term here can be implemented by a column of cells performing calculations on single bits, and in this case given sufficient patience we could show that

$$\begin{array}{l}[n\ ;\ \mathsf{map}\ bit, n\ ;\ \mathsf{map}\ bit]\ ;\\ \quad [\mathsf{tri}(\times 2)\ ;\ acc, \mathsf{tri}(\times 2)\ ;\ acc]\ ;\ sub\ ;\ acc^{-1}\ ;\ \mathsf{tri}(\times 2)^{-1}\ ;\\ \qquad n\ ;\ \mathsf{map}\ bit\\ \qquad\ = \ [n, n]\ ;\ \mathsf{zip}\ ;\ {\pi_1}^{-1}\ ;\ \mathsf{col}\ FS\ ;\ \mathsf{fst}\ zero^{-1}\ ;\ \pi_2\ ;\ n\\ \qquad\quad \text{where } FS = [[bit, bit], bit]\ ;\ [sub, (\times 2)^{-1}]\ ;\ add\ ;\ add^{-1}\ ;\ [bit, bit]\end{array}$$

but the component FS is just a full adder in an unexpected orientation, so we would be done with the refinement of the word-level sub because FS is a bit-level operation.

We could do the same for the multiply: the circuit in reference [McCabe82] makes the multiplier particularly simple to implement because the reference inputs r_i are only a single bit wide. This means that the multiplier appears in a context something like

$$\begin{array}{rcl}[bin_n, bit]\ ;\ mul & = & [bin_n, bit]\ ;\ mul\ ;\ bin_n^{-1}\ ;\ bin_n\\ & = & \mathsf{col}\ M\ ;\ \pi_2\ ;\ bin_n\\ & & \text{where } M = [bit, bit]\ ;\ \mathsf{zip}\ ;\ [\pi_2, mul]\ ;\ [bit, bit]\end{array}$$

and again M is a bit-level component. The mul in its definition is $[bit, bit]\,;mul\,;bit$ which can be implemented by the same electronics as you would use to implement a logical 'and'.

Driving the representation relations through the cell in detail would eventually result in a refined cell something like that in figure 6.7 which illustrates an instance of $[[bin_n, bin_n], bit]\,;W''\,;\mathsf{snd}[bin_n, bin_n]^{-1}$. The columns could then be interleaved giving

$$\begin{array}{l}[[bin_n, bin_n], bit]\ ;\ W''\ ;\ \pi_2\ ;\ [bin_n, bin_n]^{-1}\\ \qquad = \ \mathsf{fst}(\mathsf{zip}\ ;\ n)\ ;\ (\mathsf{snd}\ {\pi_1}^{-1}\ ;\ \mathsf{col}\ B\ ;\ \mathsf{fst}\,\mathsf{snd}\ zero^{-1})\ ;\ \pi_2\ ;\ (\mathsf{zip}\ ;\ n)^{-1}\end{array}$$

where B is a bit-level relation. Consequently for sufficiently large n, which in this case means for $n \geq N + k$,

$$\begin{array}{rl}\multicolumn{2}{l}{[[\iota, nat_k], N\ ;\ \mathsf{map}\ nat_1]\ ;\ \mathsf{snd}\,\mathsf{tri}\,\mathcal{D}\ ;\ \mathsf{slow}\ CORR''}\\ = & [[nat_n, nat_k], N\ ;\ \mathsf{map}\ nat_1]\ ;\ \mathsf{snd}\,\mathsf{tri}\,\mathcal{D}\ ;\ \mathsf{slow}\ CORR''\\ = & \mathsf{fst}[bin_n, bin_n]^{-1}\ ;\\ & \mathsf{rdl}([[bin_n, bin_n], bit]\ ;\ W''\ ;\ \pi_2\ ;\ [bin_n, bin_n]^{-1}\ ;\ [D^{-1}, \mathcal{D}])\ ;\\ & [bin_n, bin_n]\ ;\ \pi_1\ ;\ zero^{-1}\end{array}$$

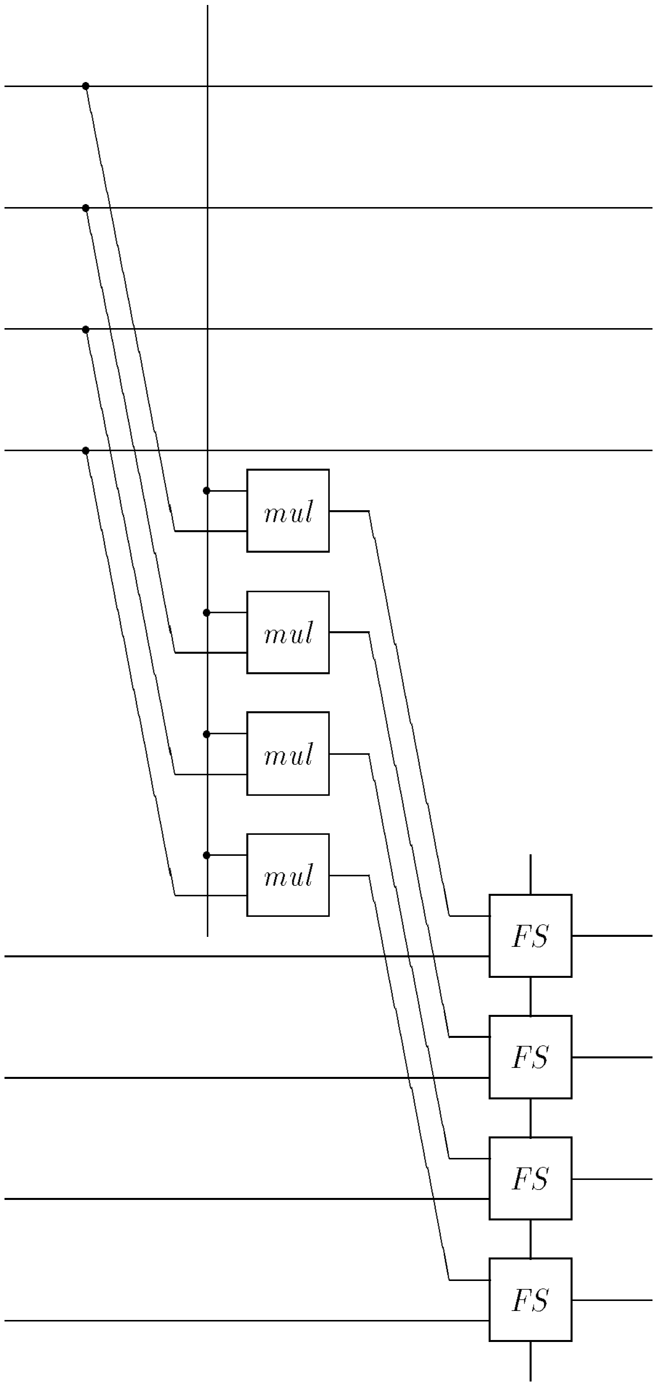

Figure 6.7: $[[bin_4, bin_4], bit]\,; W''\,; \mathsf{snd}[bin_4, bin_4]^{-1}$ implemented by an interleaving of separate columns

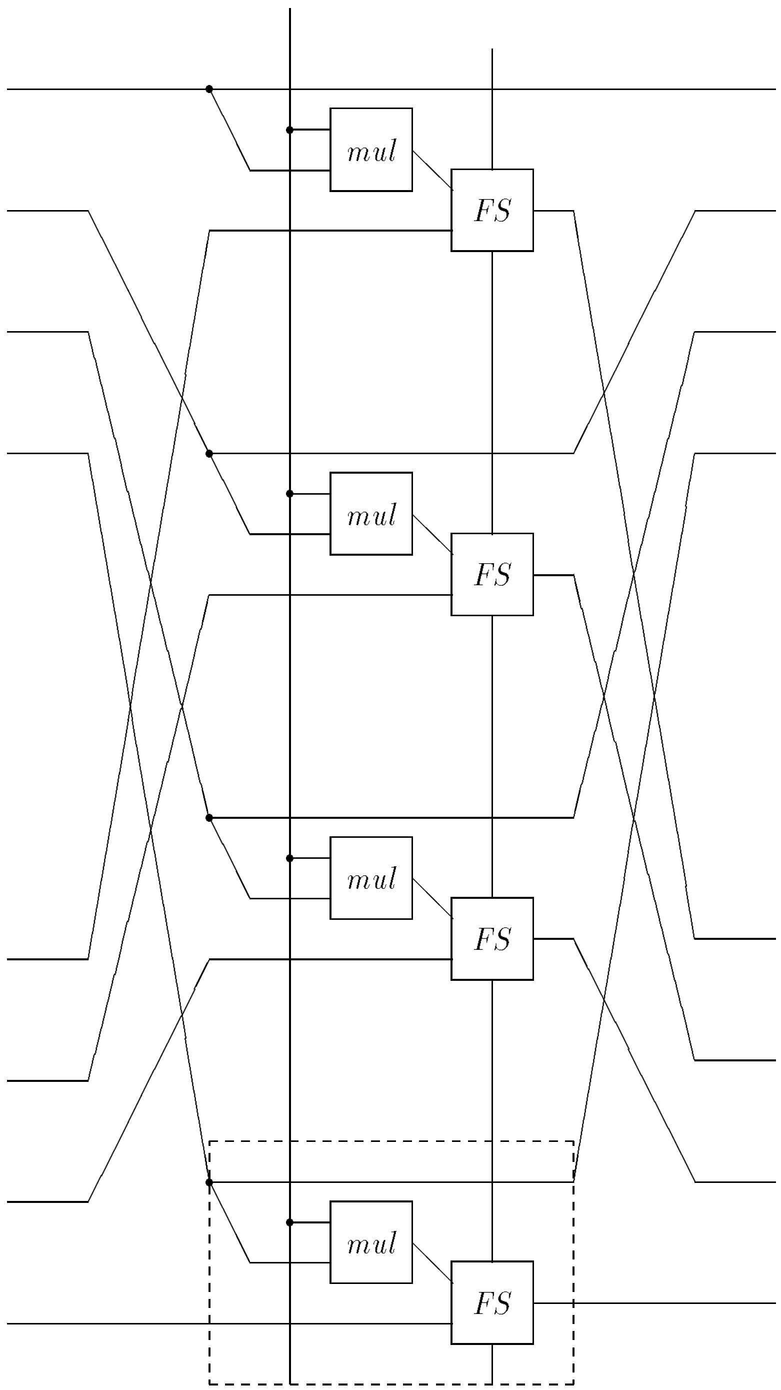

Figure 6.8: $[[bin_4, bin_4], bit]\,; W''; \mathsf{snd}[bin_4, bin_4]^{-1}$ implemented by a single column of interleaved components

