A Formal Process for Systolic Array Design Using Recurrences

Jonathan Puddicombe

Revised version of thesis submitted for degree of
Doctor of Philosophy
University of Edinburgh
October 1992
(Revised version submitted June 1993)

EX BIBL. UNIV. EDINBURGH

Declaration

This thesis was composed by me and the work described in it is my own, except where indicated.

Acknowledgements

My thanks go to my supervisors, Peter Denyer, especially for his faith, optimism and encouragement, and Peter Grant, especially for his detailed comments on my work and his faithfulness, patience and perseverance with me.

My thanks also go to Sanjay Rajopadhye, whose work was the springboard for mine, to S.E.R.C. for funding me for two years, and to everyone who gave me support of a technical, financial or moral nature.

Abstract

A systolic array is essentially a parallel processor which consists of a grid of locally-connected sub-processors which receive, process and pump out data synchronously in such a way that the pattern of data-flow to and from each processor is identical to the flow to and from the other processors. Such arrays are repetitive and modular and require little length of communication interconnection, so that they are relatively simple to design and are amenable to efficient VLSI implementation. The systolic architecture has been found suitable for implementing many of the algorithms used in the field of signal- and image-processing.

A formal design method is a well-defined process for constructing, given a well-defined function from a certain class, a well-defined object (e.g. a design) which performs that function. When proven correct, such methods are useful for designing equipment which is safety-critical or where a design fault discovered after manufacture would be expensive.

This thesis presents a formal design method for producing high-level implementations for certain signal-processing and other algorithms. These high-level implementations can themselves usually be easily implemented as systolic arrays.

As a necessary preliminary to the method, a calculus is defined. The basic concept, that of a "computation", is powerful enough to express both abstract algorithms and those whose suboperations have been assigned a place and a time to execute. Computations may be composed or abstracted (by having their variables hidden) or may have their variables renamed. The "simulation" of one computation by another is defined. Using this calculus it is possible to formalise concepts like "dependency" (of data or control) and "system of recurrence equations", which often appear in the literature on systolic array design. The design method is then presented. It consists of five stages: pipelining of data dependencies, scheduling, pipelining of the control variables, allocation of subprocessors to the subcomputations, and the final stage (in which the design is constructed). The main concepts are not new, but here they have been formalised, arranged and linked in a clearly defined way. The output of the method is a high-level design description which defines the functionality of each subprocessor in the array (for both data and control). It also defines scheduling and allocation of all the operations which are to be executed and the data and control input requirements of the array.

The method is used to design a simple one-dimensional systolic convolver and then to design a more complicated two-dimensional systolic array which performs Given's algorithm for QR-factorisation, a task required in certain signal-processing applications such as adaptive estimation and bearing measurement. Alternative designs are briefly discussed. For the convolver and the two arrays for QR-factorisation, sketches of the architectures are given but these are hand-produced and are not the product of the method.

A detailed proof is given that, subject to assumptions about the well-definedness of the computations handled and created, the design method will produce only designs which meet their specifications; however the final high-level design may imply a low-level implementation which may contain an interconnection structure which is arguably non-local. A proof is given that the well-definedness conditions hold which are required for the validation of data-pipelining.

Contents

Terminology

General

- The hand symbol "✍" signifies that what follows is a reference to a proposition and its proof in the appendices.

- The symbol "•" is used to express the composition of two functions. So (v•RENAME)var := v(RENAME(var)).

- The symbol "↘" signifies that the term or bracketed expression immediately following it is to be read as being a subscript of the one preceding it.

- The symbol "$\rfloor$" is to be read as "|" followed by "↘".

- *dom*(F) denotes the domain of a function F, and *ran*(F) denotes its range. The domain of a function written "p → e", where e is an expression in p, will often not be stated when it is implied by the context.

- Let v be a function from S to T and let S' be a subset of S. Then $v|_{S'}$ is the function from S' to T such that $v|_{S'}$ (s')= v(s') for all s' in S'.

- If F is a function then F[x → y] is defined to be the function with the same range as F and domain dom(F)∪{x} which satisfies the following equations:

  $F[x \rightarrow y](x) = y$

  $F[x \rightarrow y](x') = F(x')$ when $x' \neq x$

- A *functional* is a function which takes a function as one of its arguments.
- *w.r.t.* stands for "with respect to".
- *s.t.* stands for "such that".
- *n.p.* stands for "not proven".
- "Integer" is the set of integers.
- "Real" is the set of real numbers.
- Nat(n) is the set of natural numbers from 1 to n inclusive. Nat(n) may be written {1...n}.
- Id_S is function which has domain S and maps every element of S to itself.

Vector spaces

- A *vector space* over a field <F, +, *> (e.g. the field of real numbers with the usual addition and multiplication operations) is a triple <V, ⊕, ⊗> where

  i) <V, ⊕> is an Abelian group

  ii) $\otimes: F \times V \to V$ and, for all $\alpha, \beta \in F$ and $u, v \in V$,

  $(\alpha + \beta)\otimes u = \alpha\otimes u \oplus \beta\otimes u$

  $\alpha\otimes(u \oplus v) = \alpha\otimes u \oplus \alpha\otimes v$

  and

  $\alpha\otimes(\beta\otimes u) = (\alpha{*}\beta)\otimes u$

  $\alpha\otimes u$ and $\alpha{*}\beta$ are usually written αu and $\alpha\beta$ respectively, and the same sign may be used for ⊕ and + since no ambiguity can arise.

However there are conceptually four distinct operations, which is why four symbols were used in this definition. The term "vector space" may be loosely used to refer simply to the set V when and the field are taken as read. Ditto with the term "field".

- A *linear transformation* from a vector space $<S_1, \oplus_1, \otimes_1>$ over $<F, +, *>$ to a vector space $<S_2, \oplus_2, \otimes_2>$ over $<F, +, *>$is a function T from S_1 to S_2 which satisfies

  $T(v \oplus_1 u) = T(v) \oplus_2 T(u)$ and
  $T(\lambda \otimes_1 v) = \lambda \otimes_2 T(v)$

  for all v and u in S_1 and all λ in F.
  Let the set of linear transformations from S_1 to S_2 be called L. L itself forms a vector space $<L, \oplus_L, \otimes_L>$ over $<F, +, *>$ where

  $(T \oplus_L U)v = T(v) \oplus_2 U(v)$
  $(\alpha \otimes_L T)v = \alpha \otimes_2 (T(v))$

  for all T and U in L and all α in F.

- A linear transformation is *singular* iff it is not invertible.

- A map, $p \rightarrow A(p) + b$, from a vector space to a vector space, where A is a linear transformation and b is a constant vector is called *affine*.

- An affine map $p \rightarrow A(p) + b$ is defined to be a *translation* iff $A = I$.

- The *null space* of a linear transformation T is $\{u : T(u) = 0\}$.

- Let I be an indexing set. The set $\{v_i : i \in I\} \subseteq V$ is said to be *linearly independent* iff $\Sigma_{i \in I}\lambda_i v_i = 0 \Rightarrow \lambda_i = 0$ for all $i \in I$.

- Let I be an indexing set. The set $\{v_i : i \in I\} \subseteq V$ <u>*spans*</u> V iff, for all $v \in V$, $v = \Sigma_{i \in I}\lambda_i v_i$ for some set $\{\lambda_i : i \in I\}$.

- A <u>*basis*</u> for V is a linearly independent set which spans V. There is a theorem which states that if V has a finite basis then all bases for V have the same number of elements.

- A vector space V is said to be <u>*n-dimensional*</u> iff it has an n-element basis.

- The dimension of the null-space of a linear transformation is called its <u>*nullity*</u>.

- A <u>*matrix*</u> is (informally) an array of elements of identical type. The following is a 2×3 matrix with integer elements:

$$\begin{bmatrix} 23 & -6 & 3 \\ 4 & 9 & 0 \end{bmatrix}$$

  For a matrix A, "A(i, j)" stands for the element in the ith row and the jth column.

- The <u>*transpose*</u> of an n×m matrix A is the m×n matrix, which may be written A^T satisfying the following property: For all pairs <i, j> in Nat(n)×Nat(m), $A(i, j) = A^T(j, i)$.

  The set of n×m matrices with elements drawn from a certain field form a vector space over that field. Given an ordered basis for an n-dimensional vector space over a field, one can find a natural association between vectors in that space and n×1 matrices with elements drawn from that field.; n×1 matrices are called column (n-)vectors. Similarly any n-dimensional space over a field may be identified with the space of $1\times n$ matrices with elements drawn from the field; these are called row (n-) vectors. Given ordered bases for an n-dimensional vector space (S_1) over a field, and an m-dimensional vector space (S_2) one can find a natural association between the space formed by the set of linear transformations from S_1 to S_2 and the space of $m\times n$ matrices.The 1×2 matrix [i j] may be written [i, j] in order to separate the two elements visually; $1\times n$ matrices may be punctuated in a similar way.

- The *product* of an $m\times n$ matrix A and an $n\times p$ matrix B is the $m\times p$ matrix C, where $C(i, j) := \Sigma_{k=1 \text{ to } n} A(i, k)B(k, j)$. The product of matrices A and B is written A.B, or just AB .

- A matrix A is said to be *orthogonal* iff $AA^T = I$.

- A matrix A is said to be *upper-triangular* iff $A(i, j) = 0$ whenever $i < j$.

- The *determinant* of an $n\times n$ matrix A, written "det(A)", is defined recursively as follows: if A is the 1×1 matrix [a] then det(A) = a; otherwise $\det(A) = \Sigma_{j=1 \text{ to } n} (-1)^{j+1} A(1, j)*\det(A\rfloor_{1,j})$, where $A\rfloor_{1,j}$ is the matrix obtained from A by deleting its 1st row and jth column.

Lattices

- Let V be a vector space and let A equal $\{a_i : 1 \leq i \leq n\}$ be a subset of V; then L, defined as follows, is a *lattice*:

$$L := \{ u_1a_1 + u_2a_2 + \ldots + u_na_n : u_1, u_2 \ldots u_n \text{ are } \textit{integers}\}$$

- A (defined above) is said to be an *l-basis* for L. (There may be other l-bases for L, for example, $\{a_i' : 1 \leq i \leq n\}$, where $a_i' := \Sigma_j v(i,j)a_j$ and v is an integer matrix with det(v) equal to 1.)

- Let T be a linear transformation from V to another vector space U; let the null space of T be N. Then the <u>*null lattice*</u> of T (relative to the lattice L) is defined to be $N \cap L$.

Glossary of Terms

AR:	Affine Recurrence (see page 57)
ARMA:	"Auto-Regressive Moving Average": descriptive of a filter whose current output is a linear combination of recent inputs and outputs
CAD:	Computer-Aided Design
CCS:	Calculus of Communicating Systems: a formalism for describing the behaviour of parallel, interacting systems (see page 32)
CIRCAL:	a formalism with a similar style and purpose to CCS (see page 32)
CSP:	Communication Sequential Processes: a formalism with a similar style and purpose to CCS (see page 32)
CURE:	Conditional Uniform Recurrence Equation (see page 71)
LRA:	Linear Recurrence Algorithm (see page 34)
MIMD:	Multiple-Instruction-Multiple-Data: descriptive of a certain type of asynchronous parallel architecture in which each processor has its own control unit and memory (see page 6)
RIA:	Regular Iterative Algorithm (see page 34)
SA:	Systolic Array
SARE:	System of Affine Recurrence Equations (see page 34)

SIMD:	Single-Instruction-Multiple-Data: descriptive of a certain type of synchronous parallel architecture which operates by the broadcasting of a sequence of instructions to a set of processors. The processors generally process separate data streams (see page 6)
SURE:	System of Uniform Recurrence Equations (see page 33)
SRE:	System of Recurrence Equations (see page 71)
QR-factorisation:	the task of finding an upper-triangular matrix which, when premultiplied by some orthogonal matrix, will produce a given (square) matrix (see page 119)
UR:	Uniform Recurrence (see page 57)
URE:	Uniform Recurrence Equation (see page 71)
VLSI:	Very Large Scale Integration

1 Introduction

1.1 Subject of Thesis

Many computing tasks, especially from the areas of one-dimensional signal- and two-dimensional image-processing, have the following characteristics:

(1) The task needs to be done quickly.

(2) There are algorithms for performing it which can be parallelised.

Characteristic (2) can be used to satisfy requirement (1). Some tasks have an algorithm which will run sufficiently fast on a general purpose parallel machine. However, for real-time processing a speed of the order of 1 billion instructions per second may be necessary; in such cases it is often desirable to design a custom parallel architecture which can be implemented efficiently using VLSI. There is a certain type of parallel architecture which is particularly suitable for implementing signal- and image-processing algorithms and is also especially suited to VLSI: the systolic array.

The pioneering work on systolic arrays was done by H.T.Kung and C.E.Leiserson in the late seventies [HTKun78], though the algorithms which were found to be suitable for running on them had been studied previously [Karp67]. In [HTKun78], Kung and Leiserson concisely describe a "systolic system" as "a set of processors which rhythmically compute and pass data through the system". The synchronised "pumping" of data through such a system resembles the action of the heart on blood within the circulatory system, hence the term "systolic".

Regarding uses of the systolic architecture, Kung and Leiserson themselves showed that systolic arrays could be built which would perform certain important tasks in the field of linear algebra, such as band-matrix multiplication, triangularisation and back-substitution [HTKun78]. In the last decade systolic arrays have been designed which implement many of the algorithms used in radar-, sonar-, image-, signal- and speech-processing [SYKun88, McW92].

Also over the last decade much work has been done to develop mathematically-based languages which can be used to encapsulate hardware design specifications formally and precisely, and to develop mathematical techniques for proving that hardware designs meet those specifications. These languages and techniques are known as "formal methods" or "formal verification". To have a proof of design-correctness is particularly desirable for safety-critical hardware. It is possible to integrate the tasks of design and verification so that each step of the design process is verified as it is taken. This benefits the designer by alerting him to design errors at an early stage, avoiding costly redesign, and it also benefits the verifier since he is not fed with an uncommented, unstructured, design which he must verify without knowing the rationale behind it.

Though a validated design process will warn the designer off incorrect designs, it may still be hard for him to find a correct one, due to the plethora of red-herring options. However, if he is willing to forego some freedom, e.g. by restricting himself to a certain architecture, then he can use a specialised formal design method in which some of the steps have been frozen, leaving fewer steps to choose and verify, thereby simplifying his task. (Of course the architecture must be appropriate to the algorithm to be implemented, otherwise the task of finding a correct design may be made more difficult or impossible.) This thesis presents one such specialised method - to be used in the design of systolic arrays.

1.2 Systolic Arrays

1.2.1 What is a systolic array?

Several researchers have given more or less precise definitions of the set of systolic arrays (SAs) [HTKun78, Ull84, Rao85, SYKun88]. In this thesis the following definition is adopted:

(1) A systolic array contains a set of processors.

(2) (locality) The interconnections between these processors, and between the

processors and the outside world, are all local.

(3) (homogeneity) The network formed by the processors and their interconnections is regular and homogeneous, at least in the interior of the network, and may be extended indefinitely. The type of each processor is ignored when examining the network for homogeneity.

(4) (synchronisation) The operation of the processors is synchronised by a global clock.

(5) (pacing of data) The maximum speed at which information can travel within the array is one processor per clock tick (or cycle); i.e. a datum which is output by a processor during a particular clock-cycle cannot affect the output of any other processor during that cycle; i.e. cascading is outlawed.

These properties constitute an informal definition of the set of systolic arrays. For the purposes of this thesis, the wires used for inputting and outputting signals to and from the array are ignored when assessing its systolicity. A formal definition of a "strictly systolic computation" will appear at the end of Chapter 3. A strictly systolic computation can usually be neatly implemented on a systolic array in a straightforward manner. However, it is possible that the natural implementation may not have a local interconnection structure, if by "local" it is meant that the only connections are between nearest-neighbours or second-nearest neighbours (arguably a good definition). A property which the implementation will have is that the patterns of input wires to any two subprocessors will have the same shape, i.e. they will be congruent, in the geometrical sense. In the literature, the arrays termed "systolic" have had both of the aforementioned properties, as in fact do the arrays described in this document, but my method only guarantees that the latter property will hold. Figure 1.1 shows four interconnection structures which would be allowed by my method; the bottom two structures include directly-connected processor-pairs which are not nearest-neighbour or even second-nearest-neighbour.

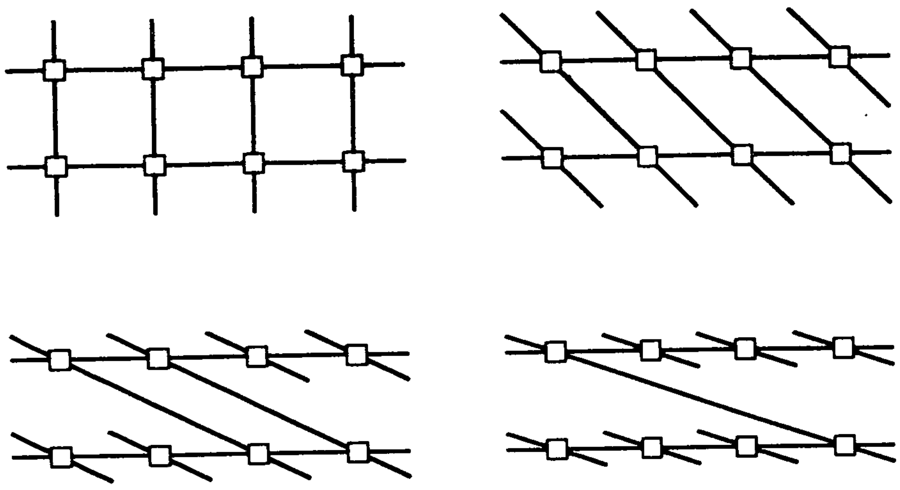


Figure 1.1 Interconnection structures allowed by my method

Figure 1.2 shows a sketch of a systolic array for a simulated annealing algorithm which uses the first-order Markov random field assumption to restore distorted images (see [SYKun88] pp. 592-599). Each processing element (subprocessor) stores an estimated value for a particular pixel in the undistorted image. As an initial estimate, the value of the corresponding pixel in the distorted image is used. The pixel-value-estimates are repeatedly updated using the value-estimates of the four nearest-neighbour pixels. A processing element / pixel $x_{i,j}$ is classified as odd or even depending on whether (i + j) is odd or even. At the beginning of each time- step, each processing element receives from its neighbours the current value-estimates of their corresponding pixels. If the parity (even/odd) of the processing element is the opposite of the parity of the time-step then the value-estimate of its pixel is updated; otherwise it is left unchanged. This means that when a processing element is updating its nearest neighbours are resting and vice-versa. The 50% processing element utilization can be increased by "processing element sharing". Similar arrays may be used to implement other algorithms such as the Jacobi method, the Gauss-Siedel algorithm and the Successive Over-relaxation algorithms for solving elliptical partial differential equations (see [SYKun88] p. 598).

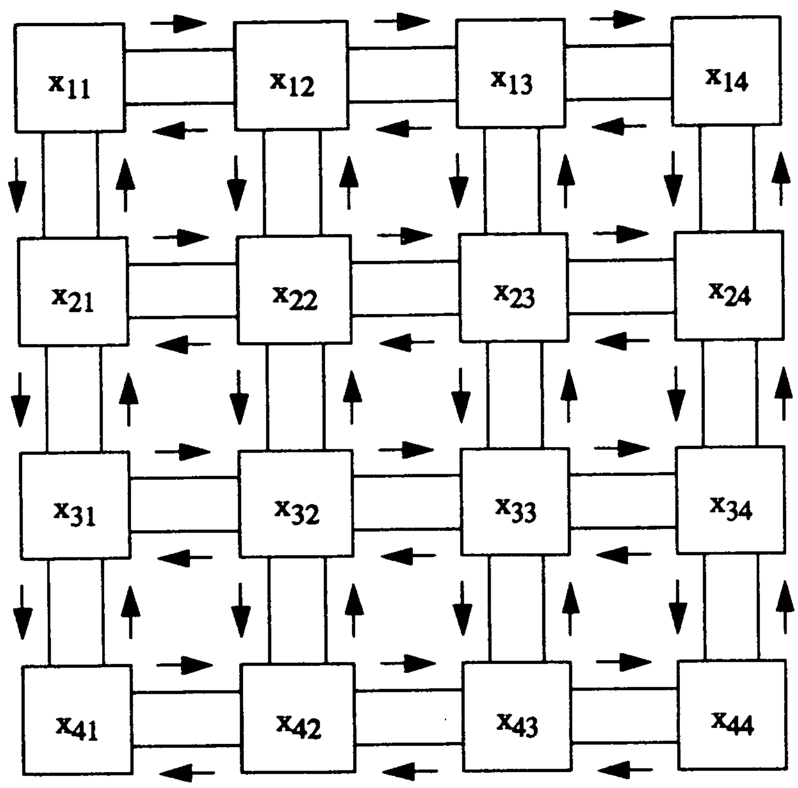


Figure 1.2 A systolic array for image restoration

1.2.2 How do SAs compare with other parallel processors?

First cousins to the systolic arrays are the *wavefront arrays*[SYKun88]. A wavefront array is like a systolic array, except that it is not clock-driven but data-driven, i.e. an operation takes place on a processor as soon as all its required inputs have arrived and the processor is available. Wavefront arrays are therefore asynchronous. The systolic and wavefront architectures are computationally equivalent. That is, if an algorithm can be executed by a systolic array then it can also be executed by a wavefront array, and vice versa.

Related to the systolic and wavefront arrays are processors of the following two types: *Single Instruction Multiple Data (SIMD)* and *Multiple Instruction Multiple Data*

(MIMD)[Rob84]. A SIMD array is similar to a systolic array except that control signals (instructions) are broadcast to the array: at each clock period all the processing elements receive an identical instruction to execute. Data may be broadcast. An MIMD array is asynchronous, the processing elements operating almost completely independently, each one having its own control unit and memory. As in a systolic or a SIMD array, the processing elements may communicate with each other and in addition may share memory. Figure 1.3 shows A Venn diagram of these parallel architectures. A detailed discussion of parallel architectures may be found in [Hwang84].

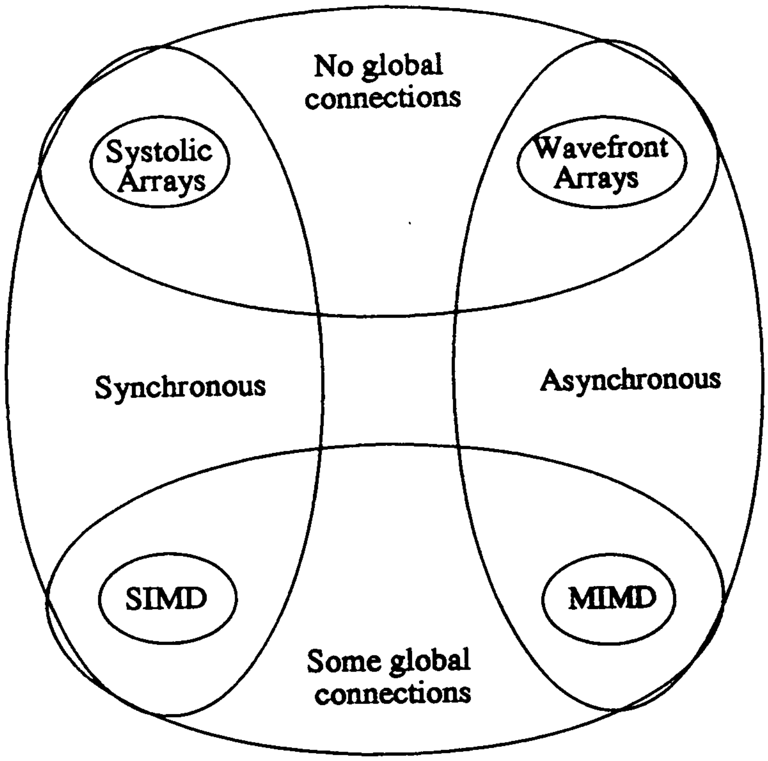


Figure 1.3 Venn diagram of parallel architectures

There are at least two advantages of systolic and wavefront arrays which SIMD and MIMD arrays do not have:

- On a VLSI chip, long wires are expensive in power consumption, area and execution time. Therefore, since these arrays don't have global communication, they are usually compact and cheap when built using VLSI technology.

- Input/Output bandwidth is small compared to computation bandwidth in these arrays, which also makes them suitable for implementation on VLSI chips.

Systolic and wavefront arrays have two more advantages, which SIMD and MIMD arrays do not necessarily have:

- Because systolic and wavefront arrays are repetitive and modular, they are relatively simple to design.

- The regularity of these arrays improves tessellation, which makes it possible to produce a more compact final implementation.

The drawback of systolic and wavefront architectures is that they can only be used to implement a restricted class of algorithms. Happily, as was mentioned earlier, many of the algorithms used in signal- and image-processing fall within that class. Some examples are:

- The Schur algorithm, which is used for certain cases of spectral estimation (a task which occurs in many fields)

- QR-factorisation, which is used for "beamforming", a process which occurs in many radar, sonar, seismic and communications systems to suppress unwanted interference.

- Kalman filtering, which is used in communications and control systems, and for tracking in radar and sonar processing

- Vector quantisation and dynamic time warping, which are used in speech-coding and speech-recognition respectively
- Rank order filtering, which is used for noise reduction and image enhancement
- Relaxation algorithms, which are used for image restoration
- Two-dimensional convolution, the Hough transform and two-dimensional normalised cross-correlation, which are used for edge-detection, curve-detection and template-matching respectively in the field of image analysis

To compare systolic arrays and wavefront arrays:

A wavefront array can be about twice as fast as the equivalent systolic array, since some operations may be allowed to execute faster than others rather than being restrained by other, slower, operations. The wavefront array generally also has the following advantages:

- It is easier to program.
- Large current surges are avoided. (These may occur in implementations of systolic arrays due to the synchronised change of components' states.)
- The problem of clock-skew is avoided completely. Clock skew means that the clock signal doesn't arrive at all the processors synchronously, due to propagation delays across the processor. It can be a problem with systolic arrays, especially large ones. However, one-dimensional systolic arrays may be synchronised by "pipelined clocks" [Fish85], and it is possible, in some two-dimensional arrays, to make clock skew less than it would otherwise

be, by routing as a recursive H-tree the wire which is to carry the clock signal, so that each processor is the same path-distance from the clock generator [SYKun88].

- Also the wavefront architectural style is more amenable to design for fault-tolerance.

However, if the systolic array is moderately-sized with simple processing elements then it may be more efficient than the equivalent wavefront array, since the disadvantages of the systolic style are not so pronounced, and the disadvantages of handshaking between the processing elements of the wavefront array are absent from the systolic array. These disadvantages are as follows:

- The average power drawn by the detection circuitry in wavefront arrays is greater than that drawn by a clock driver.

- More area is required by wavefront arrays than the corresponding systolic arrays which, as well as incurring the obvious costs, means that wavefront arrays are more subject to errors caused by radiation and processing defects.

- If "single-rail" logic handshaking is used then a wavefront array is often slower than the corresponding systolic array. "Double-rail" logic handshaking speeds things up but at the cost of an even greater area requirement: the area overhead of each wavefront array described in [McA92] is two to six times greater than its systolic equivalent.

1.3 Formal Design Methods

1.3.1 What are formal design methods and their advantages over informal methods?

A formal design method is a *well-defined process* for constructing a *well-defined object* which performs a *well-defined function*.

The function is like a label on a "black box" which tells you what the object inside does or should do. In technical jargon, the function on the label is called the *specification* and the object is called the *implementation*. If the object does indeed do what its label says it does, then we say that the implementation *satisfies* the specification. (Of course the object may in fact do *more* than its label requires, just as a Swiss army knife as well as cutting like a knife may also be used to open bottles and file fingernails.)

A formal design method for which a proof has been constructed that each implementation it produces satisfies its specification may be described as "verified" or "validated". Note that it is possible for a formal design process, even a verified one, to fail to come up with an implementation for a given specification.

Coming up with a suitable implementation will in general involve a series of design choices, which may be made by a human or by a computer.

Advantage 1

As was mentioned earlier, the fact that the product of a verified formal design method is proven correct with respect to its specification would make such methods useful for designing safety-critical equipment [Cohn88] for use in areas such as "defence", medicine and civil aviation. A formally verified design is also useful when many identical processors are used, in the area of telecommunications for example; it would be expensive to replace all of them if a design fault were discovered after manufacture. Such a design would also be useful where the processors are used in inaccessible places, for example, for sensing on pipelines or for surveillance in polar regions. [Birt88]

Advantage 2

If the designer is human, a formal design method may clarify his thoughts and lead him to solutions which he would not otherwise have thought of.

As was noted earlier, it is a good idea for each choice to be checked for correctness as soon as it is made.

If the design method is specialised (e.g. for designing ASICs with a particular architecture) then many of the choices are frozen. This has at least three advantages:

(1) The design process is comparatively fast since there are fewer design choices to make.

(2) The task of verification is eased.

(3) The design process is more likely to succeed, assuming that the specification is of a type which is appropriate to the method.

One disadvantage of a specialised method is that it may not allow the designer to proceed to valid designs which are perhaps more efficient than any which are allowed by the method.

1.4 A Formal Design Method for Systolic Arrays

The specifications to be input to the formal design method we'll be considering will consist of two parts. They will contain firstly a behavioural part, which specifies what calculation the final design must perform. Secondly they will contain the stipulation that the final design of a particular form which is easily implementable as a systolic array; to be more exact, the final design is to be an algorithm, each variable of which has an associated place and time of existence, and this space-time algorithm is of a particular form which is easily implementable on a systolic array. It should be noted that the behavioural part must itself be of a certain form, so the method can't necessarily be used to design a systolic array to do any arbitrary calculation. Sometimes it may be easy to re-write an unsuitable behavioural specification as a suitable one, but procedures for doing so are not examined in this thesis.

The formal design method is a transformational one. A sequence of designs (ALGORITHM, I_1, I_2, IMPLEMENTATION) is found such that each design satisfies the behavioural specification and IMPLEMENTATION is moreover easily implementable as a systolic array.

The designs are expressed in a formal design description language. The basic definition of the language is that of a computation. A computation has variables, which may be either inputs or outputs, and a function which relates the values of the outputs to the values of the inputs. The variables may for example be abstract, in which case the computation will express an abstract algorithm, or they may be space-time position vectors, in which case the computation will express an algorithm being executed on a processor. Computations of the latter type are called "space-time networks".

The designs in the design method are expressed as computations. Because computations can express abstract as well as "concrete" algorithms, it is possible to use the algorithm from the behavioural specification as the initial "design". (It may not be directly realisable in hardware, but that does not matter.) The complete specification for the final design is, informally: "the final design must 'simulate' the initial algorithm and be of a particular form which is easily implementable as a systolic array". The term "simulate" is defined formally in Chapter 3.

The function which produces a design in the sequence from its predecessor is called a design transformation. The sequence of design transformations associated with the method is called the transformation scheme. It consists essentially of three transformations. The initial design (algorithm) will have a regular data-dependency structure. However if it were to be directly realised in hardware, it might require non-local communication to carry some of the data. The object of the first transformation, called the "data-pipelining transformation", is to localise the data-dependencies. The combination of the initial control requirement and the one generated by data-pipelining may imply, in a direct implementation, control-broadcasting; the second transformation, the "control-pipelining transformation", removes the need for this. The design at this point has the pattern of a systolic implementation, except that its variables are still abstract. The third transformation, called the "scheduling and allocation transformation", maps the design into space-time by, for each variable, replacing its abstract position by the vector which designates the time and place of the its existence. The scheduling map maps the design into time and the allocation map maps the design into space. The complete transformation scheme is shown in Figure 1.4.

$$\begin{tikzcd}
\text{ALGORITHM} \arrow[d, "\text{Data-pipelining Transformation}"'] \\
I_1 \arrow[d, "\text{Control-pipelining Transformation}"'] \\
I_2 \arrow[d, "\text{Scheduling and Allocation Transformation}"'] \\
\text{IMPLEMENTATION}
\end{tikzcd}$$

Figure 1.4 The transformation scheme

It would be natural in the design method, to make the choice(s) associated with the ith transformation before those associated with the (i+1)th, for each i. However, there are reasons for making the choices in an 'unnatural' order. This results in the method not running quite parallel to the scheme, though the method is closely based on the scheme. The method consists of five stages (Figure 1.5).

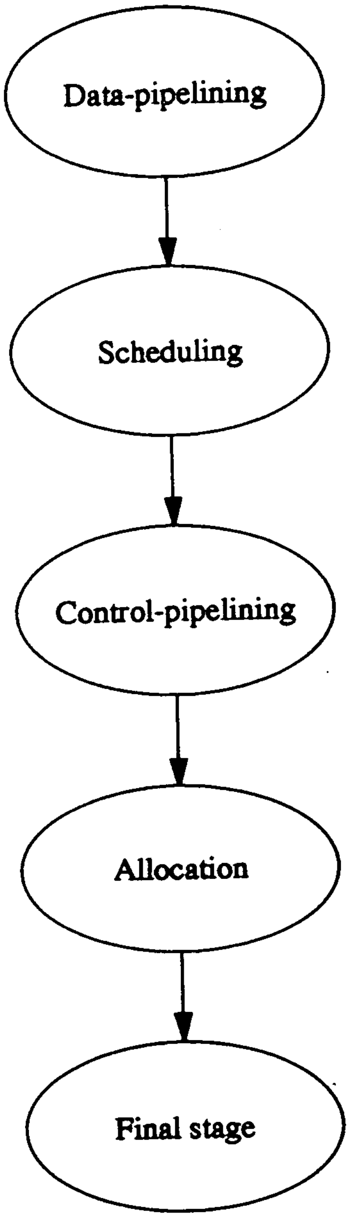


Figure 1.5 The design method

In the data-pipelining stage, the data-pipelining transformation and I_1 are found; in the scheduling stage, the schedule (the mapping of I_2 into time) is chosen, but it is not used until the final stage; in the control-pipelining stage, the control-pipelining transformation and I_2 are found; in the allocation stage, the allocation map (the mapping of I_2 into space) is chosen; in the final stage, IMPLEMENTATION is found, using the schedule and the allocation map chosen previously.

The reason that the schedule is chosen before control-pipelining is done is that the choices associated with control pipelining may be done in the light of the schedule, which may mean that an impasse which would otherwise have occurred can be avoided.

1.5 Overview of the Thesis

Chapter 2 discusses background material relevant to the formal design of systolic arrays. Chapter 3 presents the theoretical grounding of the new design method. Chapter 4 presents the method in detail with the aid of a simple example: convolution. In Chapter 5 the method is used to design a systolic implementation of the more complicated QR-factorisation algorithm, which is widely used for beamforming in antenna arrays. It is shown how different choices made during control-pipelining and allocation affect the design. Chapter 6 provides concluding remarks. In the appendices a proof is given that, subject to certain assumptions, a design produced by the method will satisfy its specification. Since the assumptions need to be made, the method cannot be described as "validated", but in Appendix H the assumptions required for two of the main theorems in the proof are proven.

2 Systolic Arrays and Formal Design Methods

This chapter starts with the presentation of two typical systolic arrays. The rest of the chapter consists of a survey of existing work in the same subject area as this thesis. The thesis presents a formal design method for systolic arrays; it belongs to two fields: formal design methods for parallel systems (not necessarily systolic), and design methods for systolic arrays (not necessarily formal). The overlap between the two fields will of course be particularly relevant. Discussion will also touch on other closely related areas such as design of regular arrays which are not quite systolic.

2.1 Examples of Systolic Arrays

Here are two examples of systolic arrays.

Example 1 (Figure 2.1) is a systolic array which implements bubble-sorting, a parallel sorting algorithm used in median filtering for noise reduction in images (see [SYKun88] pp. 122-3, 143-5, 587). It can be seen that the array consists of four processors, each connected to its neighbour(s) by communication wires.

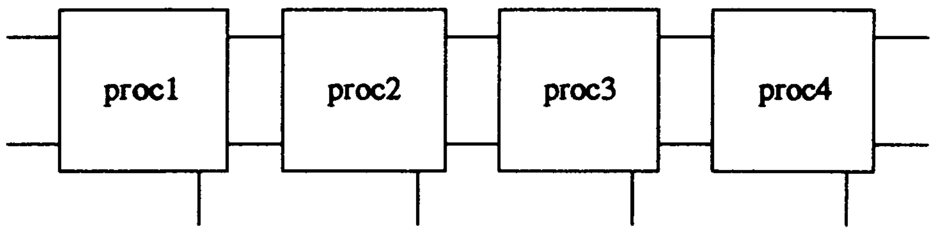


Figure 2.1 A systolic array for bubble-sorting

The input to the bubble-sorting algorithm is a sequence of four numbers, X_1, X_2, X_3 and X_4 say. The output is that sequence arranged in descending order of sample value,

Y_1, Y_2, Y_3 and Y_4. There is more than one variant of the algorithm, but all the variants operate by repeated transformation of the sequence and have as their middle phase the following characteristic motion of data. In one time-step, each datum in an odd position in the sequence is compared with the datum in the next higher (even) position. If the former is larger than the latter, the two are swapped; otherwise not. During the next time-step each datum in an odd position is compared with the one in the next *lower* even position and they are swapped if and only if the former is *smaller* than the latter. By this process each datum is buffeted towards its correct position. The algorithm is called "bubble-sorting" since the inputs can be thought of as mutually immiscible bubbles of liquid; each bubble moves to the level appropriate to its density. This method of sorting is similar to the way a squash ladder functions.

The variant of bubble-sorting presented here has an initial phase in which data is input to the array and a final phase in which data is output. The bubbling activity ramps up in the initial phase and ramps down in the final one. In order to define the algorithm formally it is helpful to introduce two sets of intermediate variables, $\{u_{i,j} : 0 \leq i \leq 4 \;\&\; 0 \leq j \leq 4\}$ and $\{d_{i,j} : 0 \leq i \leq 4 \;\&\; 0 \leq j \leq 4\}$. The former contains data which is "moving up" the sequence and the latter contains data which is "moving down". The recurrence relation defining the data-dependence is simply:

$$d_{i,j} := \min(d_{i,(j-1)}, u_{(i-1),j})$$

$$u_{i,j} := \max(d_{i,(j-1)}, u_{(i-1),j})$$

If $u_{0,1}$, $u_{1,2}$, $u_{2,3}$, $u_{3,4}$ are all $-\infty$ and $d_{1,0}$, $d_{2,0}$, $d_{3,0}$and $d_{4,0}$ are X_1, X_2, X_3 and X_4 respectively then $u_{4,1}$, $u_{4,2}$, $u_{4,3}$and $u_{4,4}$ will be Y_1, Y_2, Y_3 and Y_4 respectively. The data-dependence is illustrated in Figure 2.2. Each circle represents an operation consisting of the aforementioned pair of assignments for some i and j.

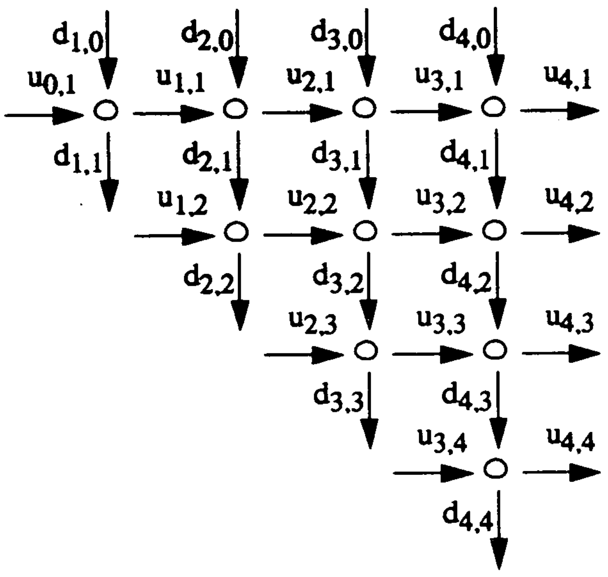


Figure 2.2 Sorting algorithm

Figure 2.2 simply shows the data-dependence of the bubble-sorting algorithm; it doesn't show when and where each operation occurs in the functioning of the systolic array (Figure 2.2). It is necessary to assign a processor and a time-step to each operation (these assignments are called allocation and scheduling respectively). Figure 2.3 shows this graphically. Each diagonal line corresponds either to a processor or to a point in time.

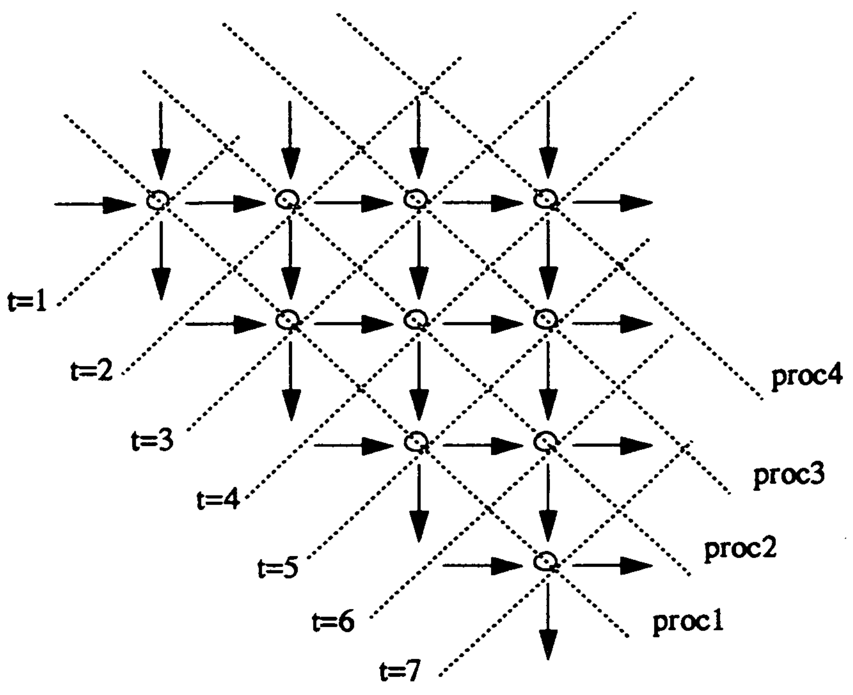


Figure 2.3 Schedule and allocation

Figure 2.4 shows four separate snapshots of the activity of the array, one being taken after each of the first four time steps (t is the time). Data which has just been generated is shown in bold print. For simplicity the multiplexers and the control signals have not been included. During the first time step, $d_{1,0}$ (X_1), which has been input to the first processor, is compared with $u_{0,1}$ (which is $-\infty$). The larger value, $d_{1,0}$, is passed to the second processor (as "$u_{1,1}$") while the smaller ($-\infty$) is discarded (as "$d_{1,1}$"). Figure 2.4(a) shows the situation when t=1. During the second time step, the second processor receives $u_{1,1}$ from the first processor as well as the new input value, $d_{2,0}$ (X_2), from the outside world. The two values are compared and, as before, the larger is passed to the right and the smaller to the left (as "$u_{2,1}$" and "$d_{2,1}$" respectively). Figure 2.4(b) shows the situation when t=2. $d_{2,1}$ is not discarded but caught by the first processor, where it is compared with $u_{1,2}$ ($-\infty$) in the next time step. Simultaneously, $u_{2,1}$ is being compared with the new input, $d_{2,0}$ (X_3), in the third processor. The larger values are passed to the right and the smaller to the left. Figure 2.4(c) shows the situation when t=3. In the fourth time step, the final input, $d_{4,0}$ (X_4) arrives at the fourth processor, where it is compared with the value just received from the third processor. A comparison is being done simultaneously on the second processor. At this time the first output, $u_{4,1}$, which is Y_1 (the largest of X_1, X_2, X_3 and X_4), appears at the fourth processor. Figure 2.4(d) shows the situation when t=4. As the sorting activity continues, $u_{4,2}$ (Y_2), $u_{4,3}$ (Y_3) and $u_{4,4}$ (Y_4) will be output in turn from the third, second and first processors respectively.

Figure 2.5 is Figure 2.4 with the variables replaced by their values, in the case where X_1, X_2, X_3 and X_4 are 4, 2, 7 and 1 respectively.

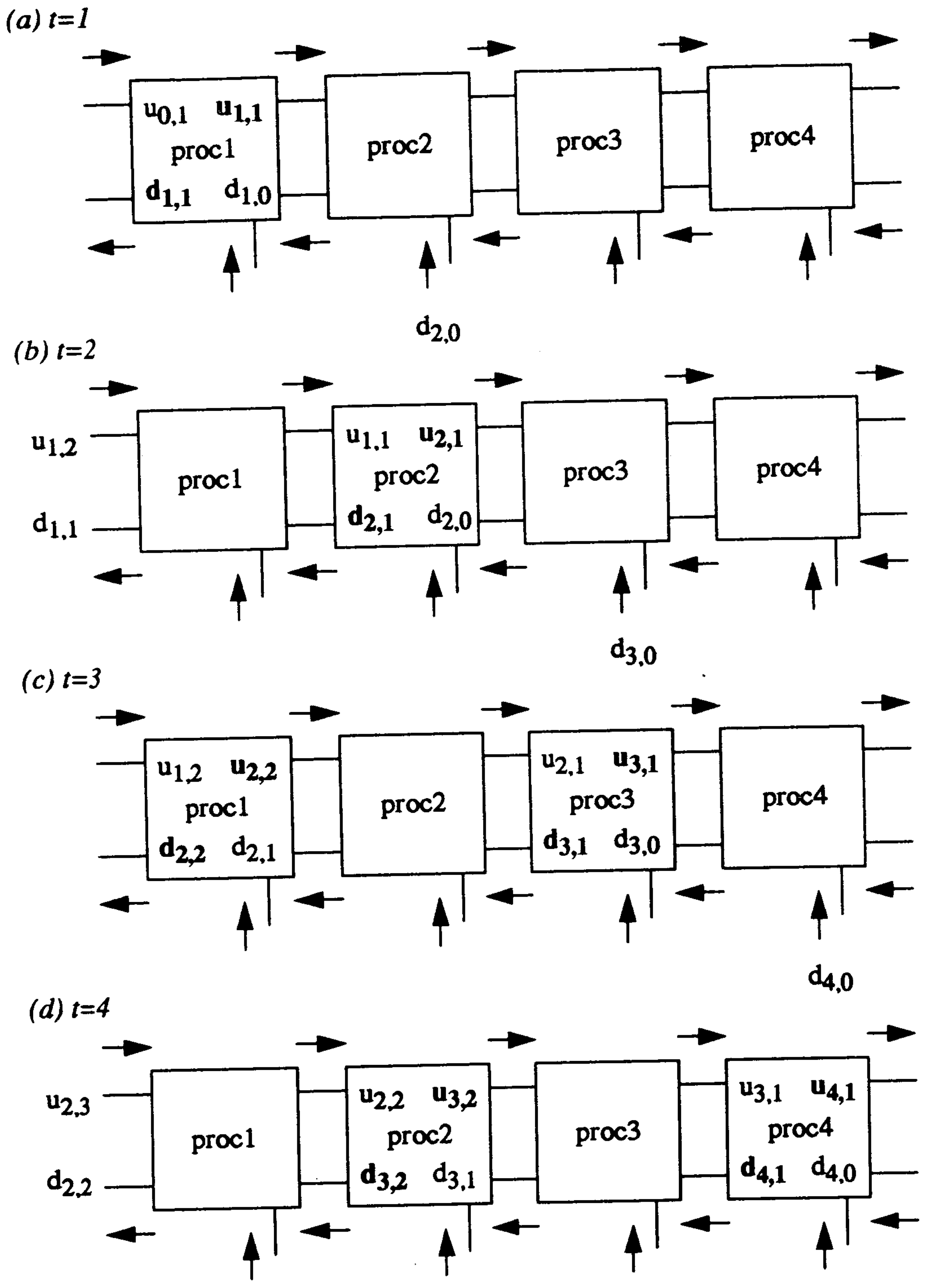


Figure 2.4 A bubble-sorter in operation

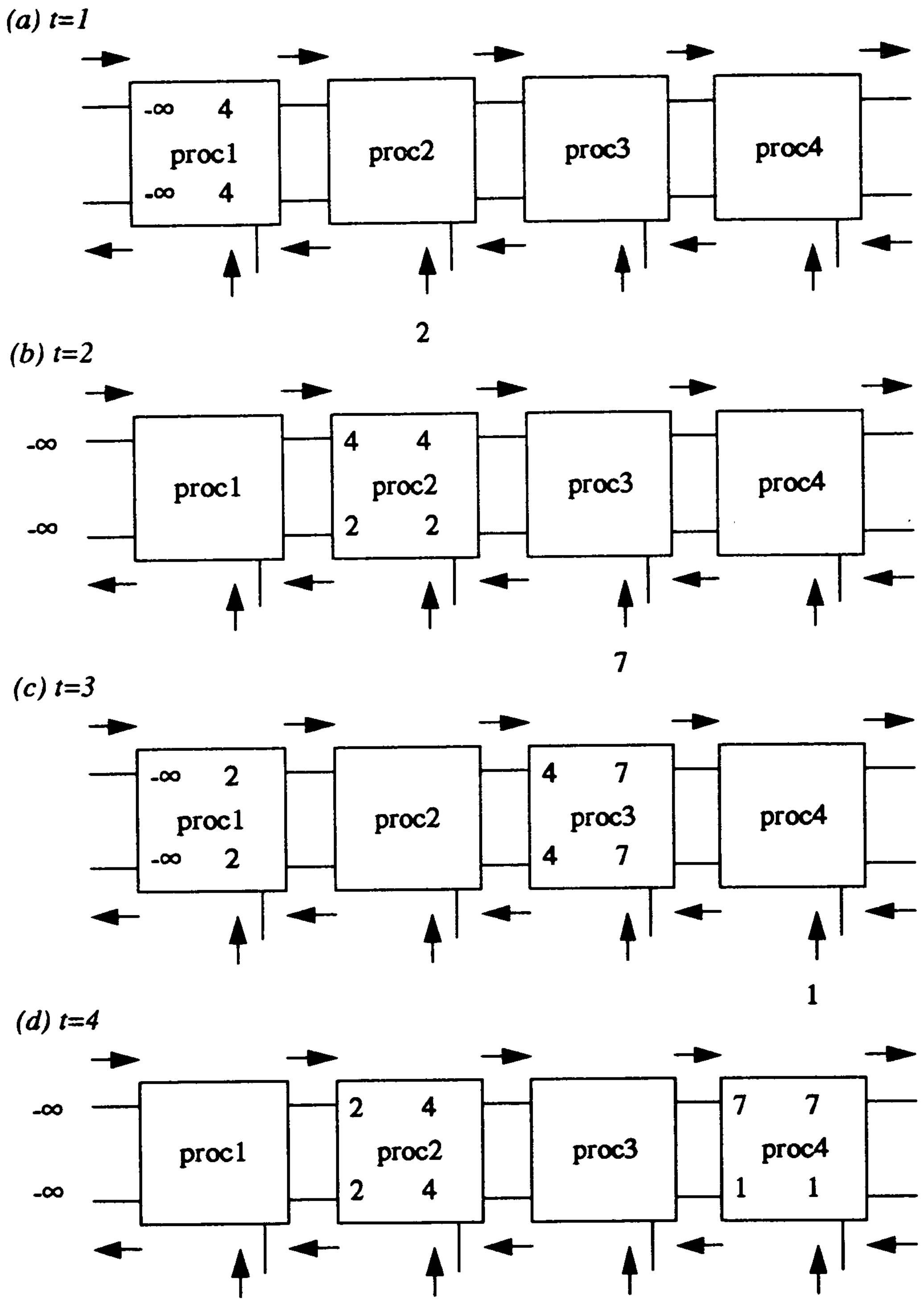


Figure 2.5 A bubble-sorter in operation: a numerical example

The second example of a systolic array is one which implements multiplication of band matrices (see [SYKun88] pp.177-8 & 200-1). A band matrix is one that has its non-zero values clustered in a band around its (top-left to bottom-right) diagonal. Let us suppose that A and B are band matrices. For all relevant pairs <i, j>, assume that $a_{i,j}$:= A(i, j) and $b_{i,j}$:= B(i, j). Let us assume that the "band" of A extends from two element-wide strips below the diagonal to one element-wide strip above it, and that B extends from one strip below the diagonal to two strips above it i.e.

$$a_{i,j} = 0 \quad \text{if } i > j+2 \text{ or } i < j-1$$

and

$$b_{i,j} = 0 \quad \text{if } j > i+2 \text{ or } j < i-1$$

Assume that C := AB and that, for all i and j, $c_{i,j}$:= C(i,j). The formula for the product of two matrices is given on page xiii. In the band matrix case, many of the products of the matrix elements are known to be zero, so we will omit these from the sums. Assume that high(i, j) := min(i+1, j+1) and that low(i, j) := max(i-2, j-2). Then we have

$$c_{i,j} = \Sigma_{j=\text{low}(i,j) \text{ to } \text{high}(i,j)}\, a_{i,k} * b_{k,j} \qquad \text{if } |i-j| \leq 3 \qquad \text{(i)}$$

$$c_{i,j} = 0 \qquad \text{otherwise} \qquad \text{(ii)}$$

So C is also a band matrix which has non-zero elements only in the band extending from three strips below to three strips above the diagonal. We may calculate the sums in (i) by introducing intermediate variables $s_{i,j,k}$ to hold the partial sums. Thus, if

$$|i-j| \leq 3$$

and

$$s_{i,j,(\text{low}(i,j)-1)} := 0$$

and

$$s_{i,j,k} := s_{i,j,(k-1)} + a_{i,k} * b_{k,j} \text{ when } k \leq \text{high}(i,j)$$

then

$$c_{i,j} = s_{i,j,\text{high}(i,j)}$$

This algorithm may be executed by a two-dimensional "hexagonal" array. The following six figures show snapshots of its state after each of the first six time-steps. The strategy is to send the band of possibly non-zero elements of A, spearheaded by $a_{1,1}$, into the array from the bottom-left and to send the band of B, spearheaded by $b_{1,1}$, into the array from the top-left. The partial sums flow from right to left through the array. When an element of A meets an element of B in a processor, their product is formed and added to the partial sum which has just arrived from the right. The new partial sum is passed out to the left. The element of A and the element of B flow out of the processor with out being deflected from their respective courses. The band of possibly non-zero elements of the product matrix C flows from right to left out of the top-left and bottom-left edges of the array.

Figure 2.6 (a) (t=0) shows $a_{1,1}$ and $b_{1,1}$ arriving at the array. In the first time-step, those elements pass into the array and $b_{1,2}$ and $a_{2,1}$ arrive. Figure 2.6 (b) shows the state of affairs when t=1. During the second time-step, the first interaction between the two matrices occurs: $a_{1,1}$ is multiplied by $b_{1,1}$, and the result is added to $s_{1,1,0}$ and passed out to the left as $s_{1,1,1}$; and $a_{1,1}$ and $b_{1,1}$ are each ready to pass out of the processor from the sides opposite their respective entrances. The state of affairs when t=2 is shown in Figure 2.6 (c). In this figure, more elements from A and B can be seen arriving. The value $s_{1,1,0}$ and all the other initial values for the partial sums must be zero; this is achieved by ensuring that all the values which are ever input to the array are zero, apart from the elements of A and B. In the third time-step, products are formed in three of the processors. The situation at the end of the third time-step is shown in Figure 2.6 (d). The processor on the far left has added its product to the partial sum just received from its rightward neighbour. The process continues, as can be seen in Figure 2.6 (e)(t=3) and Figure 2.6 (f)(t=4). Notice that in Figure 2.6 (e) the first output, $c_{1,1}$, emerges from the leftmost processor; in Figure 2.6 (f), $c_{2,1}$ and $c_{1,2}$ emerge from the neighbouring processors. Notice also how the activity of the processors in the array displays a cyclic rhythm, with each processor only doing a useful calculation one time-step in three.

The advantageous properties of systolic arrays can be clearly seen in these two examples (particularly the second): local communication, low ratio of input/output bandwidth to computation bandwidth, and a beautiful regularity in structure and activity which eases design and promotes, in the final implementation, high spatial compactness and processor utilisation. The claim of high processor utilisation may seem unfounded since utilisation seems to be 50% and 33% respectively in the first and second examples; but in each case if there is a sequence of tasks (bubble sorting or band matrix multiplication respectively) to be performed then, because of the regularity of the array's operation, it is possible to interleave the tasks to achieve virtually 100% utilisation.

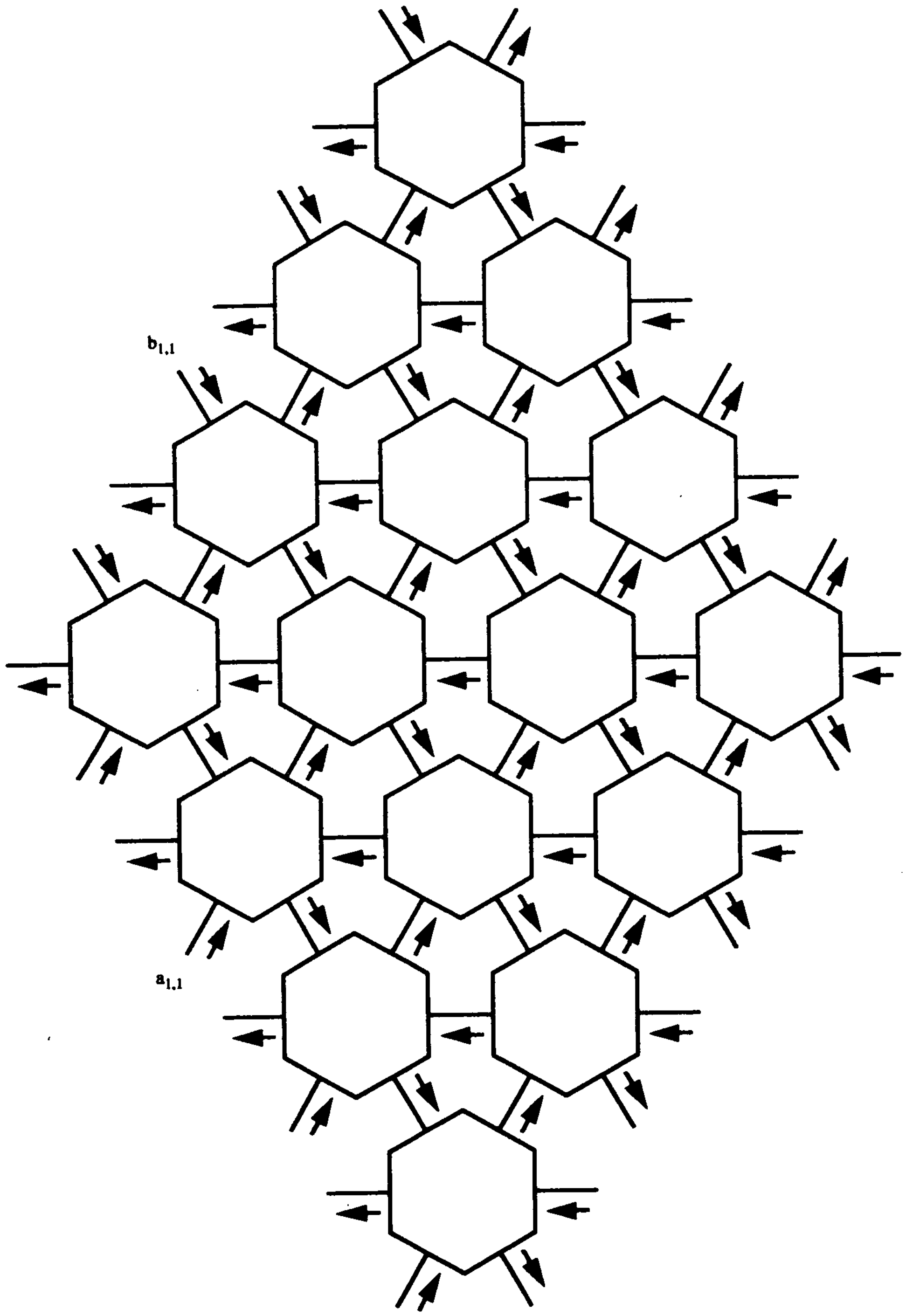


Figure 2.6 (a) Band matrix multiplier when t=0

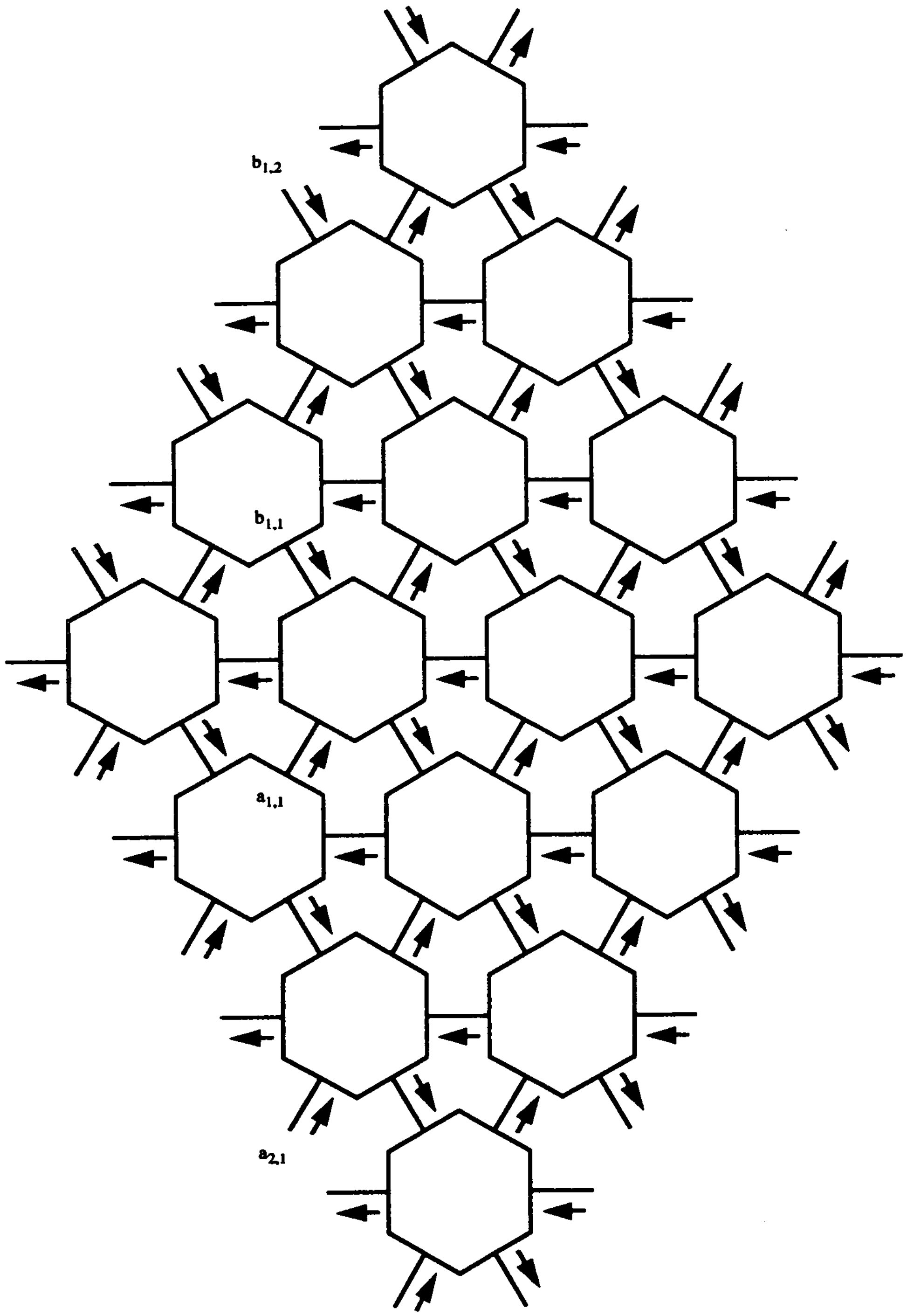


Figure 2.6 (b) t=1

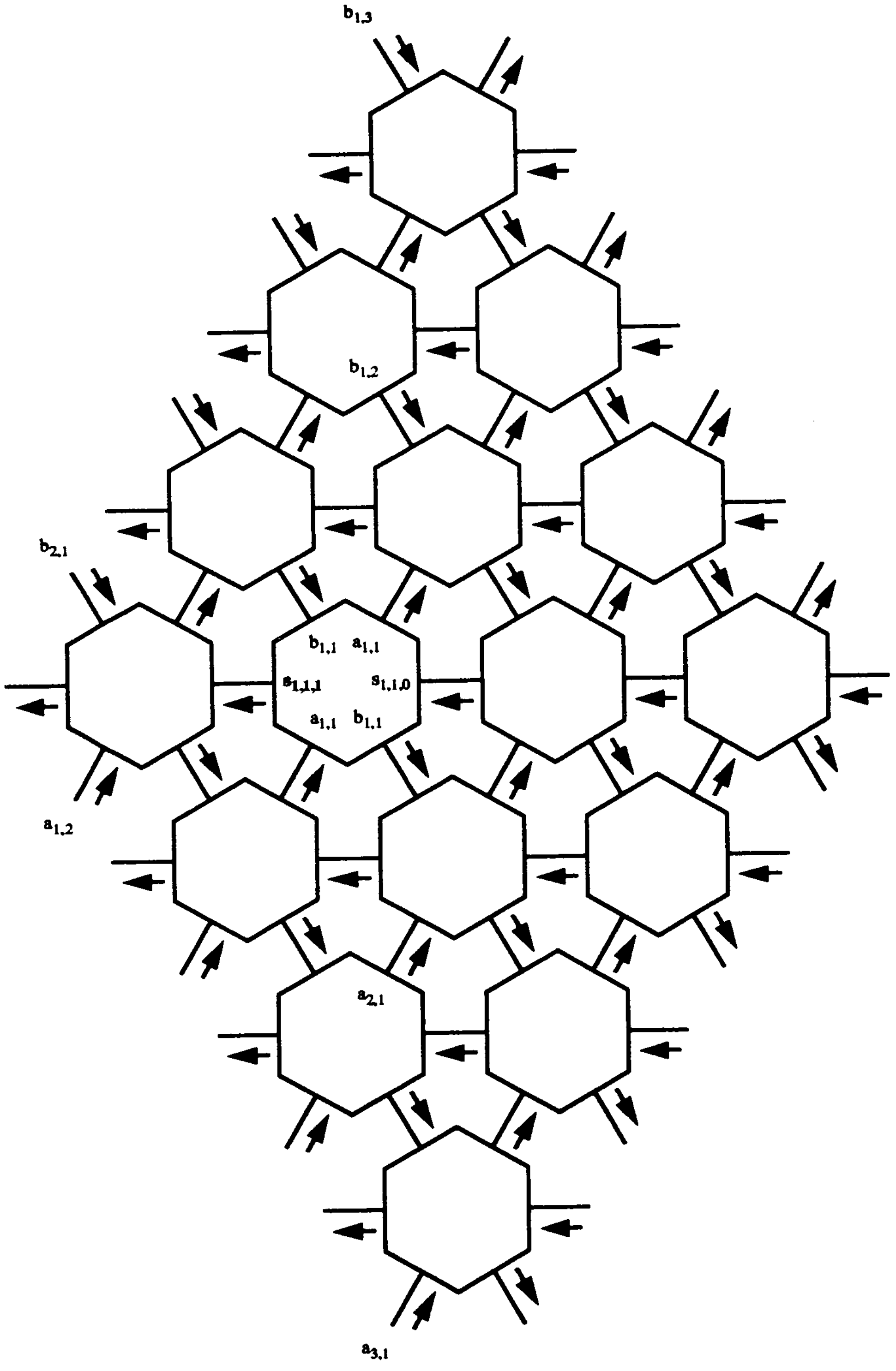


Figure 2.6 (c) t=2

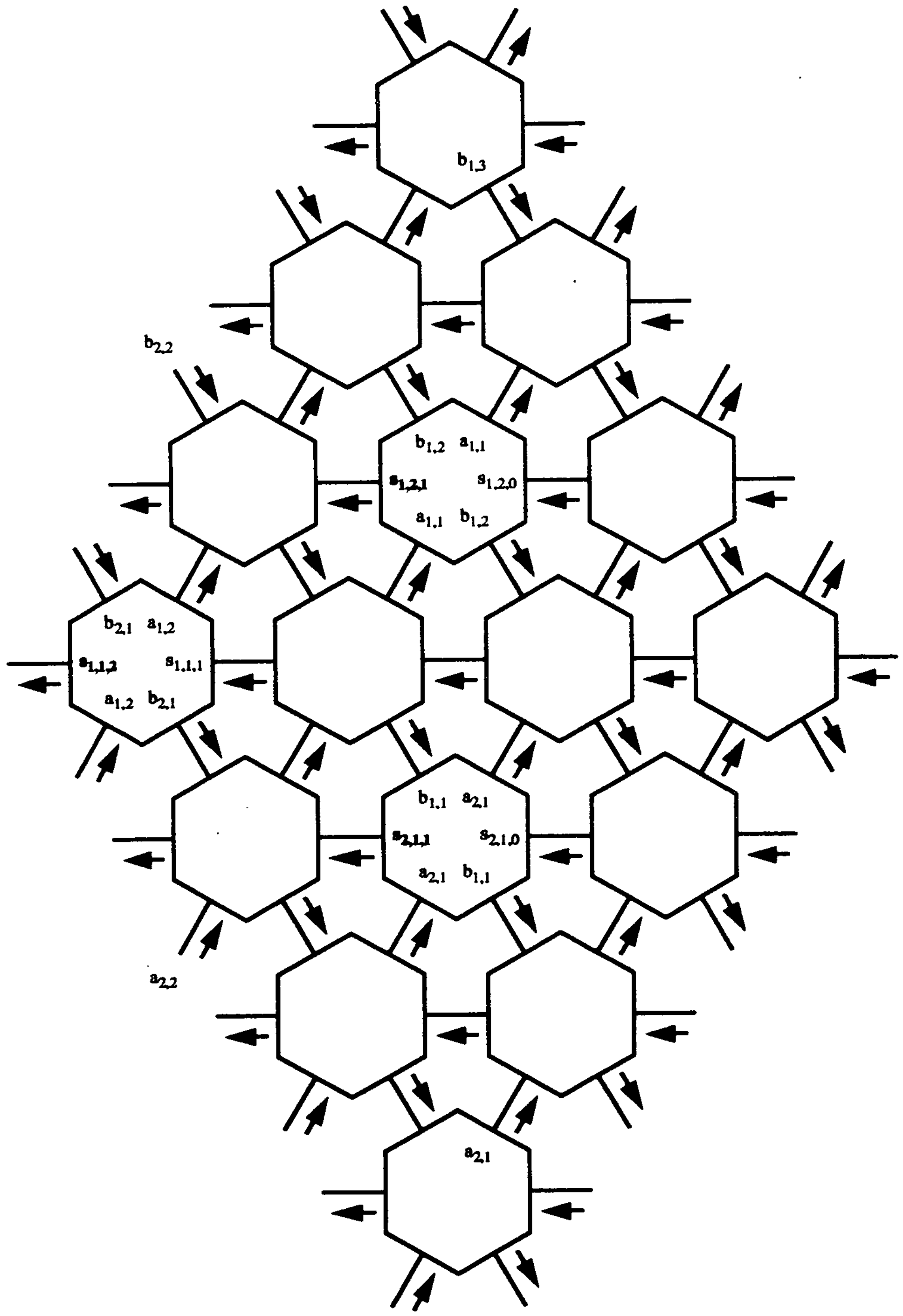


Figure 2.6 (d) t=3

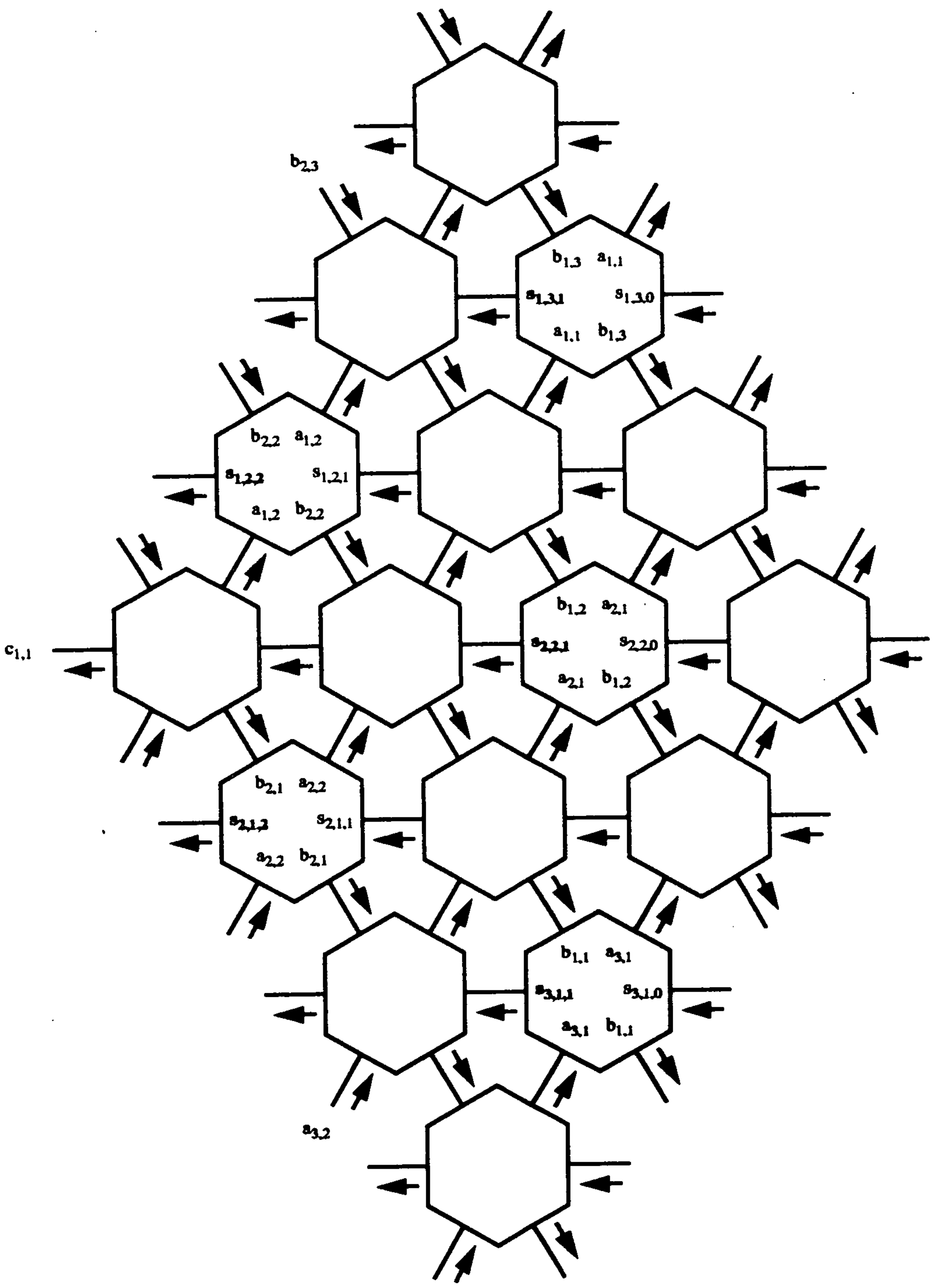


Figure 2.6 (e) t=4

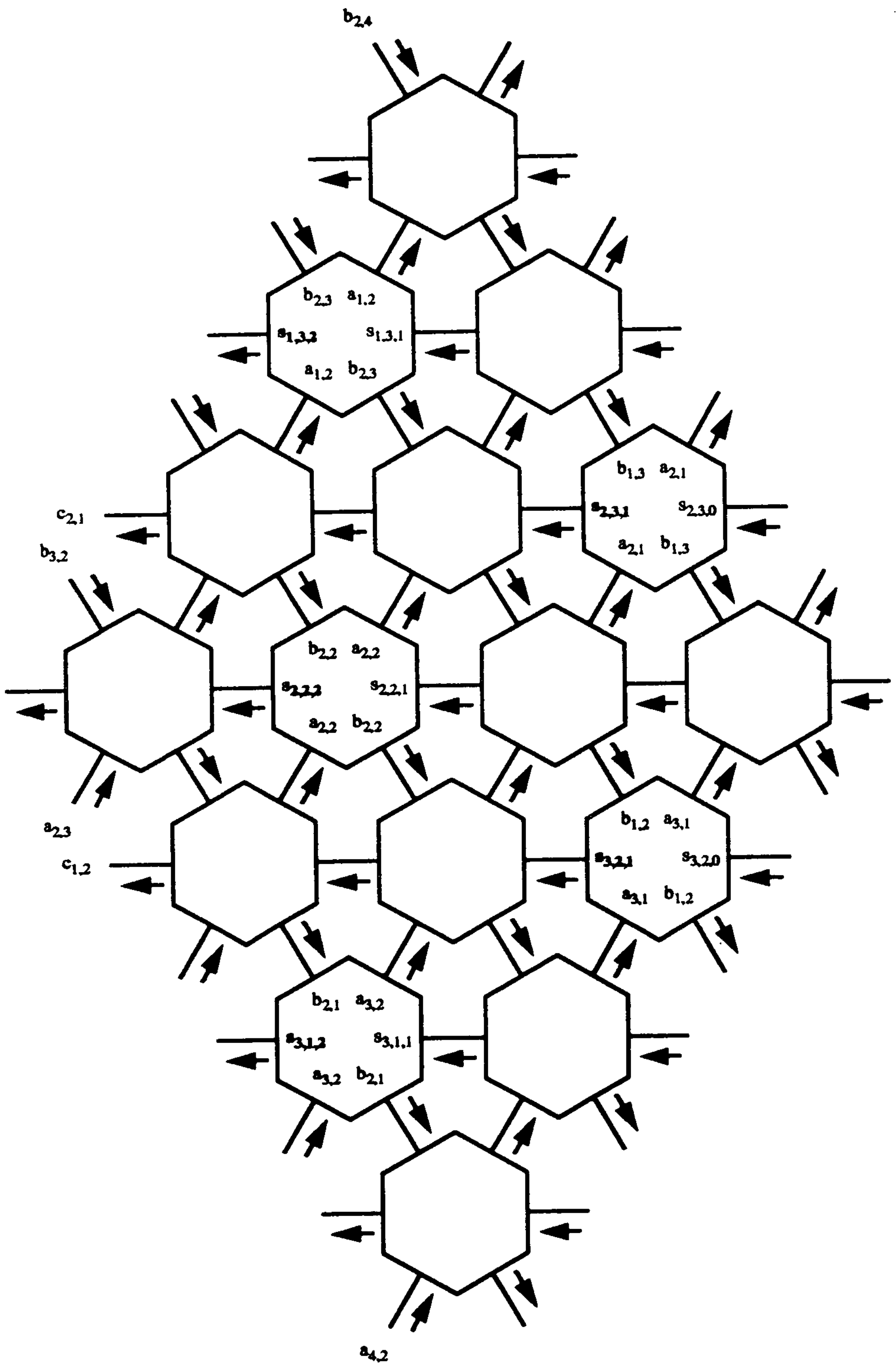


Figure 2.6 (f) t=5

2.2 Formal Design Methods

The work in this area may be classified according to the way parallel systems are modelled. A model may view the operation of a parallel system as a sequence of discrete events, without regard to time as an underlying metric, or it may view the operation of the system as a function of time. Models of the first type will be referred to as "sequential" and those of the second as "explicit-time". Sequential models are generally well-suited to describing the high-level behaviour of asynchronous systems, while the latter are better suited to describing synchronous systems and low-level behaviour in general.

Three principal languages for modelling systems sequentially are the "calculus of communicating systems" (CCS) [RMil80, RMil83, RMil89], CIRCAL [GMil83], and "communicating sequential processes" (CSP) [Hoare85]. CCS is similar to CSP in that two events cannot occur simultaneously. They have similar basic entities (an "agent" in CCS corresponds to a "process" in CSP) and are overall roughly, if not exactly, equal in expressive power; however, the set of ways in which the basic entities may be combined and the concept of equality differ between the two languages. CIRCAL has a style very similar to CCS; the key difference is that *simultaneous* events can be expressed in CIRCAL. The programming language, "occam" [Jones87, Jones88, Wex89], was designed for programming parallel systems, specifically the INMOS transputer, and is very closely related to CSP. The major difference is the lack of recursion: this was found not to be implementable in general.

Explicit-time models of parallel computer systems usually subdivide the behaviour of a system by focusing on the value of individual "ports" as a function of time. The models can be classified by how these port-functions are related to each other. In what will be termed *functional* models, all ports are classified as either "input" or "output", and each port-function of an output port is a function of the port-functions of the input ports. In *relational* models the ports are not divided into "input" and "output"; the behaviour of the system is simply a relation between all the port-functions. Work on functional models has been done by S. Johnson [John83], M. Sheeran [She84], and by A.R. Martin and J.V. Tucker [Mar87]. S. Johnson's thesis centres round the observation that a certain simple type of recursive algorithm can be directly implemented as a sequential processor. His method is closely related to methods for designing systolic arrays; in fact his sequential processors are zero-dimensional systolic arrays. M. Sheeran's language, μFP, was created for use in VLSI design. Martin and Tucker introduce an assignment language based on a functional model; the language restricts the output values at a point in time to be a function of the time and of the inputs at the immediately previous time (time is modelled on the integers rather than the real numbers); in other words there is no long-term memory. This notation is intended for simulation and testing of synchronous arrays. Work on relational models has been done by M. Gordon [Gor88], by M. Fourman [Mayg91], by M. Sheeran [She86, She88a] and by Luk and Jones [Luk88a, Luk88b]. M. Gordon's HOL is a theorem-prover for hardware verification. It is based on higher-order logic, as are all the explicit-time models. The port-functions are first-order entities; the behaviour of a circuit, being a relation between port-functions, is a second-order entity. M. Fourman's LAMBDA system has a theorem prover at its heart, and is designed for integrated synthesis and verification. Sheeran's language, Ruby, was developed from μFP; she has used it to design regular arrays, and incidentally formalises two techniques used by systolic array designers: "retiming" and "slowdown" [She88b]. Luk and Jones' work is a development of Sheeran's.

2.3 Design of Systolic Arrays

2.3.1 Beginnings

A seminal work in the area of systolic array design is [Karp67]. A "system of uniform recurrence equations" (SURE) is defined to be essentially an algorithm of a certain type. The authors give necessary and sufficient conditions for there to be a schedule for any SURE of a certain type. SUREs are significant since, by choosing a certain schedule and allocation function, it is often possible to implement them using a systolic array.

In the late seventies and early eighties, H.T. Kung and his group at Carnegie-Mellon University showed how certain algorithms could be implemented on synchronous, virtually homogeneous VLSI arrays with regular, local interconnections, which they called "systolic arrays". They showed that, because of the arrays' regularity and in particular the local communication structure, the arrays were particularly efficient. The work of H.T. Kung et al. was immediately followed by the creation of many systolic array designs by them and others [Fost80, Quin86].

2.3.2 A developing discipline

As understanding of systolic arrays and their associated algorithms grew, attention began to be paid to the development of systematic design methods. Rao [Rao85] investigates a major class of algorithms called "Regular Iterative Algorithms" (RIAs) which are essentially the same as the systems of uniform recurrence equations in [Karp67]. He carefully and precisely defines a systolic array, and shows that each RIA *of a certain type* may be directly implemented by a systolic array, (giving a procedure which produces a variety of systolic implementations for such an RIA) and that conversely every systolic array directly implements such an RIA. He also extensively analyses RIAs and provides a procedure for implementing them, and for deriving them from more general problem descriptions. Similar but less comprehensive work is described in [Far87].

One of the key properties of a regular iterative algorithm is that its "dependencies" are "uniform"; that is, if an indexed variable x(p) say depends on y(p-q) for some vectors p and q, then, for all vectors p' in the index space, x(p') depends on y(p'-q). This implies that, when the data-flow graph of the RIA is embedded in a natural way in Euclidean space, the set of vectors representing the flow of data into each node is the same, regardless of which node is chosen. "Linear Recurrence Algorithms" (LRAs), such as Gaussian Elimination and Gauss-Jordan approximation, do not necessarily have this property. A method for implementing LRAs as systolic arrays by first making the dependencies uniform is presented in [Quin89]. The set of "Systems of Affine Recurrence Equations" (SAREs) is similar, if not identical, to that of LRAs. The implementation of SAREs is tackled by Yaacoby and Cappello in [Yaa88] and S. Rajopadhye in [Raj89]. They also make the dependencies uniform as an intermediate step. [Raj89] also deals with control signals. Sometimes it isn't immediately possible to make the dependencies uniform; in [Raj90] transformations are introduced which transform awkward SAREs into SAREs of which the dependencies can be made uniform. The problem of finding affine schedules for SAREs is tackled in [Del86] and [Yaa89], separately from the problem of making their dependencies uniform. [Del86], [Yaa89], and [Rao85] implicitly or explicitly move into the area of non-systolic implementation. Other papers which deal with non-systolic implementation are [Roy89], [Teich91] and [VanSw91]. [Roy89] deals with the implementation of RIAs, such as pivoting algorithms in linear algebra and certain two-dimensional filters, which are not directly implementable as systolic arrays. [Teich91] deals with algorithms which are piecewise regular; the resulting arrays have a "dynamic configuration structure". [VanSw91] deals with algorithms in which the data-flow is even less regular than in LRAs and SAREs. In implementations of the style aimed for, the processing elements will calculate and communicate synchronously; however, their interconnections may be neither homogeneous nor local.

Several researchers express the algorithmic specification in other ways [Huang87, Len90, Xue90, Len91, Lee90, Ib90, Chen91]. However, the differences between their languages and the systems-of-recurrence-equations style is, I believe, superficial. [Huang87] presents a design method for systolic arrays. From the algorithmic specification a sequential "execution" or "trace" is derived; this is then parallelised; finally the trace is scheduled and allocated using the functions "space" and "time". The auxiliary functions, "flow" (encapsulating the velocity of data movement) and "pattern" (encapsulating the initial position of the data) are defined. [Len90], [Xue90] and [Len91] build on the work in [Huang87]. [Len90] discusses the design of a systolic array for pyramidal algorithms, [Xue90] discusses the description and design of one-dimensional systolic arrays, and [Len91] presents a scheme for compiling imperative or functional programs into "systolic programs". Design of one-dimensional systolic arrays is also the subject of [Ib90]. [Lee91] investigates the mapping of p-nested loop algorithms into q-dimensional systolic arrays (where $1 \leq q \leq p-1$).

K.Culik [Culik84, Culik85] takes a subtly but significantly different approach from the above in that his specification language doesn't even implicitly embed the algorithm in Euclidean space; in other words it is topological and not geometrical.

Several papers specialise in the design of particular types of systolic array. [Xue90] and [Ib90] have already been mentioned as dealing with the design of one-dimensional arrays; the second part of [McC87] gives examples of bit-level systolic arrays, though not a general design method for them; [Kunde86] and [Tensi88] describe work on "Instruction Systolic Arrays", where the processing elements are controlled by instructions which flow through the array in addition to the data.

Other papers present methods which produce optimal designs, or at least facilitate the choosing of an optimal design. In [Li85] the initial algorithm is constrained to be a "linear recurrence". (The class of linear recurrences includes matrix multiplication and related algorithms. These linear recurrences don't seem to bear any relation to the LRAs in [Quin89].) The design task is formulated as an optimization problem and a toolkit for solving the problem is described. [Shang89] addresses the problem of finding optimal linear schedules for an algorithm modelled as a set of indexed computations. [Chen91] presents a method for finding optimal schedules for one-dimensional "linear recurrence algorithms" such as the algorithm for an ARMA filter, which is used in signal prediction and spectrum analysis.

The papers [Raj86], [Ling90] and [LeV85] are more oriented towards formal verification of systolic arrays than the above work. [Raj86] uses techniques which have been used for verifying programs and applies them to the verification of systolic architectures. The verification problem is divided into three parts: the verification of the data representation, the processing elements and the composition of the processing elements. [Ling90] introduces a new formalism called "systolic temporal arithmetic" for specifying and verifying systolic arrays. Two plus points are that it is tailor-made for systolic arrays and is therefore efficient, and it can be unified with interval temporal logic "multilevel reasoning of systolic arrays". [LeV91] introduces a language called ALPHA which is based on recurrence equations. It is a direct descendant of a language called LUSTRE [Caspi87], which is descended from LUCID [Ash77]. It seems to be simple and straightforward.

2.4 Summary

In this chapter two examples of systolic arrays were described, and a survey was given of related work done on formal design methods and on the design of systolic arrays.

The following chapter lays the theoretical foundations for the formal design method to be presented in Chapter 4.

3 Computations and Recurrences

In this chapter the concepts are defined which are required in order to define and discuss the formal design method to be presented in Chapter 4. Firstly, the concept of a computation is defined along with three operations on computations and one relation, simulation (see page 12). Then four useful types of computation are introduced: embedded computations, recurrences, space-time networks (see page 12) and strictly systolic computations. An embedded computation is composed of subcomputations which are "located in" Euclidean space. A recurrence is a type of embedded computation; recurrences are exhibit a regularity which makes them useful for the design of systolic arrays; two types are of particular usefulness, "affine recurrences" and "uniform recurrences". The input to the design method has an affine recurrence as its main part. A space-time network is an embedded computation which models an algorithm executing on hardware, in that the Euclidean space is identified with space-time and, in the light of this identification, no data is consumed before it is produced. If a space-time network is also a uniform recurrence, then it is called a "strictly systolic computation". The output of the design method has a strictly systolic computation as its main part. Given a strictly systolic computation, one can easily design a systolic array to implement it.

3.1 Computations

A computation is similar to a function, where the inputs and outputs are given names so that they can be reasoned about separately from the function, and separately from the values they hold.

A *computation* is defined to be a triple, <I, O, F>, where I is a finite set of input variables, O is a finite set of output variables, and F is a functional[1] such that, if v_{in} is a function from input variables to their values, then $F(v_{in})$ is a function from output variables to their values. I and O must be disjoint.

1. see Terminology

We can define selector functions as follows (v is a function with domain $I \cup O$):

<u>*In*</u>(<I, O, F>)	:= I;
<u>*Out*</u>(<I, O, F>)	:= O;
<u>*Vars*</u>(<I, O, F>)	:= $I \cup O$;
<u>*Fun*</u>(<I, O, F>)	:= F;
<u>*Rel*</u>(<I, O, F>)(v)	$\Leftrightarrow$

(there exist v_{in}, v_{out} such that $v = v_{in} \cup v_{out}$ and $F(v_{in}) = v_{out}$)

Rel uniquely defines F since I and O are disjoint. (There is a simple proof of this.) Also

$$\text{Rel}(\langle I, O, F\rangle)(v) \Leftrightarrow F(v|_I) = v|_O$$

The function v is called a <u>*valuation.*</u> on $I \cup O$.

So a computation is defined uniquely by <In(C), Out(C), Fun(C)>, or, alternatively, by <In(C), Out(C), Rel(C)>. It is often more convenient to use the latter characterization (as in the four definitions given below).

We will now define three functions (composition, hiding and renaming) and one relation (simulation) on computations.

3.1.1 Composition

Consider the computations PLUS and PLUS', defined as on page 49:

In(PLUS) := {A, B}
Out(PLUS) := {TEMP}
Rel(PLUS)v $\Leftrightarrow$ v(TEMP) = v(A) + v(B)

In(PLUS') := {TEMP, C}
Out(PLUS') := {D}
Rel(PLUS')v $\Leftrightarrow$ v(D) = v(TEMP) + v(C)

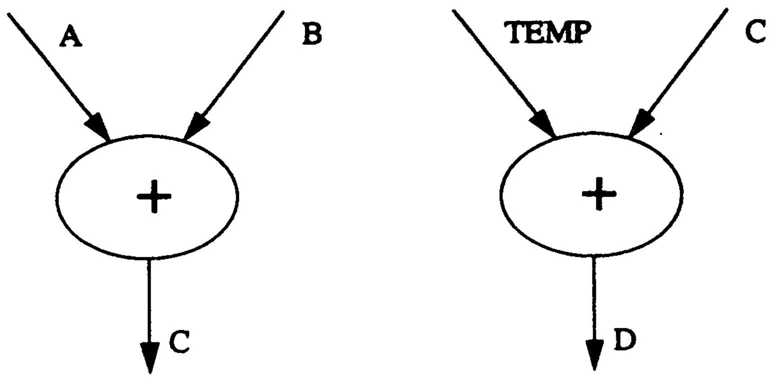


Figure 3.1 PLUS and PLUS'

It is possible to combine (technically, "compose") these in the obvious way to form TRIPLE-ADD defined as follows:

$$\begin{aligned}
&\mathrm{In(TRIPLE\text{-}ADD)} &&:= \{A, B, C\} \\
&\mathrm{Out(TRIPLE\text{-}ADD)} &&:= \{TEMP, D\} \\
&\mathrm{Rel(TRIPLE\text{-}ADD)}v &&\Leftrightarrow v(TEMP) = v(A) + v(B) \qquad \text{and} \\
& && \ v(D) = v(TEMP) + v(C)
\end{aligned}$$

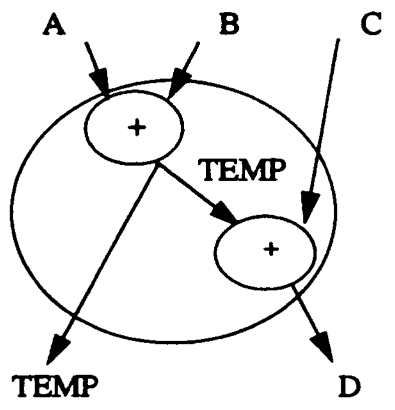


Figure 3.2 TRIPLE-ADD

In the general case, the set of output variables of the computation resulting from the composition of two or more computations is the union of the sets of output variables of the component computations. The set of input variables of the resulting computation is the union of the sets of input variables of the component computations minus the set of output variables of the resulting computation. This implies that, if a variable is both an input (of one component computation) and an output (of another component computation) then it counts as an output of the resulting computation. The relation of the resulting computation is the conjunction of the relations of the component computations. The symbol used for composition is "||". Here is its definition:

$\|_{j \in J} C_j \quad := \quad C$ (J is some finite indexing set),

where

$$\mathrm{Out}(C) \quad = \quad \bigcup_{j \in J} \mathrm{Out}(C_j)$$

$$\mathrm{In}(C) \quad = \quad \bigcup_{j \in J} \mathrm{In}(C_j) - \mathrm{Out}(C)$$

$$\mathrm{Rel}(C)(v) \iff \text{For all j. } \mathrm{Rel}(C_j)(v|_{Vars(C_j)})$$

The elements of the set $\{\mathrm{Out}(C_j) : j \in J\}$ are assumed to be mutually disjoint.

v is a function on Vars(C), which is the reason why it must be restricted to Vars(C_j) in the definition of Rel(C). || may be written as an infix operator i.e. $\|_{j \in \{0,1\}} C_j$ may be written $C_0 \| C_1$. Note that $C_0 \| C_1 = C_1 \| C_0$; no order need be specified for the composition of functions.

In(C) and Out(C) are obviously finite.

Instead of defining Rel(C), we may define Fun(C):

$$\begin{aligned} \mathrm{Fun}(C)(v_{in}) \quad = \quad & v_{out} \Leftrightarrow \text{for all } j \text{ in } J \\ & \text{there exist } v_{in(j)}, v_{out(j)} \text{ such that } \mathrm{Fun}(C_j)(v_{in(j)}) = v_{out(j)} \\ & \text{and } v_{in} \cup v_{out} = \bigcup_{j \in J} (v_{in(j)} \cup v_{out(j)}) \end{aligned}$$

In other words:

$$\begin{aligned} \mathrm{Fun}(C)(v_{in}) \quad = \quad & v_{out} \Leftrightarrow \text{for all } j \text{ in } J \\ & \mathrm{Fun}(C_j)(v|_{\mathrm{In}(C_j)}) = v|_{\mathrm{Out}(C_j)} \text{ where } v = v_{in} \cup v_{out} \end{aligned}$$

Composition may not always be well-defined since the function of the resulting computation may not be well-defined. For example, if two or more of the component computations share and output then there may be a clash when the computations are united. Even if the sets of output variables are mutually disjoint, the composition may not be well-defined. For example, let PLUS'' be defined as follows:

In(PLUS') := {TEMP, C}
Out(PLUS') := {D}
Rel(PLUS')v ⇔ v(D) = v(TEMP) + v(C)

Let CIRC-ADD be PLUS || PLUS'' and let us assume that addition is being performed on *integers*. If $v_{in}(A) = v_{in}(C) = 1$ then there is no possible v_{out} for which F(CIRC-

ADD)v_{in} = v_{out} (see Figure 3.3). Moreover if $v_{in}(A) = v_{in}(C) = 0$ then there are infinitely many v_{out} for which F(CIRC-ADD)v_{in} = v_{out} (see Figure 3.4).

In the body of the thesis we will generally assume that all computations are well-defined. In the appendices the assumptions of well-definedness will be explicitly stated.

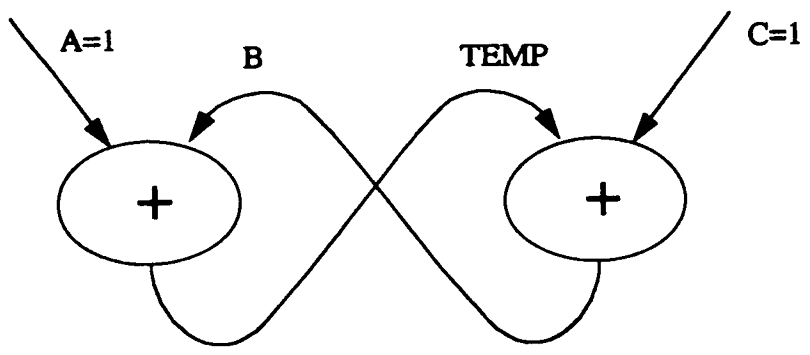


Figure 3.3 CIRC-ADD: no solution

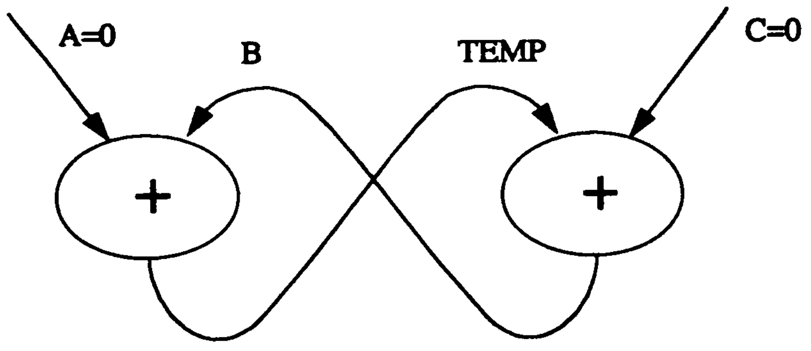


Figure 3.4 CIRC-ADD: many solutions

Figure 3.5 shows a more complicated example of composition.

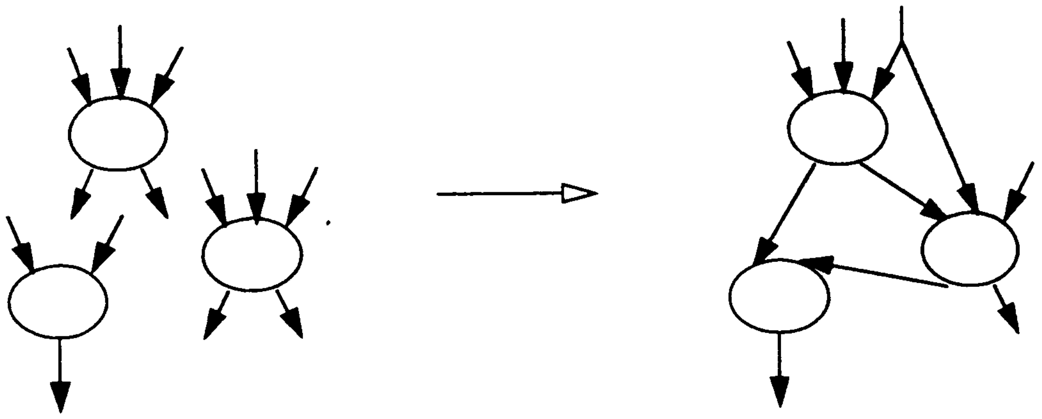


Figure 3.5 Composition: a more complicated example

3.1.2 Hiding

Internal and/or irrelevant variables may be "hidden" by removing them from the input or output variable sets. Hiding is especially useful as a sequel to composition, in order to hide the internal variables. Outputs can be hidden simply by ignoring them but, as a consequence of the way we define hiding, an input can only be hidden if its value is always the same (i.e. it is a constant) or if its value has no effect on the value of any unhidden output. The symbol for hiding is "\". Here is the definition:

$$\mathrm{In}(C\backslash \mathrm{Varset}) := \mathrm{In}(C) - \mathrm{Varset}$$
$$\mathrm{Out}(C\backslash \mathrm{Varset}) := \mathrm{Out}(C) - \mathrm{Varset}$$

and for all valuations v on Vars(C) - Varset,

$$\mathrm{Rel}(C\backslash \mathrm{Varset})(v) \Leftrightarrow (\text{for all valuations } v' \text{ on } \mathrm{Vars}(C), \mathrm{Rel}(C)v' \Rightarrow \\ (v'|_{\mathrm{In}(C)-\mathrm{Varset}} = v|_{\mathrm{In}(C)-\mathrm{Varset}} \Rightarrow \\ v'|_{\mathrm{Out}(C)-\mathrm{Varset}} = v|_{\mathrm{Out}(C)-\mathrm{Varset}}))$$

In(C\Varset) and Out(C\Varset) are obviously finite.

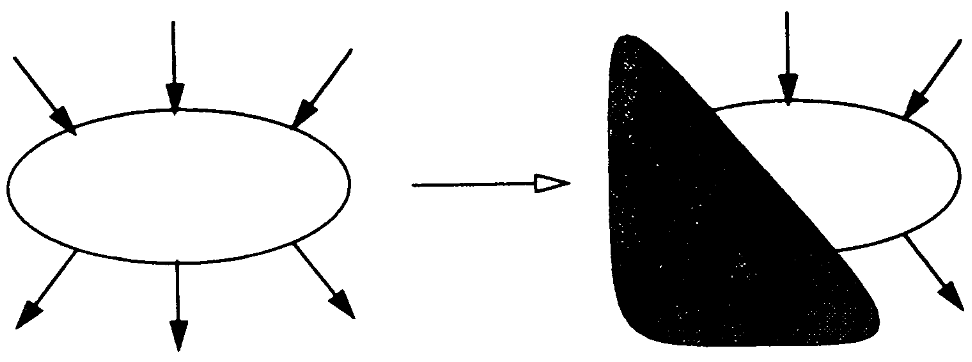


Figure 3.6 Hiding

3.1.3 Renaming

Some of the variables of a computation may be exchanged for new variables as follows:

Let RENAME be a function from Varset to Varset'; then

C ® RENAME is a computation such that:

$$\begin{aligned}
&\mathrm{Out}(C ® \mathrm{RENAME}) &&= \mathrm{ran}(\mathrm{RENAME} \mid_{\mathrm{Out}(C)}) \\
&\mathrm{In}(C ® \mathrm{RENAME}) &&= \mathrm{ran}(\mathrm{RENAME} \mid_{\mathrm{In}(C)}) - \mathrm{Out}(C ® \mathrm{RENAME}) \\
&\mathrm{Rel}(C ® \mathrm{RENAME})v &&\Leftrightarrow \mathrm{Rel}(C)(v \bullet \mathrm{RENAME})
\end{aligned}$$

(or: $\mathrm{Fun}(C ® \mathrm{RENAME})v_{in} = v_{out} \Leftrightarrow \mathrm{Fun}(C)(v_{in} \bullet \mathrm{RENAME}) = v_{out} \bullet \mathrm{RENAME}$)

Out(C ® RENAME) and In(C ® RENAME) are obviously finite.

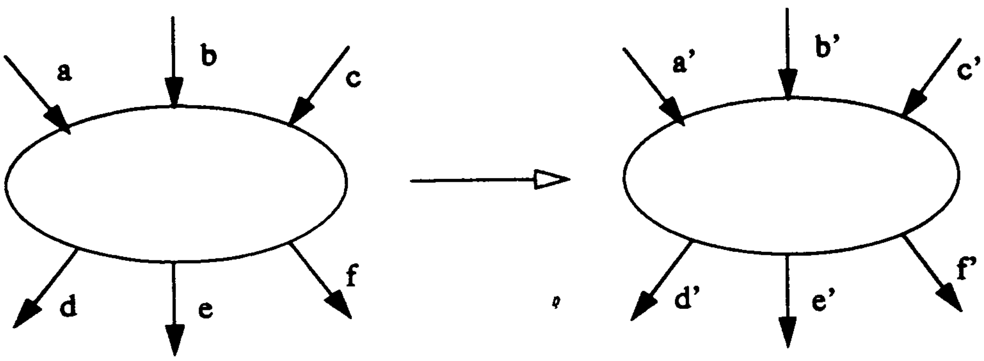


Figure 3.7 Renaming

Note that RENAME may not be 1-to-1. The freedom for it not to be is required in the definition of recurrence ($\text{RENAME}_{(p)}$, defined on page 54, may not be 1-to-1), but with the freedom comes the unwelcome side-effect that the result of the renaming may not be a well-defined function. Let PLUS''' be such that

In(PLUS''') := {A, B}
Out(PLUS''') := {C}
Rel(PLUS''')v ⇔ v(C) = v(A) + v(B)

and let TIMES be such that

In(TIMES) := {A', B'}
Out(TIMES) := {C'}
Rel(TIMES)v ⇔ v(C') = v(A') * v(B')

and let P-T be A || B (see Figure 3.8).

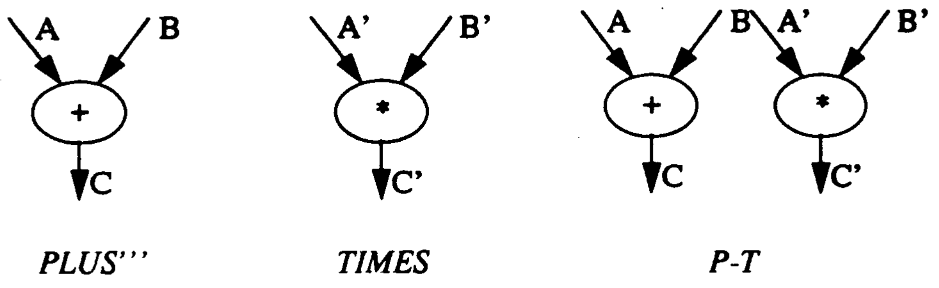


Figure 3.8 PLUS''', TIMES and P-T

Let RENAME be s.t.

RENAME(A')	:=	A
RENAME(B')	:=	B
and		
RENAME(C')	:=	C'

then P-T ® RENAME is well-defined (see Figure 3.9).

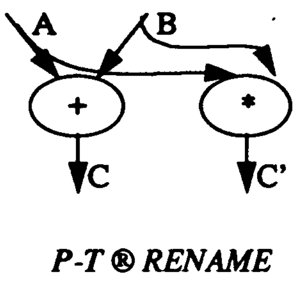


Figure 3.9 P-T ® RENAME

But if RENAME' is s.t.

RENAME'(A')	:=	A'
RENAME'(B')	:=	B'
RENAME'(C')	:=	C

then P-T ® RENAME' is obviously not well-defined if the inputs may range over the integers (see Figure 3.10).