$$
\begin{aligned}
&= \mathsf{fst}[bin_n, bin_n]^{-1} \,; \\
&\quad \mathsf{rdl}([\mathsf{zip} \,;\, n, {\pi_1}^{-1}] \,;\, \mathsf{col}\, B \,;\, \mathsf{fst}\,\mathsf{snd}\, zero^{-1} \,;\, \pi_2 \,;\, (\mathsf{zip} \,;\, n)^{-1} \,;\, [D^{-1}, \mathcal{D}]) \,; \\
&\quad [bin_n, bin_n] \,;\, \pi_1 \,;\, zero^{-1} \\
&= [[bin_n, bin_n]^{-1} \,;\, \mathsf{zip}, \mathsf{map}\, {\pi_1}^{-1}] \,; \\
&\quad \mathsf{rdl}(\mathsf{col}\, B \,;\, \mathsf{fst}\,\mathsf{snd}\, zero^{-1} \,;\, \pi_2 \,;\, \mathsf{map}[D^{-1}, \mathcal{D}]) \,; \\
&\quad \mathsf{zip}^{-1} \,;\, [bin_n, bin_n] \,;\, \pi_1 \,;\, zero^{-1} \\
&= [[bin_n, bin_n]^{-1} \,;\, \mathsf{zip}, \mathsf{map}\, {\pi_1}^{-1}] \,; \\
&\quad \mathsf{row}(\mathsf{col}(B \,;\, \mathsf{snd}[D^{-1}, \mathcal{D}])) \,; \\
&\quad [\mathsf{map}\,\mathsf{snd}\, zero^{-1}, \mathsf{map}\,\mathsf{fst}\, zero^{-1}] \,;\, any
\end{aligned}
$$

where any is relates anything to anything else. We have used facts like $zero = any \,;\, zero$ and $zero \,;\, bin_n = any \,;\, zero \,;\, bin_n = any \,;\, \mathsf{map}\, zero \,;\, n$.

The middle term on the right-hand side is just an N by n grid of identical cells, and this part would certainly be implemented as part of the circuit. The final term describes the tying of certain lines to a level representing the bit zero, and the discarding of the signals on other lines. The $[bin_n, bin_n]^{-1} \,;\, \mathsf{zip}$ in the first term though is different: this is what remains of the representation relation, and we think of it as describing way the input must be represented to the circuit.

6.6 Making the implementation systolic at the bit level

The circuit represented by $G = \mathsf{row}(\mathsf{col}(B \,;\, \mathsf{snd}[D^{-1}, \mathcal{D}]))$, illustrated in figure 6.9, has no long horizontal combinational paths because of the $[D^{-1}, \mathcal{D}]$ through which each horizontal data-path passes at each cell. Moreover since the change of representation that we used did not involve any change in the direction of the data-flow, both the delays and anti-delays in the circuit remain implementable as latches.

There are however vertical unbroken combinational paths through each column of cells. In figure 6.8 these are the long vertical line that carries the reference bit, and a route through the carry-path that joins the FS components. We might try to eliminate these long paths by skewing. The appropriate skewing law for columns is reminiscent of Horner's rule for right reductions:

$$(\mathsf{apr} \,/\, \mathsf{tri}\, \mathcal{D}) \,;\, \mathsf{col}\, R \,;\, \mathsf{snd}\,\mathsf{tri}\, \mathcal{D}^{-1} = \mathsf{col}(\mathsf{snd}\, \mathcal{D} \,;\, R)$$

and although we could skew each column in turn, it is simplest to exchange the row and column structure

$$
\begin{aligned}
&(\mathsf{apr} \,/\, \mathsf{tri}\, \mathcal{D}) \,;\, G \,;\, \mathsf{snd}\,\mathsf{tri}\, \mathcal{D}^{-1} \\
&\qquad = \quad (\mathsf{apr} \,/\, \mathsf{tri}\, \mathcal{D}) \,;\, \mathsf{row}\,\mathsf{col}(B \,;\, \mathsf{snd}[D^{-1}, \mathcal{D}]) \,;\, \mathsf{snd}\,\mathsf{tri}\, \mathcal{D}^{-1}
\end{aligned}
$$

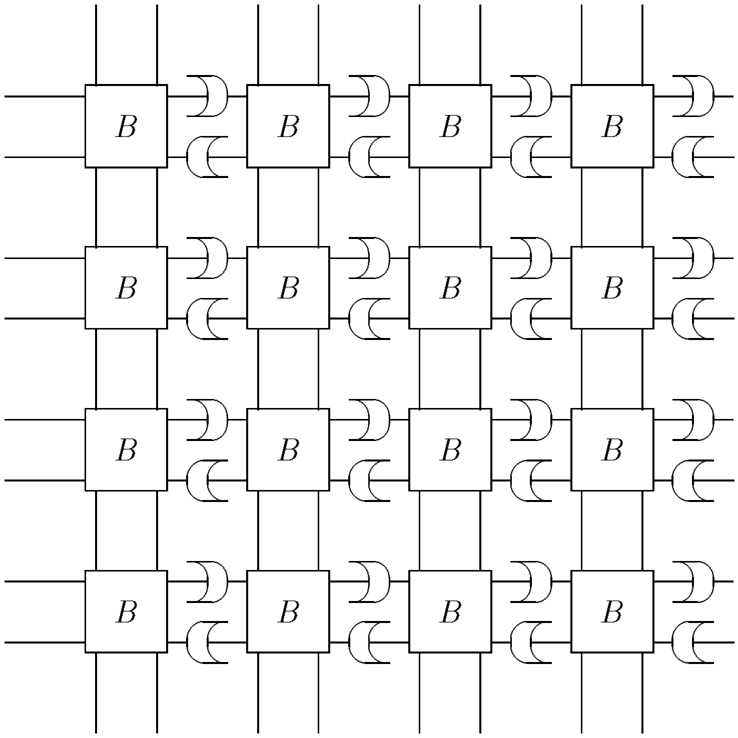

Figure 6.9: an instance of $\mathsf{row\,col}(B\,;\mathsf{snd}[D^{-1},\mathcal{D}])$

$$
\begin{aligned}
&= (\mathsf{apr}\ /\ \mathsf{tri}\,\mathcal{D})\,;\mathsf{col\,row}(B\,;\mathsf{snd}[D^{-1},\mathcal{D}])\,;\mathsf{snd\,tri}\,\mathcal{D}^{-1}\\
&= \mathsf{col}(\mathsf{snd}\,\mathcal{D}\,;\mathsf{row}(B\,;\mathsf{snd}[D^{-1},\mathcal{D}]))\\
&= \mathsf{col\,row}(\mathsf{snd}[\mathcal{D},\mathcal{D}]\,;B\,;\mathsf{snd}[D^{-1},\mathcal{D}])
\end{aligned}
$$

The trouble with this is that one of the delays introduced in each cell (the second in the expression) is going to be unimplementable because it predicts the carry signal. If we skew in the other direction, as illustrated in figure 6.10,

$$
\begin{aligned}
&(\mathsf{apr}\ /\ \mathsf{tri}\,\mathcal{D}^{-1})\,;G\,;\mathsf{snd\,tri}\,\mathcal{D}\\
&\qquad = \mathsf{col\,row}(\mathsf{snd}[\mathcal{D}^{-1},\mathcal{D}^{-1}]\,;B\,;\mathsf{snd}[D^{-1},\mathcal{D}])
\end{aligned}
$$

the first $\mathcal{D}^{-1}$ causes the same trouble by predicting the reference bit. The fix which we used for the horizontal pipelining does not work this time: one line has to be predicted by as much as the other is delayed, and slowing the circuit makes no difference to this balance.

Of course there is a circuit something like the one we have designed but which does perform essentially the right calculation. This is easily seen informally, because the path that distributes the reference bit through each column could equally well be driven from the bottom. That would amount to moving the

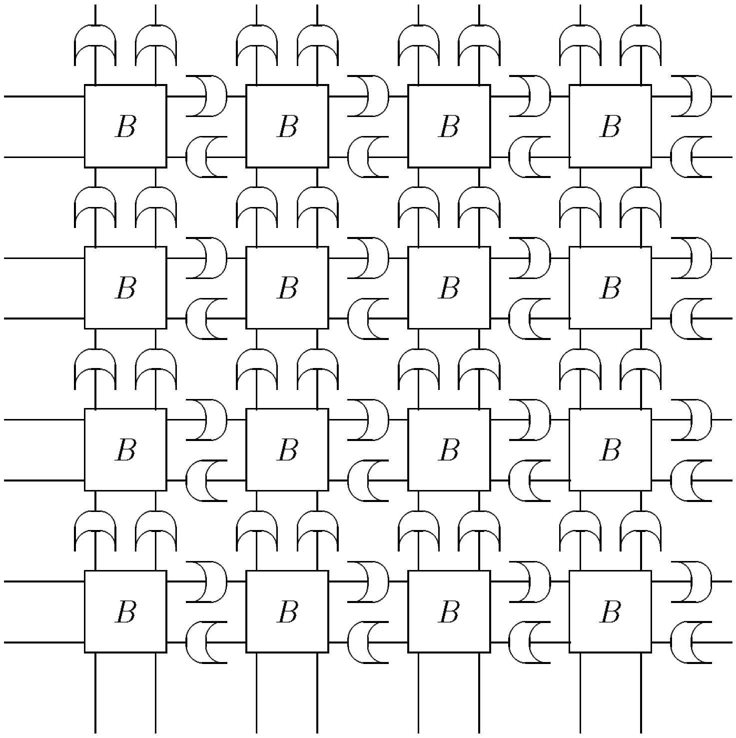

Figure 6.10: an instance of $\mathsf{row}\,\mathsf{col}(\mathsf{snd}[\mathcal{D}^{-1}, \mathcal{D}^{-1}] \,;\, B \,;\, \mathsf{snd}[D^{-1}, \mathcal{D}])$

reference input of the correlator from the second component of the domain into the first component of the range. In that case $\mathsf{snd}[\mathcal{D}^{-1}, \mathcal{D}^{-1}] \,; B \,; \mathsf{snd}[D^{-1}, \mathcal{D}]$ would be a cell with four implementable latches.

Unfortunately the formality corresponding to this late change of mind is rather more difficult: in order to change the position of the reference bit inputs we would have to go back at least as far as the decision to work on $CORR''$, and introduce at this point instead a circuit $CORR'''$ for which $\langle c, d\rangle \; CORR''' \; r$. The translation of the existing design into the new one does not however seem to fit very neatly into the formalism.