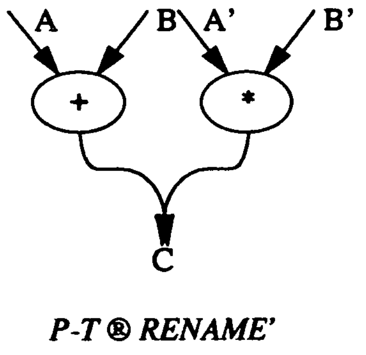


Figure 3.10 P-T ® RENAME'

3.1.4 Simulation

One computation, IMP say, is said to simulate another, ALG say, if ALG is IMP with some of its variables hidden, and other variables renamed. Formally:

IMP *simulates* ALG with respect to <Varset, RENAME>, where RENAME is a one-to-one function, if (IMP \ Varset) is well-defined and

ALG = (IMP \ Varset) ® RENAME

An example of the use of this definition is given at the end of subsection 3.1.5.

3.1.5 Example: TripleAdd

I shall now show how a very simple algorithm is defined in my language and in a sequential language.

Let us define the following procedure (in PASCAL-like language):

```
procedure TripleAdd(in A,B,C:integer;out TEMP,D:integer);
begin
        TEMP  := +(A,B);
        D     := +(TEMP,C)
end  {TripleAdd}
```

In my scheme, the computation corresponding to TripleAdd would be the composition of two subcomputations, both of which have addition as their function but which have different input and output variables...in fact, one subcomputation is a re-naming of the other.

In(PLUS) := {A, B}
Out(PLUS) := {TEMP}
Rel(PLUS)v $\Leftrightarrow$ v(TEMP) = v(A) + v(B)

PLUS' := PLUS ® RENAME

where
RENAME(A) := TEMP',
RENAME(B) := C
and
RENAME(TEMP) := D

TRIPLE-ADD := PLUS || PLUS'

Consider the similar procedure, TripleAdd', where TEMP is a local variable...this is equivalent to hiding TEMP:

```
procedure TripleAdd' (in A, B, C: integer; out D: integer);
var       TEMP: integer;
begin
     TEMP  := +(A, B);
     D     := +(TEMP, C)
end  {TripleAdd}
```

The corresponding computation in my scheme would be TRIPLE-ADD', where

TRIPLE-ADD' := TRIPLE-ADD \ {TEMP}

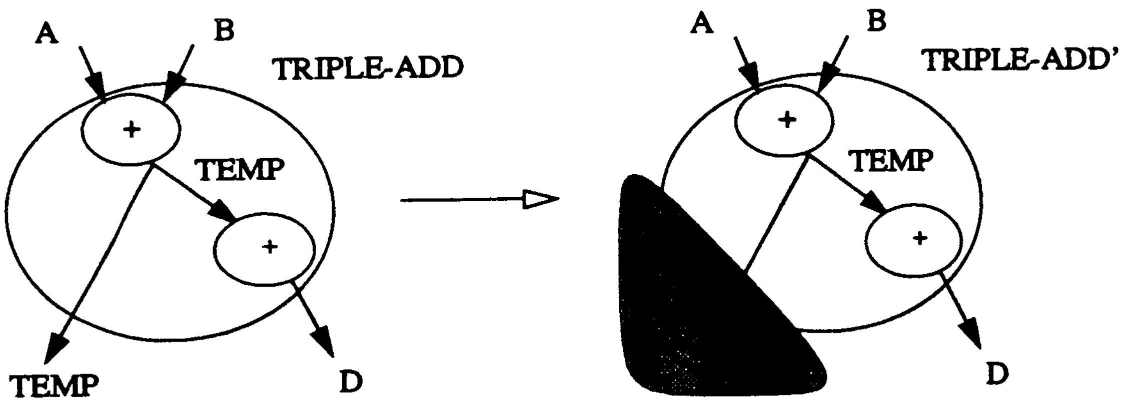


Figure 3.11 TRIPLE-ADD and TRIPLE-ADD'

Let us define TA-SPEC as follows:

In(TA-SPEC)	=	{X, Y, Z}
Out(TA-SPEC)	=	{W}
Rel(TA-SPEC)v	⇔	v(W) = v(X) + v(Y) + v(Z)

Then TRIPLE-ADD *simulates* TA-SPEC because

TA-SPEC = (TRIPLE-ADD\{TEMP}) ® RENAME'

where

RENAME'(A)	=	X
RENAME'(B)	=	Y
RENAME'(C)	=	Z
RENAME'(D)	=	W

3.2 Embedded Computations

A computation is said to be "embedded" if each of its variables is associated with a point of a lattice which is embedded in Euclidean space and, moreover, the computation is the composition of subcomputations such that, for each subcomputation, the outputs of that computation are situated at a single point. A subcomputation can be considered to be where its outputs are. Each variable is uniquely defined by its "class" and position. The formal definition is as follows:

A computation C is *embedded*[1] if, for some integer m, some finite subset D of Integer^m and some set of "variable classes", Varclasses,

(1) $\text{Vars(C)} \subseteq \text{Varclasses} \times \text{D}$,

and

(2) $\text{C} = \|_{p \in D}\, C_{(p)}$, where, for all $C_{(p)}$, $\text{Out}(C_{(p)}) \subseteq \text{Varclasses} \times \{p\}$

So each variable of C is a pair whose first component is a label (from Varclasses) and whose second is a point (in D). Note that all the output variables of $C_{(p)}$ are "located" at p (i.e. their second component is p).

The *domain* of an embedded computation, EMB, written *Dom*(EMB), is the minimal set which can be validly substituted for D in clause (1) above. ("Dom" is distinct from "dom" as defined on page ix.)

1. terminology: the word "embedded" is used simply to state that each variable in the computation is associated with a point in Euclidean space. Usually when the word is used in mathematics there is as an associated "embedding function" mapping an object into some space. There is no such function in this case...the computation is already in the space.

EX BIBL. UNIV. EDINBURGH

The *edge* of an embedded computation, EMB, written *Edge*(EMB), is the set of those points in D which have no associated output variable i.e.

$p \in \text{Edge(EMB)} \Rightarrow \text{for all var}, \langle \text{var}, p \rangle \notin \text{Out(EMB)}$

An example will clarify this definition. Let q and q' be the points $\begin{bmatrix}(0)\\(0)\end{bmatrix}$ and $\begin{bmatrix}(0)\\(0)\end{bmatrix}$ respectively. Let C_q and $C_{q'}$ be defined as follows:

$$\text{In}(C_q) = \{\langle x, \begin{bmatrix}(1)\\(0)\end{bmatrix}\rangle, \langle x, \begin{bmatrix}(1)\\(1)\end{bmatrix}\rangle\}$$

$$\text{Out}(C_q) = \{\langle x, \begin{bmatrix}(0)\\(0)\end{bmatrix}\rangle\}$$

$$\text{Rel}(C_q)v \Leftrightarrow v(\langle x, \begin{bmatrix}(0)\\(0)\end{bmatrix}\rangle) = v(\langle x, \begin{bmatrix}(1)\\(0)\end{bmatrix}\rangle) - v(\langle x, \begin{bmatrix}(1)\\(1)\end{bmatrix}\rangle)$$

$$\text{In}(C_{q'}) = \{\langle x, \begin{bmatrix}(2)\\(0)\end{bmatrix}\rangle, \langle x, \begin{bmatrix}(2)\\(1)\end{bmatrix}\rangle, \langle x, \begin{bmatrix}(2)\\(2)\end{bmatrix}\rangle\}$$

$$\text{Out}(C_{q'}) = \{\langle x, \begin{bmatrix}(1)\\(0)\end{bmatrix}\rangle\}$$

$$\text{Rel}(C_{q'})v \Leftrightarrow v(\langle x, \begin{bmatrix}(1)\\(0)\end{bmatrix}\rangle) = v(\langle x, \begin{bmatrix}(2)\\(0)\end{bmatrix}\rangle) + v(\langle x, \begin{bmatrix}(2)\\(1)\end{bmatrix}\rangle) + v(\langle x, \begin{bmatrix}(2)\\(2)\end{bmatrix}\rangle)$$

Let EMB be $C_q \parallel C_{q'}$; then EMB is an embedded computation. EMB is shown in Figure 3.12.

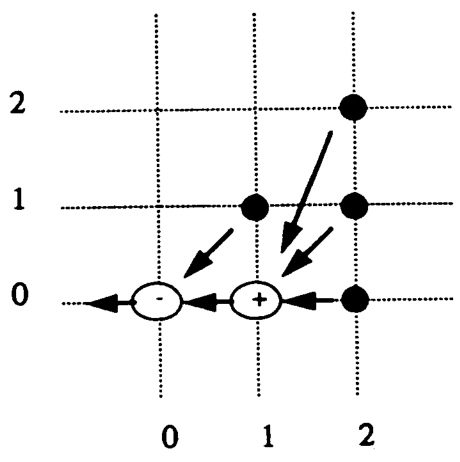


Figure 3.12 EMB

It has just one variable class, x, so we may let Varclasses be {x}. Its domain is { $\begin{bmatrix}(0)\\(0)\end{bmatrix}$, $\begin{bmatrix}(1)\\(0)\end{bmatrix}$, $\begin{bmatrix}(1)\\(1)\end{bmatrix}$, $\begin{bmatrix}(2)\\(0)\end{bmatrix}$, $\begin{bmatrix}(2)\\(1)\end{bmatrix}$, $\begin{bmatrix}(2)\\(2)\end{bmatrix}$ }, so we may let D be this set. EMB is equal to $\|_{p \in D}$ $C_{(p)}$ if, when $p \notin \{q, q'\}$, $C_{(p)}$ is defined to be the null computation, i.e. the one without inputs or outputs. Note that $Out(C_{(p)}) \subseteq \{x\} \times \{p\}$ when $p \in \{q, q'\}$.

A computation C' is an *edge-computation* of an embedded computation EMB if $Vars(C') \subseteq Varclasses \times Edge(EMB)$.

3.3 Recurrences

Recurrences are embedded computations which have a type of regularity which makes them useful in systolic array design. The subcomputations of a recurrence are arranged in a regular pattern, e.g. a rectangular grid. The subcomputations must all calculate the same function, but which variables a subcomputation has depends on its location. Formally:

Assume that $D \subseteq \text{Integer}^n$, that D is finite and that Varclasses is a set. These entities play the same role as they did in the definition of "embedded computation". Assume further that

$vc_i \in$ Varclasses for all i from 1 to m

$\Delta_i : D \rightarrow D$

and that

$\text{BASE} \subseteq D$

From the variable classes vc_i are formed the input variables to the subcomputations. Δ_i is the function that tells us from where to "fetch" vc_i, given the location of the subcomputation we are considering. BASE is the set of points which are occupied by a non-trivial (i.e. non-null) computation. (Recall that in the definition of an embedded computation, some of the subcomputations may be null.)

Let M be a computation such that

$\text{In}(M) = \{<vc_i, \Delta_i> : i = 1...m\}$

and

$\text{Out}(M) = \text{Varclasses} \times \{\text{Id}_{\text{BASE}}\}$

then the *recurrence* C constructed from *mould* M over *base* BASE is the (embedded) computation

$\|_{p \in \text{BASE}} C_{(p)}$

where $C_{(p)} = M \circledR \text{RENAME}_{(p)}$ where

$\text{RENAME}_{(p)}(<vc, fun>) = <vc, fun(p)>$

for all p in BASE and all <vc, fun> in Vars(M)

The mould M is the pattern or generator for the subcomputations. Each variable of M is a pair, the first of which is a variable class and the second is a "fetch" function. The subcomputation at a particular point p is found by replacing each fetch function fun by fun(p) in each variable of M. This is achieved by $\mathrm{RENAME}_{(p)}$. Note that the fetch function within each output variable of M is simply the identity since the outputs of each subcomputation appear simply at its location. The pairs $<vn_i, \Delta_i>$ are called the dependencies of C w.r.t. <M, BASE>. The pairs $<vn_i, \Delta_i>$ are called the <u>*dependencies*</u> of C with respect to <M, BASE>. The pair $<p, \Delta_i(p)>$ is called a <u>*dependency arc*</u> (of C with respect to <M, BASE>). Dependency arcs are depicted by arrows in my diagrams of recurrences. *Note that the arrows point in the direction opposite to that of the corresponding data-flow.* An example of an extremely simple recurrence is COPY which is defined as follows:

$$\begin{aligned} \mathrm{In(COPY)} &= \{<x, 0>\} \\ \mathrm{Out(COPY)} &= \{<x, i> : i \in \{1, 2, 3\}\} \\ \mathrm{Rel(COPY)}v &\Leftrightarrow (i \in \{1, 2, 3\}) \Rightarrow v(<x, i>) = v(<x, 0>) \end{aligned}$$

A diagram of COPY is shown in Figure 3.13.

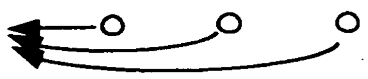


Figure 3.13 COPY

We can show this is a recurrence by finding a suitable mould and base. Let us note first of all that we may take D to be the set {0, 1, 2, 3}. (Each integer is identified with the corresponding one-dimensional vector). Varclasses is simply {x}. We may choose as the base (BASE) the set {1, 2, 3} and, letting m equal 1, set vc_1 to x and Δ_1 to $q \rightarrow 0$. We see that

$$\mathrm{COPY} = \|_{p \in \mathrm{BASE}}\, C_{(p)} \quad = \quad \|_{p \in \{1,2,3\}}\, C_{(p)}$$

where

$$C_{(p)} \quad = \quad M \circledR \mathrm{RENAME}_{(p)}$$

where $\mathrm{RENAME}_{(p)}$ is defined in the obvious way so that

$$\mathrm{In}(C_{(p)}) \quad = \quad \{<x, (q \rightarrow 0)p>\}$$
$$\mathrm{Out}(C_{(p)}) \quad = \quad \{<x, (q \rightarrow q)p>\}$$
$$\mathrm{Rel}(C_{(p)})v \Leftrightarrow v(<x, (q \rightarrow q)p>) = v(<x, (q \rightarrow 0)p>)$$

Two types of dependency are of particular interest:

A dependency $<vn_i, \Delta_i>$ is *affine* if Δ_i is an affine map, that is, if

$$\Delta_i(p) \quad = \quad A_i.p + d_i\,, \qquad \text{where } A_i \text{ is a matrix and } b_i \text{ is a vector}$$

The dependency $<x, (q \rightarrow 0)>$ in the previous example is an affine dependency. If a recurrence containing an affine dependency were to be mapped directly onto hardware, using an affine map from the space inhabited by the recurrence into space-time, a great deal of interconnect would often be needed. For example, if for some reason each subcomputation of COPY were mapped onto a separate subprocessor of a linear array, then connections would need to be made from one end of the processor to the other end and to all points in between.

A dependency $<vn_i, \Delta_i>$ is *uniform* if Δ_i is a uniform map, that is, if

$$\Delta_i(p) \quad = \quad p + d_i$$

In this case the translation vector d_i is called a *dependency vector* (of C with respect to $<M, BASE>$).

A recurrence is affine/uniform if there is a way of constructing it so that all its dependencies are affine/uniform respectively. "uniform recurrence" and "affine recurrence" may be abbreviated "UR" and "AR" respectively.

An example of a uniform recurrence is COPY' where

$$\begin{aligned} \text{In(COPY')} &= \{\langle x, 0\rangle\} \\ \text{Out(COPY')} &= \{\langle x, i\rangle : i \in \{1,2,3\}\} \\ \text{Rel(COPY')}v &\Leftrightarrow (i \in \{1,2,3\}) \Rightarrow v(\langle x, i\rangle) = v(\langle x, i\text{-}1\rangle) \end{aligned}$$

Figure 3.13 shows COPY' with its dependency arcs.

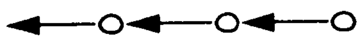


Figure 3.14 COPY'

COPY' is a uniform recurrence since it has only one dependency, $\langle x, i \rightarrow i\text{-}1\rangle$, which is uniform, its (dependency) vector being [-1]. Another example of a uniform recurrence can be seen in Figure 4.7 on page 91.

COPY is an example of a (non-uniform) affine recurrence. Again there is only one dependency, $\langle x, i \rightarrow 0\rangle$. That this is affine can be seen from the fact that

$$0 = [0].i + [0]$$

Another non-uniform affine recurrence can be seen in Figure 3.17 on page 63.

(An example of a non-affine recurrence would be COPY'', where

$$\begin{aligned} \text{In(COPY'')} &= \{\langle x, 2\rangle\} \\ \text{Out(COPY'')} &= \{\langle x, 4\rangle, \langle x, 8\rangle, \langle x, 16\rangle\} \end{aligned}$$

$\text{Rel(COPY'')} \Leftrightarrow (i \in \{1, 2, 3, 4\}) \Rightarrow v(\langle\alpha, 2^i\rangle) = v(\langle\alpha, 2^{i-1}\rangle)$

)

In general, if a uniform recurrence is mapped to hardware, only a short amount of interconnect is needed. For example, if COPY' were mapped to a linear array in a similar way to COPY, then only connections between each processor and its neighbour would be needed.

As mentioned earlier, the input to the design method has an affine recurrence as its main part. In data-pipelining (see page 14), the affine recurrence is transformed into a uniform recurrence composed with some control requirements.

There may be more than one mould-base combination which can be used to construct a particular recurrence. The sets of dependencies, dependency arcs and dependency vectors may vary according to which combination is chosen. In this document each recurrence will have just one mould-base pair, implicitly- or explicitly-stated, associated with it. The dependencies, dependency arcs and dependency vectors referred to in connection with the recurrence will be with respect to that pair.

3.3.1 Example: Convolution

Described in this section is an example of an affine recurrence ($\text{DATA}_{(\text{CONV})}$, shown in Figure 3.17 on page 63). When it is composed with its control requirements ($\text{CONTROL}_{(\text{CONV})}$), it implements a modified convolution task. Modified convolution may be defined mathematically as follows. Given two four-dimensional input vectors, W and X, we are to find the vector Y, a four-dimensional vector with components defined by the following equation (where "Y(j)" denotes the jth component of Y etc.):

$$Y(j) = \sum_{i=1}^{j} W(i)*X(j-i+1)$$

We may visualise this as follows. Let W be laid out in a horizontal line with W(1) at the left and W(4) at the right, and let X be laid alongside W but in the reverse direction. Y is found by sliding X to the left and, whenever the components of X line up with the components of W, taking the sum of the products of the components which have met and assigning the result to the next highest component of Y (Figure 3.15). Figure 3.16 is essentially the same as Figure 3.15 but it shows the input and output values when W is <1, 3, 4, 2> and X is <10, 20, 15, 11>.

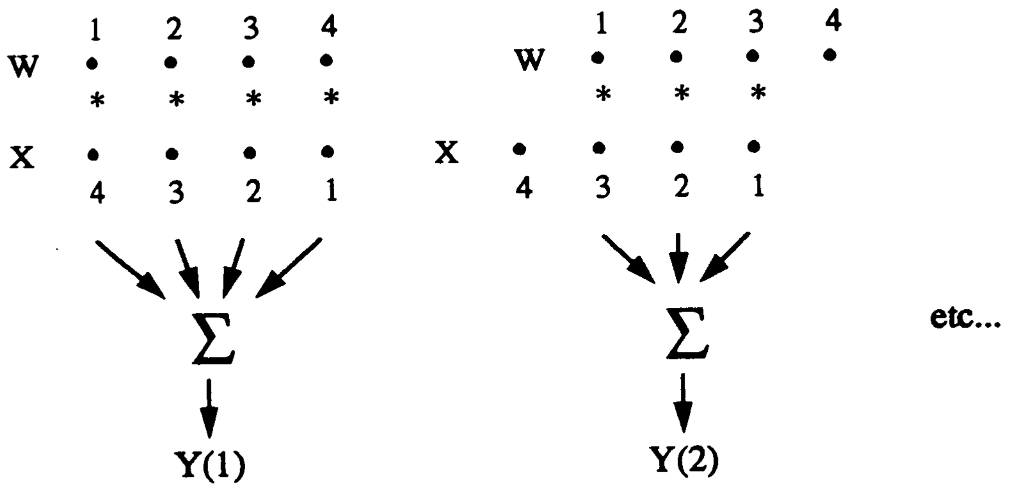


Figure 3.15 Modified convolution

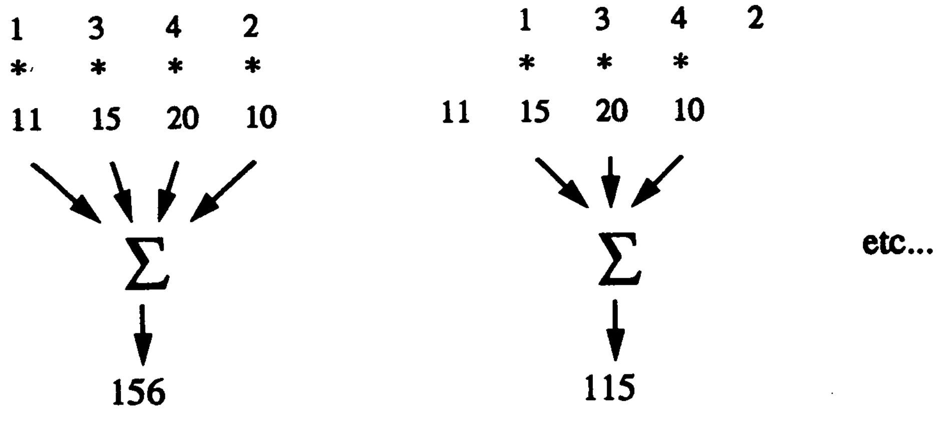


Figure 3.16 Modified convolution: a numerical example

This description has the components of Y being calculated in a certain order, but note that the order of calculation is not part of the task specification.

In the language of computations, the task may be specified by $ALG^0_{(CONV)}$, defined as follows:

Let $ALG^0_{(CONV)}$ be such that

$$\mathrm{In}(ALG^0_{(CONV)}) := \{\langle W, j\rangle \mid j = 1 \text{ to } 4\} \cup \{\langle X, i\rangle \mid i = 1 \text{ to } 4\}$$

$$\mathrm{Out}(ALG^0_{(CONV)}) := \{\langle Y, j\rangle \mid j = 1 \text{ to } 4\}$$

and $\mathrm{Rel}(ALG^0_{(CONV)})\ v \Leftrightarrow$ For all j in {1...4},

$$v(\langle Y, j\rangle) = \sum_{i=1}^{j} v(\langle W, i\rangle) * v(\langle X, j-i+1\rangle)$$

Let us now define the implementation of modified convolution, $DATA_{(CONV)} \parallel CONTROL_{(CONV)}$ (this is called $ALG_{(CONV)}$ and is shown in Figure 3.17 on page 63). $DATA_{(CONV)}$ is a recurrence with four variable classes: x, w, y, and c_y. Its base, $BASE_{(CONV)}$ is a right-angled triangle. The variable class x corresponds to X, which is input at the base of the triangle. The variable class w corresponds to W, which is presented at the left-hand edge of the triangle. The products are added one by one to the partial sums as they flow diagonally through the network from the bottom to the left of the triangle by means of the variable class y. The final sums of the products are output from the left-hand edge. c_y is a control variable class which is used to initialise the partial sums to zero. The are four dependencies, $\langle y, p \rightarrow p + \begin{bmatrix} 1 \\ -1 \end{bmatrix}\rangle$, $\langle x, p \rightarrow \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}.p\rangle$, $\langle w, p \rightarrow \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}.p\rangle$ and $\langle c_y, p \rightarrow p\rangle$. The first two dependencies fetch to each point in the base the appropriate components of the input vectors to be multiplied together and the third dependency fetches the appropriate partial sum to which the product is to be added. The final dependency simply fetches c_y from the current point. The value of c_y is required to be zero if we are currently at the bottom edge of the triangle and therefore require initialisation of the partial sum, and one if we are not. The formal definition of $\mathrm{DATA}_{(\mathrm{CONV})}$ follows. Note that the subcomputations $\mathrm{DATA}_{(\mathrm{CONV})(p)}$ are defined directly without reference to the mould.

Let us define the following region as the base of $\mathrm{DATA}_{(\mathrm{CONV})}$:

$$\mathrm{BASE}_{(\mathrm{CONV})} := \{ \begin{bmatrix} (i) \\ (j) \end{bmatrix} : i \geq 0, j \geq 0 \text{ and } j \leq 3 - i \}$$

Define $\mathrm{DATA}_{(\mathrm{CONV})(p)}$ and $\mathrm{DATA}_{(\mathrm{CONV})}$ as follows:

$$\mathrm{In}(\mathrm{DATA}_{(\mathrm{CONV})(p)}) := \{<y, p + \begin{bmatrix} 1 \\ -1 \end{bmatrix}>, <x, \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}.p>, <w, \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}.p>, <c_y, p>\}$$

$$\mathrm{Out}(\mathrm{DATA}_{(\mathrm{CONV})(p)}) := \{<y, p>\}$$

$$\mathrm{Rel}(\mathrm{DATA}_{(\mathrm{CONV})(p)})v \Leftrightarrow v(<y, p>) = v(<c_y, p>) * v(<y, p + \begin{bmatrix} 1 \\ -1 \end{bmatrix}>) + v(<x, \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}.p>) * v(<w, \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}.p>)$$

$$\mathrm{DATA}_{(\mathrm{CONV})} := (\|_{p \in \mathrm{BASE}_{(\mathrm{CONV})}} \mathrm{DATA}_{(\mathrm{CONV})(p)})$$

$\mathrm{DATA}_{(\mathrm{CONV})}$ is an example of an affine recurrence, since

$$\mathrm{In}(\mathrm{DATA}_{(\mathrm{CONV})(p)}) = \{<y, p + \begin{bmatrix} 1 \\ -1 \end{bmatrix}>, <x, \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}.p>, <w, \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}.p>, <c_y, p>\}$$

and the functions

$$p \longrightarrow p + \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

$$p \rightarrow \begin{bmatrix}1 & 0\\0 & 0\end{bmatrix}.p$$

$$p \rightarrow \begin{bmatrix}0 & 0\\0 & 1\end{bmatrix}.p$$

$$p \rightarrow p$$

are all affine. (Moreover, the dependencies $<y, p \rightarrow p + \begin{bmatrix}1\\-1\end{bmatrix}>$ and $<c_y, p \rightarrow p>$ are uniform.)

Let us now define the control requirements ($\text{CONTROL}_{(\text{CONV})}$). Firstly we need to make the following definition:

$$D_y := \{ \begin{bmatrix}i\\0\end{bmatrix} : 0 \le i \le 3\}$$

D_y is the bottom edge of the triangle. As stated previously, c_y needs to be zero in this region and one elsewhere in $\text{BASE}_{(\text{CONV})}$. This requirement is expressed in $\text{CONTROL}_{(\text{CONV})}$, defined below:

$$\begin{array}{ll}\text{In}(\text{CONTROL}_{(\text{CONV})}) & := \emptyset \\ \text{Out}(\text{CONTROL}_{(\text{CONV})}) & := \{<c_y, p> : p \in \text{BASE}_{(\text{CONV})}\} \\ \text{Rel}(\text{CONTROL}_{(\text{CONV})})v & \Leftrightarrow \text{For all } p, (\ (p \in D_y \Rightarrow v(<c_y, p>) = 0) \text{ and} \\ & \qquad (p \in \text{BASE}_{(\text{CONV})} - D_y \Rightarrow v(<c_y, p>) = 1))\end{array}$$

$\text{ALG}_{(\text{CONV})}$, which is the composition of $\text{DATA}_{(\text{CONV})}$ and $\text{CONTROL}_{(\text{CONV})}$, has domain $\text{BASE}_{(\text{CONV})} \cup D_y'$ where

$$D_y' := \{ \begin{bmatrix}i\\-1\end{bmatrix} : 1 \le i \le 4\}$$

$\text{ALG}_{(\text{CONV})}$ is illustrated in Figure 3.17. Only the data-dependency arcs between distinct points are drawn in. The shaded arrows indicate data-transfers which are not in fact required at a point. CONTROL$_{(\text{CONV})}$ is invisible.

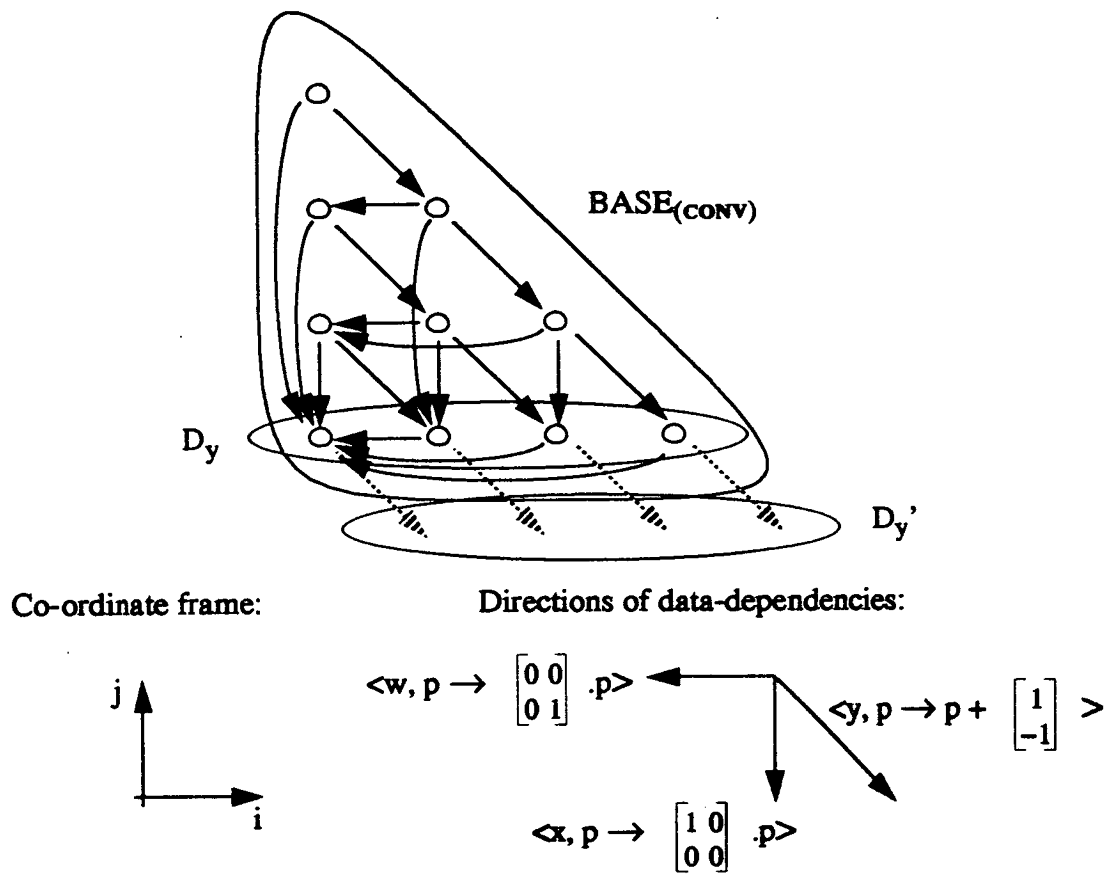


Figure 3.17 ALG$_{(\text{CONV})}$

Then ALG$_{(\text{CONV})}$ simulates ALG$^0_{(\text{CONV})}$ (defined on page 60) with respect to <Varset^0, RENAME^0>, where

$$\text{Varset}^0 := \{<y, \begin{bmatrix}(i)\\(j)\end{bmatrix}> : \begin{bmatrix}(i)\\(j)\end{bmatrix} \in \text{BASE}_{(\text{CONV})} \text{ and } i \neq 0\}$$

$$\cup \{<y, \begin{bmatrix}i\\-1\end{bmatrix}> : 1 \leq i \leq 4\}$$

$$\cup \{<w, \begin{bmatrix}(i)\\(j)\end{bmatrix}> : \begin{bmatrix}(i)\\(j)\end{bmatrix} \in \text{BASE}_{(\text{CONV})} \text{ and } i \neq 0\}$$

$$\cup \{<x, \begin{bmatrix}(i)\\(j)\end{bmatrix}> : \begin{bmatrix}(i)\\(j)\end{bmatrix} \in \mathrm{BASE}_{(\mathrm{CONV})} \text{ and } j \neq 0\}$$

$$\cup \{<c_y, p> : p \in \mathrm{BASE}_{(\mathrm{CONV})}\}$$

and RENAME^0 is such that

$$\text{for all j in } \{1...4\}, \quad \mathrm{RENAME}^0(<Y, j>) = <y, \begin{bmatrix}0\\j-1\end{bmatrix}>$$

$$\text{and for all j in } \{1...4\}, \quad \mathrm{RENAME}^0(<W, j>) = <w, \begin{bmatrix}0\\j-1\end{bmatrix}>$$

$$\text{and for all i in } \{1...4\}, \quad \mathrm{RENAME}^0(<X, i>) = <x, \begin{bmatrix}i-1\\0\end{bmatrix}>$$

In other words, ALG$_{(\mathrm{CONV})}$ equals ALG$^0{}_{(\mathrm{CONV})}$ when the internal data-transfers and all the control signals of the former are hidden and the remainder of its variables renamed appropriately.

3.3.2 Shorthand expressions for computations

There is a way of informally expressing certain computations (including all recurrences) in a briefer way:

The shorthand expression of a computation, C, is essentially a description of its relation, Rel(C). The distinction between a variable and its value which was carefully made for formal purposes is blurred for the sake of conciseness: for instance, $y(p + \begin{bmatrix}1\\-1\end{bmatrix})$ is written in place of $v(<y, p + \begin{bmatrix}1\\-1\end{bmatrix}>)$. The input and output sets of the computation are not explicitly stated in the shorthand form, but may be deduced from it: the symbol, ":=" is not symmetric, in contrast to "=" ... the variables which are represented by an expression which occurs on the left of a ":=" are outputs; all other variables are inputs.

To provide examples of shorthand form, $\text{DATA}_{(\text{CONV})}$ will generally be written as:

$$\begin{array}{lr} \lceil p \text{ in } \text{BASE}_{(\text{CONV})} \Rightarrow y(p) := c_y(p)*y(p+\begin{bmatrix}1\\-1\end{bmatrix}) + x(\begin{bmatrix}1&0\\0&0\end{bmatrix}.p)*w(\begin{bmatrix}0&0\\0&1\end{bmatrix}.p). & \rceil \\ \lfloor & \rfloor \end{array}$$

and $\text{ALG}_{(\text{CONV})}$ can be written as:

$$\begin{array}{llllr} \lceil p \text{ in } \text{BASE}_{(\text{CONV})} & \Rightarrow & y(p) & := c_y(p)*y(p+\begin{bmatrix}1\\-1\end{bmatrix}) + x(\begin{bmatrix}1&0\\0&0\end{bmatrix}.p)*w(\begin{bmatrix}0&0\\0&1\end{bmatrix}.p); & \rceil \\ \mid p \text{ in } D_y & \Rightarrow & c_y(p) & := 0; & \mid \\ \lfloor p \text{ in } \text{BASE}_{(\text{CONV})} - D_y & \Rightarrow & c_y(p) & := 1. & \rfloor \end{array}$$

Composition

The composition of several computations can be expressed in shorthand by the concatenation of the expressions representing each recurrence.

Hiding

There is, as far as I know, no general manipulation which can be done on computation expressions which corresponds to hiding.

Renaming

A renamed computation can be expressed by substituting the new variable names for the old ones in the expression of the original computation.

3.4 Space-time networks

A "space-time network" is a certain type of embedded computation; it models an algorithm executing on hardware. The Euclidean space in which a space-time network is embedded is identified with space-time, and in such a network a subcomputation can only be executed after all its inputs have been generated (i.e. the time co-ordinate of the position vector associated with a subcomputation must be greater than the time co-ordinate of the position vector of each of its input variables).

A *space-time network* is an embedded computation, C, which satisfies the following conditions:

(1) The variables of C are drawn from Varclasses $\times$ (Real $\times$ Real^{n-1}) (where Varclasses is a set of variable classes).

(2) C will have the structure $\|_{p \in D} C_p$ where $D \subseteq \text{Integer}^n$ and, for each p, C_p is a computation which produces all its output signals at point p.

(3) Let us define time(p) and space(p) to be such that

$\text{time}(p) = p{\downarrow}_1$

and

$\text{space}(p){\downarrow}_i = p{\downarrow}_{(i+1)}$

(If a variable (i.e. a signal) is $<v_n, p>$ then time(p) is the time co-ordinate of the signal and space(p) is the space co-ordinate.)

For each input <v, p'> to C_p (as defined in (2)),

$\text{time}(p') < \text{time}(p)$

If C is a recurrence then (3) is equivalent to:

(3') For all dependencies $<v, \Delta_i>$ of C, and all points p in D,

$$\text{time}(\Delta_i(p)) \quad < \quad \text{time}(p)$$

Not all recurrences are space-time networks. Furthermore, since a space-time network may not have a regular structure, not all space-time networks are recurrences.

If a computation simulates ALG and is a space-time network, then it is said to be a space-time simulation of ALG. Formally:

If C simulates ALG with respect to <Varset, RENAME> and is a space-time network, then it is called a *space-time simulation* of ALG.

Often a space-time simulation is formed from ALG, where ALG itself is an embedded computation: a one-to-one map from the domain of ALG to the domain of C is chosen and the variables are renamed accordingly (Varset is the empty set). Formally:

$$\text{ALG} \circledR \text{RENAME} \quad = \quad \text{C}$$

where RENAME : <v, p> ⟶ <v, Im(p)>, Im being some one-to-one function. We may make the following definitions:

$$\text{Im}_t(p) \quad := \quad \text{time}(\text{Im}(p))$$

and

$$\text{Im}_S(p) \quad := \quad \text{space}(\text{Im}(p))$$

Now since ALG is an embedded computation, we know that it can be decomposed into subcomputations:

$$\text{ALG} \parallel_{q \in D\text{\textbackslash}alg} \text{ALG}_q$$

where every output variable of ALG_q is situated at q.

Condition (3) is equivalent to saying that for all q, and for all inputs <v, q'> to ALG_q,

$$\text{time}(\text{Im}(q')) \quad < \quad \text{time}(\text{Im}(q))$$

That is:

$$\text{Im}_t(q') \quad < \quad \text{Im}_t(q)$$

In this case, the dependency arc <q, q'> is said to be *time-consistent* with **Im**.

Let us assume that ALG and C are uniform recurrences and $\mathbf{Im}_t$ is affine, so that

$$\mathbf{Im}_t(p) = \mathbf{A}_t.p + \mathbf{b}_t \text{ for some } \mathbf{A}_t \text{ and } \mathbf{b}_t$$

Let us say that a *vector* **b** is *time-consistent* with **Im** if

$$\mathbf{A}_t.b < 0$$

In this case, (3') further specializes to:

(3") All dependency vectors of C are *time-consistent* with **Im**.

(For future reference, when **Im** and $\mathbf{Im}_S$ are also affine, we will define **A**, **b**, $\mathbf{A}_S$ and $\mathbf{b}_S$ to be such that

$$\mathbf{Im}_S(p) = \mathbf{A}_S.p + \mathbf{b}_S$$

and

$$\mathbf{Im}(p) = \mathbf{A}.p + \mathbf{b})$$

A uniform recurrence which is also a space-time network is called a *strictly systolic computation*. The output of the design method has a strictly systolic computation as its main part. Given a strictly systolic computation, one can easily design a systolic array to implement it.

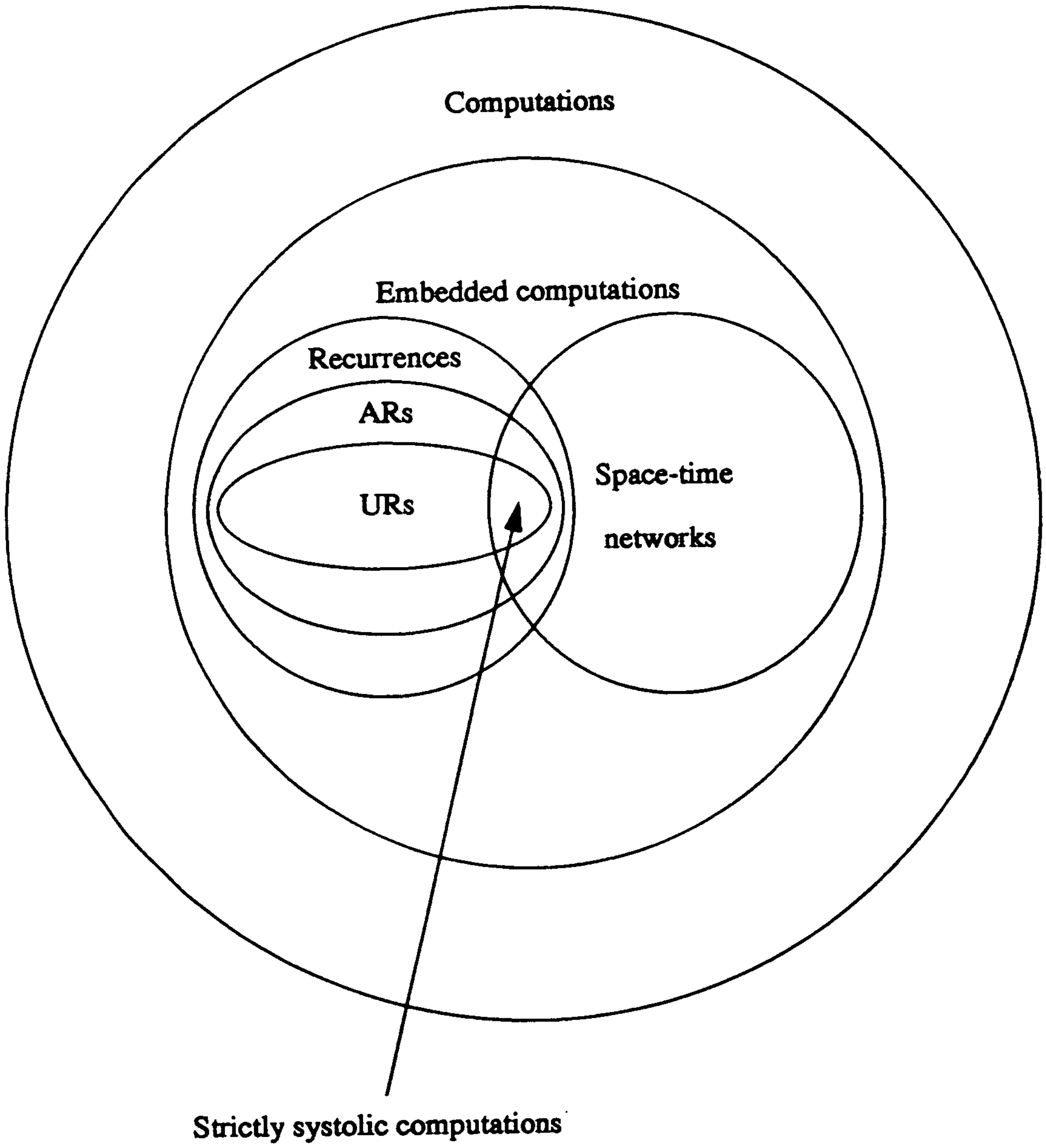


Figure 3.18 Venn diagram of the set of computations

3.5 Summary, discussion and further work

3.5.1 Summary

In this chapter the basic concepts to be used in Chapter 4 were defined. The concept of a computation was defined along with three operations on computations - composition, hiding and renaming - and one relation - simulation. The set of embedded computations, which are the compositions of subcomputations located in Euclidean space, was defined and a special type of embedded computation, the recurrence, was introduced along with some associated concepts, such as "affine recurrence" and "uniform recurrence". The set of space-time networks, embedded computations which model algorithms executing on hardware, was defined along with associated concepts. Finally, the set of strictly systolic computations was defined. Given a strictly systolic computation, one can easily design a systolic array to implement it.

3.5.2 Discussion

Computations

The basic entity in my theory is the computation. Although the behaviour of a computation is captured by a relation rather than a function (see the discussion on Formal Design Methods in Chapter 2), computations are functional in nature, with a distinction being drawn between inputs and outputs. The generality of computations means they can be used in algorithmic specifications, even those which would not be considered systolic. A distinction is drawn between a variable (input or output) and its value. In a simple function with multiple inputs and outputs, the inputs (and outputs) are ordered, and are therefore implicitly labelled by positive integers. In the explicit labelling of variables, the aim was to facilitate the combination of computations in complex ways, and to enable the capture of abstract and physical algorithmic structure by allowing as variables not only "atoms" (entities without internal structure) but also atom-vector pairs. This capturing of structure seemed to be necessary in order to define recurrences and systolic arrays.

Simulation

In this chapter not only is equality of computations defined but also what it means for one to simulate another. (As far as I know, "simulation" has not been formally defined in any of the literature on systolic array design, and yet such a definition seems essential when relating such disparate things as external behaviour, algorithms and hardware implementations. Although two things from different categories may both be expressible as computations, they are unlikely to be *equal* in any sense. Many of the more *general* parallel formalisms have a similar concept to simulation, though.

Recurrence

The concept of a recurrence is derived from the concept of a system of recurrence equations (SREs) [Raj89]. Unlike SREs, recurrences are formally defined, and therefore useful for formal verification; however, their definitions are cumbersome and hard to read, in contrast with those of the SREs and so the definition style of systems of recurrence equations is re-introduced, as the "syntactic sugar" of the shorthand form. Using this form, it should be easy to write the algorithmic specifications for input to the formal design method described in the following chapter.

Rajopadhye defines a "conditional uniform recurrence equation" (CURE) as a separate type of object from a uniform recurrence equation (URE) (a system of UREs corresponds to a uniform recurrence (UR)); a CURE is like a URE except that its output value at a point may depend directly on the position of that point, and not simply on the variable values at that or other points. In my method there is no need for conditional recurrences, uniform or affine (Rajopadhye uses an affine recurrence without introducing the type) since I hypothesise a control requirement/part right from the initial specification. Results in the "data part" never depend directly on the point at which they are generated.

Dependencies

The concept of dependency (data- and control-) occurs frequently in the literature. I give a formal definition of it.

Space-time networks

A strictly systolic computation exists in space-time and must satisfy the condition that each subcomputation must wait until all its input values have been generated and received before it can generate any of its output values. This attribute is however independent of its systolicity; hence the separate definition of a "valid space-time network" as a composite computation which has the attribute but may not be systolic.

3.5.3 Further work

It would be useful to do a detailed comparison between the formal language of this thesis with other languages, especially "Ruby" [She88a] and "ALPHA" [LeV85] with a view to designing a language which improves on them all.

If the shorthand form is to be used for writing algorithmic specifications, it will need to be given a formal semantics.

It would be good to have a more satisfactory theory of input and output. What is the essential difference between an input and an output and how can the dependency of the output-values of a computation on those of its inputs be easily determined? If the value of a certain output were found to be independent of that of a certain input, then it might be possible to schedule the production of the latter before that of the former. Computations' inputs and even computations themselves could, if redundant, be removed, which might allow the design of a more efficient implementation. Redundancy of inputs often occurs where a computation is "regulated by a control signal" (The dotted arrows e.g. in Figure 4.7 on page 91 correspond to such inputs.)

4 The Formal Design Method

In this chapter the design method is presented with the help of the convolution example introduced in Chapter 3.

The design task may be outlined as follows: given an initial computation expressing an algorithm, we want to find a space-time simulation for the computation. We will only consider initial computations of a certain form: those which are the composition of an affine recurrence and an initial control requirement. The control requirement is to be a set of control requirements of a certain form, expressed as an embedded computation. The space-time simulation - the output of the design process - is to be the composition of a uniform recurrence (which includes interior data and control signals) and a control part (which asserts constant values only and is and edge-computation of the recurrence). It is usually trivial to translate the uniform part of the space-time simulation into a systolic array; however, the edge control part may still need a little massaging before it can be encapsulated in hardware.

As described in Chapter 1, the design method is based on a transformation scheme (Figure 1.4), which can be broken down into three main transformations. These transformations are described briefly in the first part of the chapter. Most of the rest of the chapter is devoted to a detailed description of the design method, divided up into its five stages (Figure 1.5). Within this description, the transformations will be described in more detail. In tandem with its exposition, the design method is applied to the convolution example. From the space-time simulation an architecture is then constructed for the convolution algorithm. This architecture is systolic if the wires used to input and output signals to and from the array are ignored.

Transformation 1: Data-pipelining

By this transformation, the affine recurrence, which generally specifies the data-flow, is transformed into a uniform recurrence, with the generation of some control requirements which can be lumped together with the initial control part to form an aggregated control requirement. Let the initial computation be ALG, the affine recurrence be DATA, the initial control part be CONTROL, the uniform recurrence, describing the modified data-flow, be DATA', and the aggregated control part be CONTROL'. Then this transformation may be encapsulated diagrammatically as follows

$$\begin{tikzcd}
\text{ALG} \arrow[r, dotted] & \text{CONTROL} \arrow[d, dotted] & \parallel & \text{DATA} \arrow[lld, bend right] \arrow[d] \\
 & \text{CONTROL'} & \parallel & \text{DATA'}
\end{tikzcd}$$

$\longrightarrow$ = change + design decision
$\longrightarrow\triangleright$ = change without design decision
$\dashrightarrow$ = no change

CONTROL' || DATA' simulates CONTROL||DATA (✍ Theorem 1).

Transformation 2: Control-pipelining

By this transformation, a step is made towards the satisfaction of the control requirements - the aggregated control requirement is transformed into a uniform recurrence (CONTROL''') (✍ Theorem 25) and an edge control part (CONTROL'')(✍ Theorem 19). CONTROL''' has the same base as the uniform recurrence generated by the first transformation (DATA'). CONTROL'' has all its variables on the edge of the recurrence. CONTROL'' is called *EDGE* and the composition of CONTROL''' and DATA' is called *INTERIOR*. The diagram of this transformation is shown below:

$$\begin{tikzcd}
\text{CONTROL'} \arrow[d] \arrow[dr, bend left] & \parallel & \text{DATA'} \arrow[d, dotted] \\
\text{CONTROL''} \parallel (\arrow[d, dotted] & \text{CONTROL'''} & \parallel \text{DATA'}) \arrow[dl, dotted] \\
\text{EDGE} \parallel & \text{INTERIOR} &
\end{tikzcd}$$

$\longrightarrow$ = change + design decision

$-\!\!\rhd$ = change without design decision

$\dashrightarrow$ = no change

EDGE || INTERIOR simulates CONTROL' || DATA'(✍ Theorem 5).

Transformation 3: Scheduling and Allocation

By this transformation, the abstract space in which the computations are embedded is mapped to space-time by means of an affine function, **Im**. (Note that the affinity of the space-time map is logically separate from the affinity of the dependencies within the computations.) The first component of **Im**(p) forms the time co-ordinate of p and the remaining components form the space co-ordinate of p. The function $\mathbf{Im}_t$ which maps p to its time co-ordinate is the "scheduling" function and the function $\mathbf{Im}_s$ which maps p to its space co-ordinate is the "allocation" function. Formally:

$$\mathbf{Im}_t(p) := \mathbf{Im}(p)\downarrow_1$$

(i.e. $\mathbf{Im}_t(p)$ is the first component of $\mathbf{Im}(p)$)

and $\mathbf{Im}_S(p)\downarrow_i \; := \; \mathbf{Im}(p)\downarrow_{i+1}$

(i.e. the ith component of $\mathbf{Im}_S(p)$ equals the (i+1) component of $\mathbf{Im}(p)$)

Let RENAME be defined such that RENAME(<var, p>) = <var, **Im**(p)>. Then the final space-time simulation, IMP, equals EDGE' composed with INTERIOR', where EDGE' and INTERIOR' are the renamed versions of EDGE and INTERIOR respectively:

EDGE'	:=	EDGE ® RENAME
INTERIOR'	:=	INTERIOR ® RENAME
IMP	:=	EDGE' ‖ INTERIOR'

EDGE' ‖ INTERIOR' simulates EDGE ‖ INTERIOR (✍ Theorem 13).

This transformation can be expressed diagrammatically:

$$\begin{tikzcd}
\text{EDGE} \parallel \text{INTERIOR} \arrow[d, -latex] \\
(\text{EDGE} \parallel \text{INTERIOR}) \circledR \text{ RENAME} \arrow[d, -open triangle 45] \\
(\text{EDGE} \circledR \text{ RENAME}) \parallel (\text{INTERIOR} \circledR \text{ RENAME}) \arrow[d, dotted] \\
\text{EDGE}' \parallel \text{INTERIOR}' \arrow[d, dotted] \\
\text{IMP}
\end{tikzcd}$$

$$\begin{tikzcd}
{} \arrow[r, -latex] & {} & \text{= change + design decision} \\
{} \arrow[r, -open triangle 45] & {} & \text{= change without design decision} \\
{} \arrow[r, dotted] & {} & \text{= no change}
\end{tikzcd}$$

The result of the three transformations

IMP is a space-time network (✍) which simulates ALG (✍ Theorem 15) and EDGE' and INTERIOR' are of the required shape (✍ Theorem 20 and Theorem 27).

Recall from Figure 1.5 that the design process actually consists of the five-stage sequence:

Data-pipelining $\rightarrow$ Scheduling $\rightarrow$ Control-pipelining $\rightarrow$ Allocation $\rightarrow$ Final stage

We will now look at each of the five stages in detail. Each stage will be described in the general case and then in the particular case of the convolution example. Each computation or function in the example will be christened by adding the subscripted suffix, "$_{\text{(CONV)}}$" to the name of the corresponding computation or function in the general case. For example, $\text{DATA}_{\text{(CONV)}}$ in the convolution example corresponds to DATA in the general case.

4.1 Data-pipelining

As stated previously, in this stage of the design process we aim to find a computation, CONTROL', and a uniform recurrence, DATA' such that CONTROL' || DATA' simulates CONTROL || DATA. Essentially we transform the affine recurrence DATA into the uniform recurrence DATA', with the generation of some control requirements which we tack on to the control requirements from the original computation (specified by CONTROL). This transformation from uniform to affine recurrence is done by "pipelining the affine dependencies". The idea is that if the value of a variable (at a particular point) is required at more than one other point, as generally happens when there is an affine dependency, then the value doesn't have to be transmitted directly to each destination from its source, but can be passed to one point and from there circulated to all the others. The set of points depending on a single source is called a "coset". A new variable class is created to provide a channel for the value. As each affine dependency is pipelined, a control requirement is generated, since the subcomputation at each point needs to be told, by a control signal, whether it is getting the value directly from the original source or indirectly from a neighbour.