In fact having made the mistake of choosing to supply the reference input in the domain, the formalism makes it easier to recover by retreating only as far as the choice of a representation for the numbers. If instead of choosing bin_n we had used $\mathsf{rev} \,;\, bin_n$, which is to say that numbers on horizontal buses have least-significant bits at the top of the picture, then the refinement of sub would have a carry-path from the domain into the range.

Making this change would involve repeating the development from the beginning of §6.5, through essentially the same calculation. The result would be a different bit-level cell B', with different boundary conditions for the grid, but for which $G' = \mathsf{col\,row}(\mathsf{snd}[\mathcal{D},\mathcal{D}]\;;B'\;;\mathsf{snd}[D^{-1},\mathcal{D}])$ would be implementable. In fact if $rep = \mathsf{rev}\;;bin_n$ then

$$\begin{aligned}
&\mathsf{snd\,map}({\pi_1}^{-1}\;;\mathsf{snd}\;zero)\;;G'\;;\mathsf{snd\,map\,fst}\;zero^{-1}\;;any\\
&\quad= \mathsf{apr}\;/\;\mathsf{tri}\,\mathcal{D}\;;[\mathsf{zip}^{-1}\;;[rep,rep],N\;;\mathsf{map}\;nat_1]\;;\mathsf{snd\,tri}\,\mathcal{D}\;;\mathsf{slow}\;CORR''\\
&\quad= [\mathsf{tri}\,\mathcal{D},D^n]\;;[\mathsf{zip}^{-1}\;;[rep,rep],\mathsf{tri}\,\mathcal{D}\;;N\;;\mathsf{map}\;nat_1]\;;\mathsf{slow}\;CORR''\\
&\quad= [\mathsf{zip}^{-1}\;;[\mathsf{tri}\,\mathcal{D}\;;rep,\mathsf{tri}\,\mathcal{D}\;;rep],\mathsf{map}\,\mathcal{D}^n\;;\mathsf{tri}\,\mathcal{D}\;;N\;;\mathsf{map}\;nat_1]\;;\mathsf{slow}\;CORR''
\end{aligned}$$

where B' is a combinational bit-level cell containing only an 'and' gate and a full adder and few wires.

The right hand side of this equation is a precise and fairly compact description of the complex interfacing requirements of the circuit on the left:

- zip^{-1} means that the d input and the output are interleaved in space;
- $\mathsf{tri}\,\mathcal{D}\;;rep$ means that they are each represented in n-bit binary with most significant bit on the left, skewed by one bit per clock so that the least significant bit of d has to be supplied early and the least significant bit of output emerges early;
- $\mathsf{map}\,\mathcal{D}^n\;;\mathsf{tri}\,\mathcal{D}\;;N\;;\mathsf{map}\;nat_1$ means that the N one-bit reference streams have to be supplied with the first being n clocks early, and skewed so that each higher-numbered one is one clock earlier still;
- slow says that the clock needs to be twice as fast as the data-rate.

Had we only succeeded in finding a way of writing such specifications precisely and compactly, that in itself would be an achievement.

6.7 Making other design decisions

We have produced a circuit which gives exactly the specified answer when provided with any k-bit wide data stream and N one-bit wide reference streams, but it uses very much wider words than are necessary. The development would in fact be easier if we had decided early to aim to get the answer right modulo 2^n for some n, and then chosen n to be large enough after completing the design. We could have arrived at this smaller circuit by choosing a different abstraction relation, one which related each number to the n-bit binary representation of its residue modulo 2^n.

We could also consider widening the reference streams. In fact the only uses of the width of the reference stream were those in calculating the sizes of the buses, and in deciding that the components of the final circuit operated only on bits.

If we were to widen the reference values, the refinement of the multiplier which we made would deliver its result as a number of wider values of different weights, which we can think of as a representation of the answer in a redundant number system. Moreover the adder has now to deal with values which are not single bits. That means that there is at least one more refinement step before we can claim to have a bit level design, and a further skewing step before we have a bit-level systolic circuit.

Whichever way we proceed there seems to be nothing, other than the experience of getting it wrong, to guide the early design decisions to avoid the unfortunate choices which we made. That puts a premium on reducing the cost of making an unfortunate choice and finding oneself at a point in the maze where there is a clearly a better circuit just the other side of the hedge, but which is not cheaply accessible from here. The cost could be reduced either by reducing the amount of work necessary to perform a development, by mechanising the tedious parts; or perhaps by finding better ways of looking ahead to get an idea of where each of the various possible decisions may lead.

Chapter 7

The skewing theorem

The principle of this way of going about design is to hide the inductions. We work with circuits of an arbitrary size, and so have to do proofs by induction to show that our transformations are sound for all sizes of circuit. Nevertheless we want to be able to generate a design by purely equational reasoning. So as to do that we need a large repertoire of pre-packaged inductions each of which can be invoked as a single rewriting step. Most of the laws that have appeared throughout these notes – all of the ones about circuits of arbitrary size – are just such pre-packaged inductions.

An example is the statement of Horner's rule which has haunted these pages, and of which we gave a proof by induction in §4.6. Each application of Horner's rule is just a single equational rewriting step, but it has the power of the induction with which it was proved. It is as though we had done a proof by induction to justify the step: the precondition of Horner's rule is the proof obligation that we would have to meet; but we do not have to find the induction hypothesis, or repeat the proof.

As well as Horner's rule we have used other theorems about rows and triangles, and there are similar rules about columns and triangles and so on. We need a large collection of these rules to be sure that we will have the right one to hand when the need arises in the course of a design. Fortunately this does not mean having to do a vast number of tedious proofs by induction before being able to start on the real work of a design. Many of the the rules will turn out to be simple duals of others: for example if $[R, R] \,;\, S = S \,;\, [R, R]$ it is the case that $\mathsf{snd}\,\mathsf{tri}\,R \,;\, \mathsf{row}(S \,;\, \mathsf{snd}\,R) = \mathsf{row}\,S \,;\, (\mathsf{apr} \,/\, \mathsf{tri}\,R)$ and it follows immediately by taking inverses throughout that if $[R, R] \,;\, S = S \,;\, [R, R]$ it is also the case that $(\mathsf{apr} \,/\, \mathsf{tri}\,R) \,;\, \mathsf{col}\,S = \mathsf{col}(\mathsf{snd}\,R \,;\, S) \,;\, \mathsf{snd}\,\mathsf{tri}\,R$.

Moreover there is a basic similarity between all the rules about rows and columns and triangles: since the proofs have to be inductions over the same thing, all the proofs look much the same and all the results look correspondingly similar.

The *skewing theorem* is a formalisation of this similarity. It is a single result which generalises a large family of results about rows and triangles. There is a cost for the generality of the result which is that its proof is a bit daunting, indeed the statement alone is rather daunting and it does not seem to have much immediate relation to any of the circuits we have wanted to design. However the benefit which compensates for this cost is that a large number of useful theorems are substitution instances of this one result. Indeed some quite surprising results that on the face of it have nothing to do with skewing are immediate consequences of the skewing theorem.

7.1 Some new notation

In order to express the theorem we will need a few new named constants with slightly unfamiliar properties. These are all related to appending of lists, and the skewing theorem is really about flattening the structure out of two different representations of the same relation. Because it appears so often, we will name the reduction of app, which because of the associativity of the binary operation can be written as either a left or right reduction, $append = {\pi_2}^{-1} \,;\, \mathsf{fst}\, 0 \,;\, \mathsf{rdl}\,\mathsf{app} = {\pi_1}^{-1} \,;\, \mathsf{snd}\, 0 \,;\, \mathsf{rdr}\,\mathsf{app}$.

The halving relation *halve* relates any even-length list to a pair of lists of equal length which when appended give the original list, so is a subrelation of app^{-1}. Similarly, the pairing relation *pair* relates any even-length list to a list of pairs which when appended give the original list, so is also a subrelation of app^{-1}. In fact $pair = \mathsf{app}^{-1} \,;\, \mathsf{map}\, 2$ and less obviously $halve = \mathsf{app}^{-1} \,;\, (\mathsf{trn} \,/\, \mathsf{map}\, 2) = \mathsf{app}^{-1} \,;\, \mathsf{zip} \,;\, \mathsf{zip}^{-1}$.

Finally, the peeling relation $peel = \mathsf{app}^{-1} \,;\, [[\text{-}]^{-1}, \iota, [\text{-}]^{-1}]$ relates a list of at least two elements to a triple containing the first element, the body of the list, and the final element of the list, so it too is a subrelation of app^{-1}. This last one is neither primitive nor new: we have met it before in the proving the abiding property of below and beside; for $mid = halve^{-1} \,;\, 4 \,;\, peel$ and more generally $peel = \mathsf{apl}^{-1} \,;\, \mathsf{snd}\,\mathsf{apr}^{-1} \,;\, \mathsf{apl} = \mathsf{apr}^{-1} \,;\, \mathsf{fst}\,\mathsf{apl}^{-1} \,;\, \mathsf{apr}$. Indeed one of the best reasons for giving it a name is that the route from one of these expressions to the other through the associativity of app is very tedious, especially since our notation is clumsy here.

When we wanted to do inductions over the structure of all lists by taking an element from one end of the list and a shorter list from the other, these appeared as calculations involving apl and apr. Since any even-length list is either empty or exactly two elements longer than another shorter even-length list, the corresponding calculations about even-length lists can be done by picking off elements two at a time. In the case of the skewing theorem we do this symmetrically, removing an element from each end, and represent this by calculating with *peel*. For example,

$peel^{-1}$; $halve$ = $[\iota, halve, \iota]$; mid^{-1} ; $[\mathsf{apl}, \mathsf{apr}]$ translates a large instance of $halve$ into a smaller one in exactly the same way that $\mathsf{apl} ; \mathsf{tri}\, R = [R^0, \mathsf{map}\, R ; \mathsf{tri}\, R] ; \mathsf{apl}$ decomposes a large triangle into a smaller one. In fact, putting these two sorts of decomposition together

$$
\begin{aligned}
& peel^{-1} ; halve ; [\mathsf{tri}\, R, \mathsf{rev} \,/\, \mathsf{tri}\, R] \\
&\quad = [\iota, halve, \iota] ; mid^{-1} ; [\mathsf{apl}, \mathsf{apr}] ; [\mathsf{tri}\, R, \mathsf{rev} \,/\, \mathsf{tri}\, R] \\
&\quad = [\iota, halve, \iota] ; mid^{-1} ; [\mathsf{apl} ; \mathsf{tri}\, R, \mathsf{apr} ; \mathsf{rev} \,/\, \mathsf{tri}\, R] \\
&\quad = [\iota, halve, \iota] ; mid^{-1} ; [[R^0, \mathsf{map}\, R ; \mathsf{tri}\, R] ; \mathsf{apl}, [\mathsf{map}\, R ; \mathsf{rev} \,/\, \mathsf{tri}\, R, R^0] ; \mathsf{apr}] \\
&\quad = [\iota, halve, \iota] ; [R^0, [\mathsf{map}\, R ; \mathsf{tri}\, R, \mathsf{map}\, R ; \mathsf{rev} \,/\, \mathsf{tri}\, R], R^0] ; mid^{-1} ; [\mathsf{apl}, \mathsf{apr}] \\
&\quad = [R^0, \mathsf{map}\, R, R^0] ; [\iota, halve ; [\mathsf{tri}\, R, \mathsf{rev} \,/\, \mathsf{tri}\, R], \iota] ; mid^{-1} ; [\mathsf{apl}, \mathsf{apr}]
\end{aligned}
$$

gives a decomposition of circuits like $halve ; [\mathsf{tri}\, R, \mathsf{rev} \,/\, \mathsf{tri}\, R]$ into smaller instances of the same. This decomposition appears in the proof of the skewing theorem itself.

7.2 A very general theorem

The *general skewing theorem* states that if there are four numbers q, q', r, r', at least one of which is non-zero, for which $Q : \mathsf{snd}\, q \sim \mathsf{fst}\, q'$ and $R : \mathsf{snd}\, r \sim \mathsf{fst}\, r'$ and if

$$
\begin{aligned}
& \mathsf{snd}\, \mathsf{app}^{-1} ; R \leftrightarrow Q ; \mathsf{fst}\, \mathsf{app} \\
&\qquad = \mathsf{snd}(\mathsf{map}\, P ; \mathsf{app}^{-1}) ; Q \leftrightarrow R ; \mathsf{fst}(\mathsf{app} ; \mathsf{map}\, S)
\end{aligned}
$$

then

$$
\begin{aligned}
& \mathsf{snd}(append^{-1} ; pair) ; \mathsf{row}(Q \leftrightarrow R) ; \mathsf{fst}(pair^{-1} ; append) \\
&\qquad = \mathsf{snd}(append^{-1} ; halve) ; \\
&\qquad\qquad (\mathsf{snd}\, \mathsf{tri}\, \mathsf{map}\, P ; \mathsf{row}\, Q ; \mathsf{fst}\, \mathsf{tri}\, \mathsf{map}\, S) \leftrightarrow \\
&\qquad\qquad\quad (\mathsf{snd}(\mathsf{rev} \,/\, \mathsf{tri}\, \mathsf{map}\, P) ; \mathsf{row}\, R ; \mathsf{fst}(\mathsf{rev} \,/\, \mathsf{tri}\, \mathsf{map}\, S)) ; \\
&\qquad\qquad \mathsf{fst}(halve^{-1} ; append)
\end{aligned}
$$

which is, by two applications of interface rules for beside, the same as saying that if

$$
\begin{aligned}
& \mathsf{snd}\, \mathsf{app}^{-1} ; R \leftrightarrow Q ; \mathsf{fst}\, \mathsf{app} \\
&\qquad = \mathsf{snd}\, \mathsf{app}^{-1} ; \\
&\qquad\qquad (\mathsf{snd}\, \mathsf{map}\, P ; Q ; \mathsf{fst}\, \mathsf{map}\, S) \leftrightarrow (\mathsf{snd}\, \mathsf{map}\, P ; R ; \mathsf{fst}\, \mathsf{map}\, S) ; \\
&\qquad\qquad\quad \mathsf{fst}\, \mathsf{app}
\end{aligned}
$$

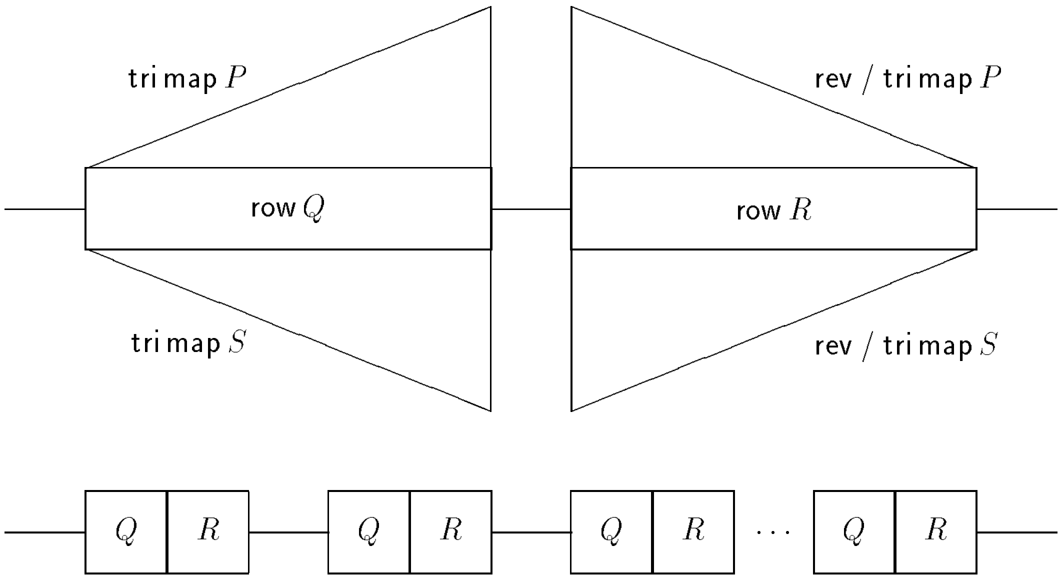

Figure 7.1: sketches of $(\mathsf{snd\,tri\,map}\,P\,;\mathsf{row}\,Q\,;\mathsf{fst\,tri\,map}\,S) \leftrightarrow (\mathsf{snd}(\mathsf{rev}\,/\,\mathsf{tri\,map}\,P)\,;\mathsf{row}\,R\,;\mathsf{fst}(\mathsf{rev}\,/\,\mathsf{tri\,map}\,S))$ and $\mathsf{row}(Q \leftrightarrow R)$ which are, under the hypotheses of the skewing theorem, equal up to regrouping of wires

then

$$\begin{array}{rl} & \mathsf{snd}(append^{-1}\,;pair)\,;\mathsf{row}(Q \leftrightarrow R)\,;\mathsf{fst}(pair^{-1}\,;append) \\ = & \mathsf{snd}(append^{-1}\,;halve\,;[\mathsf{tri\,map}\,P, \mathsf{rev}\,/\,\mathsf{tri\,map}\,P])\,; \\ & \quad \mathsf{row}\,Q \leftrightarrow \mathsf{row}\,R\,; \\ & \qquad \mathsf{fst}([\mathsf{tri\,map}\,S, \mathsf{rev}\,/\,\mathsf{tri\,map}\,S]\,;halve^{-1}\,;append) \end{array}$$

making it more obvious that the decomposition of pairs of back-to-back triangles is relevant. Having once seen the decomposition of the triangles on the right-hand side the shape of the proof follows, and all that remains is to plough through the detail in a more-or-less mechanical fashion.

How much of the detail you need in order to be convinced is a matter of taste. Essentially the existence of the four numbers that are widths of the relevant interfaces of Q and R mean that the instances of $append^{-1}$ and $append$ in the theorem are bijective, for example in the statement that

$$\begin{array}{rl} & \mathsf{snd}(append^{-1}\,;pair)\,;\mathsf{row}(Q \leftrightarrow R)\,;\mathsf{fst}(pair^{-1}\,;append) \\ = & \mathsf{snd}(append^{-1}\,;halve\,;[\mathsf{tri\,map}\,P, \mathsf{rev}\,/\,\mathsf{tri\,map}\,P])\,; \\ & \quad \mathsf{row}\,Q \leftrightarrow \mathsf{row}\,R\,; \\ & \qquad \mathsf{fst}([\mathsf{tri\,map}\,S, \mathsf{rev}\,/\,\mathsf{tri\,map}\,S]\,;halve^{-1}\,;append) \end{array}$$

the $append^{-1}$ on the left-hand side is $append^{-1}\,;(pair\,/\,[q,r])$ and that on the right-hand side is $append^{-1} ; (halve \mathbin{/} [\mathsf{map}\, q, \mathsf{map}\, r])$. This bijectivity is enough to let us reason that

$$\begin{aligned} & append^{-1} ; (halve \mathbin{/} [\mathsf{map}\, q, \mathsf{map}\, r]) ; peel \\ & \quad = \; append^{-1} ; peel ; [q, halve \mathbin{/} [\mathsf{map}\, q, \mathsf{map}\, r], r] \\ & \quad = \; append^{-1} ; [q, \iota, r] ; [\iota, append^{-1} ; (halve \mathbin{/} [\mathsf{map}\, q, \mathsf{map}\, r]), \iota] \end{aligned}$$

and similarly that

$$\begin{aligned} & append^{-1} ; (pair \mathbin{/} [q, r]) ; peel \\ & \quad = \; append^{-1} ; [q, \iota, r] ; [\iota, append^{-1} ; (pair \mathbin{/} [r, q]), \iota] \end{aligned}$$

and that means that you can use a smaller instance of the theorem as the inductive hypothesis confident that the decomposition is the same on both sides of the equation. That at least one of the four numbers is non-zero is enough to guarantee that the induction is well-founded.

The hypothesis that relates instances of $R \leftrightarrow Q$ to $Q \leftrightarrow R$ allows us to reason that $\mathsf{snd\, map\, map}\, P ; \mathsf{row}(\mathsf{snd\, app}^{-1} ; Q \leftrightarrow R ; \mathsf{fst\, app}) ; \mathsf{fst\, map\, map}\, S$ is the same as $\mathsf{row}(\mathsf{snd\, app}^{-1} ; R \leftrightarrow Q ; \mathsf{fst\, app})$. A separate induction shows that the parity of $\mathsf{row}(R \leftrightarrow Q)$ (without the app terms) can be changed by adding a Q at the left and an R at the right. Eventually everything else reduces to associativity of the appending of lists.

That I have not actually done this proof because of its length, despite believing it to be relatively straightforward, is the best sort of argument I know for mechanical assistance with such proofs. Although there may be some leverage to be gained from a better notation, or perhaps from identifying some crucial lemmas, there seems bound to be a lot of writing in any complete proof of this theorem.