Let us consider in detail how a single affine dependency may be pipelined. Figure 4.1 on page 81 shows a typical affine dependency and Figure 4.2 on page 82 shows the corresponding uniform dependency paired with the new control requirement. Recall that we have given the affine recurrence the name *DATA*. Assume that it can be constructed from mould DATA_M over base BASE and that its dependencies are affine w.r.t. this choice of mould and base. Now

$$\text{DATA} \quad = \quad \|_{p \in \text{BASE}} \quad \text{DATA\_M} \circledR \text{R\_DATA}_{(1:p)}$$

where

$R\_DATA_{(1\,:\,p)}(<vc, fun>) = <vc, fun(p)>$

for all pairs <vc, fun> in Vars(DATA_M).

Let the affine dependency we are considering be $<a_2, \Delta_2>$; we know that $\Delta_2: p \rightarrow B_2.p + d_2$ for some matrix B_2 and vector d_2. We defined C(p), the coset of p, to be the set of points which, regarding the dependency $<a_2, \Delta_2>$, depend on the same point as p. Formally:

$$C(p) \ := \ \{p' \in BASE: \Delta_2(p') = \Delta_2(p)\}$$

Let us further assume that there exists a vector r_2 s.t., for all p, there exists a p_0 and integer N s.t.

$$C(p) \ = \ \{s: s = p_0 - m^*r_2, m \in \text{Integer}, 0 \leq m \leq N\}$$

That is, C(p) is a finite row of equally spaced points parallel to r_2.

Let us now define $PIPE\_M_{(2)}$, the pattern for a section of the "pipe" which will transport the data-signal:

$$\begin{aligned} In(PIPE\_M_{(2)}) &= \{<c_2, Id_{BASE}>, <z_2, p \rightarrow p{+}r_2>, <a_2, Id_{BASE}>\} \\ Out(PIPE\_M_{(2)}) &= \{<z_2, Id_{BASE}>\} \\ Rel(PIPE\_M_{(2)}) &\Leftrightarrow \\ v(<z_2, Id_{BASE}>) &= v(<c_2, Id_{BASE}>)^*v(<z_2, p \rightarrow p{+}r_2>) \\ &\quad + \bar{v}(<c_2, Id_{BASE}>)^*v(<a_2, Id_{BASE}>) \end{aligned}$$

Note that we have introduced two new variable-classes: z_2, which is the variable-class that provides a channel for the signal, and the control variable-class c_2 which acts as a switch which determines whether the value for z_2 at a point p is obtained from z_2 at the neighbouring point (which happens if p is *not* at the beginning of its coset row) or from a_2 at point p (which happens if p *is* at the beginning of its row). (We are making the assumption here that p_0 equals $\Delta_2(p_0)$.) Note that the variables of PIPE_M$_{(2)}$ are not variable-class-vector pairs, but variable-class-function pairs, to make it suitable for forming the mould of DATA$_{(2)}$ when composed with the modified version of DATA_M. Since z_2 is the new name of a_2, a renaming must be done on DATA_M. Let us define the renaming function R_DP$_{(2)}$ to be s.t.

$$\text{R\_DP}_{(2)}(<a_2, \Delta_2>) \quad := \quad <z_2, p \rightarrow p+r_2>$$

and for all $<a', \Delta'>$in Vars(DATA_M$_{(2)}$) not equal to $<a_2, \Delta_2>$,

$$\text{R\_DP}_{(2)}(<a', \Delta'>) \quad := \quad <a', \Delta'>$$

These equations express the fact that we want to replace the dependency $<a_2, \Delta_2>$ in DATA by $<z_2, \text{Id}_{\text{BASE}}>$ but to leave every other dependency unaffected. We now compose DATA_M ® R_DP$_{(2)}$ with PIPE_M, to form the mould for DATA$_{(2)}$, which we will call DATA_M$_{(2)}$:

$$\text{DATA\_M}_{(2)} \quad := \quad \text{DATA\_M} ® \text{R\_DP}_{(2)} \parallel \text{PIPE\_M}_{(2)}$$

$$\text{DATA}_{(2)} \quad := \quad \parallel_{p \in \text{BASE}} \ \text{DATA\_M}_{(2)} ® \text{R\_DATA}_{(2\,:\,p)}$$

where

$$\text{R\_DATA}_{(2\,:\,p)}(<vc, fun>) = <vc, fun(p)>$$

for all pairs $<vc, fun>$ in Vars(DATA_M$_{(2)}$).

We must not forget the new control requirements generated by this transformation. The new control computation will be:

$$\text{CONTROL}_{(2)} := \begin{bmatrix} p \text{ in BASE} \cap \{p' \mid p' \neq \Delta_2(p')\} \Rightarrow c_2(p) := 1; \\ p \text{ in BASE} \cap \{p' \mid p' = \Delta_2(p')\} \Rightarrow c_2(p) := 0. \end{bmatrix}$$

The first line specifies that if p (in BASE) is *not* equal to $\Delta_2(p)$ then the value of $<c_2, p>$ is 1 and the second line specifies that if p *is* equal to $\Delta_2(p)$ then the value of $<c_2, p>$

is 0. (The complicated appearance of the conditions preceding the implication arrows is because they must be written in the format required for shorthand expressions of recurrences.)

Recalling that the original recurrence was called DATA, we may now state the following:

If $\text{CONTROL}_{(2)}$, $\text{DATA}_{(2)}$ and DATA are as defined previously and certain assumptions are made then

$\text{CONTROL}_{(2)} \parallel \text{DATA}_{(2)}$ simulates DATA (✍ Theorem 2)

Figure 4.1 shows pictorially a possible affine dependency of DATA. Assume it is the first one to be made uniform. Figure 4.2 shows how the uniform dependency would appear in $\text{DATA}_{(2)}$ (left) and what $\text{CONTROL}_{(2)}$ would be (right). These should be superimposed, but are displayed separately for clarity.

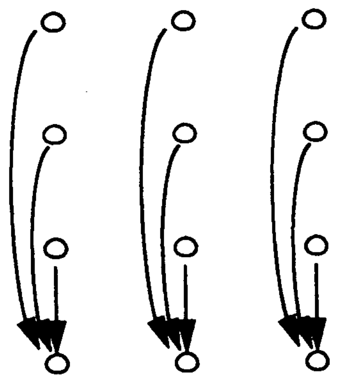


Figure 4.1 An affine dependency

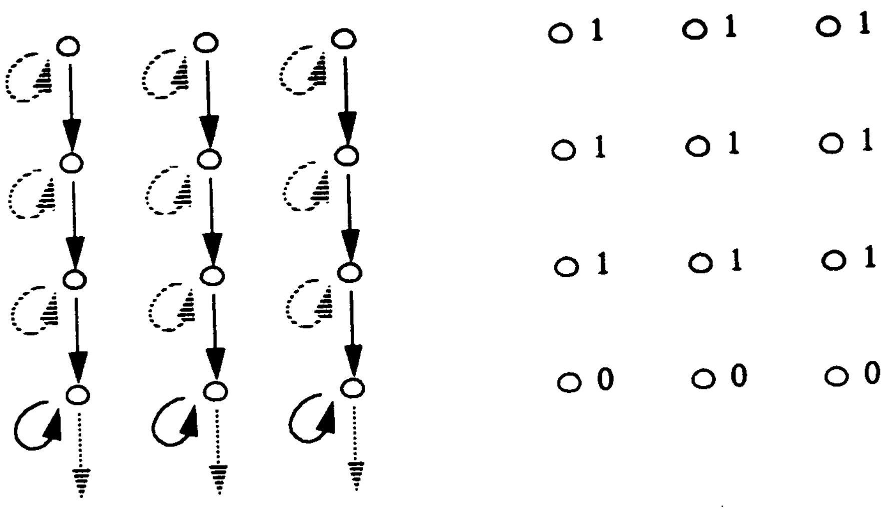


Figure 4.2 After pipelining: $DATA_{(2)}$ (left) and $CONTROL_{(2)}$ (right)

The loop arrows in Figure 4.2 correspond to the dependency $<a_2, Id_{BASE}>$ which appears in the definition of $PIPE\_M_{(2)}$, and the straight arrows correspond to the dependency $<z_2, p \rightarrow p{+}r_2>$. Solid arcs indicate that the data is actually being used, due to the value of c_2 at the destination of the arc.

We have now seen how to pipeline a single affine dependency. If there is more than one affine dependency in DATA then $DATA_{(2)}$ will have at least one such and the process must be repeated with $DATA_{(2)}$, producing $CONTROL_{(3)}$ and $DATA_{(3)}$ etc... When all the affine dependencies have been pipelined we will have the computation $(\|_{i=2 \text{ to } n} CONTROL_{(i)}) \| DATA_{(n)}$ which will simulate DATA. If we then tack on the initial control part, CONTROL (which we will call "$CONTROL_{(1)}$" for neatness' sake), we get $(\|_{i=1 \text{ to } n} CONTROL_{(i)}) \| DATA_{(n)}$, which simulates CONTROL $\|$ DATA (✍ Theorem 1). $DATA_{(n)}$ is uniform (✍ Theorem 26) so the required task, stated in the first sentence of this section, has been achieved, if we set DATA' equal to $DATA_{(n)}$ and CONTROL' equal to $(\|_{i=1 \text{ to } n} CONTROL_{(i)})$. The diagram on page 74 may now be expanded to include more details:

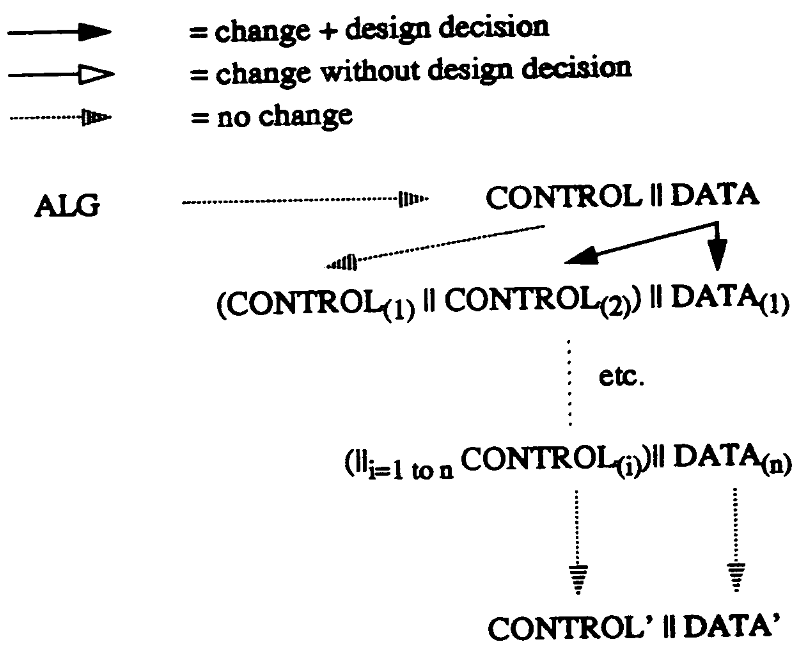

= change + design decision
= change without design decision
= no change
ALG
CONTROL || DATA
(CONTROL(1) || CONTROL(2)) || DATA(1)
etc.
(||i=1 to n CONTROL(i))|| DATA(n)
CONTROL' || DATA'

Figure 4.3 Data-pipelining

We have just seen in detail the process of data-pipelining, in which the affine dependencies in the data part of the original computation are progressively replaced by uniform dependencies. We will now see how this process operates in the particular case of the convolution example.

4.1.1 Example

Let us recall the definitions from page 61, presented this time using shorthand expressions for computations.

$$\mathrm{BASE}_{(\mathrm{CONV})} := \{ \begin{bmatrix} (i) \\ (j) \end{bmatrix} : i \geq 0, j \geq 0 \text{ and } j \leq 3 - i \}$$

$\mathrm{BASE}_{(\mathrm{CONV})}$ is the base of $\mathrm{DATA}_{(\mathrm{CONV})}$, defined below:

$$\mathrm{DATA}_{(\mathrm{CONV})} := \left\lceil \text{p in } \mathrm{BASE}_{(\mathrm{CONV})} \Rightarrow y(p) := c_y(p)*y(p + \begin{bmatrix}1\\-1\end{bmatrix}) \right\rceil$$
$$\left\lfloor \qquad +x(\begin{bmatrix}1&0\\0&0\end{bmatrix}.p)*w(\begin{bmatrix}0&0\\0&1\end{bmatrix}.p). \right\rfloor$$

$\mathrm{DATA}_{(\mathrm{CONV})}$ is the data part of the computation used in the specification of the convolution task. It states that if p is in the base then the running total at p (that is, the value of <y, p>) is equal to the running total at (p + $\begin{bmatrix}1\\-1\end{bmatrix}$) multiplied by the value of the control variable <c_y, p>, added to the relevant weighted input (the value of the input <x, $\begin{bmatrix}1&0\\0&0\end{bmatrix}$.p> multiplied by the value of the weight <w, $\begin{bmatrix}0&0\\0&1\end{bmatrix}$.p>). The value of <$c_y$, p> is defined (by the control part $\mathrm{CONTROL}_{(\mathrm{CONV})}$) to be 1 everywhere in the base except the strip D_y at the base of the triangle, where it is 0:

$$D_y := \{ \begin{bmatrix}i\\0\end{bmatrix} : 0 \le i \le 3\}$$

$$\mathrm{CONTROL}_{(\mathrm{CONV})} := \begin{bmatrix} \text{p in } D_y & \Rightarrow & c_y(p) := 0; \\ \text{p in } \mathrm{BASE}_{(\mathrm{CONV})} - D_y & \Rightarrow & c_y(p) := 1. \end{bmatrix}$$

This causes the running total to be initialised at 0 along this strip. The complete initial computation is of course the initial control part composed with the initial data part.

$$\mathrm{ALG}_{(\mathrm{CONV})} := \mathrm{CONTROL}_{(\mathrm{CONV})} \parallel \mathrm{DATA}_{(\mathrm{CONV})}$$

A diagram of $\mathrm{ALG}_{(\mathrm{CONV})}$ can be seen in Figure 3.17 on page 63.

There are two dependencies which need to be pipelined; one is <x, p $\rightarrow$ $\begin{bmatrix}1&0\\0&0\end{bmatrix}$.p> and the other is <w, p $\rightarrow$ $\begin{bmatrix}0&0\\0&1\end{bmatrix}$.p>. We will tackle the former first.

$ALG_{(CONV)}$ is of the same form as ALG on page 79 with $CONTROL_{(CONV)}$ identified with CONTROL and $DATA_{(CONV)}$ identified with DATA. $DATA_{(CONV)}$ satisfies the conditions for Theorem 2 when we identify a_2 with the variable-class x, Δ_2 with the function $p \rightarrow \begin{bmatrix}1 & 0\\0 & 0\end{bmatrix}.p$ and r_2 with the vector $\begin{bmatrix}0\\-1\end{bmatrix}$.

So x is the variable-class of the dependency to be pipelined, $p \rightarrow \begin{bmatrix}1 & 0\\0 & 0\end{bmatrix}.p$ is its function and $\begin{bmatrix}0\\-1\end{bmatrix}$ is the vector between a point in a coset and its neighbour in that coset.

Let the new variable-class for the pipe be z_x and the new control-variable class, c_x. We may now follow the pattern on page 79 in making some definitions:

$$\begin{aligned}
In(PIPE\_M_{(CONV)(2)}) &= \{<c_x, p \rightarrow p>, <z_x, p \rightarrow p+\begin{bmatrix}0\\-1\end{bmatrix}>, <x, p \rightarrow p>\}\\
Out(PIPE\_M_{(CONV)(2)}) &= \{<z_x, p \rightarrow p>\}\\
Rel(PIPE\_M_{(CONV)(2)}) &\Leftrightarrow
\end{aligned}$$

$$\begin{aligned}
v(<z_x, p \rightarrow p>) &= v(<c_x, p \rightarrow p>)*v(<z_x, p \rightarrow p+\begin{bmatrix}0\\-1\end{bmatrix}>)\\
&+ \bar{v}(<c_x, p \rightarrow p>)*v(<x, p \rightarrow p>)
\end{aligned}$$

(This definition for $PIPE\_M_{(CONV)(2)}$ corresponds to the definition for $PIPE\_M_{(2)}$ on page 79.)

$$R\_DP_{(CONV)(2)}(<x, p \rightarrow \begin{bmatrix}1 & 0\\0 & 0\end{bmatrix}.p>) := <x, p \rightarrow p+\begin{bmatrix}0\\-1\end{bmatrix}>$$

and for all <a', Δ'>in $Vars(DATA\_M_{(CONV)(2)})$ not equal to $<x, p \rightarrow \begin{bmatrix}1 & 0\\0 & 0\end{bmatrix}.p>$,

$$R\_DP_{(CONV)(2)}(<a', \Delta'>) := <a', \Delta'>$$

$$DATA\_M_{(CONV)(2)} := DATA\_M_{(CONV)} ® R\_DP_{(CONV)(2)} \| PIPE\_M_{(CONV)(2)}$$

$$\mathrm{DATA}_{(\mathrm{CONV})(2)} := \|_{p \in \mathrm{BASE}} \ \mathrm{DATA\_M}_{(\mathrm{CONV})(2)} \circledR \mathrm{R\_DATA}_{(\mathrm{CONV})(2\,:\,p)}$$

where

$\mathrm{R\_DATA}_{(\mathrm{CONV})(2\,:\,p)}(<vc, fun>) = <vc, fun(p)>$

for all pairs $<vc, fun>$ in $\mathrm{Vars}(\mathrm{DATA\_M}_{(\mathrm{CONV})(2)})$

and $\mathrm{DATA\_M}_{(\mathrm{CONV})}$ is s.t.

$\mathrm{In}(\mathrm{DATA\_M}_{(\mathrm{CONV})}) := \{<c_y, p \rightarrow p>, <y, p \rightarrow p + \begin{bmatrix} 1 \\ -1 \end{bmatrix}>, <x, p \rightarrow \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}.p>,$

$<w, p \rightarrow \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}.p>\}$

$\mathrm{Out}(\mathrm{DATA\_M}_{(\mathrm{CONV})}) := \{<y, p \rightarrow p>\}$

$\mathrm{Rel}(\mathrm{DATA\_M}_{(\mathrm{CONV})})(v) \Leftrightarrow v(<y, p \rightarrow p>) =$

$$v(<c_y, p \rightarrow p>)*v(<y, p \rightarrow p + \begin{bmatrix} 1 \\ -1 \end{bmatrix}>)$$

$$+\ v(<x, p \rightarrow \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}.p>)*v(<w, p \rightarrow \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}.p>) .$$

($\mathrm{DATA\_M}_{(\mathrm{CONV})}$ is a mould for $\mathrm{DATA}_{(\mathrm{CONV})}$. $\mathrm{DATA}_{(\mathrm{CONV})(2)}$ corresponds to $\mathrm{DATA}_{(2)}$, defined on page 80.)

$\mathrm{CONTROL}_{(\mathrm{CONV})(2)} :=$

$$\begin{bmatrix} p \text{ in } \mathrm{BASE}_{(\mathrm{CONV})} \cap \{p' \mid p' = \begin{bmatrix} (i) \\ (j) \end{bmatrix} \text{ and } j \neq 0\} \Rightarrow & c_x(p) := 1; \\ p \text{ in } \mathrm{BASE}_{(\mathrm{CONV})} \cap \{p' \mid p' = \begin{bmatrix} (i) \\ (j) \end{bmatrix} \text{ and } j = 0\} \Rightarrow & c_x(p) := 0. \end{bmatrix}$$

(This definition corresponds to the one defining $\mathrm{CONTROL}_{(2)}$ on page 80. The set $\{p' \mid p' = \begin{bmatrix} (i) \\ (j) \end{bmatrix}$ and $j \neq 0\}$ corresponds to $\{p' \mid p' \neq \Delta_2(p')\}$ since in this case Δ_2 is equated with the function $p \rightarrow \begin{bmatrix}1 & 0\\0 & 0\end{bmatrix}.p$. Similarly the set $\{p' \mid p' = \begin{bmatrix}(i)\\(j)\end{bmatrix}$ and $j = 0\}$ corresponds to $\{p' \mid p' = \Delta_2(p')\}$. Note that $\{p' \mid p' = \begin{bmatrix}(i)\\(j)\end{bmatrix}$ and $j = 0\}$ is coincidentally equal to D_y.)

Using Theorem 2, we can now deduce that, assuming certain computations are well-defined,

$\text{CONTROL}_{(\text{CONV})(2)} \parallel \text{DATA}_{(\text{CONV})(2)}$ simulates $\text{DATA}_{(\text{CONV})}$ (n.p.)

Figure 4.6 shows $\text{CONTROL}_{(\text{CONV})(2)}$ and Figure 4.5 shows $\text{DATA}_{(\text{CONV})(2)}$.

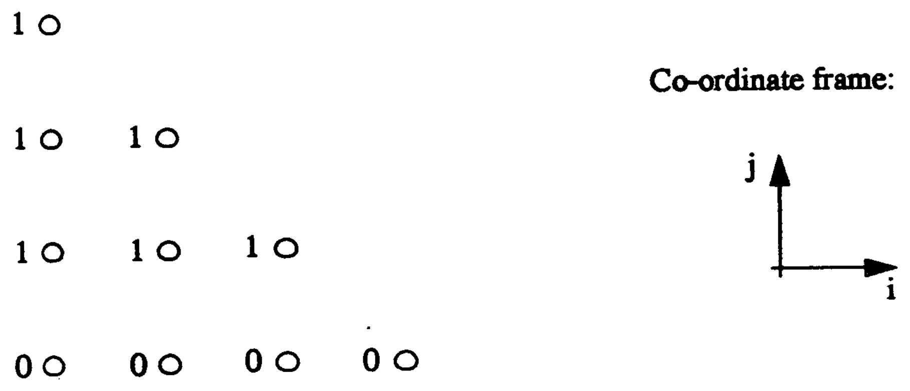


Figure 4.4 $\text{CONTROL}_{(\text{CONV})(2)}$ (showing the values of c_x at each point)

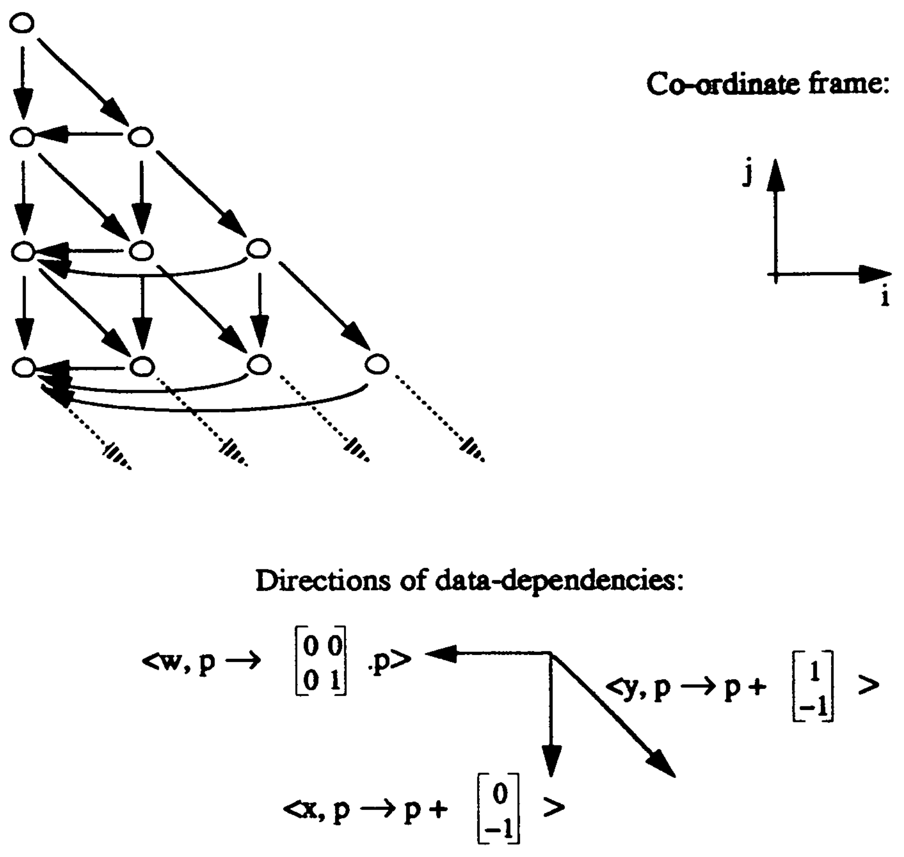

Co-ordinate frame:
j
i
Directions of data-dependencies:
$<w, p \rightarrow \begin{bmatrix}0 & 0\\0 & 1\end{bmatrix}.p>$
$<y, p \rightarrow p + \begin{bmatrix}1\\-1\end{bmatrix}>$
$<x, p \rightarrow p + \begin{bmatrix}0\\-1\end{bmatrix}>$

Figure 4.5 $DATA_{(CONV)(2)}$

We have pipelined the first dependency but we still need to pipeline the other, $<w, p \rightarrow \begin{bmatrix}0 & 0\\0 & 1\end{bmatrix}.p>$; the process must be repeated with new identifications: a_2 is identified with the variable-class w, Δ_2 is identified with the function $p \rightarrow \begin{bmatrix}0 & 0\\0 & 1\end{bmatrix}.p$ and r_2 with the vector $\begin{bmatrix}-1\\0\end{bmatrix}$. If z_2 is identified with z_w , c_2 with c_w and we make the following definitions (they are similar to the previous ones):

$$\mathrm{In}(\mathrm{PIPE\_M}_{(\mathrm{CONV})(3)}) \quad = \quad \{<c_w, p \rightarrow p>, <z_w, p \rightarrow p + \begin{bmatrix}-1\\0\end{bmatrix}>, <w, p \rightarrow p>\}$$

$$\mathrm{Out}(\mathrm{PIPE\_M}_{(\mathrm{CONV})(3)}) \quad = \quad \{<z_w, p \rightarrow p>\}$$

$$\mathrm{Rel}(\mathrm{PIPE\_M}_{(\mathrm{CONV})(3)}) \quad \Leftrightarrow$$

$$v(<z_w, p \rightarrow p>) \quad = \quad v(<c_w, p \rightarrow p>)*v(<z_w, p \rightarrow p + \begin{bmatrix}-1\\0\end{bmatrix}>)$$
$$+ \quad \bar{v}(<c_w, p \rightarrow p>)*v(<w, p \rightarrow p>)$$

(This definition for $\mathrm{PIPE\_M}_{(\mathrm{CONV})(3)}$ corresponds to the definition for $\mathrm{PIPE\_M}_{(\mathrm{CONV})(2)}$ on page 79, but w, c_w, z_w and $\begin{bmatrix}-1\\0\end{bmatrix}$ occur in place of x, c_x, z_x and $\begin{bmatrix}0\\-1\end{bmatrix}$ respectively.)

$$\mathrm{R\_DP}_{(\mathrm{CONV})(3)}(<w, p \rightarrow \begin{bmatrix}0&0\\0&1\end{bmatrix}.p>) := <w, p \rightarrow p + \begin{bmatrix}-1\\0\end{bmatrix}>$$

and for all $<a', \Delta'>$in $\mathrm{Vars}(\mathrm{DATA\_M}_{(\mathrm{CONV})(3)})$ not equal to $<w, p \rightarrow \begin{bmatrix}0&0\\0&1\end{bmatrix}.p>$,

$$\mathrm{R\_DP}_{(\mathrm{CONV})(3)}(<a', \Delta'>) \; := \; <a', \Delta'>$$

$$\mathrm{DATA\_M}_{(\mathrm{CONV})(3)} \quad := \quad \mathrm{DATA\_M}_{(\mathrm{CONV})(2)} \circledR \mathrm{R\_DP}_{(\mathrm{CONV})(3)} \,\|\, \mathrm{PIPE\_M}_{(\mathrm{CONV})(3)}$$

$$\mathrm{DATA}_{(\mathrm{CONV})(3)} \quad := \quad \|_{p \in \mathrm{BASE}} \quad \mathrm{DATA\_M}_{(\mathrm{CONV})(3)} \circledR \mathrm{R\_DATA}_{(\mathrm{CONV})(3\,:\,p)}$$

where

$$\mathrm{R\_DATA}_{(\mathrm{CONV})(3\,:\,p)}(<vc, fun>) = <vc, fun(p)>$$
for all pairs $<vc, fun>$ in $\mathrm{Vars}(\mathrm{DATA\_M}_{(\mathrm{CONV})(3)})$

(These definitions correspond to those for $\mathrm{R\_DP}_{(\mathrm{CONV})(2)}$, $\mathrm{DATA\_M}_{(\mathrm{CONV})(2)}$, $\mathrm{DATA}_{(\mathrm{CONV})(2)}$, $\mathrm{R\_DATA}_{(\mathrm{CONV})(2\,:\,p)}$, with $\begin{bmatrix}-1\\0\end{bmatrix}$ and $\begin{bmatrix}0&0\\0&1\end{bmatrix}$ in place of $\begin{bmatrix}0\\-1\end{bmatrix}$ and $\begin{bmatrix}1&0\\0&0\end{bmatrix}$ respectively, $\mathrm{DATA\_M}_{(\mathrm{CONV})}$ in place of $\mathrm{DATA\_M}_{(\mathrm{CONV})(2)}$ and 2 replaced by 3 in the subscripts.)

$$\mathrm{CONTROL}_{(\mathrm{CONV})(3)} :=$$

$$\left\lceil p \text{ in } \mathrm{BASE}_{(\mathrm{CONV})} \cap \{p' \mid p' = \begin{bmatrix}(i)\\(j)\end{bmatrix} \text{ and } i \neq 0\} \Rightarrow c_w(p) := 1; \right\rceil$$

$$\left\lfloor p \text{ in } \mathrm{BASE}_{(\mathrm{CONV})} \cap \{p' \mid p' = \begin{bmatrix}(i)\\(j)\end{bmatrix} \text{ and } i = 0\} \Rightarrow c_w(p) := 0. \right\rfloor$$

(This is the same as the definition of $\mathrm{CONTROL}_{(\mathrm{CONV})(2)}$, except that $c_w(p)$ replaces $c_x(p)$ and the strip where the value of the control variables is 0 is vertical and situated at the left-hand edge of the base, rather than being horizontal and below its base - see Figure 4.6 and Figure 4.6.)

By Theorem 2, assuming that certain computations are well-defined, we can deduce that

$$\mathrm{CONTROL}_{(\mathrm{CONV})(3)} \| \mathrm{DATA}_{(\mathrm{CONV})(3)} \text{ simulates } \mathrm{DATA}_{(\mathrm{CONV})(2)} \text{ (n.p.)}$$

(cf. the analogous deduction on page 87)

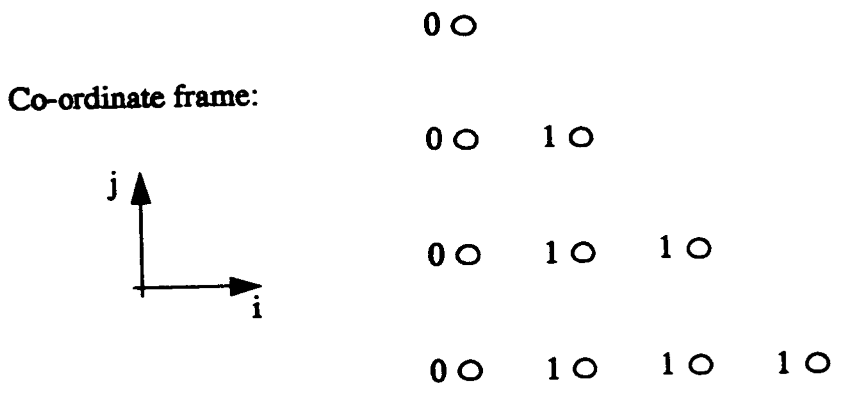


Figure 4.6 $\mathrm{CONTROL}_{(\mathrm{CONV})(3)}$ (showing the values of c_w at each point)

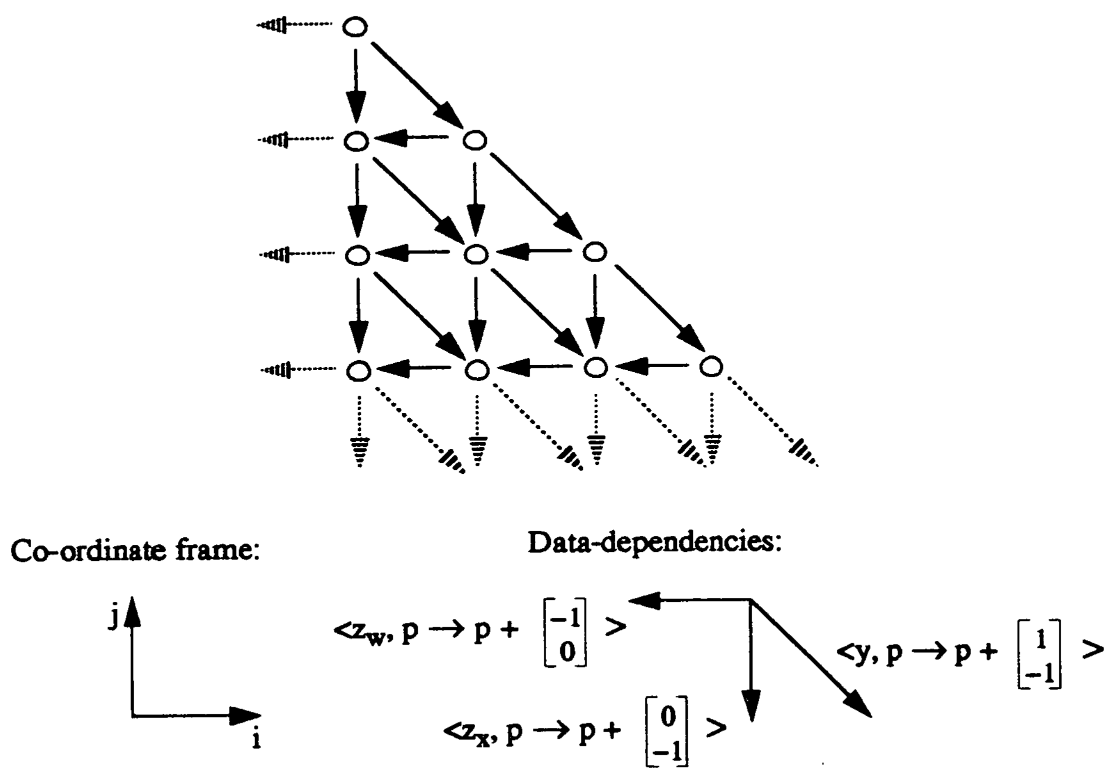

Co-ordinate frame:
j
i
Data-dependencies:
$<z_w, p \rightarrow p + \begin{bmatrix}-1\\0\end{bmatrix}>$
$<z_x, p \rightarrow p + \begin{bmatrix}0\\-1\end{bmatrix}>$
$<y, p \rightarrow p + \begin{bmatrix}1\\-1\end{bmatrix}>$

Figure 4.7 $DATA_{(CONV)(3)}$

Now $DATA_{(CONV)(3)}$ is a uniform recurrence so by the discussion on page 82 we know that if we change the name of the initial control computation $CONTROL_{(CONV)}$ to $CONTROL_{(CONV)(1)}$, for neatness:

$$CONTROL_{(CONV)(1)} := CONTROL_{(CONV)}$$

and define CONTROL' $_{(CONV)}$ to be the composition of the three control computations (the one initial one and the two just created):

$$CONTROL'_{(CONV)} := (||_{i=1 \text{ to } 3} CONTROL_{(CONV)(i)})$$

and set DATA'$_{(CONV)}$ equal to $DATA_{(CONV)(3)}$:

$$\text{DATA}'_{(\text{CONV})} := \text{DATA}_{(\text{CONV})(3)}$$

then $\text{DATA}'_{(\text{CONV})}$ is a uniform recurrence and $\text{CONTROL}'_{(\text{CONV})} \parallel \text{DATA}'_{(\text{CONV})}$ simulates $\text{CONTROL}_{(\text{CONV})} \parallel \text{DATA}_{(\text{CONV})}$ (n.p.)

($\text{CONTROL}'_{(\text{CONV})} \parallel \text{DATA}'_{(\text{CONV})}$ has not been drawn for the following reasons. $\text{DATA}'_{(\text{CONV})}$ was seen in Figure 4.7; $\text{CONTROL}'_{(\text{CONV})}$ has no dependencies and to show the values of each control signal at each point would have made the diagram confusing.)

Thus the data-pipelining task has been completed for the convolution example. We will now return to the general scheme and look at the scheduling stage.

4.2 Scheduling

We need to choose the function Im_t so that the final implementation, IMP, satisfies the conditions which would make it a space-time network(✍). The conditions are as on page 66 with IMP substituted for C namely:

(1) The variables of IMP are drawn from the set Varclasses $\times$ (Real $\times$ Real^{n-1}) (where Varclasses is a set of variable classes)

(2) IMP will have the structure $\parallel_{p \text{ in } D} \text{IMP}_p$ where D is a subset of Integer^n and, for each p, IMP_p is a computation which produces all its output signals at point p.

(3) For each input <v, p'> to IMP_p (as defined in (2)),

$$\text{time}(p') \quad < \quad \text{time}(p)$$

This condition states that each piece of data must be produced before it can be consumed.

That condition (1) is satisfied follows from the nature of the function RENAME. Condition (2) needs to be proved when the design is complete. Its satisfaction doesn't depend on the choice of $\mathbf{Im}_t$. The condition we need to consider is (3). Although the design isn't complete, the data-dependencies are in place and $\mathbf{Im}_t$ can be tested against them. Since DATA' is a uniform recurrence, it can be shown that the required test is that $A_t.b$ should be less than zero for all dependency vectors b of DATA' (where $\mathbf{Im}_t(p) = A_t.p + b_t$.) (There will be further conditions on CONTROL'' and CONTROL''' which will have to be checked when those computations are constructed.)

Now we will schedule the convolution example, choosing $\mathbf{Im}_t$, and performing the above test.

4.2.1 Example

Let $DEP_{(CONV)}$ be the set of dependency-vectors of $DATA_{(CONV)(2)}$, then

$$DEP_{(CONV)} = \{ \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \end{bmatrix} \}$$

If we set the matrix $A_{t(CONV)}$ to be equal to [1, 2] then $A_{t(CONV)}.b < 0$ for all the dependency vectors b in $DEP_{(CONV)}$ and condition (3") on page 68 will be satisfied. We will let $b_{t(CONV)}$ equal zero for simplicity so we have

$$\mathbf{Im}_{t(CONV)} := p \rightarrow [1, 2].p$$

Figure 4.8 shows the schedule for the convolution example.

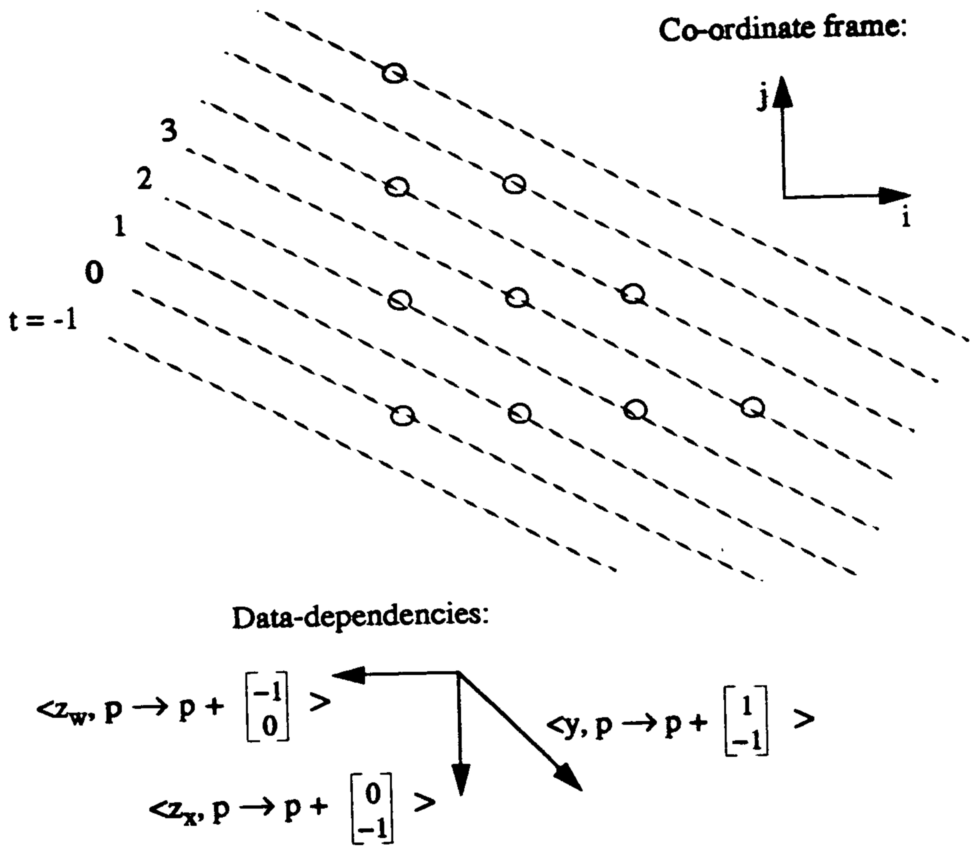

Co-ordinate frame:
j
i
3
2
1
0
t = -1
Data-dependencies:
$<z_w, p \rightarrow p + \begin{bmatrix}-1\\0\end{bmatrix}>$
$<y, p \rightarrow p + \begin{bmatrix}1\\-1\end{bmatrix}>$
$<z_x, p \rightarrow p + \begin{bmatrix}0\\-1\end{bmatrix}>$

Figure 4.8 Schedule for convolution example

The dotted lines in the figure are equitemporal. No dependency arcs are drawn in, since this might mislead: the schedule is not used until after control-pipelining has been performed. However, the data-dependency-vectors are drawn in and it can be seen that condition (3") on page 68 will be satisfied, since they all lead to earlier times.

We have now scheduled the convolution example; we will go on to the third stage: control-pipelining.

4.3 Control-pipelining

As stated earlier in this chapter, the aim of control pipelining is to transform the control computation, CONTROL', into two parts, an edge-computation, CONTROL'' which introduces control signals at the edge of the array and a uniform recurrence, CONTROL''', which transports the signals to their destinations. We will do this by dealing with each control signal separately and combining the results. Let us assume that CONTROL' may be split into several components of a certain form, each of which deals with a single control signal. Formally:

$$\text{CONTROL'} := \|_{i=1 \text{ to } n}\text{CONTROL}_{(i)} \text{ where for each } i$$

$$\text{CONTROL}_{(i)} := \begin{bmatrix} p \text{ in } \{p' \mid A_i.p' - b_i \neq 0\} & \Rightarrow & c_i(p) := 1; \\ p \text{ in } \{p' \mid A_i.p' - b_i = 0\} & \Rightarrow & c_i(p) := 0. \end{bmatrix}$$

(We are assuming here that CONTROL = $\text{CONTROL}_{(1)}$ see page 73 "Transformation 1: Data-pipelining".)

(We are here assuming that the initial control requirement is of this form, for a smaller value of n. CONTROL' may then be built up from that.)

The above definition of $\text{CONTROL}_{(i)}$ says that for each point on a certain hyperplane the control variable $c_i(p)$ has the value zero, and it has the value one elsewhere. (Note that a hyperplane is a line if the space is two-dimensional.) For each i, we will look for a computation, $\text{CONTROL}_{(i:\,1)}$, which has all its variables on the boundary of the base of DATA' and a computation, $\text{CONTROL}_{(i:\,2)}$, which is a uniform recurrence and such that $\text{CONTROL}_{(i:\,1)} \| \text{CONTROL}_{(i:\,2)}$ simulates $\text{CONTROL}_{(i)}$. Control-pipelining is similar to data-pipelining, but it is simpler since initially there are no dependencies as such; all that is required in control-pipelining is that the control-variables at several points are assigned a common value (one or zero).

Our pipelining strategy can be explained by the following analogy: imagine a light shining into a region of space and imagine that at the edge of the region there is an obstruction which casts a shadow into the region (Figure 4.9).

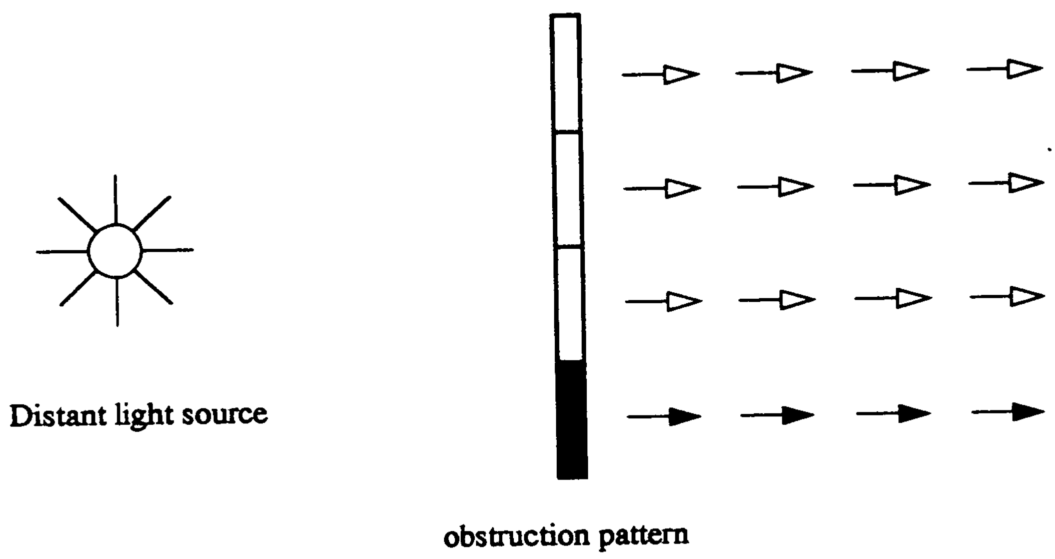


Figure 4.9 Analogy for control-pipelining

The light and dark in the region represents the value of the control variable c_i (1 or 0), the pattern of obstruction represents the edge-computation ($CONTROL_{(i)}$) and the direction the light is shining represents the direction of signal-flow through the uniform recurrence (which is transporting the control signal). We need to find a direction for the light and an obstruction pattern which will create the desired shading. The analogy breaks down slightly since we are actually dealing with a lattice of points rather than a continuous space; so we are not merely looking for a direction for the light but a vector (with a length) such that every point in the base is reachable by an integer multiple of that vector from a point on the edge of the domain. More formally we are looking for a vector r, such that for all p in D, there exists a point p_{edge} on the boundary of D and an integer n such that $p = p_{edge} - n*r$. Furthermore, in order that the shadow is cast on the correct region, r must be in the null-space of A_i (see glossary for a definition of "null-space"); this implies that r will be aligned with the dark hyperplane. If such an r can be found then we can construct the desired computations $CONTROL_{(i:\ 1)}$ and $CONTROL_{(i:\ 2)}$. If we have these for each i, then we can group all the edge-computations together to form the edge-computation CONTROL'', and all the uniform recurrences together to form the uniform recurrence CONTROL''':

$$\text{CONTROL''} := \|_{i=1 \text{ to } n}\text{CONTROL}_{(i:1)}$$

$$\text{CONTROL'''} := \|_{i=1 \text{ to } n}\text{CONTROL}_{(i:2)}$$

As mentioned earlier, there are no control dependencies at the start of the control-pipelining stage (cf. data-pipelining). Therefore it is the variable classes rather than dependencies which will be said to be pipelined. Note also that in control-pipelining, in contrast to data-pipelining, a new variable is not required to transport the signal: the control variables themselves may be used to transport it.

Figure 4.10 shows a possible CONTROL'; the numbers are the values of c_1 at each point.

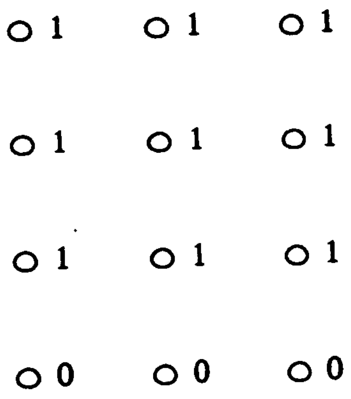


Figure 4.10 A possible CONTROL'

Figure 4.11 shows the result of pipelining.

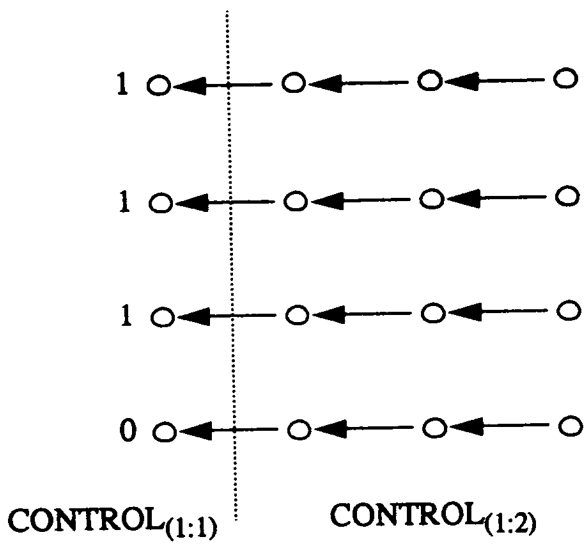


Figure 4.11 CONTROL$_{(1:1)}$ || CONTROL$_{(1:2)}$

To summarize, we have a strategy for finding an edge-computation CONTROL'' (✍ Theorem 19) and a uniform recurrence CONTROL''' (✍ Theorem 25) for which the composition CONTROL'' || CONTROL''' simulates CONTROL', which implies that CONTROL'' || (CONTROL''' || DATA') simulates CONTROL' || DATA' (✍ Theorem 12); we did this by subdividing CONTROL', operating on each subcomponent separately, and combining the results.

4.3.1 Example

In the convolution example there are three variable classes which need to be pipelined: c_y, c_x and c_w. These control variable classes correspond to the data-outputs, the data-inputs and the weights respectively. The computations which deal with these variable classes are CONTROL$_{(\text{CONV})(1)}$, CONTROL$_{(\text{CONV})(2)}$ and CONTROL$_{(\text{CONV})(3)}$ respectively (which comprise CONTROL'$_{(\text{CONV})}$ - see page 91). We can deal with each of the three subcomputations in turn.

Pipelining of the first control-variable class

Let us first consider $\text{CONTROL}_{(\text{CONV})(1)}$ (which equals $\text{CONTROL}_{(\text{CONV})}$). Looking at the definition of $\text{CONTROL}_{(\text{CONV})}$ on page 84 and noting that D_y is the set {p in D : $\begin{bmatrix}0 & 0\\0 & 1\end{bmatrix}.p = 0$}, we can see that it is of the form required for control-pipelining if we let B_1 be $\begin{bmatrix}0 & 0\\0 & 1\end{bmatrix}$ and b_1 be $\begin{bmatrix}0\\0\end{bmatrix}$. We choose our pipelining vector to be $\begin{bmatrix}-1\\0\end{bmatrix}$; it is in the null-space of B_1 and every point in $\text{BASE}_{(\text{CONV})}$ is reachable from the edge of $\text{BASE}_{(\text{CONV})}$. In fact, if $D_{(\text{CONV})(1)}$ is defined to be the set of points { $\begin{bmatrix}-1\\j\end{bmatrix}$ | j = 0...3}, then each point in $\text{BASE}_{(\text{CONV})}$ is reachable from $D_{(\text{CONV})(1)}$. Formally, for all p in $\text{BASE}_{(\text{CONV})}$ there exists p_{edge} in $D_{(\text{CONV})(1)}$ and integer n such that

$$p \;=\; p_{edge} - n^*\begin{bmatrix}-1\\0\end{bmatrix}$$

In fact if $p = \begin{bmatrix}(i)\\(j)\end{bmatrix}$ then p_{edge} is $\begin{bmatrix}-1\\j\end{bmatrix}$ and n is (i+1).Using the pipelining vector $\begin{bmatrix}-1\\0\end{bmatrix}$, we create the uniform recurrence to channel the control signals using the variable class c_y; it specifies that the value of $<c_y, p>$ is the same as that of $<c_y, p + \begin{bmatrix}-1\\0\end{bmatrix}>$ when p is in $D_{(\text{CONV})}$:

$$\text{CONTROL}_{(\text{CONV})(1:2)} \quad :=$$

$$\begin{bmatrix} p \text{ in } \text{BASE}_{(\text{CONV})} \Rightarrow c_y(p) \quad := c_y(p + \begin{bmatrix}-1\\0\end{bmatrix}). \\ \\ \end{bmatrix}$$

Then we define the edge-computation (the "obstruction pattern") as follows:

$$\text{CONTROL}_{(\text{CONV})(1:1)} \quad :=$$

$$\begin{array}{lll} \lceil \text{p in } \{ \begin{bmatrix}-1\\1\end{bmatrix}, \begin{bmatrix}-1\\2\end{bmatrix}, \begin{bmatrix}-1\\3\end{bmatrix} \} & \Rightarrow c_y(p) := 1; & \rceil \\ \lfloor \text{p in } \{ \begin{bmatrix}-1\\0\end{bmatrix} \} & \Rightarrow c_y(p) := 0. & \rfloor \end{array}$$

$\text{CONTROL}_{(\text{CONV})(1:\,1)}$ specifies that the value of $<c_y, p>$ is 1 when p is in the set $\{ \begin{bmatrix}-1\\1\end{bmatrix}, \begin{bmatrix}-1\\2\end{bmatrix}, \begin{bmatrix}-1\\3\end{bmatrix} \}$ and is 0 when p is the point $\begin{bmatrix}-1\\0\end{bmatrix}$. $\text{CONTROL}_{(\text{CONV})(1:\,1)}$ specifies the value $<c_y, p>$ only when p is in this edge-strip.

We can prove that $\text{CONTROL}_{(\text{CONV})(1:\,1)} \parallel \text{CONTROL}_{(\text{CONV})(1:\,2)}$ simulates $\text{CONTROL}_{(\text{CONV})(1)}$, so we have pipelined c_y (✍ Theorem 6).

Pipelining of the second control-variable class

The variable class c_x can be pipelined in *exactly* the same way as c_y since $\text{CONTROL}_{(\text{CONV})(2)}$ is simply a renaming of $\text{CONTROL}_{(\text{CONV})(1)}$ (c_y is replaced by c_x - see the definition of $\text{CONTROL}_{(\text{CONV})(2)}$ on page 86). We get the uniform recurrence,

$$\text{CONTROL}_{(\text{CONV})(2:\,2)} :=$$

$$\begin{array}{ll} \lceil \text{p in BASE}_{(\text{CONV})} \Rightarrow c_x(p) := c_x(p + \begin{bmatrix}-1\\0\end{bmatrix}). & \rceil \\ \lfloor & \rfloor \end{array}$$

and the edge-computation,

$$\text{CONTROL}_{(\text{CONV})(2:\,2)} :=$$

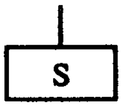
$$\begin{array}{lll} \lceil \text{p in } \{ \begin{bmatrix}-1\\1\end{bmatrix}, \begin{bmatrix}-1\\2\end{bmatrix}, \begin{bmatrix}-1\\3\end{bmatrix} \} & \Rightarrow c_x(p) := 1; & \rceil \\ \lfloor \text{p in } \{ \begin{bmatrix}-1\\0\end{bmatrix} \} & \Rightarrow c_x(p) := 0. & \rfloor \end{array}$$

We have now found a space-time simulation (n.p.) for the convolution task, but it still needs to be interpreted as hardware.

4.6 The Architecture

In this section we turn the space-time simulation $IMP_{(CONV)}$ into an architecture. This process is not part of the formal design method. The architecture is "hand-produced". Now $INTERIOR'_{(CONV)}$, corresponding to the first six lines of the shorthand expression for $IMP_{(CONV)}$, is relatively easy to turn into an architecture, but $EDGE'_{(CONV)}$, corresponding to the last six lines of the shorthand expression, is slightly awkward. The method of presentation of the control signals to the array will depend on whether a feedback loop needs to be broken into; if so, a multiplexer will be needed (otherwise not).

Figure 4.17 and Figure 4.18 show the final architecture. Figure 4.18 contains some notation which needs to be explained. The component



depicts a "black box" processor, the behaviour of which is specified by the codeword S. S signifies the set of possible character streams which may be output on the single port of the processor. There is no formal semantics for the code, but here are a few example codewords and their meaning:

"10..."signifies the set of streams such that each stream consists of a "1" followed by an infinite stream of "0"s. (There is only one element in this set.)

"1*"signifies the set of two-character lists for which the first character in the list is "1".

"1*..."signifies the set of streams which start with a "1" (which may be followed by any infinite stream of characters).

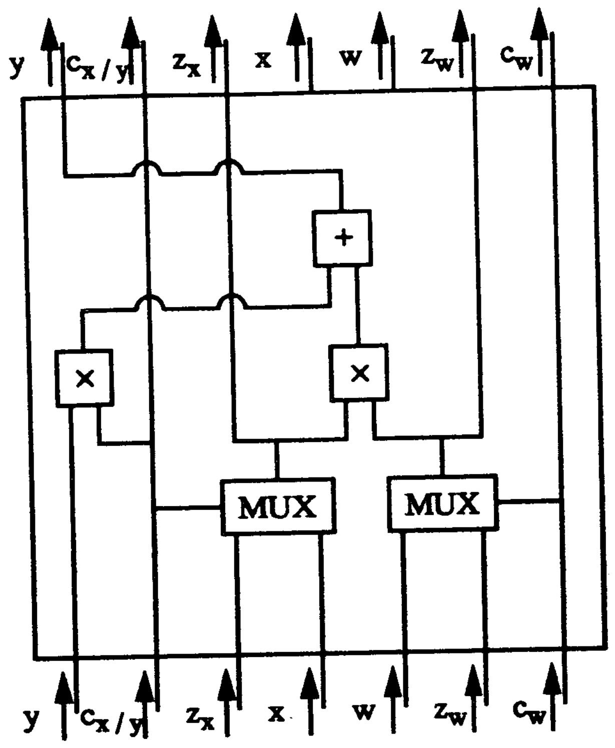


Figure 4.17 The architecture of each processor

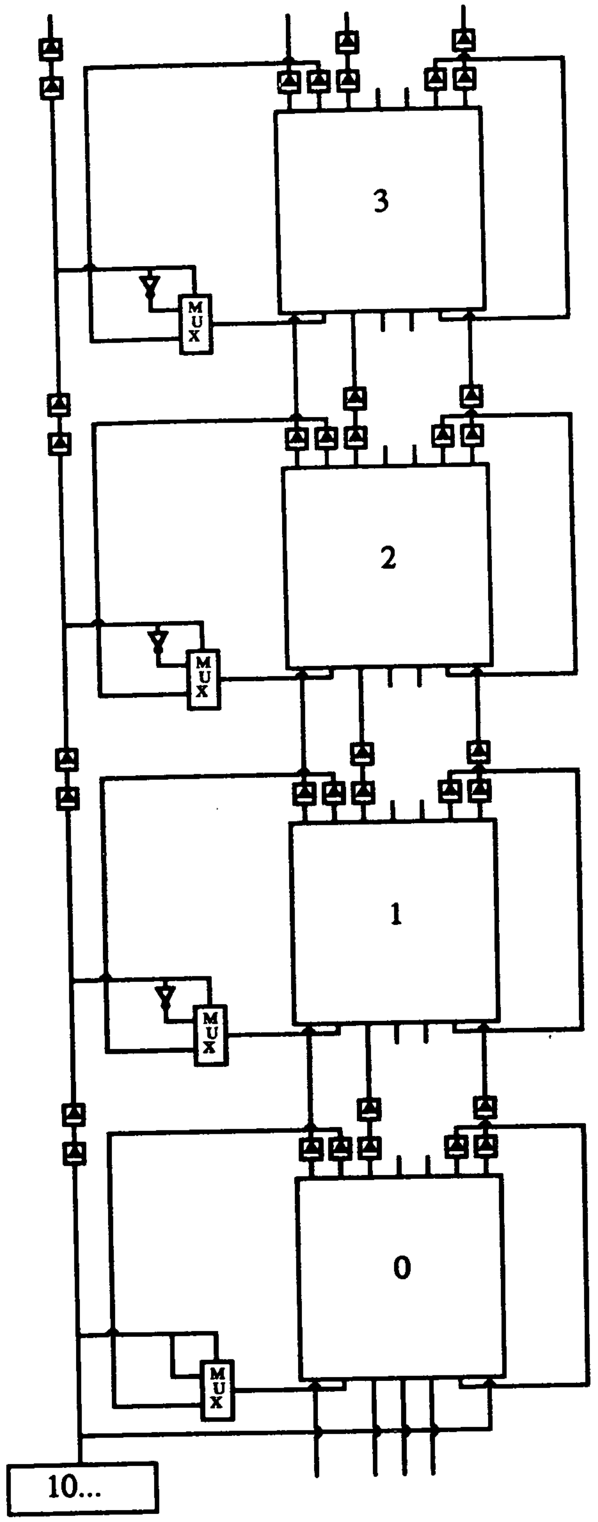


Figure 4.18 The array architecture

4.6.1 Summary of section

To summarize this section: we have turned the space-time simulation into an architecture.

4.7 Summary of chapter, discussion and further work

4.7.1 Summary

In this chapter we have seen a five-stage method of transforming a regular algorithm into an implementation which is basically systolic. Both the algorithm and the implementation are expressed in the language of computations. The method was demonstrated on a simple algorithm: convolution. The output of the method may then be transformed fairly easily into an architecture; this was seen in the case of the convolution example in the penultimate section of this chapter.

4.7.2 Discussion

The basic ideas for the steps in my design method, data-pipelining, scheduling, control-pipelining and allocation, final stage, are not new, being taken from [Raj89]. Rajopadhye's method is more sophisticated and includes many interesting ideas on pipelining; however, my method is more precisely stated than Rajopadhye's, and is verified. The sophistications of his method weren't found necessary for the convolution or QR-factorisation examples.

In Rajopadhye's method, scheduling seems to be done before data-pipelining whereas the order is reversed in my method. The rationale for the order: data-pipelining, scheduling, control-pipelining, allocation is that the more restricted choices are made before the less restricted, since each choice tends to constrain subsequent ones even more. Data-pipelining can only be done in one way. Control pipelining is more flexible: it can fit in with any schedule but not vice versa. (Of course data-pipelining *must* go before control-pipelining.) At the more detailed level, in my method of pipelining a data-dependency, p_0 can be chosen (for each coset) before the dependency vector, and can be chosen to be at the source of the data. In Rajopadhye's the dependency vector must be chosen first since its identity is completely determined by the already-chosen schedule, and the dependency vector, in turn, determines the identity of p_0. Unfortunately, p_0 may be at the other end of the line from the data source, causing an insurmountable problem. To be fair, Rajopadhye's method is also catering for situations in which his more sophisticated pipelining techniques would be used. In such situations, the choice within the data-pipelining step may be less restricted than those within the scheduling step; so by my rationale it would be sensible to schedule before data-pipelining.

If the computations used in the method are well-defined, and if a one-to-one schedule-cum-allocation function can be found, along with suitable dependency vectors for the data- and control-pipelining which are time-consistent with the function, then my method will guarantee a correct implementation to a level above the architectural level though it may not be the most efficient solution.

Automatability of the design method

If considering building a CAD system based on this method, an important question is: how automatable is the choice of pipelining vectors and the scheduling and allocation maps? If the question of optimality is ignored, this question becomes; can a pipelining vector for each data-dependency and each control variable, a schedule map and an allocation map be found which are consistent with each other? We will discuss the problem as if the choices are made in the order in which they are currently made in the method.

Data-pipelining of an affine dependency shouldn't be difficult, assuming that the following two conditions hold: the affine map (Δ_2 on page 79) is idempotent i.e. repeated application of the map to any point is the same as a single application; secondly, the base of the recurrence (BASE on page 79) is a portion of a lattice, and it doesn't have any gaps in its lattice structure i.e. it is the intersection of the lattice with a convex set of points of the Euclidean space in which the lattice is embedded. The pipelining vector can be found by performing a matrix inversion, a matrix multiplication and Euclid's algorithm (generalised to find the greatest common divisor of an arbitrary finite number of integers).

I don't know of an algorithm for finding a scheduling function which will make the data-dependency vectors time-consistent with the final space-time map. Techniques for solving integer and linear programming problems may be relevant.

Control pipelining may easily be automated. Let r be the difference between two points on ran(Δ_i). If $\mathbf{Im}_t$.b > 0, then let r_i equal b. If $\mathbf{Im}_t$.b > 0, then let r_i equal -b. There will only be a problem if $\mathbf{Im}_t$.b = 0; in this case a different pair of points may be tried.

Having chosen the scheduling map, $\mathbf{Im}_t$, allocation is done simply by finding $\mathbf{Im}_S$ such that **Im** is invertible i.e. s.t. Det(**Im**) $\neq$ 0. Assume that $\mathbf{Im}_t$, as a row vector, has a non-zero element in the i^{th} column, then we may take $\mathbf{Im}_S$ to be the identity matrix with the i^{th} row deleted.

4.7.3 Further work

Specification

The input to the method consists mainly of an affine recurrence (AR). (An AR is a formalisation of a SARE (see page 34)). In [Raj90], SARE to SARE transformations are presented which will change certain SAREs into ones of which the dependencies can more easily be made uniform. It would be interesting to see if these transformations could be formally stated and verified using the computations calculus, and to see if there are other such transformations which are valid and useful. These other transformations may rely on the associativity and commutativity of operations on the data which is drawn from a ring, as Rajopadhye's are, or they may not. In [Raj90] the transformations themselves are affine; non-affine transformations could be investigated.

It may be impossible to express some algorithms as ARs, and one could look at design methods which don't require the initial computation to be an AR. Sorter-type algorithms may fall into this category of awkward algorithm. It may be that their recursive structure makes them in general unsuitable for implementation on a lattice structure. These questions could be addressed.

Pipelining

The pipelining techniques of the method could perhaps be made more sophisticated using ideas from [Raj89], but this may not be necessary in practice. One could look at whether pipelining is always necessary for transmittant data, i.e. whether, when a signal (e.g. a control signal) travels through many subprocessors without change, it really needs to be delayed by one time step between each processor.

Scheduling and allocation

It is interesting to speculate whether scheduling and allocation could be automated. As a step towards achieving this, a constructive (in the mathematical sense) way of defining the space of valid schedules could be sought. Also, in the special case of a UR which has dependency vectors all of which are either within or on a particular plane or are a positive multiple of one of the two normals to the plane, the task of finding a schedule may reduce to scheduling within the plane. Recurrences have a "data-flow" and not a "control-flow" style: the schedule and allocation functions in my method are not conditional on the result of any computation. It would be good to incorporate such conditionality into the method. It could also be interesting to investigate non-affine schedule and allocation functions.

Implementation

The method could perhaps be adapted to allow the design of non-systolic arrays, e.g. wavefront arrays or hypercubes. The method may be more general than it appears. Non-uniformity of operations and data-flow may be simulated by introducing control signals into uniform recurrences.

Miscellaneous

Other implementations of the convolution algorithm could be investigated including those which are achieved using non-affine schedules. It would be desirable for the current method to be fully validated, i.e. for it to be proven that its computations are in fact well-defined. It would also be interesting to implement the method using LAMBDA, and to see if DIALOG could also be used as well to give the designer a graphical interface. In doing the latter project, one might see how the method could be extended to achieve the final architecture (see section 4.6 (starting on page 111)).

5 The Formal Design Method Applied to QR-Factorisation Example

We will now apply the design method to a trickier example: QR-factorisation. QR-factorisation is discussed and the algorithm to be input to the design method, $ALG_{(QR)}$, is defined. The five stages of the design are followed through. Two architectures are then shown, each resulting from a different set of design choices. The chapter finishes with a brief summary and a discussion of possible further work.

The QR-factorisation problem can be described as follows: given a square (M × M) matrix $\underline{A}$, we need to find an upper triangular matrix $\underline{R}$ which, for some orthogonal matrix $\underline{Q}$, satisfies the following equation:

$$\underline{Q}.\underline{R} \quad = \quad \underline{A} \quad (\text{that is, } \underline{R} = \underline{Q}^T.\underline{A})$$

The problem can be solved by applying a sequence of "Givens rotations" to the matrix $\underline{A}$. Each Givens rotation affects just two rows of the matrix it is applied to, and is such that it sets one of the elements in the lower of the two rows to zero. The composition (in the usual functional sense) of the rotations annihilates the lower right-hand triangle of $\underline{A}$, and can be represented by an orthogonal matrix, since each rotation can be; we can therefore set $\underline{Q}^T$ to be equal to this matrix.

We will now define the initial computation for the QR-factorization problem, $ALG^0_{(QR)}$. Firstly we need to define the domain of $ALG^0_{(QR)}$; it will be an (M × M) grid of points:

$$D_{(ALG^0)(QR)} \quad := \quad \{ \begin{bmatrix} (i) \\ (j) \end{bmatrix} \mid 1 \leq i, j \leq M \}$$

Each point in the domain corresponds to an element position in an (M × M) matrix. The variable classes A and R in $ALG^0_{(QR)}$ correspond to the matrices $\underline{A}$ and $\underline{R}$ respectively in the above problem-description; the variables which have class A are the input variables:

$$\mathrm{In}(\mathrm{ALG}^0{}_{(\mathrm{QR})}) \quad := \quad \{<A, p> \mid p \in D_{(\mathrm{ALG}^0)(\mathrm{QR})}\}$$

and those which have class R are the output variables:

$$\mathrm{Out}(\mathrm{ALG}^0{}_{(\mathrm{QR})}) \quad := \quad \{<R, p> \mid p \in D_{(\mathrm{ALG}^0)(\mathrm{QR})}\}$$

There is no variable class Q since we don't need to find $\underline{Q}$ explicitly. We then define the relation $\mathrm{Rel}(\mathrm{ALG}^0{}_{(\mathrm{QR})})$ in such a way that the values of the input and output variables are such that the corresponding matrices, $\underline{A}$ and $\underline{R}$, are related as at the start of this chapter.

$\mathrm{Rel}(\mathrm{ALG}^0{}_{(\mathrm{QR})})v \Leftrightarrow$ there exist $\underline{Q}$, $\underline{R}$ such that

(i) $\underline{R}(i, j) = v(<R, \begin{bmatrix}(i)\\(j)\end{bmatrix}>)$ if $i \leq j$

(ii) $\underline{A}(i, j) = v(<A, \begin{bmatrix}(i)\\(j)\end{bmatrix}>)$

(iii) $\underline{Q}.\underline{R} = \underline{A}$

(iv) $\underline{Q}$ is orthogonal

and

(v) $\underline{R}$ is upper-triangular.

Lines (i) and (ii) define the correspondence between the matrices and the variables of the computation: the value of the element at (i, j) in $\underline{R}$ equals the value of the variable $<R, \begin{bmatrix}(i)\\(j)\end{bmatrix}>$ and similarly for $\underline{A}$. Lines (iii) to (v) specify the constraints on and between the matrices.

Note that the value of $<R, \begin{bmatrix}(i)\\(j)\end{bmatrix}>$ is only specified when i is less than or equal to j; in other words it is only specified for the non-trivial (i.e. possibly non-zero) values. This is done so that later on the algorithm we use for solving the QR-factorization problem will not be forced to output all the zeros from the lower triangle of $\underline{R}$.

Now we will define the computation $ALG_{(QR)}$ which encapsulates the algorithm for solving QR-factorization by means of Givens rotations. Its base, $BASE_{(QR)}$, is a truncated, cube-corner pyramid (shown in Figure 5.1 for M = 5):

$$BASE_{(QR)} := \{ \begin{bmatrix}i\\j\\k\end{bmatrix} \mid k \in \{1...M\text{-}1\}, j \in \{k...M\} \text{ and } i \in \{k+1...M\}\}$$

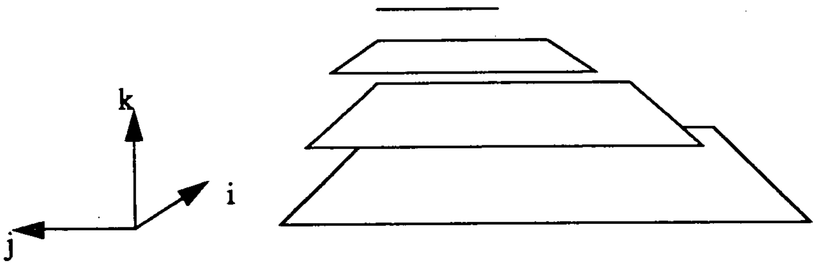


Figure 5.1 $BASE_{(QR)}$

$ALG_{(QR)}$ will be composed of a control part and a data part (as is required by my design scheme); these will be called $DATA_{(QR)}$ and $CONTROL_{(QR)}$ respectively. We will define these, but firstly we need to define a matrix, A', which will be used in the definition of $DATA_{(QR)}$:

$$A' := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

So let $\text{DATA}_{(QR)}$ be defined as follows:

$\text{DATA}_{(QR)} :=$

$$\begin{array}{lll}
\text{(i)} & \lceil p \text{ in BASE}_{(QR)} \Rightarrow \text{ox}(p) & := \text{ny}\left(p + \begin{bmatrix} -1 \\ 0 \\ -1 \end{bmatrix}\right), \quad \rceil \\
\text{(ii)} & | \quad \text{oy}(p) & := \text{cont}(p) * \text{nx}\left(p + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}\right) + \overline{\text{cont}(p)} * \text{ny}\left(p + \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}\right), \quad | \\
\text{(iii)} & | \quad \sin(p) & := \text{oy}(A'.p)/(\text{oy}(A'.p)^2 + \text{ox}(A'.p)^2)^{1/2}, \quad | \\
\text{(iv)} & | \quad \cos(p) & := \text{ox}(A'.p)/(\text{oy}(A'.p)^2 + \text{ox}(A'.p)^2)^{1/2}, \quad | \\
\text{(v)} & | \quad \text{nx}(p) & := \text{ox}(p) * \cos(p) + \text{oy}(p) * \sin(p), \quad | \\
\text{(vi)} & \lfloor \quad \text{ny}(p) & := \text{oy}(p) * \cos(p) - \text{ox}(p) * \sin(p). \quad \rfloor
\end{array}$$

Before this can be understood, more explanation of the Givens' rotation method is needed. The first rotation affects just the bottom two rows of the matrix, that is rows M-1 and M; for M = 5, the rotation matrix is:

The rotation angle θ is chosen to be such that the element position (M, 1) (that is, the Mth row and the first column) of the resultant matrix (the first of a series of intermediate matrices) is zero. For this to be true, tan (θ) must be equal to $\underline{A}(M,1)/\underline{A}(M-1,1)$. The rotation sequence ripples upwards, so the next rotation affects rows M-2 and M-1 and annihilates the element in position (M-1, 1) of the matrix it acts upon etc. When the

$$\begin{bmatrix} 1\,0\,0 & 0 & 0 \\ 0\,1\,0 & 0 & 0 \\ 0\,0\,1 & 0 & 0 \\ 0\,0\,0 & \cos\theta & \sin\theta \\ 0\,0\,0 & -\sin\theta & \cos\theta \end{bmatrix}$$

ripple reaches the top, the first column of the intermediate result matrix existing at that point consists of all zeros except for possibly the top element. The ripple then starts at the bottom again, this time eliminating elements in positions (M, 2), (M-1, 2), (M-3, 2) and so on...until row 2 is reached, at which point the ripple returns again to the bottom. This process continues until we are left with an upper-triangular matrix, as required. We may name the rotation which annihilates the element in position (i, j), "rot(i, j)"

Let us return to the definition of $DATA_{(QR)}$. The k-coordinate corresponds to the pass of the ripple through the rows: k = 1 corresponds to the first pass, k = 2 corresponds to the second pass, etc. The i and j coordinates relate in the obvious way to the position of the elements in the initial matrix, A, the intermediate matrices, and the final matrix, R.

So let us consider $DATA_{(QR)}$ at the point p where $p = \begin{bmatrix} i \\ j \\ k \end{bmatrix}$. The value of <oy, p> is the value of the element in position (i, j) of the intermediate matrix to which the rotation rot(i, k) is being or is about to be applied; the value of <ox, p> is the value in position (i-1, j) of that matrix. The cosine and sine of the rotation angle are calculated in lines (iii) and (iv) of the definition and are stored in the variables <cos, p> and <sin, p> respectively. The tangent of the angle of rot(i, k) is the value of the element in position (i, k) divided by the value of the element in position (i-1, k); the definitions in lines (iii) and (iv) follow easily from this when we note that A'.p is $\begin{bmatrix} i \\ k \\ k \end{bmatrix}$. Note that the value of <cos, p'> is going to be the same as the value of <cos, p> for all p' in the same row as p (and which are in $DATA_{(QR)}$); similarly for <sin, p>. The rotation occurs in lines (v) and (vi); in line (v) the value of the element in position (i-1, j) of the new intermediate matrix is calculated and assigned to <nx, p> and in line (vi) the value of the element in position (i, j) of the new intermediate matrix is calculated and assigned to <ny, p>. Note that the value of <ny, p> is zero, as intended, when j = k. In lines (i) and (ii), which logically precede the other lines, the values of <ox, p> and <oy, p> are brought in. The value of <ox, p> is retrieved from <ny, (p + $\begin{bmatrix}-1\\0\\-1\end{bmatrix}$)>, which belongs to the previous ripple-pass. Where the value of <oy, p> is fetched from depends on p: if k equals M then we are dealing with the first rotation in a ripple-pass, so the value of <oy, p> is fetched from <ny, (p + $\begin{bmatrix}0\\0\\-1\end{bmatrix}$)>; if k doesn't equal M then it is fetched from <nx, (p + $\begin{bmatrix}1\\0\\0\end{bmatrix}$)>, the value of which was produced by the immediately previous rotation (in the current ripple-pass). The switch between the two sources is operated by the control variable class cont, the behaviour of which is defined below in the initial control part (for an understanding of how such a switch works, see the definition of PIPE_M$_{(2)}$ on page 79).

Why are there four variable-classes as opposed to just one? Part of the reason is that the base has been made more compact than it would naturally have been - using a more straightforward approach we would have required roughly as many layers as there are intermediate matrices, whereas we use just M: one per ripple-pass plus one. We were able to do this because each rotation only affects two rows and not the whole matrix. The price we pay is that we need *two* variable classes, nx and ny; nx catches the intermediate value of each element as the ripple is passing through, and ironically it also ends up storing most of the output matrix. The variable classes ox and oy are not strictly necessary but they make the definition of DATA$_{(QR)}$ neater.

The initial control part, CONTROL$_{(QR)}$, is defined below.

$$\mathrm{CONTROL}_{(\mathrm{QR})} := \left[\begin{array}{l} p \text{ in } \mathrm{BASE}_{(\mathrm{QR})} \cap \{p' \mid [1, 0, 0].p' - M \neq 0\} \Rightarrow \mathrm{cont}(p) := 1; \\ p \text{ in } \mathrm{BASE}_{(\mathrm{QR})} \cap \{p' \mid [1, 0, 0].p' - M = 0\} \Rightarrow \mathrm{cont}(p) := 0. \end{array}\right]$$

The expression on the left-hand side of the arrow in the top line of the shorthand expression says that k doesn't equal M and the expression below it says that k equals M; so the whole definition says that if p is in $\mathrm{BASE}_{(\mathrm{QR})}$, where $p = \begin{bmatrix} i \\ j \\ k \end{bmatrix}$, then if k equals M then the value of <cont, p> is 0, otherwise it is 1.

Now we define the initial computation to be the composition of the control part and the data part:

$$\mathrm{ALG}_{(\mathrm{QR})} := \mathrm{CONTROL}_{(\mathrm{QR})} \parallel \mathrm{DATA}_{(\mathrm{QR})} .$$

We will now link up the Given's rotation algorithm, $\mathrm{ALG}_{(\mathrm{QR})}$, with the definition of QR-factorization, $\mathrm{ALG}^0_{(\mathrm{QR})}$:

$\mathrm{ALG}_{(\mathrm{QR})}$ simulates $\mathrm{ALG}^0_{(\mathrm{QR})}$ with respect to <Varset, RENAME>

where

$$\mathrm{RENAME}(<ny, \begin{bmatrix} i \\ j \\ 0 \end{bmatrix}>) := <A, \begin{bmatrix} (i) \\ (j) \end{bmatrix}>$$

$$\mathrm{RENAME}(<nx, \begin{bmatrix} i+1 \\ j \\ i \end{bmatrix}>) := <R, \begin{bmatrix} (i) \\ (j) \end{bmatrix}> \quad \text{if } i \neq M$$

$$\mathrm{RENAME}(<ny, \begin{bmatrix} i \\ j \\ i-1 \end{bmatrix}>) := <R, \begin{bmatrix} (i) \\ (j) \end{bmatrix}> \quad \text{if } i = M$$

$$\text{RENAME}(\langle nx, \begin{bmatrix} i \\ j \\ 0 \end{bmatrix}\rangle) \quad := \langle R, \begin{bmatrix} (i) \\ (j) \end{bmatrix}\rangle \quad \text{if } i > j$$

and Varset := Vars($ALG_{(QR)}$)-({<ny, p> | [0, 0, 1].p = 0}

$$\cup \{\langle nx, p\rangle \mid p = \begin{bmatrix} i+1 \\ j \\ i \end{bmatrix} \text{ for some } i, j\}$$

$$\cup \{\langle ny, p\rangle \mid p = \begin{bmatrix} i \\ j \\ i-1 \end{bmatrix} \text{ for some } i, j\}$$

$$\cup \{\langle nx, p\rangle \mid p = \begin{bmatrix} i \\ j \\ 0 \end{bmatrix} \text{ for some } i, j \text{ where } i > j\})$$

The function RENAME defines the connection between the inputs and outputs of $ALG^0_{(QR)}$ and the variables of $ALG_{(QR)}$. The first line of the definition states that the elements of the input matrix, A̲, are found on the plane below $BASE_{(QR)}$, stored in the obvious way in the variable-class ny. The output matrix doesn't appear quite so neatly; for a start only the "upper triangle" appears. (Though the rest of the matrix seems to be accounted for in line four of the definition, this part of the definition of RENAME is dummy, just put in to satisfy the criteria of simulation - that all the variables of the computation being simulated must be in the range of RENAME. The value of the variables $\langle nx, \begin{bmatrix} i \\ j \\ 0 \end{bmatrix}\rangle$ will not necessarily be zero when i > j, but this doesn't matter since, in the definition of $ALG^0_{(QR)}$, the value of $\langle R, \begin{bmatrix} (i) \\ (j) \end{bmatrix}\rangle$ is unspecified if i > j.) The possibly non-zero elements of the first M-1 rows appear on the one of the two sloping faces of $BASE_{(QR)}$ in the variable-class nx, a bit like the flotsam left on the beach by the receding tide (to pursue the ripple analogy); this is stated in the second line of the definition. The possibly non-zero element of the last row is stored in the variable <ny,

$\begin{bmatrix} M \\ M \\ M-1 \end{bmatrix}$ >, as stated in the third line. The set Varset details all the variables which are not used either for inputting the matrix $\underline{A}$ or for outputting $\underline{R}$. In other words it is all the variables of $ALG_{(QR)}$ except the ones mentioned in the four lines which define RENAME. (This can be seen in the structure of its definition.)

$ALG_{(QR)}$ (depicted in Figure 5.2 and Figure 5.3) is more complicated than $ALG_{(CONV)}$: it has more variable classes and the space in which it is embedded has three rather than two dimensions. However, the techniques which will be used in each of the four design stages are the same as those used for the convolution example and in fact no more data-dependencies and no more control-variable classes need to be pipelined than in the convolution example.

Assume that M = 5. Figure 5.1 on page 121 shows a 3-D view of the four k-planes (the planes which appear horizontal in Figure 5.1). Figure 5.2 shows the data-dependencies in the plane in which k=1. Figure 5.3 shows the dependencies in the vertical plane in which j = M. As in the case of the convolution example, the control part is invisible.

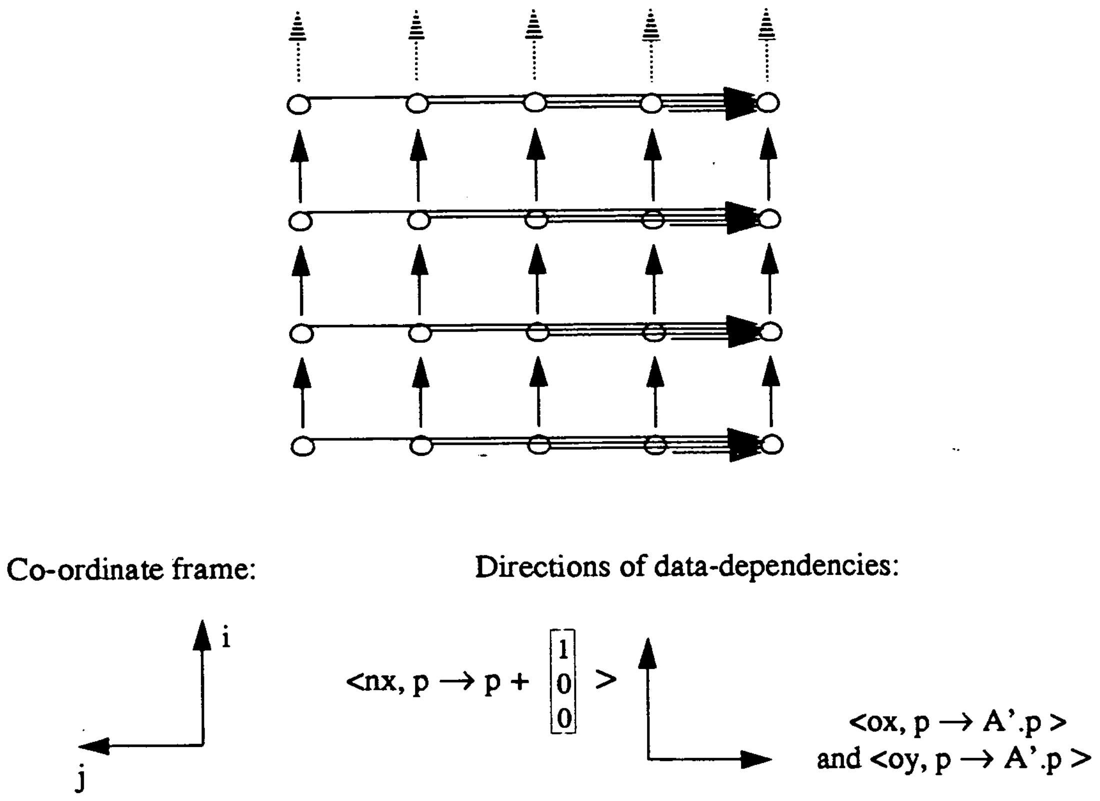


Figure 5.2 ALG$_{(QR)}$: (Horizontal Plane: k = 1)

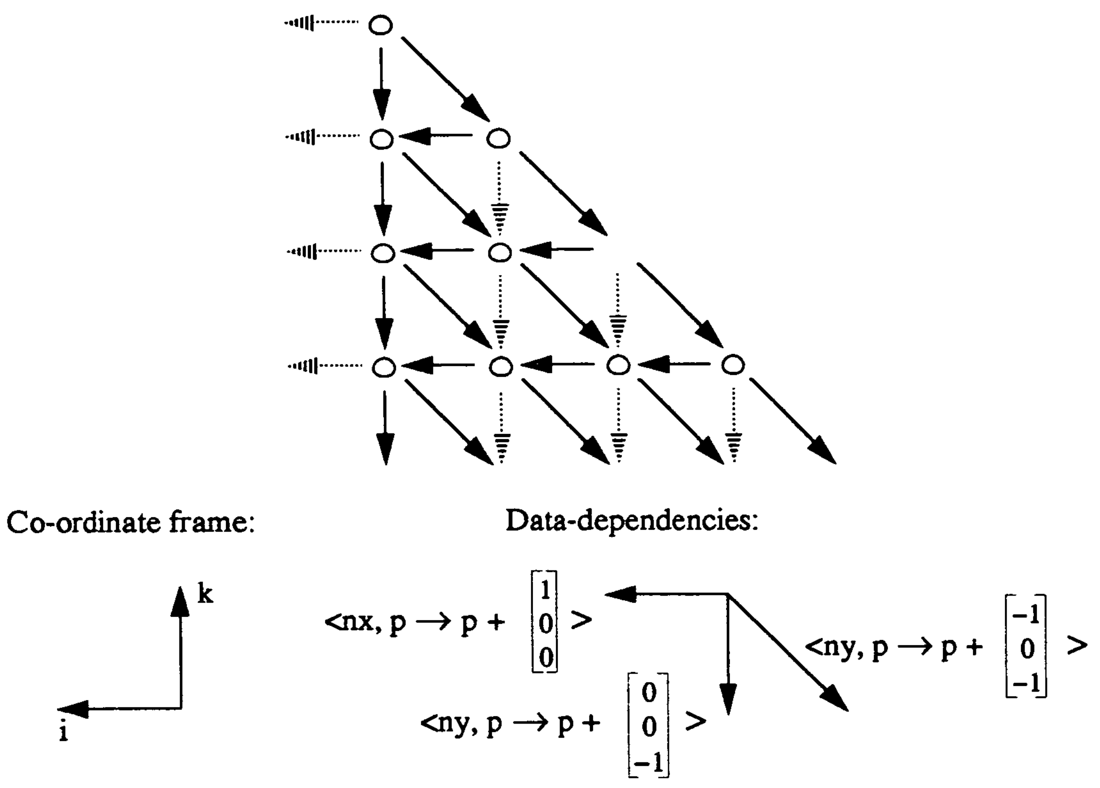


Figure 5.3 $ALG_{(QR)}$: (vertical plane: j = 5)

In this section we have given a high-level definition of QR-factorization as a computation and then defined the Givens method of performing it, also as a computation. This latter computation is of a suitable form to be input into my method and it is this and not the higher-level definition which we will treat as the initial computation. We will now go through each of the design stages. For each stage, one design choice will be presented...and then other options will be briefly investigated.

5.1 Data-pipelining

There are only two dependencies which need to be pipelined, one involving the variable class ox and the other involving the variable class oy. It turns out that the two control requirements generated will have identical values at each point. In the architecture, just one signal is used to satisfy both requirements (though in the space-time simulation,

$\mathrm{IMP}_{(\mathbf{QR})}$, there are two (identical) control signals, c_{ox} and c_{oy}).

We can find ox and oy in lines (iii) and (iv) of $\mathrm{DATA}_{(\mathbf{QR})}$. Let us pipeline the dependency <ox, p $\rightarrow$ A'.p> first. Recall from section 4.1 (starting on page 78) that we need to find a pipelining vector such that all the points in a coset are a multiple of the vector away from the first point in the coset-row; and recall furthermore that we need to name a new variable-class (z_2) to transport the data in the new pipe and a new control variable-class (c_2) to act as a switch which is off or on depending on whether or not we are at the beginning of the coset-row. In this case let the pipelining vector be $\begin{bmatrix}0\\-1\\0\end{bmatrix}$, let z_2 be identified with z_{ox} and c_2 be identified with c_{ox}. The following definitions have the same pattern as those for the convolution example (see page 85).

$$\mathrm{In}(\mathrm{PIPE\_M}_{(\mathbf{QR})(2)}) = \{<c_{ox}, p \rightarrow p>, <z_{ox}, p \rightarrow p + \begin{bmatrix}0\\-1\\0\end{bmatrix}>, <ox, p \rightarrow p>\}$$

$$\mathrm{Out}(\mathrm{PIPE\_M}_{(\mathbf{QR})(2)}) = \{<z_{ox}, p \rightarrow p>\}$$

$$\mathrm{Rel}(\mathrm{PIPE\_M}_{(\mathbf{QR})(2)}) \Leftrightarrow$$

$$\begin{aligned} v(<z_{ox}, p \rightarrow p>) \quad = \quad & v(<c_{ox}, p \rightarrow p>)*v(<z_{ox}, p \rightarrow p + \begin{bmatrix}0\\-1\\0\end{bmatrix}>) \\ & + \quad \bar{v}(<c_{ox}, p \rightarrow p>)*v(<ox, p \rightarrow p>) \end{aligned}$$

(This definition for $\mathrm{PIPE\_M}_{(\mathbf{QR})(2)}$ corresponds to the definition for $\mathrm{PIPE\_M}_{(2)}$ on page 79.)

$$\mathrm{R\_DP}_{(\mathbf{QR})(2)}(<ox, p \rightarrow A'.p>) := <ox, p \rightarrow p + \begin{bmatrix}0\\-1\\0\end{bmatrix}>$$

and for all <a', Δ'> in $\mathrm{Vars}(\mathrm{DATA\_M}_{(\mathbf{QR})(2)})$ not equal to <ox, p $\rightarrow$ A'.p>,

$$\mathrm{R\_DP}_{(\mathbf{QR})(2)}(<a', \Delta'>) := \; <a', \Delta'>$$

$$\mathrm{DATA}_{(QR)(2)} := \|_{p\in \mathrm{BASE}} \ \mathrm{DATA\_M}_{(QR)(2)} ® \mathrm{R\_DATA}_{(QR)(2:p)}$$

where

$$\mathrm{DATA\_M}_{(QR)(2)} := \mathrm{DATA\_M}_{(QR)} ® \mathrm{R\_DP}_{(QR)(2)} \| \mathrm{PIPE\_M}_{(QR)(2)}$$

and

$$\mathrm{R\_DATA}_{(QR)(2:p)}(<vc, fun>) = <vc, fun(p)>$$

for all pairs <vc, fun> in $\mathrm{Vars}(\mathrm{DATA\_M}_{(QR)(2)})$

and $\mathrm{DATA\_M}_{(QR)}$ is s.t.

$$\mathrm{In}(\mathrm{DATA\_M}_{(QR)}) := \{<ny, p \to p + \begin{bmatrix}-1\\0\\-1\end{bmatrix}>, <cont, p \to p>,$$

$$<nx, p \to p + \begin{bmatrix}1\\0\\0\end{bmatrix}>, <ny, p \to p+ \begin{bmatrix}0\\0\\-1\end{bmatrix}>,$$

$$<oy, p \to A'.p>, <ox, p \to A'.p>\}$$

$$\mathrm{Out}(\mathrm{DATA\_M}_{(QR)}) := \{<ox, p \to p>, <oy, p \to p>, <sin, p \to p>,$$

$$<cos, p \to p>, <nx, p \to p>, <ny, p \to p>\}$$

$$\mathrm{Rel}(\mathrm{DATA\_M}_{(QR)})(v) \Leftrightarrow$$

$$v(<ox, p \to p>) = v(<ny, p \to p + \begin{bmatrix}-1\\0\\-1\end{bmatrix}>)$$

and $v(<oy, p \to p>) = v(<cont, p \to p>)*v(<nx, p \to p + \begin{bmatrix}1\\0\\0\end{bmatrix}>)$

$$+ v(<cont, p \to p>)*\bar{v}(<ny, p \to p+ \begin{bmatrix}0\\0\\-1\end{bmatrix}>)$$

and $v(<sin, p \to p>) =$

$$v(<oy, p \to A'.p>)/((v(<oy, p \to A'.p>))^2 + (v(<ox, p \to A'.p>))^2)^{1/2}$$

and $v(<cos, p \to p>) =$

$$v(<ox, p \rightarrow A'.p>)/((v(<oy, p \rightarrow A'.p>))^2 + (v(<ox, p \rightarrow A'.p>))^2)^{1/2}$$

and $v(<nx, p \rightarrow p>) =$

$$v(<ox, p \rightarrow p>)*v(<cos, p \rightarrow p>) + v(<oy, p \rightarrow p>)*v(<sin, p \rightarrow p>)$$

and $v(<ny, p \rightarrow p>) =$

$$v(<oy, p \rightarrow p>)*v(<cos, p \rightarrow p>) - v(<ox, p \rightarrow p>)*v(<sin, p \rightarrow p>)$$

($DATA\_M_{(QR)}$ is a mould for $DATA_{(QR)}$. $DATA_{(QR)(2)}$ corresponds to $DATA_{(2)}$, defined on page 80.)

We need also to define the computation that defines the behaviour of the switch, c_{ox}:

$$CONTROL_{(QR)(2)} := \begin{bmatrix} p \text{ in } BASE_{(QR)} \cap \{p' \mid A'.p' \neq p'\} \Rightarrow & c_{ox}(p) := 1; \\ p \text{ in } BASE_{(QR)} \cap \{p' \mid A'.p' = p'\} \Rightarrow & c_{ox}(p) := 0. \end{bmatrix}$$

In other words the value of $c_{ox}(p)$ is 0 if p is on the sloping face of the pyramid which is the set $\{p' \mid A'.p' = p'\}$ and is 1 elsewhere in the pyramid. This definition corresponds exactly to the definition of $CONTROL_{(2)}$ on page 80 (note that in this example Δ is the function $p \longrightarrow A'.p$). Assuming that certain computations are well-defined, we may now deduce from Theorem 2 that:

$$CONTROL_{(QR)(2)} \parallel DATA_{(QR)(2)} \text{ simulates } DATA_{(QR)} \text{ (n.p.)}$$

We may operate on $DATA_{(QR)(2)}$ to pipeline the dependency $<oy, p \longrightarrow A'.p>$ in *exactly* the same way in which we operated on $DATA_{(QR)}$ to pipeline the dependency $<ox, p \longrightarrow A'.p>$:

$$In(PIPE\_M_{(QR)(3)}) = \{<c_{oy}, p \rightarrow p>, <z_{oy}, p \rightarrow p + \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}>, <oy, p \rightarrow p>\}$$

$$Out(PIPE\_M_{(QR)(3)}) = \{<z_{oy}, p \rightarrow p>\}$$

$\mathrm{Rel}(\mathrm{PIPE\_M}_{(QR)(3)}) \Leftrightarrow$

$$v(<z_{oy}, p \rightarrow p>) \;=\; v(<c_{oy}, p \rightarrow p>)*v(<z_{oy}, p \rightarrow p + \begin{bmatrix}0\\-1\\0\end{bmatrix}>)$$
$$+\; \bar{v}(<c_{oy}, p \rightarrow p>)*v(<oy, p \rightarrow p>)$$

$$\mathrm{R\_DP}_{(QR)(3)}(<oy, p \rightarrow A'.p>) := <oy, p \rightarrow p + \begin{bmatrix}0\\-1\\0\end{bmatrix}>$$

and for all $<a', \Delta'>$in $\mathrm{Vars}(\mathrm{DATA\_M}_{(QR)(3)})$ not equal to $<oy, p \rightarrow A'.p>$,

$$\mathrm{R\_DP}_{(QR)(3)}(<a', \Delta'>) := \;<a', \Delta'>$$

$$\mathrm{DATA}_{(QR)(3)} \quad := \; \|_{p \in \mathrm{BASE}} \;\; \mathrm{DATA\_M}_{(QR)(3)} \circledR \mathrm{R\_DATA}_{(QR)(3\,:\,p)}$$

where

$$\mathrm{DATA\_M}_{(QR)(3)} \;:=\; \mathrm{DATA\_M}_{(QR)(2)} \circledR \mathrm{R\_DP}_{(QR)(3)} \| \mathrm{PIPE\_M}_{(QR)(3)}$$

and

$$\mathrm{R\_DATA}_{(QR)(3\,:\,p)}(<vc, fun>) = \;<vc, fun(p)>$$
for all pairs $<vc, fun>$ in $\mathrm{Vars}(\mathrm{DATA\_M}_{(QR)(3)})$

We need also to define the computation that defines the behaviour of the switch, c_{oy}:

$\mathrm{CONTROL}_{(QR)(3)} :=$

$$\begin{bmatrix} p \text{ in } \mathrm{BASE}_{(QR)} \cap \{p' \mid A'.p' \neq p'\} \Rightarrow c_{oy}(p) := 1; \\ p \text{ in } \mathrm{BASE}_{(QR)} \cap \{p' \mid A'.p' = p'\} \Rightarrow c_{oy}(p) := 0. \end{bmatrix}$$

We may now deduce from Theorem 2 that:

$\mathrm{CONTROL}_{(QR)(3)} \| \mathrm{DATA}_{(QR)(3)}$ simulates $\mathrm{DATA}_{(QR)(2)}$ (n.p.)

We have pipelined the two dependencies $<ox, p \rightarrow A'.p>$ and $<oy, p \rightarrow A'.p>$ and in doing so transformed the affine recurrence $\mathrm{DATA}_{(QR)}$ into the uniform recurrence

$\mathrm{DATA}_{(\mathrm{QR})(3)}$, with the generation of the two control computations: $\mathrm{CONTROL}_{(\mathrm{QR})(2)}$ and $\mathrm{CONTROL}_{(\mathrm{QR})(3)}$. Let us give the new name $\mathrm{CONTROL}_{(\mathrm{QR})(1)}$ to the initial control computation $\mathrm{CONTROL}_{(\mathrm{QR})}$ and let us define $\mathrm{CONTROL'}_{(\mathrm{QR})}$ to be the composition of the three control computations:

$$\mathrm{CONTROL'}_{(\mathrm{QR})} \quad := \quad (\|_{i=1\ \mathrm{to}\ 3}\mathrm{CONTROL}_{(\mathrm{QR})(i)})$$

and set $\mathrm{DATA'}_{(\mathrm{QR})}$ equal to $\mathrm{DATA}_{(\mathrm{QR})(3)}$:

$$\mathrm{DATA'}_{(\mathrm{QR})} \qquad := \quad \mathrm{DATA}_{(\mathrm{QR})(3)}$$

Then $\mathrm{DATA'}_{(\mathrm{QR})}$ is a uniform recurrence, all the variables of $\mathrm{CONTROL'}_{(\mathrm{QR})}$ are on the boundary of the base of $\mathrm{DATA'}_{(\mathrm{QR})}$, and $\mathrm{CONTROL'}_{(\mathrm{QR})} \| \mathrm{DATA'}_{(\mathrm{QR})}$ simulates $\mathrm{CONTROL}_{(\mathrm{QR})} \| \mathrm{DATA}_{(\mathrm{QR})}$. That is:

$$\mathrm{CONTROL'}_{(\mathrm{QR})} \| \mathrm{DATA'}_{(\mathrm{QR})} \text{ simulates } \mathrm{ALG}_{(\mathrm{QR})} \text{ (n.p.)}$$

We have now completed the data-pipelining stage for the QR-factorization example. The question is, "Can it be pipelined in any other way?"...

5.1.1 Other Options

Only two dependencies were pipelined, resulting in the same dependency vector, $\begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}$.

There is no other way to pipeline these dependencies. The same reason applies to each. Each coset has only two ends, and only one of these is the source of the data, so p_0 must be the point at this end; having chosen p_0 there is only one choice of dependency vector (see Theorem 3 on page 218).

5.2 Scheduling

Let $\mathrm{DEP}_{(\mathrm{QR})}$ be the set of data-dependency vectors in $\mathrm{DATA'}_{(\mathrm{QR})}$, then

$$\mathrm{DEP}_{(QR)}=\{\begin{bmatrix}-1\\0\\-1\end{bmatrix},\begin{bmatrix}1\\0\\0\end{bmatrix},\begin{bmatrix}0\\0\\-1\end{bmatrix},\begin{bmatrix}0\\-1\\0\end{bmatrix}\}$$

In choosing the scheduling function, $\mathbf{Im}_{t(QR)}$, where $\mathbf{Im}_{t(QR)}(p) = \mathrm{A}_{t(QR)}.p + \mathbf{b}_{t(QR)}$, we must satisfy the condition that $\mathrm{A}_{t(QR)}.b$ is less than zero for all dependency vectors b in $\mathrm{DEP}_{(QR)}$ (see section 4.2 (starting on page 92)). From this we can deduce that if $\mathrm{A}_t = [\alpha, \beta, \gamma]$ then $\alpha < 0, \beta > 0, \gamma > 0$ and $\alpha + \gamma > 0$.

The matrix [-1, 1, 2] fits the bill for $\mathrm{A}_{t(QR)}$. Any value for $\mathbf{b}_{t(QR)}$ is satisfactory, so let $\mathbf{b}_{t(QR)}$ be zero for simplicity.

5.2.1 Other Options

If a, b, and c are to be integral then the matrix which was chosen, namely [-1, 1, 2], is the best, i.e. the modulus of each component is no bigger than the modulus of the corresponding component of every other suitable matrix. (✍ Theorem 14)

5.3 Control Pipelining

As in the case of convolution, there are three control-variable classes, namely cont, c_{ox} and c_{oy}; each of them needs to be pipelined. They correspond (respectively) to the three computations, $\mathrm{CONTROL}_{(QR)(1)}$, $\mathrm{CONTROL}_{(QR)(2)}$ and $\mathrm{CONTROL}_{(QR)(3)}$ which, we recall, comprise the control part resulting from data-pipelining, $\mathrm{CONTROL'}_{(QR)}$ (see page 134).

Let us first consider $\mathrm{CONTROL}_{(QR)(1)}$.

5.3.1 Pipelining of cont

Let $\mathrm{A}_{1(QR)}$ and $\mathrm{b}_{1(QR)}$ be the row-vector and the integer which characterize the value-pattern of cont ($\mathrm{CONTROL}_{(QR)}$ equals $\mathrm{CONTROL}_{(QR)(1)}$ from the definition on page

125). From this definition, we know that $A_{1(QR)} = [1, 0, 0]$ and $b_{1(QR)} = M$.

We will define $D_{(QR)(1)}$ to be the part of the recurrence edge where the control signals corresponding to the variable name "cont" are to be fed in. Before we do this, we need to choose the pipelining vector. Let us choose the vector $\begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}$ (note that we have chosen the vector to be in the null space of $A_{1(QR)}$). Then let us define $D_{(QR)(1)}$ to be those points which are in the image of the map $p \rightarrow p + \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}$ which are not in the base $BASE_{(QR)}$; formally:

$$D_{(QR)(1)} := \{ (p + \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}) \mid p \in BASE_{(QR)} \} - BASE_{(QR)}$$

which equals the set of points:

$$\{ \begin{bmatrix} (i) \\ (j) \\ (k) \end{bmatrix} \mid k = j+1 \text{ and } 1 \leq k \leq M-1 \text{ and } k+1 \leq i \leq M \}$$

This set borders one of the sloping faces of the pyramid which is $BASE_{(QR)}$. Let us divide this region into two disjoint subsets, $D_{(QR)(1:0)}$ and $D_{(QR)(1:1)}$:

$$D_{(QR)(1:0)} := D_{(QR)(1)} \cap \{ \begin{bmatrix} (i) \\ (j) \\ (k) \end{bmatrix} : i = M \}$$

$D_{(QR)(1:0)}$ therefore consists of the points in $D_{(QR)(1)}$ for which $i = M$.

$$D_{(QR)(1:1)} := D_{(QR)(1)} \cap \{ \begin{bmatrix} (i) \\ (j) \\ (k) \end{bmatrix} : i \neq M \}$$

$D_{(QR)(1:0)}$ consists of all the other points in $D_{(QR)(1)}$.