7.3 Instances of the skewing theorem

The skewing theorem itself has no obvious connection with the sorts of circuits we want to describe, but a number of the rules we have given earlier in these notes turn out to be substitution instances of it. Suppose for example that $Q = \mathsf{snd}\, [\text{-}]^{-1} ; Q' ; \mathsf{fst}\, [\text{-}]$ and $R = \mathsf{snd}\, [\text{-}]^{-1} ; R' ; \mathsf{fst}\, [\text{-}]$ then certainly the four numbers exists, for they are all equal to one. If moreover $[P, P] ; (Q' \leftrightarrow R') ; [S, S] = R' \leftrightarrow Q'$ then it follows that

$$\begin{aligned} & \mathsf{snd\, app}^{-1} ; R \leftrightarrow Q ; \mathsf{fst\, app} \\ & \quad = \; \mathsf{snd\, app}^{-1} ; (\mathsf{snd}\, [\text{-}]^{-1} ; R' ; \mathsf{fst}\, [\text{-}]) \leftrightarrow (\mathsf{snd}\, [\text{-}]^{-1} ; Q' ; \mathsf{fst}\, [\text{-}]) ; \mathsf{fst\, app} \\ & \quad = \; \mathsf{snd}(\mathsf{app}^{-1} ; [[\text{-}]^{-1}, [\text{-}]^{-1}]) ; R' \leftrightarrow Q' ; \mathsf{fst}([[\text{-}], [\text{-}]] ; \mathsf{app}) \\ & \quad = \; \mathsf{snd}(\mathsf{app}^{-1} ; [[\text{-}]^{-1}, [\text{-}]^{-1}]) ; [P, P] ; (Q' \leftrightarrow R') ; [S, S] ; \mathsf{fst}([[\text{-}], [\text{-}]] ; \mathsf{app}) \end{aligned}$$

$$
\begin{aligned}
= \;& \mathsf{snd}(\mathsf{app}^{-1} \,;\, [\mathsf{map}\, P, \mathsf{map}\, P]) \,; \\
& \quad (\mathsf{snd}\, [\text{-}]^{-1} \,;\, R' \,;\, \mathsf{fst}\, [\text{-}]) \leftrightarrow (\mathsf{snd}\, [\text{-}]^{-1} \,;\, Q' \,;\, \mathsf{fst}\, [\text{-}]) \,; \\
& \qquad \mathsf{fst}([\mathsf{map}\, S, \mathsf{map}\, S] \,;\, \mathsf{app}) \\
= \;& \mathsf{snd}(\mathsf{map}\, P \,;\, \mathsf{app}^{-1}) \,;\, Q \leftrightarrow R \,;\, \mathsf{fst}(\mathsf{app} \,;\, \mathsf{map}\, S)
\end{aligned}
$$

so the hypotheses of the skewing theorem are satisfied, and it follows that

$$
\begin{aligned}
& \mathsf{snd}(append^{-1} \,;\, pair) \,;\, \mathsf{row}(Q \leftrightarrow R) \,;\, \mathsf{fst}(pair^{-1} \,;\, append) \\
& \quad = \; \mathsf{snd}(append^{-1} \,;\, halve) \,; \\
& \qquad\quad (\mathsf{snd}\,\mathsf{tri}\,\mathsf{map}\, P \,;\, \mathsf{row}\, Q \,;\, \mathsf{fst}\,\mathsf{tri}\,\mathsf{map}\, S) \leftrightarrow \\
& \qquad\qquad (\mathsf{snd}(\mathsf{rev} \,/\, \mathsf{tri}\,\mathsf{map}\, P) \,;\, \mathsf{row}\, R \,;\, \mathsf{fst}(\mathsf{rev} \,/\, \mathsf{tri}\,\mathsf{map}\, S)) \,; \\
& \qquad\quad \mathsf{fst}(halve^{-1} \,;\, append)
\end{aligned}
$$

Substituting into the left hand side and simplifying

$$
\begin{aligned}
& \mathsf{snd}(append^{-1} \,;\, pair) \,;\, \mathsf{row}(Q \leftrightarrow R) \,;\, \mathsf{fst}(pair^{-1} \,;\, append) \\
& \quad = \; \mathsf{snd}(append^{-1} \,;\, pair) \,; \\
& \qquad\quad \mathsf{row}((\mathsf{snd}\, [\text{-}]^{-1} \,;\, Q' \,;\, \mathsf{fst}\, [\text{-}]) \leftrightarrow (\mathsf{snd}\, [\text{-}]^{-1} \,;\, R' \,;\, \mathsf{fst}\, [\text{-}])) \,; \\
& \qquad\qquad \mathsf{fst}(pair^{-1} \,;\, append) \\
& \quad = \; \mathsf{snd}(append^{-1} \,;\, pair \,;\, \mathsf{map}[[\text{-}]^{-1}, [\text{-}]^{-1}]) \,; \\
& \qquad\quad \mathsf{row}(Q' \leftrightarrow R') \,; \\
& \qquad\qquad \mathsf{fst}(\mathsf{map}[[\text{-}], [\text{-}]] \,;\, pair^{-1} \,;\, append) \\
& \quad = \; \mathsf{snd}(append^{-1} \,;\, \mathsf{map}\, [\text{-}]^{-1} \,;\, pair) \,; \\
& \qquad\quad \mathsf{row}(Q' \leftrightarrow R') \,; \\
& \qquad\qquad \mathsf{fst}(pair^{-1} \,;\, \mathsf{map}\, [\text{-}] \,;\, append) \\
& \quad = \; \mathsf{snd}\, pair \,;\, \mathsf{row}(Q' \leftrightarrow R') \,;\, \mathsf{fst}\, pair^{-1}
\end{aligned}
$$

and similarly with the right-hand side

$$
\begin{aligned}
& \mathsf{snd}\,\mathsf{tri}\,\mathsf{map}\, P \,;\, \mathsf{row}\, Q \,;\, \mathsf{fst}\,\mathsf{tri}\,\mathsf{map}\, S \\
& \quad = \; \mathsf{snd}\,\mathsf{tri}\,\mathsf{map}\, P \,;\, \mathsf{row}(\mathsf{snd}\, [\text{-}]^{-1} \,;\, Q' \,;\, \mathsf{fst}\, [\text{-}]) \,;\, \mathsf{fst}\,\mathsf{tri}\,\mathsf{map}\, S \\
& \quad = \; \mathsf{snd}(\mathsf{tri}\,\mathsf{map}\, P \,;\, \mathsf{map}\, [\text{-}]^{-1}) \,;\, \mathsf{row}\, Q' \,;\, \mathsf{fst}(\mathsf{map}\, [\text{-}] \,;\, \mathsf{tri}\,\mathsf{map}\, S) \\
& \quad = \; \mathsf{snd}\,\mathsf{map}\, [\text{-}]^{-1} \,;\, (\mathsf{snd}\,\mathsf{tri}\, P \,;\, \mathsf{row}\, Q' \,;\, \mathsf{fst}\,\mathsf{tri}\, S) \,;\, \mathsf{fst}\,\mathsf{map}\, [\text{-}]
\end{aligned}
$$

and

$$
\begin{aligned}
& append^{-1} \,;\, halve \,;\, [\mathsf{map}\, [\text{-}]^{-1}, \mathsf{map}\, [\text{-}]^{-1}] \\
& \quad = \; append^{-1} \,;\, \mathsf{map}\, [\text{-}]^{-1} \,;\, halve \\
& \quad = \; halve
\end{aligned}
$$

so putting everything together, under the originally stated hypothesis that $[P, P]$; $(Q' \leftrightarrow R')$; $[S, S] = R' \leftrightarrow Q'$ we are allowed to conclude that

$$\begin{aligned}
&\mathsf{snd}\,\mathit{pair} \,;\, \mathsf{row}(Q' \leftrightarrow R') \,;\, \mathsf{fst}\,\mathit{pair}^{-1} \\
&\quad = \mathsf{snd}\,\mathit{halve} \,; \\
&\qquad\quad (\mathsf{snd}\,\mathsf{tri}\,P \,;\, \mathsf{row}\,Q' \,;\, \mathsf{fst}\,\mathsf{tri}\,S) \leftrightarrow \\
&\qquad\qquad (\mathsf{snd}(\mathsf{rev} \,/\, \mathsf{tri}\,P) \,;\, \mathsf{row}\,R' \,;\, \mathsf{fst}(\mathsf{rev} \,/\, \mathsf{tri}\,S)) \,; \\
&\qquad\quad \mathsf{fst}\,\mathit{halve}^{-1}
\end{aligned}$$

For ease of reference, let us call this the *special skewing theorem.*

This looks quite like the general skewing theorem: indeed if you try to draw a picture of an instance it will look confusingly similar to the picture of the general theorem, the only difference being that the triangles are triangles of maps in the general theorem. This indicates once again the danger of reasoning from pictures. The special theorem is as we have seen a consequence of the general theorem; as far as I know there is no simple way of deducing the general theorem from the special theorem.

In fact the special theorem seems to be strictly weaker because although the consequences are similar, the hypothesis of the general theorem is a much stricter requirement than is the hypothesis of the special theorem. To require that $[P, P]$; $(Q' \leftrightarrow R'); [S, S] = R' \leftrightarrow Q'$ is to require that wires which are connected (through P and S) to Q' can be connected to R' and vice versa, without changing the effect. To require that $\mathsf{snd}(\mathsf{map}\,P \,;\, \mathsf{app}^{-1}) \,;\, Q \leftrightarrow R \,;\, \mathsf{fst}(\mathsf{app} \,;\, \mathsf{map}\,S) = R \leftrightarrow Q$ is only to require that a bundle of wires divided between Q and R can also be divided between R and Q with the same effect. There are a number of very general consequences of the general theorem which derive from giving the whole of the bundle to either Q or R, so that this part of the hypothesis says little about Q and R.