We will pipe in the value 0 from $D_{(QR)(1:0)}$ and the value 1 from $D_{(QR)(1:1)}$ using the vector $\begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}$ .The validity of this piping depends on the fact that each point which needs a 0 (that is, each point p in $BASE_{(QR)}$ for which [1, 0, 0].p - M = 0) can be reached by the vector from $D_{(QR)(1:0)}$ and each point which needs the value 1 (all the other points in $BASE_{(QR)}$) can be reached from $D_{(QR)(1:1)}$; formally:

For all p in $BASE_{(QR)}$,

[1, 0, 0].p - M = 0 implies that there in exist p_{edge} in $D_{(QR)(1:0)}$ and integer n such that

$$p = p_{edge} + n\begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}$$

and [1, 0, 0].p - M ≠ 0 implies that there exist p_{edge} in $D_{(QR)(1:1)}$ and integer n such that

$$p = p_{edge} + n\begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}$$

This fact is true because of our careful choice of the pipelining vector and the regions $D_{(QR)(1)}$, $D_{(QR)(1:0)}$ and $D_{(QR)(1:1)}$ (see Theorem 9).

We note also that $\begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}$ is time-consistent (with $\mathbf{Im}_{(QR)}$); this fact is not needed right now, but is necessary for the validity of the final space-time simulation.

The pipelining process results in the uniform recurrence $CONTROL_{(QR)(1:2)}$ and the edge-computation $CONTROL_{(QR)(1:1)}$, defined below:

$$\text{CONTROL}_{(\mathbf{QR})(1:1)} := \begin{bmatrix} \text{p in } D_{(\mathbf{QR})(1:0)} \Rightarrow \text{cont(p)} := 0; \\ \text{p in } D_{(\mathbf{QR})(1:1)} \Rightarrow \text{cont(p)} := 1. \end{bmatrix}$$

$$\text{CONTROL}_{(\mathbf{QR})(1:2)} := \begin{bmatrix} \text{p in BASE}_{(\mathbf{QR})} \Rightarrow \text{cont(p)} := \text{cont}(\text{p} + \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}). \\ \\ \end{bmatrix}$$

$\text{CONTROL}_{(\mathbf{QR})(1:2)}$ passes the values of cont from point to point in the direction $\begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}$, and $\text{CONTROL}_{(\mathbf{QR})(1:1)}$ feeds in the values at the edge of the array as already described. (It forms the obstruction pattern, to return to the light analogy.) We can prove that the composition of these two computations simulates $\text{CONTROL}_{(\mathbf{QR})(1)}$:

$\text{CONTROL}_{(\mathbf{QR})(1:1)} \parallel \text{CONTROL}_{(\mathbf{QR})(1:2)}$ simulates $\text{CONTROL}_{(\mathbf{QR})(1)}$.

(✍ Theorem 9)

We can pipeline c_{ox} using similar reasoning:

5.3.2 Pipelining of c_{ox}

Recall the definition of its corresponding computation, $\text{CONTROL}_{(\mathbf{QR})(2)}$, from page 132:

$$\text{CONTROL}_{(\mathbf{QR})(2)} :=$$
$$\begin{bmatrix} \text{p in BASE}_{(\mathbf{QR})} \cap \{\text{p}' \mid \text{A}'.\text{p}' \neq \text{p}'\} \Rightarrow c_{ox}(\text{p}) := 1; \\ \text{p in BASE}_{(\mathbf{QR})} \cap \{\text{p}' \mid \text{A}'.\text{p}' = \text{p}'\} \Rightarrow c_{ox}(\text{p}) := 0. \end{bmatrix}$$

The conditions on the left-hand sides of the double arrows are not in the same form as the general definition of $\text{CONTROL}_{(i)}$ on page 95, so we need to re-jig them to work out what $A_{2(\mathbf{QR})}$ and $b_{2(\mathbf{QR})}$ are. In fact we just have to look at the precondition on the second line. The condition that the matrix product of A' and p be equal to p,

$$A'.p = p$$

is equivalent to

$$(A' - I).p = 0 (I \text{ is the identity matrix})$$

which is equivalent to

$$[0, -1, 1].p = 0 \text{ (since } A' - I = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix})$$

From this we can see that

$$A_{2(QR)} = [0, -1, 1]$$

and $b_{2(QR)} = 0$

As before, let us choose a pipelining vector, and let it be $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ (which is in the null space of $A_{2(QR)}$). Let us define $D_{(QR)(2)}$ to be the part of the array boundary where the control signals corresponding to the variable name c_{ox} are fed in. $D_{(QR)(2)}$ can be deduced in exactly the same way as $D_{(QR)(1)}$, by taking the image of the function $p \rightarrow p + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ on $BASE_{(QR)}$ and then discarding the points of $BASE_{(QR)}$ itself:

$$D_{(QR)(2)} := \{ (p + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}) \, p \in BASE_{(QR)} \} - BASE_{(QR)}$$

which equals

$$\{ \begin{bmatrix} (i) \\ (j) \\ (k) \end{bmatrix} \mid i = M+1 \text{ and } 1 \leq k \leq M-1 \text{ and } k \leq j \leq M\}$$

This set of points neighbours the vertical back plane of the pyramid as drawn in Figure 5.1.

As before, we define the subregions from which we feed the values 0 and 1 into the array:

$$D_{(QR)(2:0)} := D_{(QR)(2)} \cap \{ \begin{bmatrix} (i) \\ (j) \\ (k) \end{bmatrix} : j = k\}$$

$$D_{(QR)(2:1)} := D_{(QR)(2)} \cap \{ \begin{bmatrix} (i) \\ (j) \\ (k) \end{bmatrix} : j \neq k\}$$

$D_{(QR)(2:0)}$ consists of all the points in $D_{(QR)(2)}$ for which $j = k$ and $D_{(QR)(2:1)}$ consists of all other points in $D_{(QR)(2)}$. We then find that each point in $BASE_{(QR)}$ which requires a zero (that is, each point p in $BASE_{(QR)}$ for which $[0, -1, 1].p = 0$) is reachable from $D_{(QR)(2:0)}$ using the vector and each the point which requires a one is reachable from $D_{(QR)(2:1)}$; formally (see Theorem 10):

For all p in $BASE_{(QR)}$, $[0, -1, 1].p = 0$ implies that there exist $p_{edge} \in D_{(QR)(2:0)}$ and integer n such that

$$p = p_{edge} - n \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

and $[0, -1, 1].p \neq 0$ implies that there exist $p_{edge} \in D_{(QR)(2:1)}$ and integer n such that

$$p = p_{edge} - n\begin{bmatrix}1\\0\\0\end{bmatrix}$$

This means that we can pipeline c_{ox} using the dependency vector $\begin{bmatrix}1\\0\\0\end{bmatrix}$, feeding in the value 0 from the region $D_{(QR)(2:0)}$ and the value 1 from the region $D_{(QR)(2:1)}$. Note that $\begin{bmatrix}1\\0\\0\end{bmatrix}$ is time-consistent (with $\mathbf{Im}_{(QR)}$). The pipelining process results in the uniform recurrence $\mathrm{CONTROL}_{(QR)(2:2)}$ and the edge-computation $\mathrm{CONTROL}_{(QR)(2:1)}$, defined below:

$$\mathrm{CONTROL}_{(QR)(2:1)} := \begin{bmatrix} p \text{ in } D_{(QR)(2:0)} \Rightarrow c_{ox}(p) := 0; \\ p \text{ in } D_{(QR)(2:1)} \Rightarrow c_{ox}(p) := 1. \end{bmatrix}$$

$$\mathrm{CONTROL}_{(QR)(2:2)} := \begin{bmatrix} p \text{ in } \mathrm{BASE}_{(QR)} \Rightarrow c_{ox}(p) := c_{ox}(p + \begin{bmatrix}1\\0\\0\end{bmatrix}). \\ \\ \end{bmatrix}$$

$\mathrm{CONTROL}_{(QR)(2:2)}$ passes the values of cont from point to point in the direction $\begin{bmatrix}0\\-1\\0\end{bmatrix}$, and $\mathrm{CONTROL}_{(QR)(2:1)}$ feeds in the values at the edge of the array as already described. We can prove that the composition of these two computations simulates $\mathrm{CONTROL}_{(QR)(2)}$:

$\mathrm{CONTROL}_{(QR)(2:1)} \| \mathrm{CONTROL}_{(QR)(2:2)}$ simulates $\mathrm{CONTROL}_{(QR)(2)}$ (✍ Theorem

10)

5.3.3 Pipelining of c_{oy}

We can now apply *exactly* the same method to the variable-class c_{oy} (and its corresponding computation, $\text{CONTROL}_{(QR)(3)}$) .

The definitions of the resulting computations are

$$\text{CONTROL}_{(QR)(3:1)} := \begin{bmatrix} p \text{ in } D_{(QR)(3:0)} \Rightarrow c_{oy}(p) := 0; \\ p \text{ in } D_{(QR)(3:1)} \Rightarrow c_{oy}(p) := 1. \end{bmatrix}$$

and

$$\text{CONTROL}_{(QR)(3:2)} := \begin{bmatrix} p \text{ in } \text{BASE}_{(QR)} \Rightarrow c_{oy}(p) := c_{oy}(p + \begin{bmatrix}1\\0\\0\end{bmatrix}). \\ \end{bmatrix}$$

These definitions are the same as the definitions for the corresponding computations produced by the pipelining of c_{ox}, with c_{ox} replaced by c_{oy}.

The composition of these two computations simulates $\text{CONTROL}_{(QR)(3)}$:

$\text{CONTROL}_{(QR)(3:1)} \| \text{CONTROL}_{(QR)(3:2)}$ simulates $\text{CONTROL}_{(QR)(3)}$ (✍ Theorem 11)

.

(Note that c_{ox} and c_{oy} will have everywhere the same values (as c_x and c_y had in the previous example - see page 100).)

5.3.4 Amalgamation of just-generated computations

We can now splice the pipelines, composing the three edge-computations to form $\mathrm{CONTROL''}_{(\mathbf{QR})}$, and composing the three uniform recurrences to form $\mathrm{CONTROL'''}_{(\mathbf{QR})}$:

$$\mathrm{CONTROL''}_{(\mathbf{QR})} := \|_{i=1 \text{ to } 3}\, \mathrm{CONTROL}_{(\mathbf{QR})(i:1)}$$

$$\mathrm{CONTROL'''}_{(\mathbf{QR})} := \|_{i=1 \text{ to } 3}\, \mathrm{CONTROL}_{(\mathbf{QR})(i:2)}$$

$\mathrm{CONTROL'''}_{(\mathbf{QR})}$ is a uniform recurrence and all the variables of $\mathrm{CONTROL''}_{(\mathbf{QR})}$ are on the boundary of the array [see Theorem 19 and Theorem 25] which is what we need (see page 74). So we have

$$\mathrm{EDGE}_{(\mathbf{QR})} := \mathrm{CONTROL''}_{(\mathbf{QR})}$$

and

$$\mathrm{INTERIOR}_{(\mathbf{QR})} := \mathrm{CONTROL'''}_{(\mathbf{QR})} \,\|\, \mathrm{DATA'}_{(\mathbf{QR})}$$

as required. (The composition of $\mathrm{EDGE}_{(\mathbf{QR})}$ and $\mathrm{INTERIOR}_{(\mathbf{QR})}$ simulates the initial computation.)

So we have found one way to perform control-pipelining for the QR-factorization example. Are there any others?...

5.3.5 Other Options

We can also pipeline cont using the dependency vector $\begin{bmatrix}0\\0\\-1\end{bmatrix}$ instead of $\begin{bmatrix}0\\-1\\0\end{bmatrix}$. (We must make obvious changes to $D_{(\mathbf{QR})(1)}$, $D_{(\mathbf{QR})(1:0)}$ and $D_{(\mathbf{QR})(1:1)}$.) Similarly, we can pipeline

c_{ox} (or c_{oy}) using $\begin{bmatrix}0\\-1\\-1\end{bmatrix}$ instead of $\begin{bmatrix}1\\0\\0\end{bmatrix}$, changing $D_{(QR)(2)}$, $D_{(QR)(2:0)}$ and $D_{(QR)(2:1)}$ (or $D_{(QR)(3)}$, $D_{(QR)(3:0)}$ and $D_{(QR)(3:1)}$).

Now we have looked at alternative design choices for the control-pipelining stage, let us proceed to the allocation stage.

5.4 Allocation

Recall that the allocation function maps the original domain of computation into space (as opposed to the scheduling function, which maps it into time). Following from the definitions in section 4.4 on page 105, $\mathbf{Im}_{s(QR)}$ will be the allocation function, and $\mathbf{Im}_{s(QR)}(p)$ will be equal to $(\mathbf{A}_{s(QR)}.p + \mathbf{b}_{s(QR)})$. $\mathbf{Im}_{(QR)}$ will be the complete space-time map and $\mathbf{Im}_{(QR)}(p)$ will be equal to $(\mathbf{A}_{(QR)}.p + \mathbf{b}_{(QR)})$.

Let us set $\mathbf{A}_{s(QR)}$ to be the simple matrix $\begin{bmatrix}1&0&0\\0&1&0\end{bmatrix}$. Then the determinant of the matrix $\mathbf{A}_{(QR)}$ will be non-zero as required (see section 4.4 on page 105 again). We will let $\mathbf{b}_{s(QR)}$ be zero for simplicity. So $\mathbf{Im}_{(QR)}$ equals the function $p \rightarrow \begin{bmatrix}-1&1&2\\1&0&0\\0&1&0\end{bmatrix}.p$, $\mathbf{Im}_{(QR)}$ is invertible, and RENAME maps the variable <v, p> to <v, $\mathbf{Im}_{(QR)}(p)$>. ("v" stands for an arbitrary variable class and p an arbitrary point in the domain for which <v, p> is a variable of $\mathrm{EDGE}_{(QR)} \parallel \mathrm{INTERIOR}_{(QR)}$.)

Let us see if there are any other choices for the allocation function...

5.4.1 Other Options

Let the desired alternative allocation function be $\mathbf{Im}_{s(QR)}'$, and let $\mathbf{Im}_{s(QR)}'(p)$ be equal to $(\mathbf{A}_{s(QR)}'.p + \mathbf{b}_{s(QR)}')$. ($\mathbf{Im}_{(QR)}'$ will be the alternative space-time map and $\mathbf{Im}_{s(QR)}'(p)$

will be equal to ($A_{(QR)}'.p + b_{(QR)}'$).) If we let $A_{s(QR)}'$ be $\begin{bmatrix}1&0&0\\0&0&1\end{bmatrix}$ then the determinant of $A_{(QR)}'$ will be non-zero and so the $\mathbf{Im}_{(QR)}'$ (equal to $p \rightarrow \begin{bmatrix}-1&1&2\\1&0&0\\0&0&1\end{bmatrix}.p$) will give rise to a space-time simulation. The resulting architecture can be seen in Figure 5.9 on page 153.

5.5 The Final Stage

Let us return to the design choice corresponding to the first allocation function, $\mathbf{Im}_{(QR)}$; with this choice, the final array will consist of twenty processors arranged in a rectangular grid (when M = 5). The required interconnections will be made in section 5.6 on page 149. We can now, without having to make any more design choices, construct the final space-time simulation. We rename CONTROL''$_{(QR)}$ to create the edge-computation EDGE'$_{(QR)}$ and we rename CONTROL'''$_{(QR)}$ and DATA'$_{(QR)}$ and compose them to form the uniform recurrence INTERIOR'$_{(QR)}$; finally we compose EDGE'$_{(QR)}$ and INTERIOR'$_{(QR)}$ to form the final space-time simulation, IMP$_{(QR)}$.

EDGE'$_{(QR)}$:= CONTROL''$_{(QR)}$ ® RENAME$_{(QR)}$

INTERIOR'$_{(QR)}$:=

(CONTROL'''$_{(QR)}$ ® RENAME$_{(QR)}$) || (DATA'$_{(QR)}$ ® RENAME$_{(QR)}$)

IMP$_{(QR)}$:= EDGE'$_{(QR)}$ || INTERIOR'$_{(QR)}$

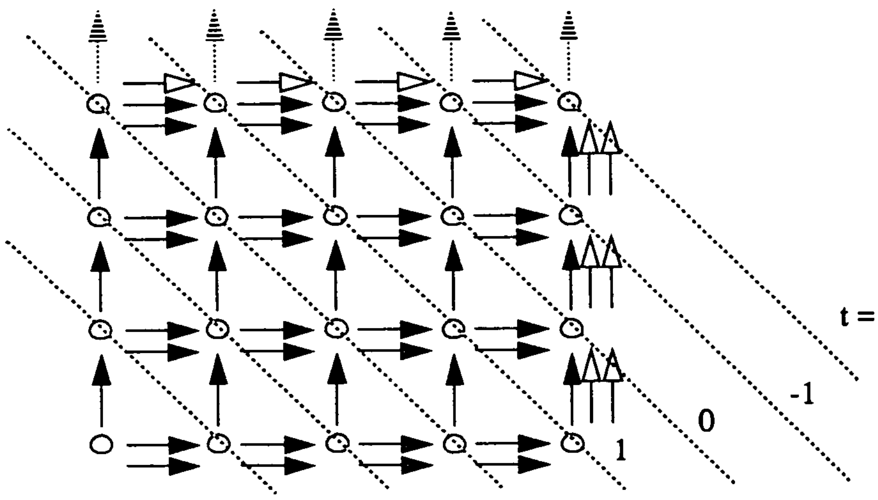
IMP$_{(QR)}$ is equal to:

$$\ulcorner \quad p \text{ in } \mathrm{BASE}_{(QR)} \Rightarrow c_{ox}(p) := c_{ox}\left(p + \begin{bmatrix}-1\\1\\0\end{bmatrix}\right), \qquad \urcorner$$

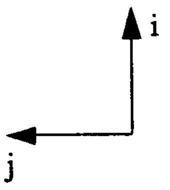
$$
\begin{array}{|llll|}
 & c_{oy}(p) & := & c_{oy}(p + \begin{bmatrix}-1\\1\\0\end{bmatrix}), \\
 & cont(p) & := & cont(p + \begin{bmatrix}-1\\0\\-1\end{bmatrix}), \\
 & ox(p) & := & ny(p + \begin{bmatrix}-1\\-1\\0\end{bmatrix}), \\
 & oy(p) & := & cont(p)*nx(p + \begin{bmatrix}-1\\1\\0\end{bmatrix}) + \overline{cont(p)}*ny(p + \begin{bmatrix}-2\\0\\0\end{bmatrix}), \\
 & z_{ox}(p) & := & c_{ox}(p)*z_{ox}(p + \begin{bmatrix}-1\\0\\-1\end{bmatrix}) + \bar{c}_{ox}(p)*ox(p), \\
 & z_{oy}(p) & := & c_{oy}(p)*z_{oy}(p + \begin{bmatrix}-1\\0\\-1\end{bmatrix}) + \bar{c}_{oy}(p)*oy(p), \\
 & sin(p) & := & z_{oy}(p)/(z_{oy}(p)^2 + z_{ox}(p)^2)^{1/2}, \\
 & cos(p) & := & z_{ox}(p)/(z_{oy}(p)^2 + z_{ox}(p)^2)^{1/2}, \\
 & nx(p) & := & ox(p)*cos(p) + oy(p)*sin(p), \\
 & ny(p) & := & ox(p)*sin(p) - oy(p)*cos(p), \\
p \text{ in } D_{(QR)(2:1)} & \Rightarrow & c_{ox}(p) & := 1; \\
p \text{ in } D_{(QR)(2:0)} & \Rightarrow & c_{ox}(p) & := 0; \\
p \text{ in } D_{(QR)(3:1)} & \Rightarrow & c_{oy}(p) & := 1; \\
p \text{ in } D_{(QR)(3:0)} & \Rightarrow & c_{oy}(p) & := 0; \\
p \text{ in } D_{(QR)(1:1)} & \Rightarrow & cont(p) & := 1; \\
p \text{ in } D_{(QR)(1:0)} & \Rightarrow & cont(p) & := 0. \\
\hline
\end{array}
$$

The first three lines show the channelling of the three control signals through the variable-classes c_{ox}, c_{oy} and cont. Note that the dependency vectors have now been transformed by $\mathbf{Im}_{(\mathbf{QR})}$ from $\begin{bmatrix}1\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\-1\\0\end{bmatrix}$, $\begin{bmatrix}-1\\0\\-1\end{bmatrix}$, $\begin{bmatrix}0\\0\\-1\end{bmatrix}$ to $\begin{bmatrix}-1\\1\\0\end{bmatrix}$, $\begin{bmatrix}-1\\0\\-1\end{bmatrix}$, $\begin{bmatrix}-1\\-1\\0\end{bmatrix}$ and $\begin{bmatrix}-2\\0\\0\end{bmatrix}$ respectively. The fourth and fifth lines show the required values being loaded into the variables <ox, p> and <oy, p> ; in line four, the value of <ny, p + $\begin{bmatrix}-1\\-1\\0\end{bmatrix}$> is loaded into <ox, p> and in line five the value of <nx, p + $\begin{bmatrix}-1\\1\\0\end{bmatrix}$> is loaded into <oy, p> if the value of <cont, p> is 1 and the value of <ny, p + $\begin{bmatrix}-2\\0\\0\end{bmatrix}$> is loaded in if the value of <cont, p> is 0. Lines six and seven assign values to the variables <z_{ox}, p> and <z_{oy}, p>. Recall that the variable-classes z_{ox} and z_{oy} were created in the data-pipelining stage to ferry the values of ox and oy respectively from the beginning of the row. If p is at the beginning of a row then c_{ox} and c_{oy} will be 0 and <z_{ox}, p> and <z_{oy}, p> will be assigned the values of <ox, p> and <oy, p> respectively; otherwise c_{ox} and c_{oy} will be 1 and <z_{ox}, p> and <z_{oy}, p> will each be assigned the value of the corresponding variable at the previous point. In the eighth and ninth lines the values <cos, p> and <sin, p> are calculated using the values of <z_{ox}, p> and <z_{oy}, p>. In the tenth and eleventh lines, the Givens rotation is executed and values are assigned to <nx, p> and <ny, p>. The final six lines, grouped in pairs, correspond to the three edge-computations which deal with the three control-variable-classes c_{ox} , c_{oy} and cont. (That is, these lines describe the "obstruction pattern" at the edge of the region, in the light analogy.) In each pair of lines, the first line defines the region where the control variable has the value 1, and the second line defines the region where the control signal is 0 (cf. the convolution example, page 108).

Figure 5.4 and show the complete implementation, $\mathrm{IMP}_{(\mathbf{QR})}$, with schedule lines drawn in. As in Chapter 4, the hollow arrows represent control dependencies. Only those corresponding to a zero signal are drawn.



Co-ordinate frame:



Data-dependencies:

$<nx, p \rightarrow p + \begin{bmatrix}1\\0\\0\end{bmatrix}>$

$<ox, p \rightarrow p + \begin{bmatrix}0\\-1\\0\end{bmatrix}>$

and $<oy, p \rightarrow p + \begin{bmatrix}0\\-1\\0\end{bmatrix}>$

Control-dependencies:

$<c_{ox}, p + \begin{bmatrix}1\\0\\0\end{bmatrix}>$

and $<c_{oy}, p + \begin{bmatrix}1\\0\\0\end{bmatrix}>$

$<cont, p \rightarrow p + \begin{bmatrix}0\\-1\\0\end{bmatrix}>$

Figure 5.4 IMP$_{(QR)}$ (horizontal cross-section: k = 1)

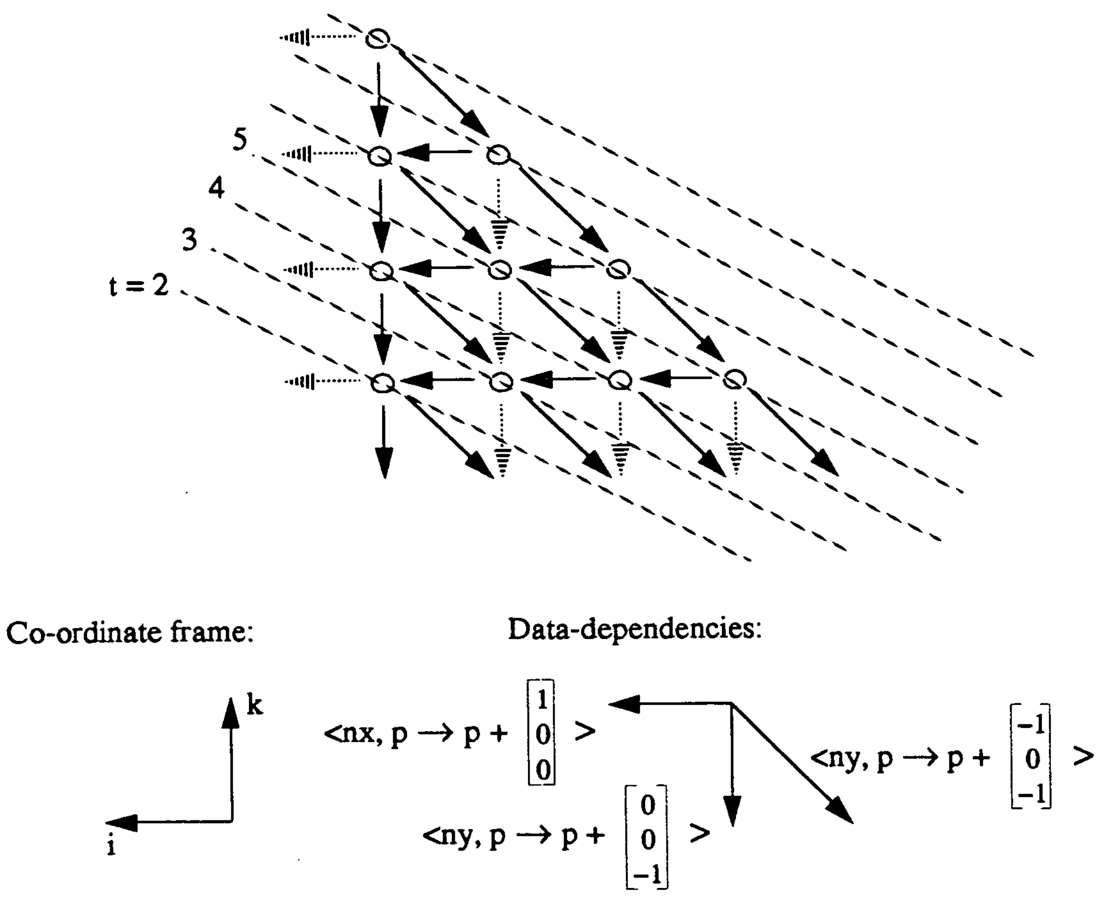


Figure 5.5 IMP$_{(QR)}$ (vertical cross-section: j = 5)

5.6 The Architecture

Figure 5.6 and Figure 5.7 show the final architecture for the design.

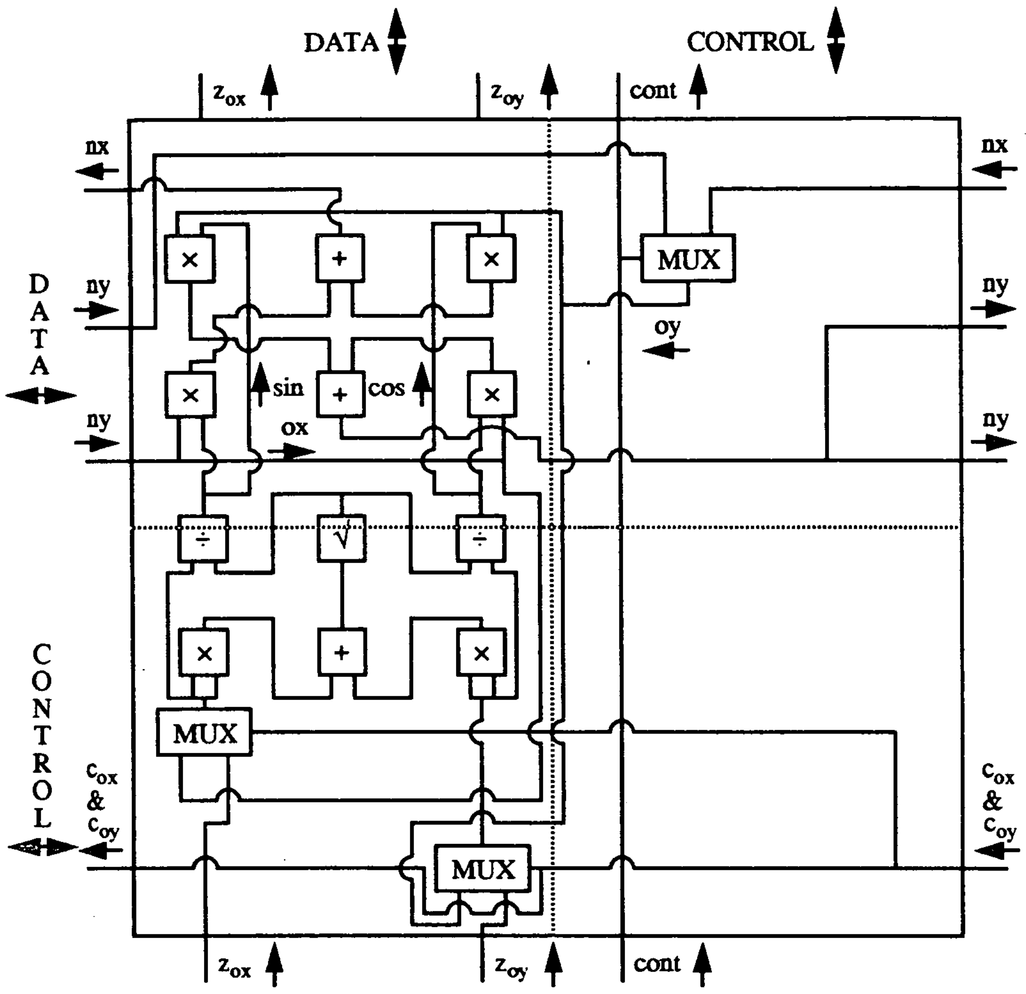


Figure 5.6 The architecture of each processor

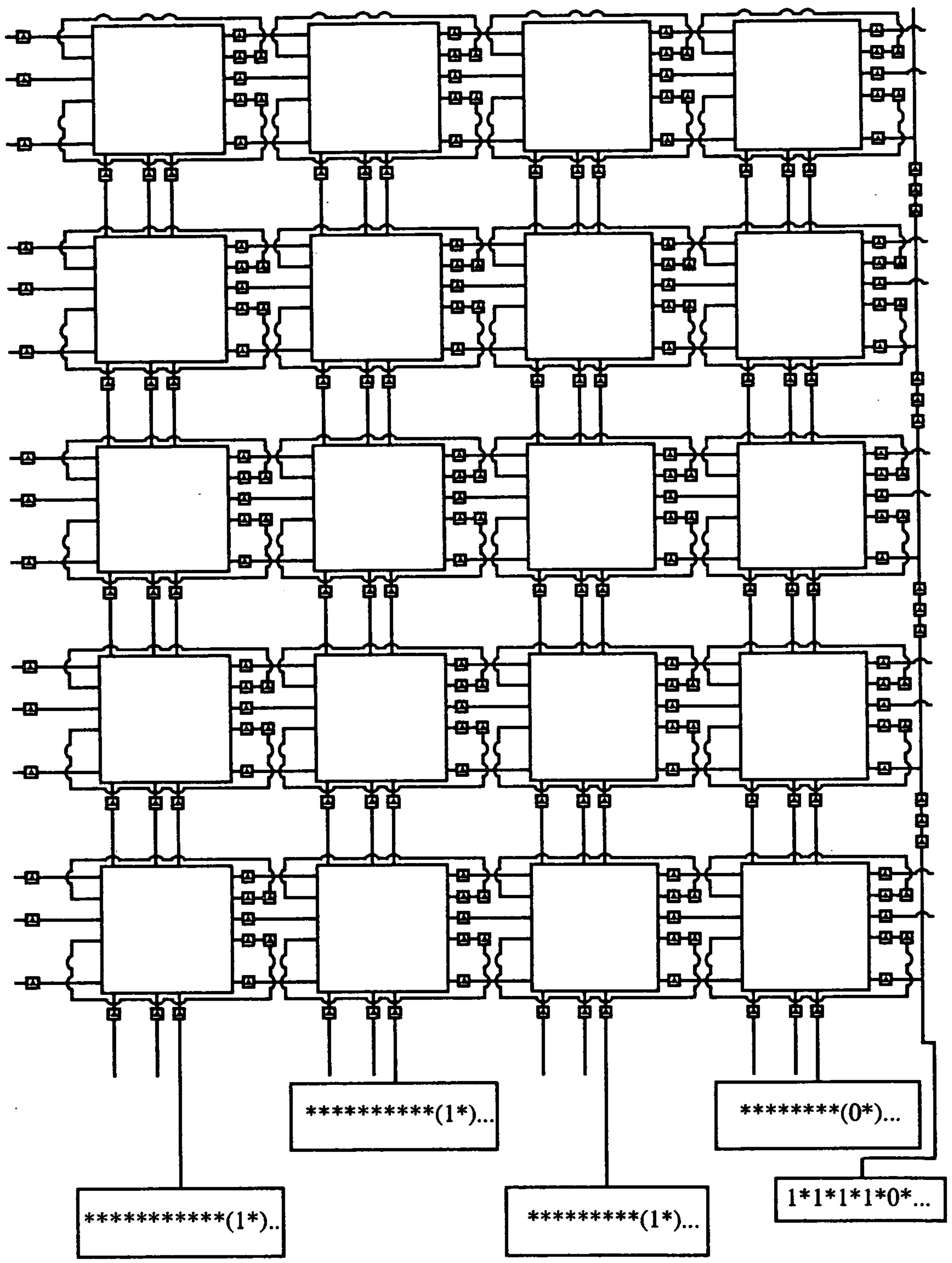


Figure 5.7 The architecture of the complete array

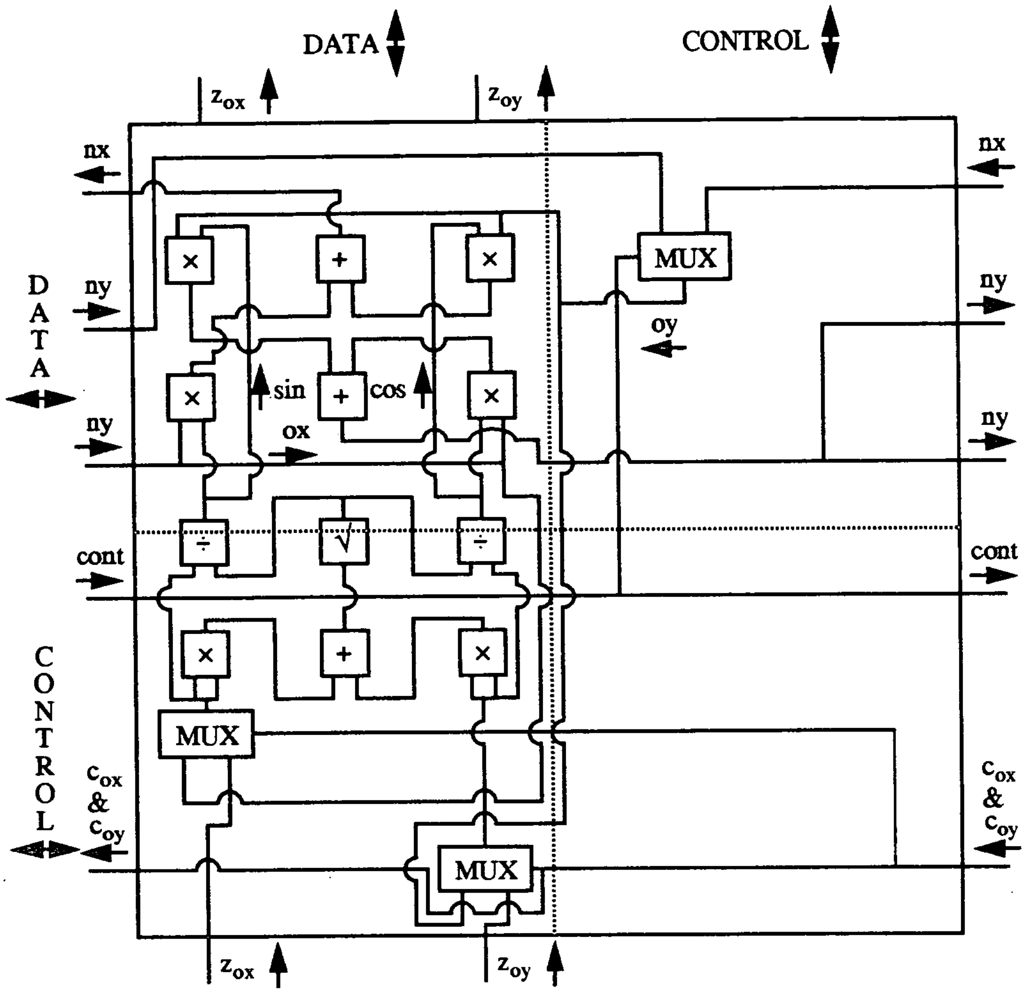


Figure 5.8 Processor for alternative design

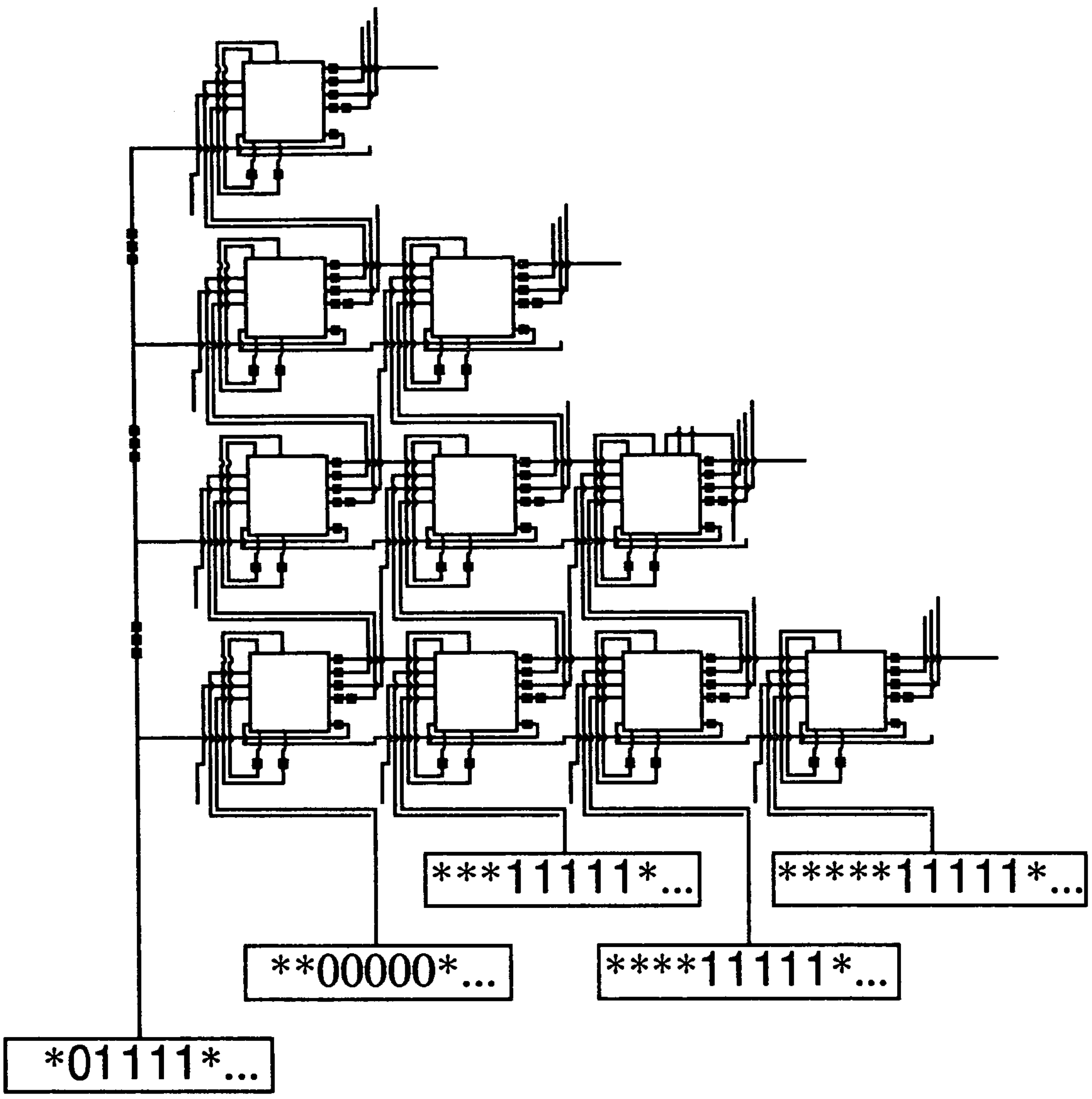


Figure 5.9 Alternative architecture

5.7 Summary of chapter and further work

5.7.1 Summary

In this chapter the method was used to achieve a systolic implementation of Given's algorithm for QR-factorisation. An alternative implementation was achieved by choosing a different allocation function. Alternative ways of data-pipelining, scheduling and control pipelining were briefly investigated.

5.7.2 Further work

It might be instructive to compare the implementations of QR-factorisation in it with those of others (e.g. Gentleman and Kung's) and to see if the other implementations can be achieved by the method. There is inefficiency in my implementations of QR-factorisation: the calculation of the coefficients for the Givens rotations ("Givens Generation") is done by every computation in INTERIOR'$_{(\mathbf{QR})}$. This is unnecessary. It would be good if the design process could be simply modified so that this redundancy didn't occur.

6 Conclusions

This chapter summarises the contribution made by this thesis and suggests some avenues which could be explored in future.

6.1 Contribution

The contribution of this thesis is as follows

6.1.1 Formalisation of concepts

- The concept of a computation is defined. It is possible to express many, if not all, algorithms as computations.
- Explicit labelling of variables in computations facilitates their composition in complex ways and enables physical as well as abstract algorithmic structure to be captured.
- The concept of simulation is formally defined.
- Two key concepts in the literature on systolic array design - recurrence equations and dependency - are clarified by formal definition. Other important concepts are also clarified: uniformity, affinity and conditionality.

6.1.2 The method

- A design method is formulated which is simple and yet sufficiently powerful for the high-level design of a systolic array for QR-factorisation.
- The ordering of the design steps is chosen to minimise the chance of an impasse in the design path.

- The method has been mathematically proven, subject to the assumption that the computations in the method are well-defined. In Appendix H, the well-definedness assumptions required to validate data-pipelining have been proven to hold.

6.2 Further work

Here are some suggestions for further work; ideas from previous chapters are summarised.

6.2.1 Priority work

It would be good to have a proof, with minimal assumptions, that the computations in the method *are* well-defined. (In Appendix H this is done for the well-definedness assumptions of Appendix D.) It might be useful to implement the method on a proof assistant for hardware design, like LAMBDA. It would also be interesting to see whether more efficient implementations of QR-factorisation and convolution could be achieved using the method.

6.2.2 Analysis, extension and automation of the method

One could investigate the feasibility of the method, e.g. when is it possible to make ARs uniform or to schedule URs?

One could also extend the method. AR to AR transformations to make the input computation more amenable could be sought, and the class of input computations could perhaps be extended beyond the class of computations which are the composition of an AR with an initial control requirement; the class of output implementations could perhaps be extended to include e.g. wavefront arrays or hypercubes. It could be investigated whether pipelining could be made more sophisticated, using the ideas in [Raj89], and scheduling and allocation could be modified to allow the space-time mapping to be conditional on the output of the computations. One could adapt the method to take fault-tolerance and optimisation into account. It would be useful to extend the method down to architectural level.

The possibility could be investigated of automating the currently unautomated parts of the method, in particular the scheduling and allocation tasks.

6.2.3 Theoretical foundation

It would be useful to perform a critical survey of formal design languages, with a careful look at the relative merits of relational and functional styles, to come up with a more satisfactory theory of input and output and to consider how the concept of a systolic array should be defined.

6.2.4 Wider issues

Is there a connection between the design of systolic arrays and boundary value problems? Are there analogue methods for problems which now use systolic arrays? Is there a connection between systolic arrays and neural networks? They are all regular, parallel architectures which have simple processing elements and local connections.

In my method the candidate "algorithms" for direct implementation by systolic arrays are the URs. URs already have a geometry since they are embedded in Euclidean space. It might be possible to abstract away from the class of URs their topological structure as networks. Culik does something similar to this [Culik84, Culik85]. (It might be possible to embed these networks in Riemannian or Spherical rather than Euclidean space.) If this abstraction could be done it would call into question the usefulness of abstract ARs and URs, affine scheduling and allocation, and in fact the whole geometrical design paradigm.

6.3 In Conclusion

This thesis provides an underlying theory for formal design methods for systolic arrays, which use "recurrence equations". The use of the theory is illustrated by a describing such a method. The method is simple and it is hoped that, now the theory is in place, the method could be considerably extended to make it more useful for the design of practical systolic arrays.

7 References

[Ash77] E.A. Ashcroft & W.W. Wadge,
LUCID: a non-procedural language with iteration, *Communications of the ACM*, 20(7), 1977, pp. 519-529

[Birt88] G. Birtwhistle & P.A. Subrahmanyam,
Preface to: *VLSI Specification, Verification and Synthesis*, G. Birtwhistle & P.A. Subrahmanyam (eds.), Kluwer Academic Publishers, 1988

[Caspi87] P. Caspi, D. Pilaud, N. Halbwachs & J.R. Plaice,
LUSTRE: A declarative language for programming synchronous systems, *14th Annual ACM Symposium of Principles of Programming Languages, 1987*, ACM, Munich (W. Ger.) pp.178-188

[Chen91] C.Y.R. Chen & M.Z. Moricz,
A Delay Distribution Methodology for the Optimal Systolic Synthesis of Linear Recurrence Algorithms, *IEEE Transactions on Computer-Aided Design*, 10(6), June 1991

[Cohn88] A. Cohn,
A Proof of Correctness of the Viper Microprocessor: The first Level, *VLSI Specification, Verification and Synthesis*, G. Birtwhistle & P.A. Subrahmanyam (eds.), Kluwer Academic Publishers, 1988, pp. 27-71

[Culik84] K. Culik & Fris,
Topological Transformations as a Tool in the Design of Systolic Networks, Department of Computer Science, University of Waterloo, Waterloo, Ont., U.S.A., 1984

[Culik85] K. Culik & Yu,
Translation of Systolic Algorithms between Systems of Different Topology, Department of Computer Science, University of Waterloo, Waterloo, Ont., U.S.A., 1985

[Dav88] B.S. Davie,
A Formal, Hierarchical Design and Validation Methodology for VLSI, Ph.D. Thesis, University of Edinburgh, 1988

[Del86] J.-M. Delosme & I.C.F. Ipsen,
Systolic Array Synthesis: Computability & Time Cones, *Parallel Algorithms & Architectures*, M. Cosnard et al. (eds.), Elsevier Science Publishers B.V. (North Holland), 1986, pp. 295-312

[Far87] N. Faroughi & M. A. Shanblatt,
An Improved Systematic Method for Constructing Systolic Arrays from Algorithms, *24th ACM/IEEE Design Automation Conference, 1987*, Paper 3.1

[Fish85] A.L. Fisher & H.T. Kung,
Synchronizing Large VLSI Processor Arrays, *IEEE Transactions on Computers*, 34(8), 1985, pp. 734-740

[Fost80] M.J. Foster & H.T. Kung,
Design of Special-Purpose VLSI Chips, *IEEE Computer*, Jan. 1980, pp. 26-40

[Gen81] W.M. Gentleman & H.T. Kung,
Matrix Triangularisation by Systolic Arrays, *Proceedings of the SPIE, Real Time Signal Processing IV*, 298, 1981, pp. 19-26

[GMil83] G.J. Milne,
CIRCAL: A Calculus for Circuit Description, *Integration, the VLSI Journal*, 1(2&3), Oct. 1983, pp.121-160

[Golub83] G.H. Golub & C.F. Van Loan,
Matrix Computations, North Oxford Academic Publishing Company Ltd., Oxford, England, 1983

[Gor88] M. Gordon,
HOL: A Proof Generating System for Higher-Order Logic, *VLSI Specification, Verification and Synthesis*, G. Birtwhistle & P.A. Subrahmanyam (eds.), Kluwer Academic Publishers, 1988

[Hoare85] C.A.R. Hoare,
Communicating Sequential Processes, Prentice-Hall, 1985

[HTKun78] H.T. Kung & C.E. Leiserson,
Systolic Arrays (for VLSI), *Sparse Matrix Proceedings, 1978*, SIAM, 1979, pp. 256-282,

[HTKun82] H.T. Kung,
Why Systolic Architectures?, *IEEE, Computer*, 15(1), Jan. 1982,

pp. 37-46

[Huang87] C.-H. Huang & C. Lengauer,
The Derivation of Systolic Implementations of Programs, *Acta Informatica*, 24, 1987, pp.595-632

[Hwang84] K. Hwang and F.A. Briggs,
Computer Architecture and Parallel Processing, McGraw Hill, 1984

[Ib90] O.H. Ibarra, T. Jiang, J.H. Chang & M.A. Palis,
Systolic Algorithms for some Scheduling and Graph Problems, *Journal of VLSI Signal Processing*, 1(4), 1990, pp. 307-320

[John83] S. Johnson,
Synthesis of Digital Designs from Recursion Equations, MIT Press, 1983

[Jones87] G. Jones,
Programming in occam, Prentice Hall 1987

[Jones88] G. Jones, E.M. Goldsmith,
Programming in occam2, Prentice Hall, 1988

[Karp67] R.M. Karp, R.E. Miller & S. Winograd,
The Organization of Computations for Uniform Recurrence Equations, *JACM*, 14(3), 1967, pp.563-590

[Kunde86] M. Kunde, H.W. Lang, M. Schimmler, J. Schmeck, H. Schöder,
The Instruction Systolic Array and its Relation to Other Models of Parallel Computers, *Parallel Computing'85*, M. Feilmeier, G. Joubert, U. Schendel (eds.), 1986, pp. 491-497

[Lee90] P.-Z. Lee & Z.M. Kedem,
Mapping Nested Loop Algorithms into Multidimensional Systolic Arrays, *IEEE Transactions on Parallel and Distributed Systems*, 1, Jan. 1990, pp. 64-76

[Len90] C. Lengauer & J. Xue,
A Systolic Array for Pyramidal Algorithms, Laboratory for Foundations of Computer Science, University of Edinburgh, ECS-LFCS-90-114, 1990

[Len91] C. Lengauer, M. Barnett & D. G. Hudson,
Towards Systolizing Compilation, *Distributed Computing*, 5(1),

1991, pp. 7-24

[LeV85] H. Le Verge, C. Mauras & P. Quinton,
The ALPHA Language and its Use for the Design of Systolic Arrays, *Journal of VLSI Signal Processing*, 3(3), 1991, pp.173-182,

[Li85] G.J. Li & B.W. Wah,
The Design of Optimal Systolic Arrays, *IEEE Transactions on Computers*, C-34(1), Jan. 1985

[Lin90] N. Ling & M.A. Bayoumi,
Systolic Temporal Arithmetic: A New Formalism for Specification & Verification of Systolic Arrays, *IEE Transactions on CAD*, 9(8), Aug. 1990

[Luk88a] W. Luk & G. Jones,
From Specification to Parametrized Architectures, *Proceedings of the International Working Conference on "The Fusion of Hardware Design and Verification" (IFIP WG 10.2)*, Glasgow, July 1988, (participants edition published by the University of Strathclyde), pp. 263-284

[Luk88b] W. Luk & G. Jones,
The derivation of regular synchronous circuits, *Proceedings of the International Conference on Systolic Arrays*, K. Bromley, S.Y. Kung, & E. Swartzlander eds., IEEE Computer Society Press, 1988, pp. 305-314

[Mar87] A.R. Martin & J.V. Tucker,
The Concurrent Assignment Representation of Synchronous Systems, Report 8.87, Centre for Theoretical Computer Science, The University of Leeds, January, 1987

[Mayg91] E.M. Mayger & M.P. Fourman,
Integration of Formal Methods with System Design, *VLSI 91 (Proceedings of the IFIP TC 10/WG 10.5 on VLSI)*, Edinburgh, 1991

[McA92] A.J. McAuley,
Four State Asynchronous Architectures, *IEEE Transactions on Computers*, 41(2), Feb. 1992, pp.129-142

[McC87] J.V. McCanny & J.G. McWhirter,
Some Systolic Array Developments in the United Kingdom,

Computer, 20(7), July 1987, pp.51-63

[McW83] J.G. McWhirter,
Recursive least-squares minimization using a systolic array, *Proceedings of the SPIE, Real Time Signal Processing VI*, 1983, pp. 105-110

[McW92] J.G. McWhirter,
Algorithmic Engineering in Adaptive Signal Processing, Proceedings of the IEE, Vol. 139, Part F, June 1992, pp. 226-232

[Mein86] C. Meinel,
The Parallelization Index of Synchronous Systems, *Parallel Processing by Cellular Automata and Arrays*, Sept. 1986, pp. 226-233

[O'K86] M.T. O'Keefe & J.A.B. Fortes,
A Comparative Study of Two Systematic Design Methodologies, *Parallel Algorithms & Architectures*, M. Cosnard et al. (eds.), Elsevier Science Publishers B.V. (North Holland), 1986

[Quin86] P. Quinton, B. Joinnault & P. Gachet,
A New Matrix Multiplication Systolic Array, *Parallel Algorithms & Architectures*, M. Cosnard et al. (eds.), Elsevier Science Publishers B.V. (North Holland), 1986

[Quin89] P. Quinton & V. Van Dongen,
The Mapping of Linear Recurrence Equations on Regular Arrays, *Journal of VLSI Signal Processing*, 1(2), Kluwer Academic Publishers,1989, pp. 95-113

[Rao85] S.K. Rao,
Regular Iterative Algorithms and their Implementations on Processor Arrays, Ph.D. Thesis, Stanford University, October 1985

[Raj86] S.V. Rajopadhye & P. Panangaden,
Verification of Systolic Arrays: A stream functional approach, *IEEE International Conference on Parallel Processing, 1986*, pp.773-775

[Raj89] S.V. Rajopadhye,
Synthesizing Systolic Arrays with Control Signals from Recurrence Equations, *Distributed Computing*, 3, Springer-

Verlag 1989, pp. 88-105

[Raj90] S.V. Rajopadhye,
Algebraic Transformations in Systolic Array Synthesis: A Case Study, *Formal VLSI SPecification and Synthesis: VLSI Design Methods-1*, L.J.M. Claesen (ed.), North Holland, 1990

[RMil80] R. Milner,
A Calculus of Communicating Systems, *Lecture Notes in Computer Science*, 92, Springer Verlag, 1980

[RMil83] R. Milner,
Calculi for Synchrony and Asynchrony, *Theoretical Computer Science*, 25(3), 1983, pp.267-310

[RMil89] R. Milner,
Communication and Concurrency, Prentice Hall, 1989

[Rob84] J.B.G. Roberts, P. Simpson, B.C. Merrifield and J.F. Cross,
Signal Processing Applications of Distributed Array Processors, *IEE Proceedings*, Vol. 131, Part F, No. 6, Oct. 1984, pp. 603-609

[Roy89] V.P. Roychowdhury & T. Kailath,
Subspace Scheduling and Parallel Implementation of Non-Systolic Regular Iterative Algorithms, *Journal of VLSI Signal Processing*, 1(2), Kluwer Academic Publishers, 1989, pp.127-142

[Shang89] W. Shang & J.A.B. Fortes,
On the Optimality of Linear Schedules, *Journal of VLSI Signal Processing*, 1(3), Kluwer Academic Publishers, 1989, pp. 209-220

[She84] M. Sheeran,
μFP, a language for VLSI Design, *Proceedings of the ACM Symposium on LISP and Functional Programming, 1984*, pp.104-112

[She86] M. Sheeran,
Describing and Reasoning about Circuits using Relations, *Proceedings of The Leeds Workshop on Theoretical Aspects of VLSI Design 1986*, (was to be published in the CUP)

[She88a] M. Sheeran,

Describing Hardware Algorithms in Ruby, *Proceedings of the Workshop on Concepts and Characteristics of Declarative Systems (IFIP WG10.1)*, Budapest, 1988

[She88b] M. Sheeran,
Retiming and Slowdown in Ruby, *Proceedings of the International Working Conference on "The Fusion of Hardware Design and Verification" (IFIP WG 10.2)*, Glasgow, July 1988, (participants edition published by the University of Strathclyde), pp. 285-304

[SYKun88] S.Y. Kung, 1988,
VLSI Array Processors, Prentice Hall, 1988

[Teich91] J. Teich & L. Thiele,
Control Generation in the Design of Processor Arrays, *Journal of VLSI Signal Processing*, 3(1/2), 1991, pp.77-92

[Tensi88] T.Tensi,
Worst Case Analysis for Reducing Algorithms on Instruction Systolic Arrays with Simple Instruction Sets, *Parallel Processing by Cellular Automata and Arrays, 1988*, pp. 347-352

[Ull84] J.D. Ullman,
Computational Aspects of VLSI, Computer Science Press, 1984

[VanSw91] M. Van Swaaij, J. Rosseel, F. Catthoor, H. De Man,
Synthesis of ASIC Regular Arrays for Real-Time Image Processing Systems, *Journal of VLSI Signal Processing*, 3 (3), 1991, pp. 183-192

[Wadge85] W.W. Wadge & E.A. Ashcroft,
LUCID, the Data-flow Programming Language, London Academic Press, 1985

[Wat82] D.S. Watkins
Understanding the QR Algorithm, *SIAM Review*, 24(4), Oct. 1982

[Wex89] J. Wexler,
Concurrent Programming in occam2, Ellis Horwood, 1989

[Xue90] J. Xue & C. Lengauer,
On the Description & Development of One-Dimensional Systolic Arrays, Laboratory for Foundations of Computer

Science, University of Edinburgh, ECS-LFCS-90-116, 1990

[Yaa88] Y. Yaacoby & P.R. Cappello,
Converting Affine Recurrence Equations to Quasi-Uniform Recurrence Equations, *AWOC 1988: 3rd International Workshop on Parallel Computation and VLSI Theory*, Springer, Berlin Heidelberg New York Tokyo

[Yaa89] Y. Yaacoby & P.R. Cappello,
Scheduling a System of Nonsingular Affine Recurrence Equations onto a Processor Array, *Journal of VLSI Signal Processing*, 1(2), 1991, pp. 183-192

Appendix A : Overview of Appendices

These appendices contain the proof that, subject to assumptions about the well-definedness of the computations handled and created, the design method will produce only designs which meet their specifications. Appendix B and Appendix C contain basic results which are used by the other appendices. Appendix D, Appendix E, Appendix F contain propositions relating to the data-pipelining, control-pipelining and schedule-and-allocation transformations respectively, the principle results being that, if certain conditions hold, the output of each transformation simulates the input to the transformation. Appendix G contains three theorems which state that the output to the method satisfies the specification, if certain conditions hold; the theorems are proved using the main results of Appendix D, Appendix E, and Appendix F. Appendix H contains the proofs of the assumptions made in Appendix D that certain computations are well-defined.

The following three pages show how the proofs of the theorems and lemmas in each appendix use other theorems and lemmas. The key lemmas and theorems of each appendix are written white-on-black. A small black blob on an intersection of lines indicates the forking of an arrow.

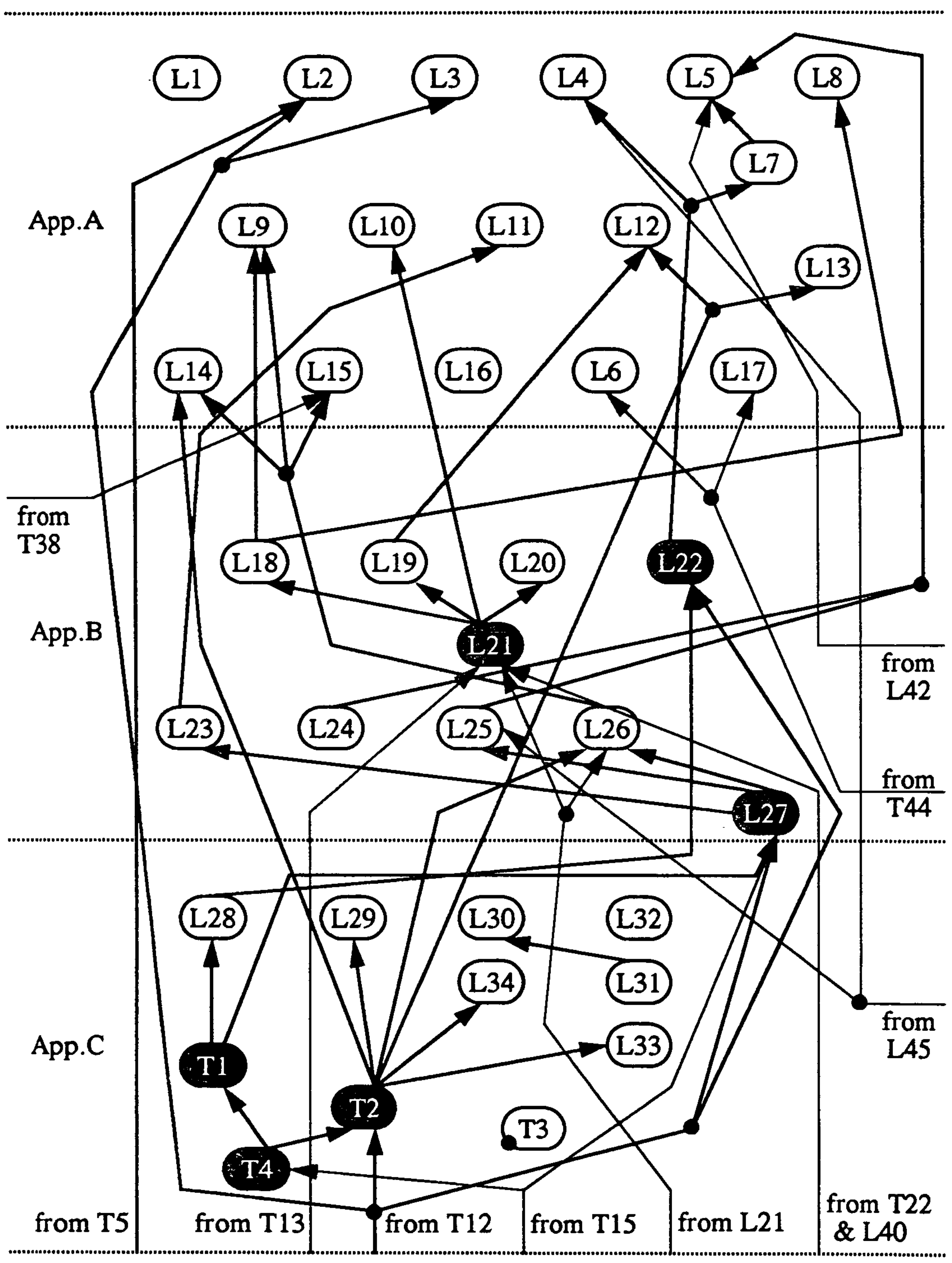


Figure 6.1 Appendices A to C

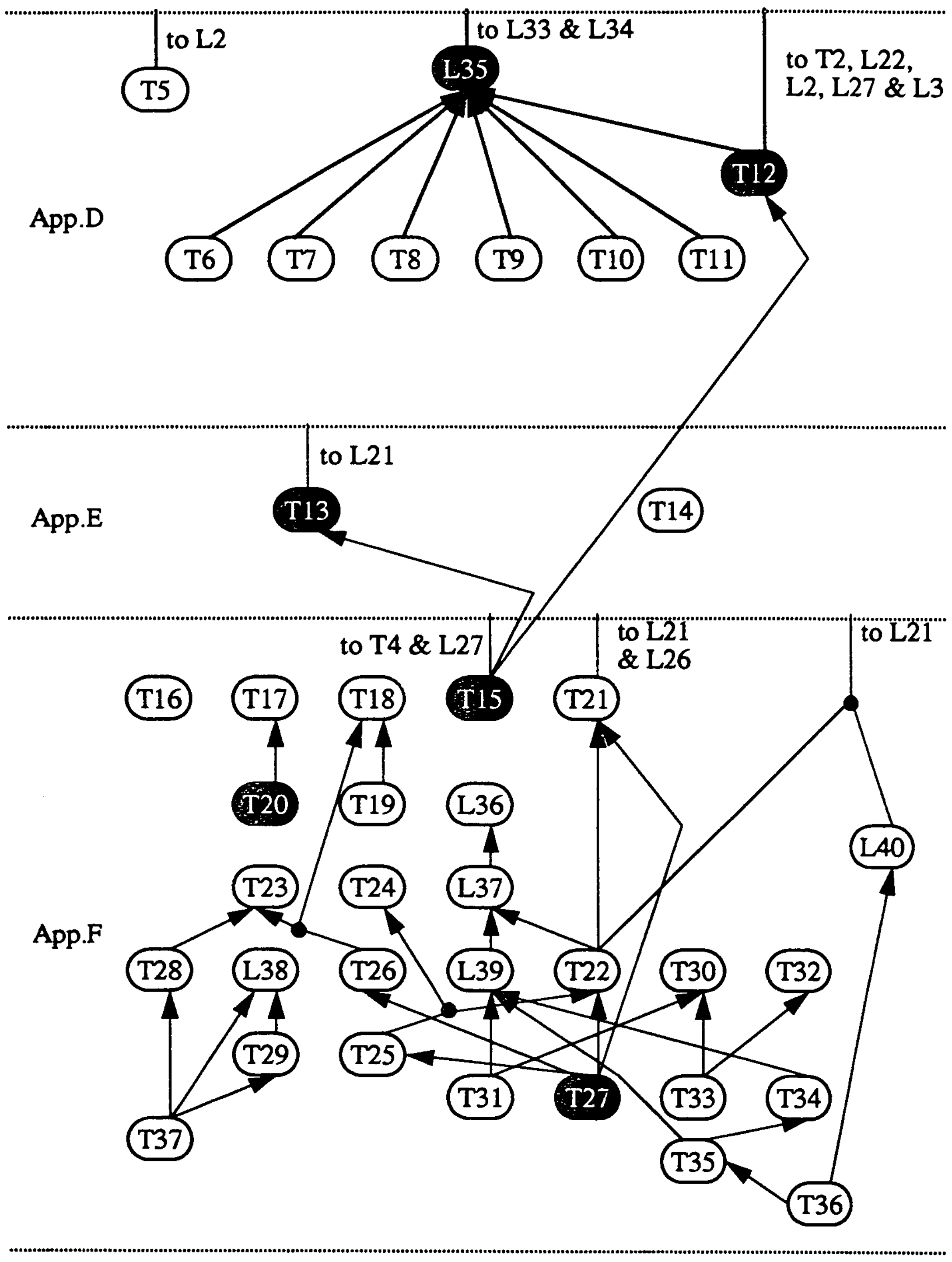


Figure 6.2 Appendices D to F

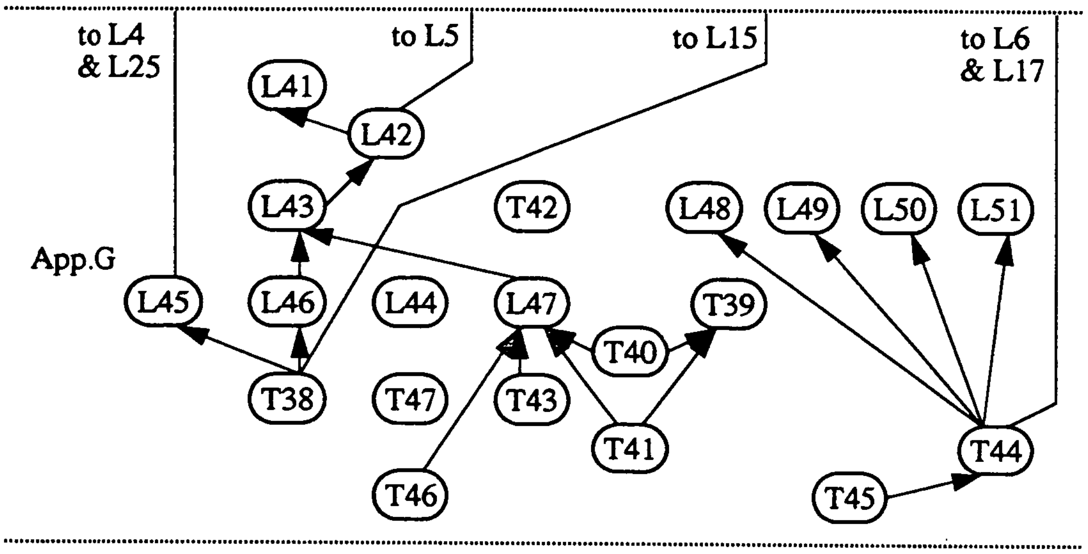


Figure 6.3 Appendix G

Appendix B : Basic Propositions I

In this section are proved some basic properties of sets, functions and computations which will be used later.

Lemma 1 "Commutativity of Composition"

If $A \parallel B$ is well-defined then so is $B \parallel A$ and $A \parallel B = B \parallel A$

Proof

Trivial from definition of "$\parallel$"

Lemma 2 "Associativity of Composition"

If $(A \parallel B)$, $(B \parallel C)$, $(A \parallel B) \parallel C$ and $A \parallel (B \parallel C)$ are well-defined, then

$(A \parallel B) \parallel C = A \parallel (B \parallel C)$

Proof

Trivial from definition of "$\parallel$"

Lemma 3 "Generalised Associativity of Composition"

If $(\parallel_{i \in \{1..k-1\}} A_i)$, $(\parallel_{i \in \{1..k\}} A_i)$and $A_k \parallel (\parallel_{i \in \{1..k-1\}} A_i)$ are well-defined, then

$A_k \parallel (\parallel_{i \in \{1...k-1\}} A_i) = \parallel_{i \in \{1..k\}} A_i$

Proof

Trivial from definition of "$\parallel$".

Lemma 4

If C is a computation and R is 1-to-1, then C ® R is well-defined.

Proof

Obviously Out(C ® R) and In(C ® R) are well-defined (see page 45). We therefore simply need to prove that Rel(C ® R) corresponds to a functional on valuations on In(C ® R). i.e. that

For all v', $v'|_{In(C ® R)} = v|_{In(C ® R)} \Rightarrow v'|_{Out(C ® R)} = v|_{Out(C ® R)}$ (i)

and

For all valuations v_{in} on In(C ® R),

there exists v_{out} s.t. Rel(C ® R)($v_{in} \cup v_{out}$) (ii)

Proof of (i)

Assume Rel(C ® R)v and Rel(C ® R)v'. By the definition of Rel(C ® R), we know that Rel(C)v•R and Rel(C)v'•R.

Assume further that

$$v'|_{In(C \circledR R)} = v|_{In(C \circledR R)}$$

so that

$$v' \bullet R|_{In(C)} = v \bullet R|_{In(C)}$$

This implies, by the fact that Rel(C) corresponds to a valuation on In(C), that

$$v' \bullet R|_{In(C)} = v \bullet R|_{In(C)}$$

which implies that

$$v'|_{Out(C \circledR R)} = v|_{Out(C \circledR R)}$$

Proof of (ii)

Let v_{in}'' equal $v_{in} \bullet R$. Then there exists v_{out}'' s.t. Rel(C) $(v_{in}'' \cup v_{out}'')$ so

$$Rel(C \circledR R)((v_{in}'' \cup v_{out}'') \bullet R^{-1}) = Rel(C \circledR R)v_{in} \cup (v_{out}'' \bullet R|_{Out(C \circledR R)}{}^{-1})$$

So let v_{out} equal $v_{out}'' \bullet R^{-1}$.

Lemma 5

Let C be a computation. Every valuation v on In_0, where $In_0 \subseteq In(C)$ can be extended to v' on Vars(C) for which Rel(C)v' holds.

Proof

Extend v arbitrarily to v'' on In(C). Let v' be v'' $\cup$ v''' where Fun(C)v'' = v'''

Lemma 6

$$ran(R|_{S \cup T}) = ran(R|_S) \cup ran(R|_T)$$

Lemma 7

If (A' || B), (A' || B) \Varset, (A' \Varset) and (A' \Varset) || B are well-defined and if Vars(B) $\cap$ Varset = $\emptyset$ then

(A' || B)\Varset = (A'\Varset) || B

<u>Proof</u>

Out((A' || B)\Varset) = Out(A' || B) - Varset
= Out(A') $\cup$ Out(B) - Varset
= (Out(A') - Varset) $\cup$ Out(B)
by the fact that Vars(B) $\cap$ Varset = $\emptyset$
= Out((A'\Varset) || B)

In((A' || B)\Varset) = (In(A') $\cup$ In(B) - Out(A' || B)) - Varset
= (In(A') $\cup$ In(B) - Out(A') $\cup$ Out(B)) - Varset
= (In(A') - Out(A')) $\cup$ (In(B) - Out(A'))
$\cup$ (In(A') - Out(B)) $\cup$ (In(B) - Out(B))
- Varset
= ((In(A') - Varset) - (Out(A') - Varset))
$\cup$ ((In(B) - Varset) - (Out(A') - Varset))
$\cup$ ((In(A') - Varset) - (Out(B) - Varset))
$\cup$ ((In(B) - Varset) - (Out(B) - Varset))
= ((In(A') - Varset) - (Out(A') - Varset))
$\cup$ (In(B) - (Out(A') - Varset))
$\cup$ ((In(A') - Varset) - Out(B))
$\cup$ (In(B) - Out(B))
by the fact that Vars(B) $\cap$ Varset = $\emptyset$
= In((A'\Varset) || B)

Rel((A' || B) \ Varset)v

$\Leftrightarrow$ for all v', Rel(A' || B)v' $\Rightarrow$

$(v'|_{In(A' || B) - Varset} = v|_{In(A' || B) - Varset}$

$\Rightarrow$

$v'|_{Out(A' || B) - Varset} = v|_{Out(A' || B) - Varset})$

$\Leftrightarrow$ for all v', (Rel(A')$v'|_{vars(A')}$ and Rel(B)$v'|_{vars(B)}$) $\Rightarrow$

$(v'|_{In(A' || B) - Varset} = v|_{In(A' || B) - Varset}$

$\Rightarrow$

$v'|_{Out(A' || B) - Varset} = v|_{Out(A' || B) - Varset})$

by definition of Rel(A' || B)

and

Rel((A'\Varset) || B)v

$\Leftrightarrow$ Rel(B)$v|_{Vars(B)}$ and Rel(A'\Varset)$v|_{Vars(A'\backslash Varset)}$

by definition

So it is sufficient to prove that the re-written versions of Rel((A' || B) \ Varset)v and Rel((A'\Varset) || B)v are equivalent, i.e. that

(for all v', (Rel(A')$v'|_{vars(A')}$ and Rel(B)$v'|_{vars(B)}$) $\Rightarrow$

$(v'|_{In(A' || B) - Varset} = v|_{In(A' || B) - Varset}$

$\Rightarrow$

$v'|_{Out(A' || B) - Varset} = v|_{Out(A' || B) - Varset}))$

$\Leftrightarrow$ Rel(B)$v|_{Vars(B)}$ and Rel(A'\Varset)$v|_{Vars(A'\backslash Varset)}$)

Let us prove the implication "$\Rightarrow$" and then the implication "$\Leftarrow$".

$\underline{\Rightarrow}$

We will assume the L.H.S. of the implication and attempt to prove the R.H.S.

Choose v' s.t.

$v'|_{In(A' \| B)} = v|_{In(A' \| B)}$ and $Rel(A' \| B)v'$

(This is possible, by Lemma 5.)

Then, by L.H.S., $v'|_{Out(A' \| B) - Varset} = v|_{Out(A' \| B) - Varset}$ which implies that

$$v'|_{Vars(A' \| B) - Varset} = v|_{Vars(A' \| B) - Varset}$$

so

$$v'|_{Vars(B)} = v|_{Vars(B)}$$

since $Vars(B) \cap Varset = \emptyset$. So $Rel(B)v|_{Vars(B)}$ holds. We now need to prove that $Rel(A'\backslash Varset)v|_{Vars(A'\backslash Varset)}$ i.e. that

for all v'', $Rel(A')v'' \Rightarrow$

$$\begin{array}{rcl} v''|_{In(A') - Varset} & = & v|_{In(A') - Varset} \\ \Rightarrow & & \\ v''|_{Out(A') - Varset} & = & v|_{Out(A') - Varset} \end{array}$$

Take an arbitrary v'' s.t. $Rel(A')v''$ and

$$v''|_{In(A') - Varset} = v|_{In(A') - Varset}$$

We just need to prove that

$$v''|_{Out(A') - Varset} = v|_{Out(A') - Varset}$$

Extend v'' to v''' on $Vars(A' \| B)$ s.t. $Rel(A' \| B)v'''$ and

$$v'''|_{In(A' \| B) - Varset} = v|_{In(A' \| B) - Varset}$$

Is this possible? Yes: let v'''' be s.t.

$$v''''|_{In(A' \| B) - Varset} = v|_{In(A' \| B) - Varset}$$

and

$$v''''|_{In(A') \cap Varset} = v|_{In(A') \cap Varset}$$

$dom(v'''') = In(A' \| B)$ so by Lemma 5 it can be extended to v''' s.t. $Rel(A' \| B)v'''$ holds.

Then, by the L.H.S.,

$$v'''|_{Out(A' \| B) - Varset} = v|_{Out(A' \| B) - Varset}$$

which implies that

$$v'''|_{Out(A') - Varset} = v|_{Out(A') - Varset}$$

But

$$v'''|_{Out(A') - Varset} = v''|_{Out(A') - Varset}$$

so

$$v'''|_{Out(A') - Varset} = v|_{Out(A') - Varset}$$

Q.E.D.

$\Leftarrow$

We will assume the R.H.S. and attempt to prove the L.H.S.

So assume that

$$Rel(B)v|_{Vars(B)}$$

holds and also that

$$Rel(A'\backslash Varset)v|_{Vars(A'\backslash Varset)}$$

holds, i.e. that

for all v' $Rel(A')v' \Rightarrow$

$$(v'|_{In(A') - Varset} = v|_{In(A') - Varset} \Rightarrow v'|_{Out(A') - Varset} = v|_{Out(A') - Varset})$$

Furthermore assume that

$$Rel(A')v''|_{Vars(A')}$$

and

$$Rel(B)v''|_{Vars(B)}$$

and

$$v''|_{In(A' \parallel B) - Varset} = v|_{In(A' \parallel B) - Varset}$$

hold, where v'' is an arbitrary valuation on Vars(A' $\parallel$ B).

We want to prove that

$$v''|_{Out(A' \parallel B) - Varset} = v|_{Out(A' \parallel B) - Varset}$$

We know, since $v''|_{In(A') - Varset} = v|_{In(A') - Varset}$, that

$$v''|_{Out(A') - Varset} \quad = \quad v|_{Out(A') - Varset}$$

Also

$$v''|_{In(B)} \quad = \quad v|_{In(B)}$$

and

$$Rel(B)v|_{Vars(B)} \text{ and } Rel(B)v|_{Vars(B)}$$

so

$$v''|_{Out(B)} \quad = \quad v|_{Out(B)}$$

So we have the desired result.

<u>Lemma 8</u>

$$ran(R|_{S-T}) \quad \subseteq \quad ran(R|_S)$$

<u>Lemma 9</u>

$$ran(R|_{S-T}) \quad \supseteq \quad ran(R|_S) - ran(R|_T)$$

<u>Lemma 10</u>

$$ran(R|_{\bigcup_{i \in I} S_i}) = \bigcup_{i \in I} ran(R|_{S_i})$$

<u>Lemma 11</u>

$$ran(R|_{S-T}) \quad = \quad ran(R|_S) - ran(R|_T) \qquad \text{if R is 1-to-1}$$

<u>Lemma 12</u>

$$\bigcup_{i \in I} (S_i - T_i) \subseteq \bigcup_{i \in I} S_i$$

<u>Lemma 13</u>

$$\bigcup_{i \in I} (S_i - T_i) \supseteq \bigcup_{i \in I} S_i - \bigcup_{i \in I} T_i$$

<u>Lemma 14</u>

$$S - T - U = S - U \text{ if } T \subseteq U$$

<u>Lemma 15</u>

$$S \subseteq T \Rightarrow S - U \subseteq T - U$$

<u>Lemma 16</u>

$$A \cup B - (B - A) = A$$

<u>Lemma 17</u>

$$S \subseteq T \Rightarrow \mathrm{ran}(R|_S) \subseteq \mathrm{ran}(R|_T)$$

<u>Proofs</u>

Easy

Appendix C : Basic Propositions II

The key results in this appendix are Lemma 21, Lemma 22 and Lemma 27. Lemma 21 states that renaming distributes over composition.Lemma 22 states that (providing certain conditions hold) if A' simulates A then A' || B simulates A || B. Lemma 27 states that if A simulates B and B simulates C then A simulates C, and gives the relationship between the parameter pairs of the simulations. These two propositions play an important role in the proofs of the later results which state that the transformations of the method preserve behaviour. The other propositions in this section are required for the proofs of the key results.

Lemma 18

For all i in I, let C_i be a computation. If $\|_{i \in I} C_i$ is well-defined and dom(R) = Vars($\|_{i \in I} C_i$), then

$$\begin{aligned} &\mathrm{ran}(R\rfloor(\bigcup_{i \in I} \mathrm{In}(C_i) - \mathrm{Out}(\|_{i \in I} C_i))) - \mathrm{Out}((\|_{i \in I} C_i) ® R) \\ = \;& \mathrm{ran}(R\rfloor(\bigcup_{i \in I} \mathrm{In}(C_i))) - \mathrm{Out}((\|_{i \in I} C_i) ® R) \end{aligned}$$

Proof

$\subseteq$

by Lemma 8 and Lemma 15

$\supseteq$

by Lemma 9 and Lemma 14

Comment: this lemma, and the following two, are used in the proof of Lemma 21. The proof uses the fact that

$\mathrm{Out}((\|_{i \in I} C_i) \circledR R) = R|_{\mathrm{Out}(\|\searrow(i \in I)\, C\searrow i)}$

Lemma 19

For all i in I, let C_i be a computation. If $\|_{i \in I} C_i$ is well-defined and $\mathrm{dom}(R) = \bigcup_{i \in I} \mathrm{Vars}(C_i)$, then

$$\bigcup_{i \in I} \mathrm{ran}(R|_{\mathrm{In}(C\searrow i)}) - \bigcup_{i \in I} \mathrm{ran}(R|_{\mathrm{Out}(C\searrow i)})$$

$$= \bigcup_{i \in I} (\mathrm{ran}(R|_{\mathrm{In}(C\searrow i)}) - \mathrm{ran}(R|_{\mathrm{Out}(C\searrow i)})) - \bigcup_{i \in I} \mathrm{ran}(R|_{\mathrm{Out}(C\searrow i)})$$

Proof

$\subseteq$

by Lemma 14 and Lemma 13

$\supseteq$

by Lemma 12

Lemma 20

Assume that $\mathrm{dom}(v) = \mathrm{ran}(R)$; then

$\mathrm{Rel}(A)((v \bullet R)|_{\mathrm{Vars}(A)}) \Leftrightarrow \mathrm{Rel}(A \circledR R|_{\mathrm{Vars}(A)})v|_{\mathrm{Vars}(A \circledR R|\mathrm{Vars}(A))}$

Proof

Now, using the definition of renaming on page 45 and the fact that

$$\mathrm{Vars}(A \circledR R|_{\mathrm{Vars}(A)}) = \mathrm{ran}(R|_{\mathrm{Vars}(A)})$$

$$\mathrm{Rel}(A \circledR R|_{\mathrm{Vars}(A)})v|_{\mathrm{Vars}(A \circledR R\rfloor \mathrm{Vars}(A))} \Leftrightarrow$$
$$\mathrm{Rel}(A)(v|_{\mathrm{ran}(R\rfloor \mathrm{Vars}(A))}{\bullet}R|_{\mathrm{Vars}(A)})$$

so we just need to show that

$$(v{\bullet}R)|_{\mathrm{Vars}(A)} = v|_{\mathrm{ran}(R\rfloor \mathrm{Vars}(A))}{\bullet}R|_{\mathrm{Vars}(A)}$$

This is obviously true if both sides are well-defined and have the same domain. The L.H.S. is well-defined, since ran(R) = dom(v). The R.H.S. is also well-defined, since $\mathrm{ran}(R|_{\mathrm{Vars}(A)}) = \mathrm{dom}(v|_{\mathrm{ran}(R\rfloor \mathrm{Vars}(A))})$. The domain of the L.H.S. = (dom(R) $\cap$ Vars(A)) = the domain of the R.H.S.

Lemma 21

If $(\|_{i \in I} C_i) \circledR R$ is well-defined and, for all i, $C_i \circledR R|_{\mathrm{Vars}(C\searrow i)}$ is well-defined, then

$$(\|_{i \in I} C_i) \circledR R = \|_{i \in I} (C_i \circledR R|_{\mathrm{Vars}(C\searrow i)})$$

Proof

$$\mathrm{Out}((\|_{i \in I} C_i) \circledR R)$$
$$= \mathrm{ran}(R\rfloor(\mathrm{Out}(\|_{i \in I} C_i)))$$

by definition of renaming

$$= \mathrm{ran}(R\rfloor(\bigcup_{i \in I} \mathrm{Out}(C_i)))$$

by definition of composition

$$= \bigcup_{i \in I} \operatorname{ran}(R|_{\operatorname{Out}(C \searrow i)})$$

by Lemma 10

$$= \bigcup_{i \in I} \operatorname{Out}(C_i \circledR R)$$

by definition of renaming

$$= \operatorname{Out}(\|_{i \in I} (C_i \circledR R))$$

$$\operatorname{In}((\|_{i \in I} C_i) \circledR R) = \operatorname{ran}(R \rfloor \operatorname{In}(\|_{i \in I} C_i)) - \operatorname{Out}((\|_{i \in I} C_i) \circledR R)$$

by definition of renaming

$$= \operatorname{ran}(R \rfloor (\bigcup_{i \in I} \operatorname{In}(C_i) - \operatorname{Out}(\|_{i \in I} C_i))) - \operatorname{Out}((\|_{i \in I} C_i) \circledR R)$$

by definition of composition

$$= \operatorname{ran}(R \rfloor (\bigcup_{i \in I} \operatorname{In}(C_i))) - \operatorname{Out}((\|_{i \in I} C_i) \circledR R)$$

by Lemma 18

$$= \bigcup_{i \in I} \operatorname{ran}(R|_{\operatorname{In}(C \searrow i)}) - \operatorname{Out}((\|_{i \in I} C_i) \circledR R)$$

by Lemma 10

$$= \bigcup_{i \in I} \operatorname{ran}(R|_{\operatorname{In}(C \searrow i)}) - \bigcup_{i \in I} \operatorname{ran}(R|_{\operatorname{Out}(C \searrow i)})$$

by definitions of renaming and composition

$$= \bigcup_{i \in I} (\operatorname{ran}(R|_{\operatorname{In}(C \searrow i)}) - \operatorname{ran}(R|_{\operatorname{Out}(C \searrow i)}))$$

$$- \bigcup_{i \in I} \operatorname{ran}(R|_{\operatorname{Out}(C \searrow i)})$$

by Lemma 19

$$= \bigcup_{i \in I} \operatorname{In}(C_i \circledR R|_{\operatorname{Vars}(C \searrow i)}) - \bigcup_{i \in I} \operatorname{Out}(C_i \circledR R|_{\operatorname{Vars}(C \searrow i)})$$

$$= \operatorname{In}(\|_{i \in I} (C_i \circledR R))$$

by definition of composition.

Now to prove the equivalence of $\operatorname{Rel}((\|_{i \in I} C_i) \circledR R)v$ and $\operatorname{Rel}(\|_{i \in I} C_i \circledR R|_{\operatorname{Vars}(C \searrow i)})$:

Let v be a valuation on ran(R); then

$\mathrm{Rel}((\|_{i \in I} C_i) \circledR R)v \Leftrightarrow \mathrm{Rel}(\|_{i \in I} C_i)(v \bullet R)$

by definition of renaming

$\Leftrightarrow$ (For all i in I, $\mathrm{Rel}(C_i)((v \bullet R)|_{Vars(C \searrow i)})$)

by definition of composition

$\Leftrightarrow \mathrm{Rel}(\|_{i \in I} C_i \circledR R|_{Vars(C \searrow i)})$

byLemma 20 and definition of composition

Lemma 22

If A' || B is well-defined

and A || B is well-defined

and (A' || B)\Varset is well-defined

and (A'\Varset) || B is well-defined

and $Vars(B) \cap Varset = \emptyset$

and A' simulates A w.r.t. $<Varset, R_1>$

and $R_1|_{Vars(B)} \cap Vars(A'\backslash Varset) \subseteq Id_{Vars(B)}$

then if R_2 is s.t.

$dom(R_2) = Vars(A) \cup Vars(B)$

and $R_2|_{Vars(A'\backslash Varset)} = R_1$

and $R_2|_{Vars(B)} = Id_{Vars(B)}$

then A' || B simulates A || B w.r.t. $<Varset, R_2>$

Proof

$$
\begin{aligned}
(A' \parallel B)\backslash Varset &= (A'\backslash Varset) \parallel B && \text{by Lemma 7, so} \\
(A' \parallel B)\backslash Varset \circledR R_2 &= ((A'\backslash Varset) \parallel B) \circledR R_2 \\
&= (A'\backslash Varset \circledR R_2|_{Vars(A'\backslash Varset)}) \parallel (B \circledR R_2|_{Vars(B)}) \\
& && \text{by Lemma 4} \\
&= A \parallel B
\end{aligned}
$$

Lemma 23

Assume that R is invertible (i.e. 1-to-1) with dom(R) equal to Vars(C) and that $(C \circledR R)\backslash Varset$ and $(C\backslash Varset')$ are well-defined, where

$$Varset' = ran(R^{-1}|_{Varset})$$

then

$$(C \circledR R)\backslash Varset = (C\backslash Varset') \circledR (R|_{Vars(C) - Varset'})$$

Proof

$$
\begin{aligned}
Out(C \circledR R \backslash Varset) &= Out(C \circledR R) - Varset \\
&= ran(R|_{Out(C)}) - Varset \\
& && \text{by definition of renaming} \\
&= ran(R|_{Out(C)}) - ran(R|_{Varset'}) \\
&= ran(R|_{Out(C) - Varset'}) && \text{by Lemma 11} \\
&= ran((R|_{Vars(C) - Varset'})|_{Out(C) - Varset'}) \\
&= Out((C\backslash Varset') \circledR (R|_{Vars(C) - Varset'}))
\end{aligned}
$$

$$
\begin{aligned}
In(C \circledR R \backslash Varset) &= In(C \circledR R) - Varset \\
&= ran(R|_{In(C)}) - Varset
\end{aligned}
$$

(Since R is 1-to-1 we do not need to subtract Out(C ® R))

$= \mathrm{ran}(R|_{In(C)}) - \mathrm{ran}(R|_{Varset'})$

$= \mathrm{ran}(R|_{In(C) - Varset'})$ by Lemma 11

$= \mathrm{ran}((R|_{Vars(C) - Varset'})|_{In(C) - Varset'})$

$= In((C\backslash Varset') ® (R|_{Vars(C) - Varset'}))$

Rel(C ® R \ Varset)v $\Leftrightarrow$

For all v', Rel(C ® R)v' $\Rightarrow$

$(v'|_{In((C ® R)\backslash Varset)} = v|_{In((C ® R)\backslash Varset)}$

$\Rightarrow$

$v'|_{Out((C ® R)\backslash Varset)} = v|_{Out((C ® R)\backslash Varset)})$

by definition of hiding

$\Leftrightarrow$ For all v', Rel(C)(v'•R) $\Rightarrow$

$(v'|_{In((C ® R)\backslash Varset)} = v|_{In((C ® R)\backslash Varset)}$

$\Rightarrow$

$v'|_{Out((C ® R)\backslash Varset)} = v|_{Out((C ® R)\backslash Varset)})$

by definition of renaming

$\Leftrightarrow$ For all v'', Rel(C)(v'') $\Rightarrow$

$(v''\bullet R^{-1})|_{\mathrm{ran}(R\rfloor In(C) - Varset')} = v|_{\mathrm{ran}(R\rfloor In(C) - Varset')}$

$\Rightarrow$

$(v''\bullet R^{-1})|_{\mathrm{ran}(R\rfloor Out(C) - Varset')} = v|_{\mathrm{ran}(R\rfloor Out(C) - Varset')}$

(We are setting v'' equal to v'.R. We can then write v' as v''.R^{-1} since R is 1-to-1.)

$\Leftrightarrow$ For all v'', Rel(C)(v'') $\Rightarrow$

$v''|_{In(C) - Varset'} = v\bullet(R|_{In(C) - Varset'})$

$\Rightarrow$

$v''|_{Out(C) - Varset'} = v\bullet(R|_{Out(C) - Varset'})$

$\Leftrightarrow$ Rel(C\Varset') ® $R|_{Vars(C) - Varset'}$

by definition of hiding

Lemma 24

C\Varset is well-defined $\Leftrightarrow$

for all v and v'', (Rel(C)v and Rel(C)v'') $\Rightarrow$

$(v|_{In(C) - Varset} = v''|_{In(C) - Varset}$

$\Rightarrow$

$v|_{Out(C) - Varset} = v''|_{Out(C) - Varset})$

Proof

C\Varset is well-defined $\Leftrightarrow$
Fun(C\Varset) is a well-defined function

Hence it is sufficient to prove that

for all v_{in} on In(C) - Varset, there exists v_{out} on Out(C) - Varset s.t.

(Rel(C\Varset) $v_{in} \cup v_{out}$

and

for all valuations v_{in}' on In(C) - Varset and v_{out}' on Out(C) - Varset,

(Rel(C\Varset) $v_{in}' \cup v_{out}' \Rightarrow$

$(v_{in}' = v_{in} \Rightarrow$

$v_{out}' = v_{out}))$

is equivalent to

for all v and v'', (Rel(C)v and Rel(C)v'') $\Rightarrow$

$(v|_{In(C) - Varset} = v''|_{In(C) - Varset}$

$\Rightarrow$

$$v|_{Out(C) - Varset} = v''|_{Out(C) - Varset})$$

We will prove "$\Rightarrow$" and then "$\Leftarrow$".

$\underline{\Rightarrow}$

Assume the L.H.S. is true, that Rel(C)v and Rel(C)v'' hold and that

$$v|_{In(C) - Varset} = v''|_{In(C) - Varset}$$

It is sufficient to prove that

$$v|_{Out(C) - Varset} = v''|_{Out(C) - Varset})$$

Let v_{in} equal $v|_{In(C) - Varset}$: then Rel(C\Varset)$v|_{Vars(C)-Varset}$ holds, so

$$v|_{Out(C) - Varset} = v_{out}$$

(letting v_{in}' equal $v|_{In(C) - Varset}$ and v_{out}' equal $v|_{Out(C) - Varset}$ in the L.H.S.)

By a similar argument,

$$v''|_{Out(C) - Varset} = v_{out}$$

so

$$v|_{Out(C) - Varset} = v''|_{Out(C) - Varset}$$

$\underline{\Leftarrow}$

Assume the R.H.S. is true, and consider an arbitrary valuation v_{in} on In(C) - Varset. By Lemma 5 there exists v s.t.

$$v|_{In(C) - Varset} = v_{in}$$

and

Rel(C)v holds

Let v_{out} equal $v|_{Out(C) - Varset}$. Firstly we will prove that Rel(C\Varset)$v_{in} \cup v_{out}$ holds. Let v'' be s.t.

Rel(C)v''

and

$$v''|_{In(C) - Varset} = v|_{In(C) - Varset} = v_{in}$$

then by assumption of the R.H.S.,

$$v''|_{Out(C) - Varset} = v|_{Out(C) - Varset}$$

so we know that Rel(C\Varset)$v|_{Vars(C) - Varset}$ holds. But

$v|_{Vars(C) - Varset} = v_{in} \cup v_{out}$ so we have the desired result.

We now just need to prove that, for arbitrary valuations v_{in}' on In(C) - Varset and v_{out}' on Out(C) - Varset,

$$(\text{Rel}(C\backslash\text{Varset})\ v_{in}' \cup v_{out}' \Rightarrow (v_{in}' = v_{in} \Rightarrow v_{out}' = v_{out}))$$

So let us assume that Rel(C\Varset)($v_{in}' \cup v_{out}'$) holds, that v_{in}' equals v_{in}. By Lemma 5, we may extend v_{in}' to v' s.t. Rel(C)v' holds. Then

$v'|_{Out(C) - Varset} = v_{out}'$ (from the definition of Rel(C\Varset), since Rel(C\Varset)$v_{in}' \cup v_{out}'$ holds)

so

$$\begin{aligned} v_{out}' &= v'|_{Out(C) - Varset} \\ &= v|_{Out(C) - Varset} \quad \text{by R.H.S. (setting v'' equal to v')} \\ &= v|_{Out(C) - Varset} \\ &= v_{out} \end{aligned}$$

and so the L.H.S. is true.

<u>Lemma 25</u>

If $C\backslash V_1$ and $(C\backslash V_1)\backslash V_2$ are well-defined then $C\backslash(V_1 \cup V_2)$ is well-defined and $(C\backslash V_1)\backslash V_2 = C\backslash(V_1 \cup V_2)$

<u>Proof</u>

$\mathrm{Out}((C\backslash V_1)\backslash V_2) \;=\; \mathrm{Out}(C) - V_1 - V_2 \;=\; \mathrm{Out}(C) - (V_1 \cup V_2)$
$= \;\mathrm{Out}(C\backslash(V_1 \cup V_2))$

$\mathrm{In}((C\backslash V_1)\backslash V_2) \;=\; \mathrm{In}(C) - V_1 - V_2 \;=\; \mathrm{In}(C) - (V_1 \cup V_2)$
$= \;\mathrm{In}(C\backslash(V_1 \cup V_2))$

We therefore simply need to prove that, for all valuations v on $\mathrm{Vars}(C) - (V_1 \cup V_2)$,

$\mathrm{Rel}((C\backslash V_1)\backslash V_2)v \Leftrightarrow \mathrm{Rel}(C\backslash(V_1 \cup V_2))v$

This equivalence will imply that $C\backslash(V_1 \cup V_2)$ is well-defined, since $(C\backslash V_1)\backslash V_2$ is. We will prove "$\Rightarrow$" and then "$\Leftarrow$".

$\underline{\Leftarrow}$

Let us assume the R.H.S., i.e. that, for all v',

$\mathrm{Rel}(C)v' \Rightarrow$

$(v'|_{\mathrm{In}(C\backslash V_1) - V_2} \;=\; v|_{\mathrm{In}(C\backslash V_1) - V_2}$

$\Rightarrow$

$v''|_{\mathrm{Out}(C\backslash V_1) - V_2} \;=\; v|_{\mathrm{Out}(C\backslash V_1) - V_2})$

But

$$\mathrm{In}((C\backslash V_1)\backslash V_2) \quad = \quad \mathrm{In}(C\backslash(V_1 \cup V_2))$$

and

$$\mathrm{Out}((C\backslash V_1)\backslash V_2) \quad = \quad \mathrm{Out}(C\backslash(V_1 \cup V_2))$$

so this is the same as saying that, for all v',

Rel(C)v' $\Rightarrow$

$$\begin{aligned} (v'|_{\mathrm{In}(C) - V\searrow 1 \cup V\searrow 2} &= v|_{\mathrm{In}(C) - V\searrow 1 \cup V\searrow 2} \\ &\Rightarrow \\ v'|_{\mathrm{Out}(C) - V\searrow 1 \cup V\searrow 2} &= v|_{\mathrm{Out}(C) - V\searrow 1 \cup V\searrow 2}) \end{aligned}$$

In order to prove this, let us assume that Rel($C\backslash V_1$)v' i.e. that for all v''', Rel(C)v''' $\Rightarrow$

$$\begin{aligned} (v'''|_{\mathrm{In}(C) - V\searrow 1} &= v'|_{\mathrm{In}(C) - V\searrow 1} \\ &\Rightarrow \\ v'''|_{\mathrm{Out}(C) - V\searrow 1} &= v'|_{\mathrm{Out}(C) - V\searrow 1}) \end{aligned}$$

and also that

$$v'|_{\mathrm{In}(C) - V\searrow 1 \cup V\searrow 2} = v|_{\mathrm{In}(C) - V\searrow 1 \cup V\searrow 2}$$

It will be sufficient to prove

$$v'|_{\mathrm{Out}(C) - V\searrow 1 \cup V\searrow 2} = v|_{\mathrm{Out}(C) - V\searrow 1 \cup V\searrow 2}$$

By Lemma 5, we may extend $v'|_{\mathrm{In}(C) - V\searrow 1 \cup V\searrow 2}$ to a valuation v'''' on Vars(C) s.t. Rel(C)v'''' holds; and now

$$\begin{aligned} v''''|_{\mathrm{In}(C) - V\searrow 1 \cup V\searrow 2} &= v'|_{\mathrm{In}(C) - V\searrow 1 \cup V\searrow 2} \\ &= v|_{\mathrm{In}(C) - V\searrow 1 \cup V\searrow 2} \end{aligned}$$

so

$$v''''|_{Out(C) - V{\searrow}1 \cup V{\searrow}2} = v|_{Out(C) - V{\searrow}1 \cup V{\searrow}2}$$

by assumed R.H.S.

but also

$$v''''|_{Out(C) - V{\searrow}1} = v|_{Out(C) - V{\searrow}1}$$

by assumption that Rel(C\V$_1$)v' holds

So

$$\begin{aligned} v'|_{Out(C) - V{\searrow}1 \cup V{\searrow}2} &= v''''|_{Out(C) - V{\searrow}1 \cup V{\searrow}2} \\ &= v|_{Out(C) - V{\searrow}1 \cup V{\searrow}2} \end{aligned}$$

which is what we were aiming to prove.

$\Rightarrow$

Assume the L.H.S., i.e. (in doubly expanded form) that

for all v',

(for all v'', Rel(C)v'' $\Rightarrow$

$$\begin{aligned} &(v''|_{In(C) - V{\searrow}1} = v'|_{In(C) - V{\searrow}1} \\ &\quad\Rightarrow \\ &v''|_{Out(C) - V{\searrow}1} = v'|_{Out(C) - V{\searrow}1})) \end{aligned}$$

$\Rightarrow$

$$\begin{aligned} &(v'|_{In(C) - V{\searrow}1 \cup V{\searrow}2} = v|_{In(C) - V{\searrow}1 \cup V{\searrow}2} \\ &\quad\Rightarrow \\ &v'|_{Out(C) - V{\searrow}1 \cup V{\searrow}2} = v|_{Out(C) - V{\searrow}1 \cup V{\searrow}2}) \end{aligned}$$

We want to prove the R.H.S. To do this we will assume Rel(C)v''' holds and that

$$(v'''|_{In(C) - V{\searrow}1 \cup V{\searrow}2} = v|_{In(C) - V{\searrow}1 \cup V{\searrow}2}$$

and prove that

$$v'''|_{Out(C) - V\searrow 1 \cup V\searrow 2} \quad = \quad v|_{Out(C) - V\searrow 1 \cup V\searrow 2})$$

Since $C\backslash V_1$ is well-defined and Rel(C)v''' holds, we may deduce from Lemma 24 (with v' specialized to v''') that

for all v, Rel(C)v $\Rightarrow$

$$((v|_{In(C) - V\searrow 1} \quad = \quad v'''|_{In(C) - V\searrow 1}$$
$$\Rightarrow$$
$$v|_{Out(C) - V\searrow 1} \quad = \quad v'''|_{Out(C) - V\searrow 1}))$$

which is the L.H.S. of the first hypothesis with v'' replaced by v and v''' replaced by v'''. (Note that v here is a dummy variable and does not necessarily equal the other v.) Therefore we may deduce that

$$(v'''|_{In(C) - V\searrow 1 \cup V\searrow 2} \quad = \quad v|_{In(C) - V\searrow 1 \cup V\searrow 2}$$
$$\Rightarrow$$
$$v'''|_{Out(C) - V\searrow 1 \cup V\searrow 2} \quad = \quad v|_{Out(C) - V\searrow 1 \cup V\searrow 2})$$

The hypothesis of this statement is true, so the conclusion is.

<u>Lemma 26</u>

$$C ® R_{(1)} ® R_{(2)} \quad = \quad C ® (R_{(2)} \bullet R_{(1)})$$

(assuming that $dom(R_{(2)}) = ran(R_{(1)})$ and $dom(R_{(1)}) = Vars(C)$)

<u>Proof</u>

The proof uses repeated application of the definition of renaming.

$$\begin{aligned} Out(C ® R_{(1)} ® R_{(2)}) &= ran(R_{(2)}|_{Out(C ® R\searrow(1))}) \\ &= ran(R_{(2)}|ran(R_{(1)}|_{Out(C)}) \\ &= ran(R_{(2)} \bullet R_{(1)}|_{Out(C)}) \end{aligned}$$

$$= \mathrm{Out}(C \circledR (R_{(2)} \bullet R_{(1)}))$$

$$\begin{aligned}
\mathrm{In}(C \circledR R_{(1)} \circledR R_{(2)}) &= \mathrm{ran}(R_{(2)}|_{\mathrm{In}(C \circledR R\searrow(1))}) - \mathrm{ran}(R_{(2)}|_{\mathrm{Out}(C \circledR R\searrow(1))}) \\
&= \mathrm{ran}(R_{(2)}|_{\mathrm{ran}(R\searrow(1)/\mathrm{In}(C)) - \mathrm{ran}(R\searrow(1)/\mathrm{Out}(C))}) \\
&\qquad - \mathrm{ran}(R_{(2)}|_{\mathrm{Out}(C \circledR R\searrow(1))}) \\
&= \mathrm{ran}(R_{(2)}|_{\mathrm{ran}(R\searrow(1)/\mathrm{In}(C)) - \mathrm{ran}(R\searrow(1)/\mathrm{Out}(C))}) \\
&\qquad - \mathrm{ran}(R_{(2)}|_{\mathrm{ran}(R\searrow(1)/\mathrm{Out}(C))}) \\
&= \mathrm{ran}(R_{(2)}|_{\mathrm{ran}(R\searrow(1)/\mathrm{In}(C))}) \\
&\qquad - \mathrm{ran}(R_{(2)}|_{\mathrm{ran}(R\searrow(1)/\mathrm{Out}(C))})
\end{aligned}$$

by Lemma 8, Lemma 9, Lemma 14 and Lemma 15

$$\begin{aligned}
&= \mathrm{ran}(R_{(2)} \bullet R_{(1)}|_{\mathrm{In}(C)}) - \mathrm{ran}(R_{(2)} \bullet R_{(1)}|_{\mathrm{Out}(C)}) \\
&= \mathrm{In}(C \circledR R_{(2)} \bullet R_{(1)})
\end{aligned}$$

$$\begin{aligned}
\mathrm{Rel}(C \circledR R_{(1)} \circledR R_{(2)})v &\Leftrightarrow \mathrm{Rel}(C \circledR R_{(1)})v \bullet R_{(2)} \\
&\Leftrightarrow \mathrm{Rel}(C)(v \bullet R_{(2)}) \bullet R_{(1)} \\
&\Leftrightarrow \mathrm{Rel}(C)v \bullet (R_{(2)} \bullet R_{(1)}) \\
&\Leftrightarrow \mathrm{Rel}(C \circledR (R_{(2)} \bullet R_{(1)}))v
\end{aligned}$$

Lemma 27

A simulates B w.r.t. $<V_{AB}, R_{AB}>$

and B simulates C w.r.t. $<V_{BC}, R_{BC}>$

$\Rightarrow$ A simulates C,

w.r.t. $<\mathrm{Vars}(A) - V_{AB} \cup V_{BC}', R_{BC} \bullet (R_{AB}|_{V\searrow AB \cup (V\searrow BC)'})>$

Proof

Assume the L.H.S. Then we know that $A \backslash V_{AB}$ and $B \backslash V_{BC}$ are well-defined and

$$A \backslash V_{AB} \circledR R_{AB} = B$$

and $B \backslash V_{BC} \circledR R_{BC} = C$

so

$C = (((A \backslash V_{AB}) \circledR R_{AB}) \backslash V_{BC}) \circledR R_{BC}$

$= (A \backslash V_{AB} \backslash V_{BC}') \circledR (R_{AB}|_{\mathrm{Vars}(A)\, -\, V\searrow AB\, -\, (V\searrow BC)'}) \circledR R_{BC}$

where $V_{BC}' = R_{AB}{}^{-1}|_{V\searrow BC}$, by Lemma 23 with C in Lemma 23 equal to $A \backslash V_{AB}$ and Varset' equal to V_{BC}'.