An example of this is that if $Q = \mathsf{snd}\,0 \,;\, \pi_1 \,;\, Q' \,;\, \mathsf{fst}\,[\text{-}]$ and $R = \mathsf{snd}\,0 \,;\, \pi_1 \,;\, R' \,;\, {\pi_2}^{-1} \,;\, \mathsf{fst}\,0$ then if $R' \,;\, Q' = Q' \,;\, [S, R']$ by substituting into the general skewing theorem and simplifying it follows that

$$\begin{aligned}
&\mathsf{snd}(0 \,;\, \mathit{append}^{-1} \,;\, \mathsf{map}\,\mathit{any}) \,;\, \mathsf{row}(\pi_1 \,;\, Q' \,;\, \mathsf{snd}\,R') \\
&\quad = \mathsf{snd}(0 \,;\, \mathit{append}^{-1} \,;\, \mathsf{map}\,\mathit{any}) \,;\, \mathsf{row}(\pi_1 \,;\, Q') \leftrightarrow \mathsf{row}(\pi_1 \,;\, R' \,;\, {\pi_2}^{-1}) \,; \\
&\qquad \mathsf{fst}(\mathsf{fst}\,\mathsf{tri}\,S \,;\, [\mathsf{map}\,[\text{-}], \mathsf{map}\,0] \,;\, \mathit{halve} \,;\, \mathit{append})
\end{aligned}$$

We can dispose of the $\mathsf{snd}(0 \,;\, \mathit{append}^{-1} \,;\, \mathsf{map}\,\mathit{any})$ term by composing with a projection, but there seems to be no way of getting any further with the right-hand side as it stands. Consider however composing both sides with some n,

$$\begin{aligned}
&{\pi_1}^{-1} \,;\, \mathsf{row}(\pi_1 \,;\, Q' \,;\, \mathsf{snd}\,R') \,;\, \mathsf{fst}\,n \\
&\quad = {\pi_1}^{-1} \,;\, \mathsf{row}(\pi_1 \,;\, Q') \leftrightarrow \mathsf{row}(\pi_1 \,;\, R' \,;\, {\pi_2}^{-1}) \,; \\
&\qquad \mathsf{fst}(\mathsf{fst}\,\mathsf{tri}\,S \,;\, [\mathsf{map}\,[\text{-}], \mathsf{map}\,0] \,;\, [n, n] \,;\, \mathit{halve} \,;\, \mathit{append})
\end{aligned}$$

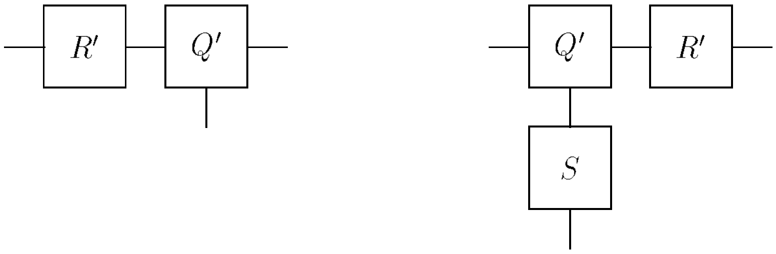

Figure 7.2: the hypothesis that $R' \,;\, Q' = Q' \,;\, [S, R']$

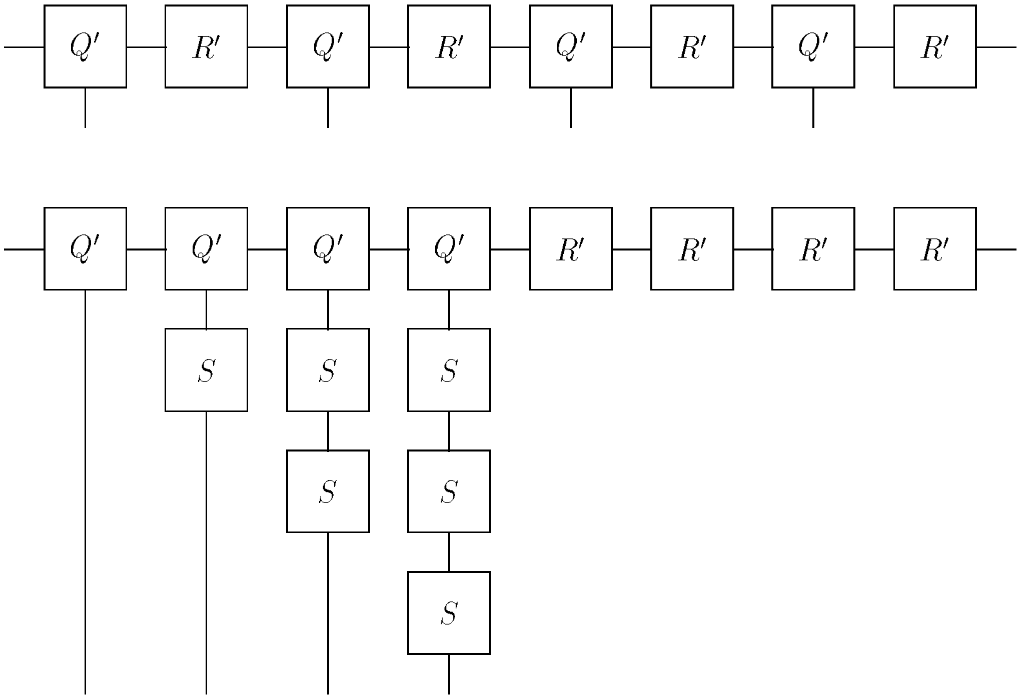

Figure 7.3: ${\pi_1}^{-1} \,;\, \mathsf{row}(\pi_1 \,;\, Q' \,;\, \mathsf{snd}\, R') \,;\, \mathsf{fst}\, 4$ and ${\pi_1}^{-1} \,;\, \mathsf{row}(\pi_1 \,;\, Q') \,;\, [\mathsf{tri}\, S \,;\, 4, R'^4]$ which are instances of circuits equal by the skewing theorem, under the hypothesis illustrated in figure 7.2

$$\begin{aligned}
&= {\pi_1}^{-1} ; \mathsf{row}(\pi_1 ; Q') \leftrightarrow \mathsf{row}(\pi_1 ; R' ; {\pi_2}^{-1}) ; \mathsf{fst}(\mathsf{fst}\,\mathsf{tri}\,S ; [n,n] ; \pi_1) \\
&= {\pi_1}^{-1} ; (\mathsf{row}(\pi_1 ; Q') ; \mathsf{fst}(\mathsf{tri}\,S ; n)) \leftrightarrow (\pi_1 ; R'^n ; {\pi_2}^{-1}) ; \mathsf{fst}\,\pi_1 \\
&= {\pi_1}^{-1} ; \mathsf{row}(\pi_1 ; Q') ; [\mathsf{tri}\,S ; n, R'^n]
\end{aligned}$$

and moreover if $R' = S$, the last term is just $\mathsf{apr} \mathbin{/} \mathsf{tri}\,S ; \mathsf{fst}\,n$ so by generalising over all n we conclude that

$$ {\pi_1}^{-1} ; \mathsf{row}(\pi_1 ; Q' ; \mathsf{snd}\,R') \quad = \quad {\pi_1}^{-1} ; \mathsf{row}(\pi_1 ; Q') ; \mathsf{apr} \mathbin{/} \mathsf{tri}\,S $$

and this is of course just the dual of the familiar statement of Horner's rule yet again. Taking inverses throughout we have shown that if $[S, S] ; Q' = Q' ; S$ then $\mathsf{rdr}(\mathsf{snd}\,S ; Q') = \mathsf{apr} \mathbin{/} \mathsf{tri}\,S ; \mathsf{rdr}\,Q'$ as an instance of the general skewing theorem.

7.4 Arithmetic and carries

There are instances of the skewing theorem which seem not to have anything to do with skewing, since they take $P = S = \iota$ and the triangles disappear entirely. Suppose that

$$ \mathsf{snd}\,\mathsf{app}^{-1} ; R \leftrightarrow Q ; \mathsf{fst}\,\mathsf{app} = \mathsf{snd}\,\mathsf{app}^{-1} ; Q \leftrightarrow R ; \mathsf{fst}\,\mathsf{app} $$

then by substitution into the general skewing theorem

$$\begin{aligned}
&\mathsf{snd}(\mathit{append}^{-1} ; \mathit{pair}) ; \mathsf{row}(Q \leftrightarrow R) ; \mathsf{fst}(\mathit{pair}^{-1} ; \mathit{append}) \\
&\quad = \mathsf{snd}(\mathit{append}^{-1} ; \mathit{halve}) ; \mathsf{row}\,Q \leftrightarrow \mathsf{row}\,R ; \mathsf{fst}(\mathit{halve}^{-1} ; \mathit{append})
\end{aligned}$$

This turns out to be a more surprising theorem than you might expect. Had we done the same with the special theorem the result would have been more obvious: if R beside Q is the same as Q beside R then clearly a row of Rs beside a row of Qs is much the same as a row of circuits each of which is either Q beside R or indeed R beside Q.

The more general theorem however says much more. Consider substituting $Q = \mathsf{snd}\,[\text{-}]^{-1} ; Q' ; \pi_2 ; \mathsf{fst}\,0$ and $R = \mathsf{snd}\,0 ; \pi_1 ; R' ; \mathsf{fst}\,[\text{-}]$, then the theorem gives something which simplifies to saying that if $Q' ; R' = \mathsf{snd}\,R' ; \mathit{lsh} ; \mathsf{fst}\,Q'$ then $\mathsf{snd}\,n ; \mathsf{rdl}\,Q' ; (\mathsf{fst}\,n ; \mathsf{rdr}\,R'^{-1})^{-1} = \mathsf{snd}\,n ; \mathsf{row}(Q' ; R') ; \mathsf{fst}\,n$.

In particular you can show by considering the arithmetic that $\mathit{add} ; \mathit{add}^{-1} = \mathsf{snd}\,\mathit{add}^{-1} ; \mathit{lsh} ; \mathsf{fst}\,\mathit{add}$, at least so long as you are talking about arithmetic on all integers, both positive and negative. It follows that $\mathsf{snd}\,n ; \mathsf{rdl}\,\mathit{add} ; (\mathsf{rdr}\,\mathit{add})^{-1} ; \mathsf{fst}\,n = \mathsf{snd}\,n ; \mathsf{row}(\mathit{add} ; \mathit{add}^{-1}) ; \mathsf{fst}\,n$. An instance of this appears in figure 7.4 and you can probably see informally that since the circuit relates two collections of numbers if their sums are the same, the value on one of the internal wires has to have

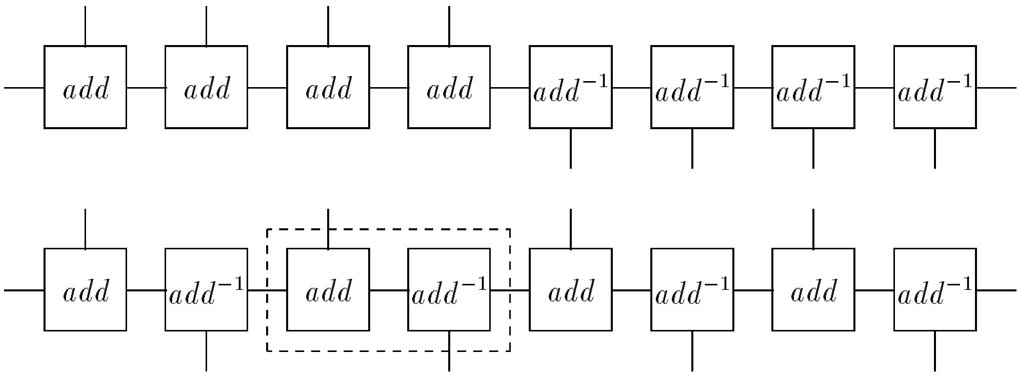

Figure 7.4: $\mathsf{snd}\,4 ; \mathsf{rdl}\,add ; (\mathsf{rdr}\,add)^{-1} ; \mathsf{fst}\,4 = \mathsf{snd}\,4 ; \mathsf{row}(add ; add^{-1}) ; \mathsf{fst}\,4$

something to do with the difference of the sums accumulated left and right of that wire. Fortunately the theorem has taken care of such details and we need neither know nor care what the value on an internal wire is in either circuit.

Consider now the relations $abs = \mathsf{tri}(\times 2) ; {\pi_2}^{-1} ; \mathsf{fst}\,zero ; \mathsf{rdl}\,add$ which relates a list of numbers (not necessarily all zero or one) to their weighted sum, where each number is given twice the weight to the number to its left. It is called abs here to encourage you to think of it as abstracting from that list of numbers to the weighted sum which they represent. If you are given two n-bit binary numbers and asked for their sum, this could be calculated by abstracting each list of bits to the number that it represents, and adding the abstractions. To do this requires two instances of $n ; \mathsf{map}\,bit ; abs$ and an adder, but using a zipping theorem and simplifying

$$
\begin{aligned}
&[n ; abs, n ; abs] ; add \\
&\quad = [n, n] ; [\mathsf{tri}(\times 2), \mathsf{tri}(\times 2)] ; \\
&\quad\ [{\pi_2}^{-1} ; \mathsf{fst}\,zero ; \mathsf{rdl}\,add, {\pi_2}^{-1} ; \mathsf{fst}\,zero ; \mathsf{rdl}\,add] ; add \\
&\quad = [n, n] ; [\mathsf{tri}(\times 2), \mathsf{tri}(\times 2)] ; \\
&\quad\ {\pi_2}^{-1} ; \mathsf{fst}[zero, zero] ; \mathsf{snd}\,\mathsf{zip} ; \mathsf{rdl}(\mathsf{zip} ; [add, add]) ; add \\
&\quad = [n, n] ; \mathsf{zip} ; \mathsf{tri}[(\times 2), (\times 2)] ; \\
&\quad\ {\pi_2}^{-1} ; \mathsf{fst}[zero, zero] ; \mathsf{rdl}(\mathsf{zip} ; [add, add]) ; add \\
&\quad = [n, n] ; \mathsf{zip} ; \mathsf{tri}[(\times 2), (\times 2)] ; \\
&\quad\ {\pi_2}^{-1} ; \mathsf{fst}([zero, zero] ; add) ; \mathsf{rdl}(\mathsf{snd}\,add ; add) \\
&\quad = [n, n] ; \mathsf{zip} ; \mathsf{tri}[(\times 2), (\times 2)] ; \mathsf{map}\,add ; {\pi_2}^{-1} ; \mathsf{fst}\,zero ; \mathsf{rdl}\,add \\
&\quad = \mathsf{zip} ; \mathsf{map}\,add ; n ; abs
\end{aligned}
$$

so we could interleave the numbers, add corresponding bits, and then take the weighted sum with abs. That means that $[\mathsf{map}\,bit, \mathsf{map}\,bit] ; \mathsf{zip} ; \mathsf{map}\,add ; n ; abs$

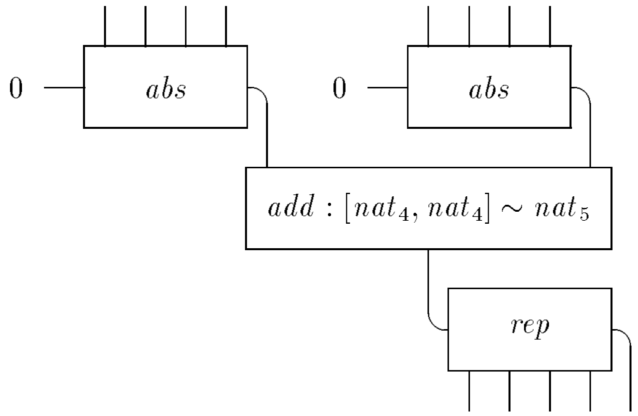

Figure 7.5: $[4 ; abs, 4 ; abs] ; add ; rep ; 5$

is an abstraction taking pairs of n-bit numbers to their sum.

The sum is just a number, and if we wanted to build a circuit to calculate the sum we would need to choose a representation for it. Since the sum of two n-bit binary numbers can be bigger than the largest number representable in n bits we cannot use $(n ; \mathsf{map}\ bit ; abs)^{-1}$ to represent the sum. We might use $((n+1) ; \mathsf{map}\ bit ; abs)^{-1}$ and indeed this would eventually lead to a circuit. With an application of the skewing theorem in mind, however, let us choose a different abstraction. Since adding from the left required us to provide an extra ('carry') bit with the same weight as the least significant bit, adding from the right is going to require an additional high-weight bit. Indeed $(n + 1) ; \mathsf{map}\ bit ; \mathsf{tri}(\times 2) ; \mathsf{apr}^{-1} ; \mathsf{rdr}\ add = \mathsf{apr}^{-1} ; [n ; \mathsf{map}\ bit, bit] ; (\mathsf{apr} \ / \ \mathsf{tri}(\times 2)) ; \mathsf{rdr}\ add$ is also an abstraction from lists of bits to the numbers they represent, taking any $n + 1$ bit number to its value. We can use its inverse to represent the sum of the two n-bit numbers.

Suppose we call this inverse rep then we propose to add two numbers with the circuit

$$
\begin{aligned}
& [n ; abs, n ; abs] ; add ; rep \\
& \quad = [\mathsf{map}\ bit, \mathsf{map}\ bit] ; \mathsf{zip} ; \mathsf{map}\ add ; n ; abs ; rep \\
& \quad = [\mathsf{map}\ bit, \mathsf{map}\ bit] ; \mathsf{zip} ; \mathsf{map}\ add ; n ; \\
& \qquad \mathsf{tri}(\times 2) ; {\pi_2}^{-1} ; \mathsf{fst}\ zero ; \mathsf{rdl}\ add ; \\
& \qquad \mathsf{rdr}\ add^{-1} ; (\mathsf{apr} \ / \ \mathsf{tri}(\times 2)^{-1}) ; [n ; \mathsf{map}\ bit, bit] ; \mathsf{apr} \\
& \quad = [\mathsf{map}\ bit, \mathsf{map}\ bit] ; \mathsf{zip} ; \mathsf{map}\ add ; n ; \mathsf{tri}(\times 2) ; {\pi_2}^{-1} ; \mathsf{fst}\ zero ; \\
& \qquad \mathsf{row}(add ; add^{-1}) ; \\
& \qquad (\mathsf{apr} \ / \ \mathsf{tri}(\times 2)^{-1}) ; [n ; \mathsf{map}\ bit, bit] ; \mathsf{apr}
\end{aligned}
$$

But now appealing to the arithmetic on integers you can show that $\mathsf{snd}(\times 2) ; add ; add^{-1} ; [(\times 2)^{-1}, (\times 2)^{-1}] = \mathsf{fst}(\times 2)^{-1} ; add ; add^{-1}$ because the sum of two even

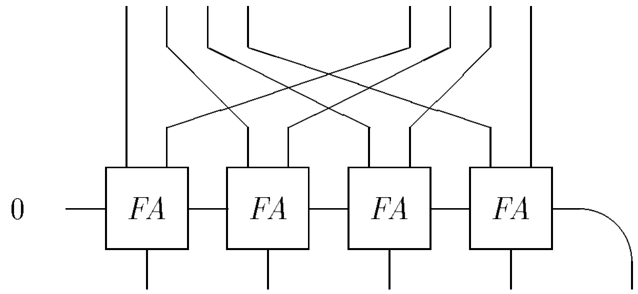

Figure 7.6: $[4,4]$; zip ; ${\pi_2}^{-1}$; fst $zero$; row FA ; apr ; 5

numbers can only differ from an even number by a fourth even number, and the numbers related by the adders on the two sides of the equation are themselves related by dividing throughout by two. Now this fact about the components turns out to be just what we need to apply an instance of the general skewing theorem the conclusion of which is that

$$\begin{aligned}&\mathsf{snd\,tri}(\times 2)\ ;\ \mathsf{row}(add\ ;\ add^{-1})\ ;\ (\mathsf{apr}\ /\ \mathsf{tri}(\times 2)^{-1})\\ &\quad=\ \mathsf{row}(add\ ;\ add^{-1}\ ;\ \mathsf{snd}(\times 2)^{-1})\end{aligned}$$

so returning to the design for an adder circuit

$$\begin{aligned}&[n\ ;\ abs, n\ ;\ abs]\ ;\ add\ ;\ rep\\ &\quad=\ [\mathsf{map}\ bit, \mathsf{map}\ bit]\ ;\ \mathsf{zip}\ ;\ \mathsf{map}\ add\ ;\ n\ ;\ {\pi_2}^{-1}\ ;\ \mathsf{fst}\ zero\ ;\\ &\qquad\ \mathsf{row}(add\ ;\ add^{-1}\ ;\ \mathsf{snd}(\times 2)^{-1})\ ;\\ &\qquad\ [n\ ;\ \mathsf{map}\ bit, bit]\ ;\ \mathsf{apr}\\ &\quad=\ [n, n]\ ;\ \mathsf{zip}\ ;\ {\pi_2}^{-1}\ ;\ \mathsf{fst}\ zero\ ;\\ &\qquad\ \mathsf{snd\,map}([bit, bit]\ ;\ add)\ ;\ \mathsf{row}(add\ ;\ add^{-1}\ ;\ \mathsf{snd}(\times 2)^{-1})\ ;\ [\mathsf{map}\ bit, bit]\ ;\\ &\qquad\ \mathsf{apr}\\ &\quad=\ [n, n]\ ;\ \mathsf{zip}\ ;\ {\pi_2}^{-1}\ ;\ \mathsf{fst}\ zero\ ;\ \mathsf{row}\ A\ ;\ \mathsf{snd}\ bit\ ;\ \mathsf{apr}\\ &\qquad\ \text{where } A = \mathsf{snd}([bit, bit]\ ;\ add)\ ;\ add\ ;\ add^{-1}\ ;\ [bit, (\times 2)^{-1}]\end{aligned}$$

At this point we have to appeal to the arithmetic again, to show that fst bit ; $A =$ fst bit ; A ; snd bit $= A$; snd bit and so by induction that

$$\begin{aligned}&[n\ ;\ abs, n\ ;\ abs]\ ;\ add\ ;\ rep\\ &\quad=\ [n, n]\ ;\ \mathsf{zip}\ ;\ {\pi_2}^{-1}\ ;\ \mathsf{fst}\ zero\ ;\ \mathsf{row}\ FA\ ;\ \mathsf{apr}\\ &\qquad\ \text{where } FA = [bit, [bit, bit]\ ;\ add]\ ;\ add\ ;\ add^{-1}\ ;\ [bit, (\times 2)^{-1}\ ;\ bit]\end{aligned}$$

where FA is now just a circuit that operates on bits, and turns out to be a *full adder* which is a well-known component in the library of standard circuits.

What we have done in this section is to design a bit-level circuit for adding two numbers purely by calculating the consequences of choosing particular representations for the numbers and their sum. To be strictly honest the word 'representation' as used here means a little more than you might think. We had to decide not only that we would choose to abstract a binary number, that is that the abstraction was going to be $\mathsf{tri}(\times 2)\ ;\ acc$, but also whether to implement the acc by a left or right reduction. There were a number of such arbitrary choices to be made, so the development was not entirely mechanical. Reference [Jones90a] expands on this design-by-refinement, which is still 'work in progress' both in Oxford and Glasgow.

If you have a copy of a draft of reference [Jones90a], beware that all versions in existence at the time of writing these notes state the *special* skewing theorem and claim incorrectly that various theorems are consequences of it.

Bibliography

This bibliography contains a number of things in a rather confused muddle: there are the references from the notes, there is almost everything published about this design technique, and there are the volumes in which the papers appear. Some time I will separate these various things into separate categories.

[Backus78] J. Backus, *Can programming be liberated from the von Neumann style? a functional style and its algebra of programs*, Commun ACM, Vol. 21, No. 8, 1978. pp. 613–641.

[Bird87a] R. S. Bird, *An introduction to the theory of lists*, in [Broy87]. pp. 3–42. (Programming Research Group technical monograph PRG–56)

[Bird87b] R. S. Bird, *A calculus of functions for program derivations*, Programming Research Group technical monograph PRG–64, December 1987.

[Bird88] R. S. Bird, *Lectures on constructive functional programming*, Programming Research Group technical monograph PRG–69, September 1988.

[Birt90] G. Birtwistle (ed.), *Proc 3rd Banff workshop on hardware verification*, Springer 1990.

[Brom88] K. Bromley, S. Y. Kung and E. Schwartzlander (eds.), *Proc Int Conf on Systolic arrays*, IEEE Computer Society Press, 1988.

[Broy87] M. Broy (ed.), *Logic of programming and calculi of discrete design*, Springer, 1987.

[Claes89] L. J. M. Claesen, *Applied formal methods for correct VLSI design*, North-Holland, 1989.

[David89] David et al (eds.), *Proc IFIP workshop on Concepts and characteristics of declarative systems*, North-Holland, 1989.

[Dew89] P. M. Dew, R. A. Earnshaw and T. R. Heywood (eds.), *Parallel processing for computer vision and display*, Addison-Wesley, 1989.

[Finn83] S. Finn, *LVIS – a VLSI transformation system*, M.Sc. dissertation, University of Oxford, 1983.

[Hansen89] B. W. Hansen and J. Jørgensen, *Graphical relational algebra for the synthesis of VLSI*, M.Sc. thesis, Technical University of Denmark.

[Hughes90] R. J. M. Hughes et al (eds.), *Proc Glasgow Functional Programming Group workshop*, Fraserburgh, 1989. (To appear from Springer 1990)

[Jones86] G. Jones and W. Luk, *Exploring designs by circuit transformation*, in [Moore86]. pp. 91–98.

[Jones88] G. Jones and M. Sheeran, *Timeless truths about sequential circuits*, in [Tewk88]. pp. 245–259.

[Jones89] G. Jones, *Deriving the fast Fourier algorithm by calculation*, in [Hughes90]. (Programming Research Group technical report PRG–TR–4–89)

[Jones90a] G. Jones and M. Sheeran, *Relations and refinement in circuit design*, in [Morgan90].

[Jones90b] G. Jones and M. Sheeran, *Circuit design in Ruby*, in [Staun90].

[Jouann85] J-P. Jouannaud (ed.), *Functional programming languages and computer architecture*, Springer LNCS 201, 1985.

[Kung80] H. T. Kung, *Special-purpose devices for signal processing and image processing: an opportunity in VLSI*, 24th SPIE Annual Technical Symposium, San Diego, August 1980.

[Kung88] S. Y. Kung, *VLSI array processors*, Prentice Hall, New Jersey, 1988.

[Leeser89] M. Leeser et al (eds.), *Hardware specification, verification and synthesis: mathematical aspects*, Springer LNCS, 1989.

[Leise81] C. E. Leiserson, *Area efficient VLSI computation*, Ph.D. thesis, Carnegie-Mellon University, Pittsburg, Pennsylvania, October 1981.

[Luk87] W. Luk and G. Jones, *Systolic arrays for recursive digital filtering*, Electronic Letters, Vol. 23, No. 22, October 1987. pp. 1174–75.

[Luk88a] W. Luk and G. Jones, *Systolic recursive filters*, IEEE Trans Circuits and Systems, Vol. 35, No. 8, August 1988. pp. 1067–68.

[Luk88b] W. Luk and G. Jones, *From specification to parametrised architectures*, in [Milne88]. pp. 267–288.

[Luk88c] W. Luk and G. Jones, *The derivation of regular synchronous circuits*, in [Brom88]. pp. 305–314.

[Luk88d] W. Luk, *Parametrised design of regular processor arrays*, D.Phil. thesis, University of Oxford, 1988.

[Luk89a] W. Luk and G. Jones, *Parametrised retiming of regular computational arrays*, in [Dew89]. pp. 50–63.

[Luk89b] W. Luk, G. Jones and M. Sheeran, *Computer-based tools for regular array design*, in [McCann89]. pp. 589–598.

[Luk89c] W. Luk, *Specifying and developing regular heterogeneous designs*, in [Claes89].

[Luk89d] W. Luk and G. Brown, *A systolic LRU processor and its top-down development*, October 1989 (unpublished).

[McCabe82] M. M. McCabe, A. P. H. McCabe, B. Abrambepola, I. N. Robinson and A. G. Cory, *New algorithms and architectures for VLSI*, GEC Journal of Science & Technology, Vol. 48, No. 2, 1982. pp. 68–75.

[McCann82] J. V. McCanny and J. G. McWhirter, *Implementation of signal-processing functions using 1-bit systolic arrays*, Electronic Letters, Vol. 18 No. 16, March 1982. pp. 241–243.

[McCann89] J. McCanny, J. McWhirter and E. Schwartzlander (eds.), *Systolic array processors*, Prentice Hall, 1989.

[Milne88] G. J. Milne (ed.), *The fusion of hardware design and verification*, North-Holland, 1988.

[Morgan90] C. C. Morgan (ed.), *Proc BCS FACS workshop on Refinement*, Hursley, January 1990. (To appear from Springer 1990)

[Moore86] W. Moore, A. McCabe and R. Urquhart (eds.), *Systolic arrays*, Adam Hilger, Bristol, 1986.

[ODonn88] J. T. O'Donnell, *Hydra: hardware description in a functional language using recusion equations and high order combining forms*, in [Milne88]. pp. 309–328.

[Proeb87] W. E. Proebster and H. Reiner (eds.), *Proc IEEE Comp Euro 87: VLSI and Computers*, Hamburg, May 1987.

[Ross89a] L. Rossen, *Relationsbaseret Specifikation af VLSI*, M.Sc. thesis, Technical University of Denmark, January 1989. (In Danish)

[Ross89b] L. Rossen, *HOL Formalisation of Ruby*, Technical University of Denmark, 1989. (Technical report, shortened translation of [Ross89a])

[Sheer81] M. Sheeran, *Functional geometry and integrated circuit layout*, M.Sc. dissertation, University of Oxford, 1981.

[Sheer83] M. Sheeran, *μFP, a language for VLSI design*, D.Phil. thesis, University of Oxford, 1983. (Programming Research Group technical monograph PRG–39)

[Sheer84] M. Sheeran, *muFP an algebraic VLSI design language*, in [Steele84]. pp. 104–112.

[Sheer85] M. Sheeran, *Designing regular array architectures using higher-order functions*, in [Jouann85]. pp. 220–237.

[Sheer86a] M. Sheeran, *Design and verification of regular synchronous circuits*, IEE Proceedings, Vol. 133, Pt. E, No. 5, September 1986. pp. 295–304.

[Sheer86b] M. Sheeran, *Describing and reasoning about circuits using relations*, in [Tucker90]. (Revised form appears as [Sheer90])

[Sheer87] M. Sheeran and G. Jones, *Relations + higher-order functions = hardware descriptions*, in [Proeb87]. pp. 303–306. (University of Glasgow Department of Computing Science technical report 87/R1)

[Sheer88] M. Sheeran, *Retiming and slowdown in Ruby*, in [Milne88]. pp. 289–308.

[Sheer89a] M. Sheeran, *Describing hardware algorithms in Ruby*, in [David89]. (Revised form appears as [Sheer89c])

[Sheer89b] M. Sheeran, *Categories for the working hardware designer*, in [Leeser89].

[Sheer89c] M. Sheeran, *Describing butterfly networks in Ruby*, in [Hughes90].

[Sheer90] M. Sheeran, *Ruby – a language of relations and higher-order functions*, in [Birt90].

[Singh89] S. Singh, *Application of non-standard interpretation: testability*, in [Claes89].

[Staun90] Jørgen Staunstrup (ed.), *Formal methods for VLSI design*, North-Holland, 1990.

[Steele84] G. L. Steele Jr. et al (eds.), *Proc ACM Symp on LISP and functional programming*, 1984.

[Tewk88] S. K. Tewksbury, B. W. Dickinson and S. C. Schwartz (eds.), *Concurrent computations: algorithms, architecture and technology*, Plenum Press, New York, 1988.

[Tucker90] J. Tucker et al (eds.), *Proc workshop on theoretical aspects of VLSI design*, Leeds 1986. (To appear from CUP 1990)

Index