$= (A \backslash V_{AB} \cup V_{BC}') \circledR (R_{BC} \bullet R_{AB}|_{\mathrm{Vars}(A)\, -\, (V\searrow AB \cup (V\searrow BC)')})$

by Lemma 25 andLemma 26

so A simulates C w.r.t. $<\mathrm{Vars}(A) - (V_{AB} \cup V_{BC}')$, $R_{BC} \bullet R_{AB}|_{\mathrm{Vars}(A)\, -\, (V\searrow AB \cup (V\searrow BC)')}>$

Appendix D : Propositions relating to data-pipelining

The important result of this section is Theorem 4, which states that under certain conditions data-pipelining preserves behaviour. Theorem 1 and Theorem 2 are also key. The former states that data-pipelining is valid if certain conditions are fulfilled; the latter states that under certain conditions the pipelining of each data-dependency is valid, which is one of the conditions of Theorem 1. The other propositions of the section support the main ones, apart from Theorem 3, which states that there is only one way to pipeline two of the dependencies in the QR-factorisation example (see subsection 5.1.1 on page 134).

The definitions and assumptions made at the start of the previous appendices are assumed to hold for this one. The following ones also hold:

Definitions

DATA is an affine recurrence with mould $DATA\_M_{(1)}$ over base BASE. Let its set of dependency vectors relative to this mould and BASE be $\{<a_i, \Delta_i> : 1 \leq i \leq n\}$.

CONTROL is an embedded computation defined as follows

In(CONTROL) $= \emptyset$

Out(CONTROL) $= \{<c_1, p> : p \in BASE\}$

Rel(CONTROL)v $\Leftrightarrow$

For all p in BASE,

$(p \in BASE_{(1:0)} \Rightarrow v(<c_1, p>) = 0$ and

$p \in BASE_{(1:1)} \Rightarrow v(<c_1, p>) = 1)$

where $\{BASE_{(1:0)}, BASE_{(1:1)}\}$ is a partition of BASE

$\text{CONTROL}_{(1)} := \text{CONTROL}$

Let $\text{R\_DP}_{(i)}$ be defined on $\text{Vars}(\text{DATA\_M}_{(i-1)})$ as follows:

$\text{R\_DP}_{(i)}(\langle a_i, \Delta_i\rangle) := \langle z_i, \text{Id}_{\text{BASE}}\rangle$

and for all $\langle a', \Delta'\rangle$ not equal to $\langle a_i, \Delta_i\rangle$,

$\text{R\_DP}_{(i)}(\langle a', \Delta'\rangle) := \langle a', \Delta'\rangle$

Let r_i be chosen to satisfy the aforementioned assumption in which it appears.

Let $\text{DATA}_{(i)}$ be defined for each i in {1...n} recursively as follows:

$\text{DATA}_{(1)} := \text{DATA}$

If $i \in \{2...n\}$, $\text{DATA}_{(i)}$ is the recurrence with mould

$\text{DATA\_M}_{(i-1)} ® \text{R\_DP}_{(i)} \parallel \text{PIPE\_M}_{(i)}$

over base BASE

where $\text{DATA\_M}_{(i-1)}$ is such that

$\text{DATA}_{(i-1)} = \parallel_{p \in \text{BASE}} \text{DATA\_M}_{(i-1)} ® \text{R\_DATA}_{(i-1\,:\,p)}$

where

$\text{R\_DATA}_{(i-1\,:\,p)}(\langle vc, fun\rangle) = \langle vc, fun(p)\rangle$

for all $\langle vc, fun\rangle$ in $\text{Vars}(\text{DATA\_M}_{(i-1)})$

and $\text{PIPE\_M}_{(i)}$ is defined to be s.t.

$\text{In}(\text{PIPE\_M}_{(i)}) = \{\langle c_i, \text{Id}_{\text{BASE}}\rangle, \langle z_i, p \rightarrow p+r_i\rangle, \langle a_i, \text{Id}_{\text{BASE}}\rangle\}$

$\text{Out}(\text{PIPE\_M}_{(i)}) = \{\langle z_i, \text{Id}_{\text{BASE}}\rangle\}$

$\text{Rel}(\text{PIPE\_M}_{(i)})v \Leftrightarrow$

$$\begin{aligned} v(<z_i, Id_{BASE}>) &= v(<c_i, Id_{BASE}>)*v(<z_i, p \rightarrow p+r_i>) \\ &\quad + \bar{v}(<c_i, Id_{BASE}>)*v(<a_i, Id_{BASE}>) \end{aligned}$$

$$\begin{aligned} Varset_{(i)} := &\{<z_i, p+r_i> : p \in BASE\} \cup \{<z_i, p> : p \in BASE\} \\ &\cup \{<c_i, p> : p \in BASE\} \\ &\cup \{<a_i, p> : p \in BASE \text{ and } <a_i, p> \notin In(DATA_{(i-1)})\} \\ &\qquad \text{when } 1 < i \leq n \end{aligned}$$

$$\begin{aligned} R_{(i)} := &\ Id_{Vars((CONTROL\searrow(i) \parallel DATA\searrow(i)) \setminus Varset\searrow(i))} \\ &\qquad \text{when } 1 < i \leq n \end{aligned}$$

For i in {2...n}, $CONTROL_{(i)}$ is defined to be s.t.

$In(CONTROL_{(i)}) = \emptyset$

$Out(CONTROL_{(i)}) = \{<c, p> : p \in BASE\}$

$Rel(CONTROL_{(i)})v \Leftrightarrow$

For all p in BASE, $(v(<c_i, p>) = 1 \Leftrightarrow p \neq \Delta_i(p))$ and $(v(<c_i, p>) = 0 \Leftrightarrow p = \Delta_i(p))$

$$CONTROL' := \parallel_{i \in \{1...n\}} CONTROL_{(n)}$$

$$DATA' := DATA_{(n)}$$

$$BASE_{(QR)} := \{ \begin{bmatrix} i \\ j \\ k \end{bmatrix} \mid k \in \{1...M-1\}, j \in \{k...M\} \text{ and } i \in \{k+1...M\}\}$$

$$A' := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Assumptions

For all i, there exists r_i s.t. for all p in BASE, there exist S and N s.t.

$\{s : s = \Delta_i(p) - M*r_i \text{ where } m \in \text{Integer and } 0 \leq m \leq N\} \quad = \quad \text{Coset}_i(p)$

where

$\text{Coset}_i(p) \quad = \quad \{s : s \in \text{Base and } \Delta_i(s) = \Delta_i(p)\}$ (iii)

This assumption is used on page 214 in the proof of Theorem 2.

The following are well-defined

$\text{DATA\_M}_{(i)}$ for i in {1...n}

$\text{DATA}_{(i)}$ for i in {1...n}

$\text{CONTROL}_{(i)}$ for i in {1...n}

$\text{CONTROL}_{(i)} \parallel \text{DATA}_{(i)}$ for i in {1...n}

$\parallel_{j \in \{1...i\}}\text{CONTROL}_{(j)}$ when $1 < i \leq n$

$\text{Varset}_{(i)} \cap \text{Vars}(\parallel_{j \in \{1...i-1\}}\text{CONTROL}_{(j)}) = \emptyset$ when $1 < i \leq n$

$(\parallel_{j \in \{1...i-1\}}\text{CONTROL}_{(j)}) \parallel (\text{CONTROL}_{(i)} \parallel \text{DATA}_{(i)})$,

$((\parallel_{j \in \{1...i-1\}}\text{CONTROL}_{(j)}) \parallel (\text{CONTROL}_{(i)} \parallel \text{DATA}_{(i)}))\backslash\text{Varset}_{(i)}$,

$(\text{CONTROL}_{(i)} \parallel \text{DATA}_{(i)})\backslash\text{Varset}_{(i)}$,

$(\parallel_{j \in \{1...i-1\}}\text{CONTROL}_{(j)}) \parallel ((\text{CONTROL}_{(i)} \parallel \text{DATA}_{(i)})\backslash\text{Varset}_{(i)})$

when $1 < i \leq n$

$(\parallel_{j \in \{1...i\}}\text{CONTROL}_{(j)}) \parallel \text{DATA}_{(i)})$ when $1 < i \leq n$

Lemma 28

If C_1' simulates C_1 w.r.t. <Varset, R>

and $R = \mathrm{Id}_{Vars(C_1\backslash Varset)}$

and $Vars(B) \cap Varset = \emptyset$

and $C_1' \parallel C_2$ is well-defined

and $C_1 \parallel C_2$ is well-defined

and $(C_1' \parallel C_2)\backslash Varset$ is well-defined

and $(C_1\backslash Varset) \parallel C_2$ is well-defined

then $C_1' \parallel C_2$ simulates $C_1 \parallel C_2$

Proof

Since

$R = \mathrm{Id}_{Vars(C_1\backslash Varset)}$

obviously

$R|_{Vars(C_2) \cap Vars(C_1\backslash Varset)} \subseteq \mathrm{Id}_{Vars(C_2)}$ and the hypotheses are satisfied for Lemma 22 with $R_1 = R$, and R_2 defined appropriately.

Theorem 1

Let n be a positive integer; if, for all i s.t. $1 < i \leq n$, $CONTROL_{(i)} \parallel DATA_{(i)}$ simulates $DATA_{(i-1)}$ w.r.t. <$Varset_{(i)}$, $R_{(i)}$> then

CONTROL' $\parallel$ DATA' simulates CONTROL $\parallel$ DATA

Proof

...by induction on n

Base case

The theorem is trivially true when n=1.

Inductive case

Assume the theorem is true when n = N-1, and assume the hypotheses of the theorem for n = N.

From the hypotheses of the theorem for n = N, we have that $\mathrm{CONTROL}_{(N)} \parallel \mathrm{DATA}_{(N)}$ is well-defined and simulates $\mathrm{DATA}_{(N-1)}$ w.r.t. $<\mathrm{Varset}_{(N)}, R_{(N)}>$; so, by Lemma 28,

$$(\parallel_{i \in \{1...N-1\}}\mathrm{CONTROL}_{(i)}) \parallel (\mathrm{CONTROL}_{(N)} \parallel \mathrm{DATA}_{(N)})$$

simulates

$$(\parallel_{i \in \{1...N-1\}}\mathrm{CONTROL}_{(i)}) \parallel \mathrm{DATA}_{(N-1)}$$

by the fact that these two computations are well-defined,

$$R_{(N)} = \mathrm{Id}_{\mathrm{Vars}(\mathrm{CONTROL}\searrow(N) \parallel \mathrm{DATA}\searrow(N))}$$

$$\mathrm{Varset}_{(N)} \cap \mathrm{Vars}(\parallel_{j \in \{1...N-1\}}\mathrm{CONTROL}_{(j)}) = \emptyset$$

and the other well-definedness conditions for Lemma 28 hold.

Now $(\parallel_{i \in \{1...N-1\}}\mathrm{CONTROL}_{(i)}) \parallel \mathrm{DATA}_{(N-1)}$ simulates CONTROL $\parallel$ DATA by the induction hypothesis.

So

CONTROL' $\parallel$ DATA' simulates CONTROL $\parallel$ DATA

by Lemma 27

Lemma 29

$$F\bullet(G[x \to y]) = (F\restriction_{\mathrm{Ran}(G)}\bullet G)[x \to F(y)]$$

(For a definition of the arrow notation, see Terminology (General) on page ix)

Proof

$$F\bullet(G[x \rightarrow y])x = F(G[x \rightarrow y](x)) = F(y)$$

$$F\bullet(G[x \rightarrow y])x' = F(G(x')) = (F|_{Ran(G)}\bullet G)x'$$

$$\text{if } x' \neq x \text{ and } x' \in \text{dom}(G)$$

and the domains of the two functions are obviously equal.

F needs to be restricted to Ran(G) before being composed with G, since its domain is Ran(G[x $\rightarrow$ y]), which may be a strict superset of Ran(G) if $y \notin$ Ran(G)

Lemma 30

Let the coset of p, Coset(p), be defined as follows:

$$\text{Coset}(p) = \{p' : p' \in \text{BASE and } \Delta(p') = \Delta(p)\}$$

Assume that there exists an r s.t., for all p in BASE, there exists N_p s.t.

$$\text{Coset}(p) = \{s : s = \Delta(p) - m*r, \text{ where } m \in \text{Integer and } 0 \leq m \leq N_p\}$$

Then, if Rem[oteness] is a function defined as follows:

$$\text{Rem}(p) := 0 \quad \text{if } p = \Delta(p)$$

$$\text{Rem}(p) := \text{Rem}(p+r) + 1 \quad \text{if } p \neq \Delta(p)$$

then Rem is well-defined.

Proof

Let p be in S and assume that

$$p = \Delta(p) - m*r$$

We can prove that Rem(p) is well-defined by induction on m, using the inductive hypothesis, "Rem(Δ(p) - (m−1)*r) is well-defined."

Base case

$m = 0$ so $Rem(p) = 0$

Inductive case

$m \neq 0 \Rightarrow p \neq \Delta(p)$ (if $r \neq 0$; otherwise $Rem(p) = 0$ as for base case)

so

$Rem(p) = Rem(p\text{-}r) + 1 = Rem(p_0 - (m\text{-}1)*r) + 1$ which is well defined, by the inductive hypothesis.

Lemma 31

If the hypotheses of Lemma 30 hold then

(For all p in BASE,

$$(p \neq \Delta(p) \Rightarrow v(<z, p>) = v(<z, p\text{-}r>))$$

and $$(p = \Delta(p) \Rightarrow v(<z, p>) = v(<a, p>)))$$

$\Rightarrow$ for all p in BASE, $v(<z, p>) = v(<a, \Delta(p)>)$

Proof

The proof will proceed by induction on Rem(p) (which is well-defined, by Lemma 30) using the inductive hypothesis, "For all p' s.t. Rem(p') < p, v(<z,

p'> = v<a, Δ(p')>".

Base case

Rem(p) = 0, so

$$p = \Delta(p),$$

and so

$$v(\langle z, p\rangle) = v(\langle a, p\rangle) = v(\langle a, \Delta(p)\rangle)$$

Inductive case

Rem(p) > 0, so

$$p \neq \Delta(p)$$

(assuming that $r \neq 0$; If r=0 then the same argument holds as in the base case.)

so

$$v(\langle z, p\rangle) = v(\langle z, p+r\rangle)$$

but Rem(p-r) = Rem(p) - 1, so, by the inductive hypothesis and the fact that p+r ∈ Coset(p),

$$v(\langle z, p+r\rangle) = v(\langle a, \Delta(p+r)\rangle) = v(\langle a, \Delta(p)\rangle)$$

Lemma 32

$$v'|_{In(C)} = v''|_{In(C)} \quad \text{and} \quad Rel(C)v' \quad \text{and } Rel(C)v''$$

$$\Rightarrow v'|_{Out(C)} = v''|_{Out(C)}$$

Proof

...directly from the fact that Rel corresponds to a function from valuations on In(C) to valuations on Out(C).

Lemma 33

If C_1 and C_2 are computations and $\mathrm{Vars}(C_2) \subseteq \mathrm{Vars}(C_1)$ and $\mathrm{Rel}(C_1)v' \Rightarrow \mathrm{Rel}(C_2)v'|_{\mathrm{Vars}(C{\searrow}2)}$ and $C_1\backslash\mathrm{Varset}$ is well-defined

then $\mathrm{Rel}(C_1\backslash\mathrm{Varset})v \Rightarrow \mathrm{Rel}(C_2)v$

where $\mathrm{Varset} = \mathrm{Vars}(C_1) - \mathrm{Vars}(C_2)$

Proof

Assume $\mathrm{Rel}(C_1\backslash\mathrm{Varset})v$. Now, from the definition of hiding, we know that

$$\begin{aligned}&\mathrm{Rel}(C_1\backslash\mathrm{Varset})v \Leftrightarrow\\ &\quad \text{For all } v',\ \mathrm{Rel}(C_1)v' \Rightarrow (v'|_{\mathrm{In}(C{\searrow}1\backslash\mathrm{Varset})} = v_{\mathrm{In}(C{\searrow}1\backslash\mathrm{Varset})}\\ &\qquad\qquad \Rightarrow v'|_{\mathrm{Out}(C{\searrow}1\backslash\mathrm{Varset})} = v_{\mathrm{Out}(C{\searrow}1\backslash\mathrm{Varset})})\end{aligned}$$

Let v' be constructed s.t. $v'|_{\mathrm{In}(C{\searrow}1\backslash\mathrm{Varset})} = v|_{\mathrm{In}(C{\searrow}1\backslash\mathrm{Varset})}$ and $\mathrm{Rel}(C_1)v'$ holds (we know from Lemma 5 that this can be done) then $\mathrm{Rel}(C_2)v'|_{\mathrm{Vars}(C{\searrow}2)}$ holds by hypothesis and $v'|_{\mathrm{Vars}(C{\searrow}2)} = v$ by the above equivalence and so $\mathrm{Rel}(C_2)v$ holds.

Lemma 34

If C_1 and C_2 are computations where $\mathrm{Vars}(C_2) \subseteq \mathrm{Vars}(C_1)$ and $\mathrm{In}(C_1)|_{\mathrm{Vars}(C{\searrow}2)} = \mathrm{In}(C_2)$ and $\mathrm{Out}(C_1)|_{\mathrm{Vars}(C{\searrow}2)} = \mathrm{Out}(C_2)$ and $C_1\backslash\mathrm{Varset}$ is well-defined and $\mathrm{Rel}(C_1)v' \Rightarrow \mathrm{Rel}(C_2)v'|_{\mathrm{Vars}(C{\searrow}2)}$

then

$\mathrm{Rel}(C_2)v \Rightarrow \mathrm{Rel}(C_1\backslash\mathrm{Varset})v$ where $\mathrm{Varset} = \mathrm{Vars}(C_1) - \mathrm{Vars}(C_2)$

Proof

Again, from the definition of hiding, we know that

$\mathrm{Rel}(C_1\backslash\mathrm{Varset})v \Leftrightarrow$

For all v' $\mathrm{Rel}(C_1)v' \Rightarrow (v'|_{\mathrm{In}(C_1\backslash\mathrm{Varset})} = v|_{\mathrm{In}(C_1\backslash\mathrm{Varset})}$

$\Rightarrow v'|_{\mathrm{Out}(C_1\backslash\mathrm{Varset})} = v|_{\mathrm{Out}(C_1\backslash\mathrm{Varset})})$

so

$(\mathrm{Rel}(C_2)v \Rightarrow \mathrm{Rel}(C_1\backslash\mathrm{Varset})v)$

$\Leftrightarrow$

For all v'$((\mathrm{Rel}(C_2)v$ and $\mathrm{Rel}(C_1)v'$ and $v'|_{\mathrm{In}(C_1\backslash\mathrm{Varset})} = v|_{\mathrm{In}(C_1\backslash\mathrm{Varset})})$

$\Rightarrow v'|_{\mathrm{Out}(C_1\backslash\mathrm{Varset})} = v|_{\mathrm{Out}(C_1\backslash\mathrm{Varset})})$

To prove it is therefore sufficient to prove the R.H.S. Now since

$\mathrm{Rel}(C_1)v' \Rightarrow \mathrm{Rel}(C_2)v'|_{\mathrm{Vars}(C_2)}$ by hypothesis

and

$\mathrm{Vars}(C_2) = \mathrm{Vars}(C_1\backslash\mathrm{Varset})$

we have by Lemma 32 with C_2 substituted for C and v for v'' that

$v'|_{\mathrm{Out}(C_1\backslash\mathrm{Varset})} = v|_{\mathrm{Out}(C_1\backslash\mathrm{Varset})}$

Here we have used the fact that

$\mathrm{In}(C_1\backslash\mathrm{Varset}) = \mathrm{In}(C_1)|_{\mathrm{Vars}(C_2)} = \mathrm{In}(C_2)$

and

$\mathrm{Out}(C_1\backslash\mathrm{Varset}) = \mathrm{Out}(C_1)|_{\mathrm{Vars}(C_2)} = \mathrm{Out}(C_2)$

Theorem 2

$\mathrm{CONTROL}_{(i)} \| \mathrm{DATA}_{(i)}$ simulates $\mathrm{DATA}_{(i-1)}$ w.r.t. $<\mathrm{Varset}_{(i)}, R_{(i)}>$ for all i s.t. $1 < i \leq n$

Proof

An overview of the proof

Firstly $DATA_{(i)}$ is examined and much rewriting of $Out(DATA_{(i)})$ and $In(DATA_{(i)})$ is done to obtain useful expressions for these sets. Then $Rel(DATA_{(i)})$ is rewritten and expanded. Using this work, expressions are obtained for $Out(CONTROL_{(i)} \parallel DATA_{(i)})$, $In(CONTROL_{(i)} \parallel DATA_{(i)})$ and $Rel(CONTROL_{(i)} \parallel DATA_{(i)})$. Using these expressions, the statements (iv), (v) and (vi) (see page 213) are proven which are together equivalent to Theorem 2. This is the core of the proof. The proofs of (iv) and (v) are relatively easy, but proof of (vi) is more difficult. It is eased by the use of an intermediate result, (vii), which can be used for proving both that the L.H.S. implies the R.H.S. and vice versa.

Expressions for Out(DATA$\searrow$(i)), In(DATA$\searrow$(i)) and Rel(DATA$\searrow$(i))

Let us expand $DATA_{(i)}$:

$$DATA_{(i)} =$$
$$\parallel_{p\in BASE} \quad (DATA\_M_{(i-1)} ® R\_DP_{(i)} \parallel PIPE\_M_{(i)}) ® R\_DATA_{(i\,:\,p)}$$

where $R\_DATA_{(i\,:\,p)}(<vc, fun>) \quad = \quad <vc, fun(p)>$ for all $<vc, fun>$ in $Vars(DATA\_M_{(i-1)} ® R\_DP_{(i)} \parallel PIPE\_M_{(i)})$ (by definition)

$$Out(DATA_{(i)})$$
$$= \bigcup_{p\in BASE} Out((DATA\_M_{(i-1)} ® R\_DP_{(i)} \parallel PIPE\_M_{(i)}) ® R\_DATA_{(i\,:\,p)})$$
$$= \bigcup_{p\in BASE} Out(DATA\_M_{(i-1)} ® R\_DP_{(i)} ® R\_DATA_{(i\,:\,p)}|_{Vars(DATA\_M\searrow(i-1) ® R\_DP\searrow(i))}$$
$$\parallel PIPE\_M_{(i)} ® R\_DATA_{(i\,:\,p)}|_{Vars(PIPE\_M(i))})$$

by Lemma 21 on page 182

$= \bigcup_{p \in BASE} Out(DATA\_M_{(i-1)}$ ® $(R\_DATA_{(i\ :\ p)}|_{Vars(DATA\_M\searrow(i-1)}$ ® $_{R\_DP\searrow(i))} \bullet R\_DP_{(i)})$

$\| \; PIPE\_M_{(i)}$ ® $R\_DATA_{(i\,:\,p)}|_{Vars(PIPE\_M(i))})$

by Lemma 26 on page 193 applied to the expression on the L.H.S. of the "||"

Let us define "f" to be $R\_DATA_{(i\,:\,p)}|_{Vars(DATA\_M\searrow(i-1)\text{ ® } R\_DP\searrow(i))} \bullet R\_DP_{(i)}$; then

$$dom(f) \quad = \quad dom(R\_DP_{(i)}) = \quad Vars(DATA\_M_{(i-1)})$$

Now

$f \quad = \quad R\_DATA_{(i\,:\,p)}|_{Vars(DATA\_M\searrow(i-1)\text{ ® } R\_DP\searrow(i))}$

$\bullet(Id|_{Vars(DATA\_M\searrow(i-1))}[<a_i, \Delta_i> \rightarrow <z_i, Id_D>])$

from definition of $R\_DP_{(i)}$ on page 197

$\quad = \quad R\_DATA_{(i\,:\,p)}|_{Vars(DATA\_M\searrow(i-1))}[<a_i, \Delta_i> \rightarrow <z_i, p>]$

by Lemma 29 on page 201

and definition of $R\_DATA_{(i\,:\,p)}$

So

$\bigcup_{p \in BASE} Out(DATA\_M_{(i-1)}$ ® $(R\_DATA_{(i\ :\ p)}|_{Vars(DATA\_M\searrow(i-1)}$ ® $_{R\_DP\searrow(i))} \bullet R\_DP_{(i)})$

$\| \; PIPE\_M_{(i)}$ ® $R\_DATA_{(i\,:\,p)}|_{Vars(PIPE\_M(i))})$

$= \bigcup_{p \in BASE} Out((DATA\_M_{(i-1)}$ ® $(R\_DATA_{(i\,:\,p)}|_{Vars(DATA\_M\searrow(i-1))}$

$[<a_i, \Delta_i> \rightarrow <z_i, p>])|_{Out(DATA\_M\searrow(i-1))})$

$\| \quad PIPE\_M_{(i)}$ ® $(R\_DATA_{(i\,:\,p)}|_{Out(PIPE\_M\searrow(i))})$

$$= \bigcup_{p \in BASE} ran((R\_DATA_{(i:p)}|_{Vars(DATA\_M\searrow(i-1))}$$

$$[<a_i, \Delta_i> \rightarrow <z_i, p>])|_{Out(DATA\_M\searrow(i-1))})$$

$$\cup \bigcup_{p \in BASE} ran(R\_DATA_{(i:p)}|_{Vars(PIPE\_M\searrow(i))}|_{Out(PIPE\_M\searrow(i))})$$

by definition of renaming and composition

$$= \bigcup_{p \in BASE} ran((R\_DATA_{(i:p)}|_{Vars(DATA\_M\searrow(i-1))}$$

$$[<a_i, \Delta_i> \rightarrow <z_i, p>])|_{Out(DATA\_M\searrow(i-1))})$$

$$\cup \{<z_i, p> : p \in BASE\}$$

by definition of $PIPE\_M_{(i)}$ on page 197

Let us now consider $In(DATA_{(i)})$...

$In(DATA_{(i)})$

$$= \bigcup_{p \in BASE} In(DATA\_M_{(i-1)} \circledR (R\_DATA_{(i\ :\ p)}|_{Vars(DATA\_M\searrow(i-1)} \circledR R\_DP\searrow(i))^{\bullet}R\_DP_{(i)})$$

$$\| \; PIPE\_M_{(i)} \circledR R\_DATA_{(i:p)}|_{Vars(PIPE\_M(i))})$$

$$- \; Out(DATA_{(i)})$$

by definition of composition

$$= \bigcup_{p \in BASE} (In(DATA\_M_{(i-1)} \circledR (R\_DATA_{(i:p)}|_{Vars(DATA\_M\searrow(i-1))}$$

$$[<a_i, \Delta_i> \rightarrow <z_i, p>])) \; \| \; PIPE\_M_{(i)} \circledR R\_DATA_{(i\ :\ p)}|_{Vars(PIPE\_M(i))})$$

$$- \; Out(DATA_{(i)})$$

by rewriting f as on page 208

$$= \bigcup_{p \in BASE} (\; (In(DATA\_M_{(i-1)} \circledR (R\_DATA_{(i:p)}|_{Vars(DATA\_M\searrow(i-1))}$$

$$[<a_i, \Delta_i> \rightarrow <z_i, p>]))$$

$\cup \quad In(PIPE\_M_{(i)} ® R\_DATA_{(i:p)}|_{Vars(PIPE\_M(i))}))$

$- (Out(DATA\_M_{(i-1)} ® (R\_DATA_{(i:p)}|_{Vars(DATA\_M\searrow(i-1))}$

$[<a_i, \Delta_i> \rightarrow <z_i, p>]))$

$\cup \quad Out(PIPE\_M_{(i)} ® R\_DATA_{(i:p)}|_{Vars(PIPE\_M(i))})))$

$- Out(DATA_{(i)})$

by definition of composition

$= \bigcup_{p \in BASE} ((In(DATA\_M_{(i-1)} ® (R\_DATA_{(i:p)}|_{Vars(DATA\_M\searrow(i-1))}$

$[<a_i, \Delta_i> \rightarrow <z_i, p>]))$

$\cup \quad In(PIPE\_M_{(i)} ® R\_DATA_{(i:p)}|_{Vars(PIPE\_M(i))}))$

$- Out(DATA_{(i)})$

by repeated application of Lemma 12, Lemma 13 and Lemma 14 on page 179

$= \bigcup_{p \in BASE} ran((R\_DATA_{(i:p)}|_{Vars(DATA\_M\searrow(i-1))}$

$[<a_i, \Delta_i> \rightarrow <z_i, p>])|_{In(DATA\_M\searrow(i-1))})$

$\cup \bigcup_{p \in BASE} ran(R\_DATA_{(i:p)}|_{Vars(PIPE\_M\searrow(i))}|_{In(PIPE\_M\searrow(i))})$

$- Out(DATA_{(i)})$

by definition of renaming, Lemma 26, and by rewriting f as on page 208

$= \bigcup_{p \in BASE} ran((R\_DATA_{(i:p)}|_{Vars(DATA\_M\searrow(i-1))}$

$[<a_i, \Delta_i> \rightarrow <z_i, p>])|_{In(DATA\_M\searrow(i-1))})$

$\cup \bigcup_{p \in BASE} \{<c_i, p>, <z_i, p+r_i>, <a_i, p>\}$

$- (\bigcup_{p \in BASE} ran((R\_DATA_{(i:p)}|_{Vars(DATA\_M\searrow(i-1))}$

$[<a_i, \Delta_i> \rightarrow <z_i, p>])|_{Out(DATA\_M\searrow(i-1))})$

$\cup \{<z_i, p> : p \in BASE\})$

Now for $\mathrm{Rel}(\mathrm{DATA}_{(i)})$

$\mathrm{Rel}(\mathrm{DATA}_{(i)})v \quad \Leftrightarrow$

$\Leftrightarrow \mathrm{Rel}(\|_{p \in \mathrm{BASE}}(\mathrm{DATA\_M}_{(i-1)} ® \mathrm{R\_DP}_{(i)} \| \mathrm{PIPE\_M}_{(i)}) ® \mathrm{R\_DATA}_{(i : p)})v$

$\Leftrightarrow \mathrm{Rel}(\|_{p \in \mathrm{BASE}}(\mathrm{DATA\_M}_{(i-1)} ® (\mathrm{R\_DATA}_{(i : p)}|_{\mathrm{Vars}(\mathrm{DATA\_M}\searrow(i-1))}$
$[<a_i, \Delta_i> \rightarrow <z_i, p>])$
$\| (\mathrm{PIPE\_M}_{(i)} ® \mathrm{R\_DATA}_{(i : p)}|_{\mathrm{Vars}(\mathrm{PIPE\_M}(i))}))v$

$\Leftrightarrow$ For all p in BASE,
$(\mathrm{Rel}(\mathrm{DATA\_M}_{(i-1)} ® (\mathrm{R\_DATA}_{(i : p)}|_{\mathrm{Vars}(\mathrm{DATA\_M}\searrow(i-1))}$
$[<a_i, \Delta_i> \rightarrow <z_i, p>])))$
$v|_{\mathrm{Vars}(\mathrm{DATA\_M}\searrow(i-1) ® (\mathrm{R\_DATA}\searrow(i : p))/\mathrm{Vars}(\mathrm{DATA\_M}\searrow(i-1))[<a\searrow i, \Delta\searrow i> \rightarrow <z\searrow i, p>]))}$
and
$(\mathrm{Rel}(\mathrm{PIPE\_M}_{(i)} ® (\mathrm{R\_DATA}_{(i : p)}|_{\mathrm{Vars}(\mathrm{PIPE\_M}\searrow(i))})))$
$v|_{\mathrm{Vars}(\mathrm{PIPE\_M}\searrow(i) ® (\mathrm{R\_DATA}\searrow(i : p))/\mathrm{Vars}(\mathrm{PIPE\_M}\searrow(i))))}$

by definition of composition and the definition of the variables of a renamed computation

$\Leftrightarrow$ For all p in BASE,
$(\mathrm{Rel}(\mathrm{DATA\_M}_{(i-1)} ® (\mathrm{R\_DATA}_{(i : p)}|_{\mathrm{Vars}(\mathrm{DATA\_M}\searrow(i-1))}$
$[<a_i, \Delta_i> \rightarrow <z_i, p>])))$
$v|_{\mathrm{Vars}(\mathrm{DATA\_M}\searrow(i-1) ® (\mathrm{R\_DATA}\searrow(i : p))/\mathrm{Vars}(\mathrm{DATA\_M}\searrow(i-1))[<a\searrow i, \Delta\searrow i> \rightarrow <z\searrow i, p>]))}$
and
$v(<z_i, p>) = v(<c_i, p>) * v(<z_i, p+r_i>) + \bar{v}(<c_i, p>) * v(<a_i, p>)$
)

by definition of $\mathrm{PIPE\_M}_{(i)}$

<u>Expressions for In(CONTROL↘(i) || DATA↘(i)).</u>

<u>$\mathrm{Out}(\mathrm{CONTROL}\searrow(i) \parallel \mathrm{DATA}\searrow(i))$ and $\mathrm{Rel}(\mathrm{CONTROL}\searrow(i) \parallel \mathrm{DATA}\searrow(i))$</u>

Let us now expand $\mathrm{CONTROL}_{(i)}$.

$\mathrm{In}(\mathrm{CONTROL}_{(i)}) = \emptyset$

$\mathrm{Out}(\mathrm{CONTROL}_{(i)}) = \{<c_i, p> : p \in \mathrm{BASE}\}$

$\mathrm{Rel}(\mathrm{CONTROL}_{(i)})v \Leftrightarrow$ For all p in BASE, $(v(<c_i, p>) = 1 \Leftrightarrow p \neq \Delta_i(p))$ and $(v(<c_i, p>) = 0 \Leftrightarrow p = \Delta_i(p))$

$\mathrm{In}(\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)}) = \mathrm{In}(\mathrm{DATA}_{(i)}) - \{<c_i, p> : p \in \mathrm{BASE}\}$

by definition of composition and $\mathrm{CONTROL}_{(i)}$

$$= \bigcup_{p \in \mathrm{BASE}} \mathrm{ran}(\mathrm{R\_DATA}_{(i:p)}|_{\mathrm{In}(\mathrm{DATA\_M}\searrow(i-1))}[<a_i, \Delta_i> \rightarrow <z_i, p>])$$

$$\cup \bigcup_{p \in \mathrm{BASE}} \{<z_i, p+r_i>, <a_i, p>\}$$

$$- (\bigcup_{p \in \mathrm{BASE}} \mathrm{ran}(\mathrm{R\_DATA}_{(i:p)}|_{\mathrm{Out}(\mathrm{DATA\_M}\searrow(i-1))})$$

$$\cup \{<z_i, p> : p \in \mathrm{BASE}\})$$

rewriting $\mathrm{In}(\mathrm{DATA}_{(i)})$ and simplifying, using the fact that $<a_i, \Delta_i> \in \mathrm{In}(\mathrm{DATA\_M}_{(i-1)})$ and $<a_i, \Delta_i> \notin \mathrm{Out}(\mathrm{DATA\_M}_{(i-1)})$

$\mathrm{Out}(\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)}) =$

$\mathrm{Out}(\mathrm{DATA}_{(i)}) \cup \mathrm{Out}(\mathrm{CONTROL}_{(i)})$

$$= \bigcup_{p \in \mathrm{BASE}} \mathrm{ran}(\mathrm{R\_DATA}_{(i:p)}|_{\mathrm{Out}(\mathrm{DATA\_M}\searrow(i-1))})$$

$$\cup \{<z_i, p> : p \in \mathrm{BASE}\}$$

$$\cup \{<c_i, p> : p \in \mathrm{BASE}\}$$

rewriting $\mathrm{Out}(\mathrm{DATA}_{(i)})$, $\mathrm{Out}(\mathrm{CONTROL}_{(i)})$ and simplifying

$\mathrm{Rel}(\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)}) \Leftrightarrow$

For all p in BASE,

$(\mathrm{Rel}(\mathrm{DATA\_M}_{(i-1)} \circledR (\mathrm{R\_DATA}_{(i:p)}|_{\mathrm{Vars}(\mathrm{DATA\_M}\searrow(i-1))}$

$[<a_i, \Delta_i> \rightarrow <z_i, p>])))$

$v|_{\mathrm{Vars}(\mathrm{DATA\_M}\searrow(i-1)\ \circledR\ (\mathrm{R\_DATA}\searrow(i\ :\ p))/\mathrm{Vars}(\mathrm{DATA\_M}\searrow(i-1))[<a\searrow i,\ \Delta\searrow i>}$
$\rightarrow\ _{<z\searrow i,\ p>]))}$

and

$v(<z_i, p>) = v(<c_i, p>) * v(<z_i, p+r_i>) + \overline{v}(<c_i, p>) * v(<a_i, p>)$

)

and, for all p in BASE,

$(v(<c_i, p>) = 1 \Leftrightarrow p \neq \Delta_i(p))$ and $(v(<c_i, p>) = 0 \Leftrightarrow p = \Delta_i(p))$

using rewriting of $\mathrm{Rel}(\mathrm{DATA}_{(i)})$ and the definition of $\mathrm{Rel}(\mathrm{CONTROL}_{(i)})$

<u>The core of the proof</u>

It is necessary and sufficient to show that

$$\mathrm{Out}(((\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)})\backslash\mathrm{Varset}_{(i)}) \circledR R_{(i)}) = \mathrm{Out}(\mathrm{DATA}_{(i-1)}) \quad \text{(iv)}$$

$$\mathrm{In}(((\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)})\backslash\mathrm{Varset}_{(i)}) \circledR R_{(i)}) = \mathrm{In}(\mathrm{DATA}_{(i-1)}) \quad \text{(v)}$$

$$\mathrm{Rel}(((\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)})\backslash\mathrm{Varset}_{(i)}) \circledR R_{(i)}) \Leftrightarrow \mathrm{Rel}(\mathrm{DATA}_{(i-1)}) \quad \text{(vi)}$$

<u>Proof of (iv)</u>

$\mathrm{Out}(((\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)})\backslash\mathrm{Varset}_{(i)}) \circledR R_{(i)})$

$= \mathrm{Out}((\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)})\backslash\mathrm{Varset}_{(i)})$

$= \bigcup_{p \in \mathrm{BASE}} \mathrm{ran}(\mathrm{R\_DATA}_{(i:p)}|_{\mathrm{Out}(\mathrm{DATA\_M}\searrow(i-1))})$

$\cup \{<z_i, p> : p \in \mathrm{BASE}\}$

$\cup \{<c_i, p> : p \in \mathrm{BASE}\}$

$$- \ \mathrm{Varset}_{(i)}$$

$$= \ \mathrm{Out}(\mathrm{DATA}_{(i-1)})$$

Proof of (v)

$$\mathrm{In}(((\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)}) \backslash \mathrm{Varset}_{(i)}) \circledR R_{(i)})$$

$$= \ \mathrm{In}((\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)}) \backslash \mathrm{Varset}_{(i)})$$

trivially from the definition of $R_{(i)}$ on page 198

$$= \ \bigcup_{p \in \mathrm{BASE}} \mathrm{ran}(\mathrm{R\_DATA}_{(i:p)}|_{\mathrm{In}(\mathrm{DATA\_M}\searrow(i-1))}[\langle a_i, \Delta_i\rangle \to \langle z_i, p\rangle])$$

$$\cup \ \bigcup_{p \in \mathrm{BASE}} \{\langle z_i, p+r_i\rangle, \langle a_i, p\rangle\}$$

$$- \ (\bigcup_{p \in \mathrm{BASE}} \mathrm{ran}(\mathrm{R\_DATA}_{(i:p)}|_{\mathrm{Out}(\mathrm{DATA\_M}\searrow(i-1))})$$

$$\cup \ \{\langle z_i, p\rangle : \ p \in \mathrm{BASE}\})$$

$$- \ \mathrm{Varset}_{(i)}$$

from definition of $\mathrm{Varset}_{(i)}$ on page 198

$$= \ \bigcup_{p \in \mathrm{BASE}} \mathrm{ran}(\mathrm{R\_DATA}_{(i:p)}|_{\mathrm{In}(\mathrm{DATA\_M}\searrow(i-1)) - \{\langle a\searrow i, \Delta\searrow i\rangle\}})$$

$$\cup \ \bigcup_{p \in \mathrm{BASE}} \{\langle a_i, p\rangle\}$$

$$- \ \bigcup_{p \in \mathrm{BASE}} \mathrm{ran}(\mathrm{R\_DATA}_{(i:p)}|_{\mathrm{Out}(\mathrm{DATA\_M}\searrow(i-1))})$$

$$- \ \{\langle a_i, p\rangle : p \in \mathrm{BASE} \text{ and } \langle a_i, p\rangle \notin \mathrm{In}(\mathrm{DATA}_{(i-1)})\}$$

From (iii) on page 199, we may deduce that $\Delta_i(p) \in \mathrm{BASE}$ for all p in BASE, so $\{\langle a_i, \Delta_i(p)\rangle : p \in \mathrm{BASE}\} \subseteq \bigcup_{p \in \mathrm{BASE}} \{\langle a_i, p\rangle\}$; therefore we know that the above expression equals

$$\bigcup_{p \in \mathrm{BASE}} \mathrm{ran}(\mathrm{R\_DATA}_{(i:p)}|_{\mathrm{In}(\mathrm{DATA\_M}\searrow(i-1))})$$

$$\cup \ \bigcup_{p \in \mathrm{BASE}} \{\langle a_i, p\rangle\}$$

$$- \bigcup_{p \in BASE} ran(R\_DATA_{(i\,:\,p)}|_{Out(DATA\_M\searrow(i-1))})$$

$$- \{<a_i, p> : p \in BASE \text{ and } <a_i, p> \notin In(DATA_{(i-1)})\}$$

$$= \bigcup_{p \in BASE} ran(R\_DATA_{(i\,:\,p)}|_{In(DATA\_M\searrow(i-1))})$$

$$- \bigcup_{p \in BASE} ran(R\_DATA_{(i\,:\,p)}|_{Out(DATA\_M\searrow(i-1))})$$

by Lemma 15 on page 179 with A equal to

$$\bigcup_{p \in BASE} ran(R\_DATA_{(i\,:\,p)}|_{In(DATA\_M\searrow(i-1))})$$

and B equal to $\bigcup_{p \in BASE} \{<a_i, p>\}$, since A - B

will then be

$$\{<a_i, p> : p \in BASE \text{ and } <a_i, p> \notin In(DATA_{(i-1)})\}$$

$$= In(DATA_{(i-1)})$$

<u>Proof of (vi)</u>

$Rel(((CONTROL_{(i)} \parallel DATA_{(i)})\backslash Varset_{(i)}) \circledR R_{(i)})$

$\Leftrightarrow Rel((CONTROL_{(i)} \parallel DATA_{(i)})\backslash Varset_{(i)})\, v$

$\Leftrightarrow$ For all v',

$Rel(CONTROL_{(i)} \parallel DATA_{(i)})v' \Rightarrow$

$(v'|_{In((CONTROL\searrow(i) \parallel DATA\searrow(i))\backslash Varset\searrow(i))}$

$= v|_{In((CONTROL\searrow(i) \parallel DATA\searrow(i))\backslash Varset\searrow(i))}$

$\Rightarrow v'|_{Out((CONTROL\searrow(i) \parallel DATA\searrow(i))\backslash Varset\searrow(i))}$

$= v|_{Out((CONTROL\searrow(i) \parallel DATA\searrow(i))\backslash Varset\searrow(i))})$

by definition of hiding

We want to show that this is equivalent to $Rel(DATA_{(i-1)})v$. Now

$Rel(DATA_{(i-1)})v \Leftrightarrow$

For all p, $Rel(DATA\_M_{(i-1)} \circledR R\_DATA_{(i-1\,:\,p)})v|_{Vars(DATA\_M\searrow(i-1) \circledR R\_DATA\searrow(p))}$

$\Leftrightarrow$ For all p, $\text{Rel}(\text{DATA\_M}_{(i-1)})(v|_{\text{Vars}(\text{DATA\_M}\searrow(i-1)} ® \text{R\_DATA}\searrow(p))^{\bullet}\text{R\_DATA}_{(i-1\,:\,p)}$

We will divide the proof of (vi) into "$\Rightarrow$" and "$\Leftarrow$", but first we will prove

$\text{Rel}(\text{CONTROL}_{(i)} \,||\, \text{DATA}_{(i)})v' \Rightarrow \text{Rel}(\text{DATA}_{(i-1)})v'|_{\text{Vars}(\text{DATA}\searrow(i-1))}$ (vii)

<u>Proof of (vii)</u>

Now

$\text{Rel}(\text{CONTROL}_{(i)} \,||\, \text{DATA}_{(i)})v \Leftrightarrow$

For all p in BASE,

$(\text{Rel}(\text{DATA\_M}_{(i-1)}$ ® $(\text{R\_DATA}_{(i\,:\,p)}|_{\text{Vars}(\text{DATA\_M}\searrow(i-1))}$

$[<a_i, \Delta_i> \rightarrow <z_i, p>])))$

$v|_{\text{Vars}(\text{DATA\_M}\searrow(i-1)}$ ® $(\text{R\_DATA}\searrow(i\,:\,p))|_{\text{Vars}(\text{DATA\_M}\searrow(i-1))}[<a\searrow i, \Delta\searrow i> \rightarrow <z\searrow i, p>]))$

and

$v(<z_i, p>) = v(<c_i, p>) * v(<z_i, p+r_i>) + \bar{v}(<c_i, p>) * v(<a_i, p>)$

)

and, for all p in BASE,

$(v(<c_i, p>) = 1 \Leftrightarrow p \neq \Delta_i(p))$ and $(v(<c_i, p>) = 0 \Leftrightarrow p = \Delta_i(p))$

from previous work

The last two subclauses of the R.H.S. imply that

for all p in BASE,

$(p \neq \Delta_i(p) \Rightarrow v(<z_i, p>) = v(<z_i, p+r_i>))$

and $(p = \Delta_i(p) \Rightarrow v(<z_i, p>) = v(<a_i, p>))$

which implies that, for all p in BASE, $v(<z_i, p>) = v(<a_i, \Delta_i(p)>)$

by Lemma 31 and (iii) on page 199

So L.H.S. of (vi) $\Rightarrow$

(For all p in BASE,

$(\mathrm{Rel}(\mathrm{DATA\_M}_{(i-1)}) \circledR (\mathrm{R\_DATA}_{(i\,:\,p)}|_{\mathrm{Vars(DATA\_M}\searrow(i-1))}[<a_i, \Delta_i> \rightarrow <z_i, p>])))$

$v'|_{\mathrm{Vars(DATA\_M}\searrow(i-1)\ \circledR\ (\mathrm{R\_DATA}\searrow(i\,:\,p))/\mathrm{Vars(DATA\_M}\searrow(i-1))[<a\searrow i,\ \Delta\searrow i> \rightarrow\ <z\searrow i,\ p>]))}$

and, for all p in BASE,

$v'(<z_i, p>) = v'(<a_i, \Delta_i(p)>)$

)

$\Rightarrow$

(For all p in BASE,

$(\mathrm{Rel}(\mathrm{DATA\_M}_{(i-1)})$

$(v'|_{\mathrm{Vars(DATA\_M}\searrow(i-1)\ \circledR\ (\mathrm{R\_DATA}\searrow(i\,:\,p))/\mathrm{Vars(DATA\_M}\searrow(i-1))[<a\searrow i,\ \Delta\searrow i> \rightarrow\ <z\searrow i,\ p>]))})^{\bullet}(\mathrm{R\_DATA}_{(i\,:\,p)}|_{\mathrm{Vars(DATA\_M}\searrow(i-1))}[<a_i, \Delta_i> \rightarrow <z_i, p>])$

and, for all p in BASE,

$v'(<z_i, p>) = v'(<a_i, \Delta_i(p)>)$

)

by definition of renaming

$\Rightarrow$

(For all p in BASE,

$(\mathrm{Rel}(\mathrm{DATA\_M}_{(i-1)})$

$(v'|_{\mathrm{Vars(DATA\_M}\searrow(i-1)\ \circledR\ (\mathrm{R\_DATA}\searrow(i\,:\,p))/\mathrm{Vars(DATA\_M}\searrow(i-1))))})$

${}^{\bullet}(\mathrm{R\_DATA}_{(i\,:\,p)}|_{\mathrm{Vars(DATA\_M}\searrow(i-1))})$

)

using the fact that for all p in BASE,

$v'(<z_i, p>) = v'(<a_i, \Delta_i(p)>)$

$\Leftrightarrow$ $\mathrm{Rel}(\mathrm{DATA}_{(i-1)})v'|_{\mathrm{Vars}(\mathrm{DATA}\searrow(i-1))}$

by definition of $\mathrm{DATA}_{(i-1)}$ on page 197

and definition of re-naming

$\Rightarrow$

...follows from (vii), the fact that

$\mathrm{Vars}(\mathrm{DATA}_{(i-1)}) \subseteq \mathrm{Vars}(\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)})$

and

Lemma 33 on page 205

$\Leftarrow$

...follows directly from (vii) and the fact that

$\mathrm{Vars}(\mathrm{DATA}_{(i-1)}) \subseteq \mathrm{Vars}(\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)})$

and

$\mathrm{In}(\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)})|_{\mathrm{Vars}(\mathrm{DATA}\searrow(i-1))} = \mathrm{In}(\mathrm{DATA}_{(i-1)})$

and

$\mathrm{Out}(\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)})|_{\mathrm{Vars}(\mathrm{DATA}\searrow(i-1))} = \mathrm{Out}(\mathrm{DATA}_{(i-1)})$

and

Lemma 34 on page 205

Theorem 3

There is no other way to pipeline the dependencies < ox, p $\longrightarrow$ A'.p> and <oy, p $\longrightarrow$ A'.p> (see page 134) i.e.

If

r, a vector with integer components is such that

for all p in $\mathrm{BASE}_{(QR)}$, there exists a positive integer m s.t.

$$p = A'.p - m*r$$

then

$$r = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}$$

Proof

Let the components of r be d, e, and f.

Assume the hypothesis, that, for all $\begin{bmatrix} i \\ j \\ k \end{bmatrix}$ in $BASE_{(QR)}$, there exists a positive integer m s.t.

$$(A' - I).\begin{bmatrix} i \\ j \\ k \end{bmatrix} = m*r$$

i.e.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} i \\ j \\ k \end{bmatrix} = m*r$$

i.e.

$$\begin{bmatrix} 0 \\ k-j \\ 0 \end{bmatrix} = m*\begin{bmatrix} d \\ e \\ f \end{bmatrix}$$

Now let p equal $\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$. This is in $BASE_{(QR)}$ so we know that

0 = m*d, -1 = m*e and 0 = m*f

This implies that

m = 1, d = 0, e = -1 and f = 0

Theorem 4

CONTROL' || DATA' simulates CONTROL || DATA

Proof

...using Theorem 1.

By Theorem 2 on page 206,

$CONTROL_{(i)}$ || $DATA_{(i)}$ simulates $DATA_{(i-1)}$ w.r.t. $<Varset_{(i)}, R_{(i)}>$, for all i s.t. $1 < i \leq n$

If we can prove that, for all k s.t. $1 < k \leq n$

$$\left(\bigcup_{i \in Nat(k-1)} Vars(CONTROL_{(i)}) \right) \cap Varset_{(k)} = \emptyset \qquad \text{(viii)}$$

then all the hypotheses, and therefore the conclusion of Theorem 1 will hold and Theorem 4 will be proven; but (viii) is true because the only control variables in $Varset_{(k)DATA}$ are of class c_k and $\bigcup_{i \in Nat(k-1)}$ $Vars(CONTROL_{(i)})$ consists solely of control variables in classes $c_1 \ldots c_k$.

Appendix E : Propositions relating to control-pipelining

The main result of this section is Theorem 12, which states that under certain conditions control-pipelining preserves behaviour. A key result is Lemma 35 which states sufficient conditions for the pipelining of each control-variable-class to be valid. Most of the other propositions (i.e. Theorem 7 to Theorem 11) in this section prove the validity of pipelining the particular control-variable-classes in the convolution and QR-factorisation examples, assuming the well-definedness of certain computations.

The definitions and assumptions made at the start of the previous appendices are assumed to hold for this one. The following ones also hold:

Definitions

$\mathrm{BASE}_{(i:0)} := \{p : p = \Delta_i(p)\}$

$\mathrm{BASE}_{(i:1)} := \{p : p \neq \Delta_i(p)\}$

(Consequently the definition of $\mathrm{CONTROL}_{(i)}$ on on page 198 may be rewritten:

$\mathrm{In}(\mathrm{CONTROL}_{(i)}) = \emptyset$

$\mathrm{Out}(\mathrm{CONTROL}_{(i)}) = \{<c_i, p> : p \in \mathrm{BASE}\}$

$\mathrm{Rel}(\mathrm{CONTROL}_{(i)})v \Leftrightarrow$

For all p in BASE,

$(p \in \mathrm{BASE}_{(i:0)} \Rightarrow v(<c_i, p>) = 0$ and

$p \in \mathrm{BASE}_{(i:0)} \Rightarrow v(<c_i, p>) = 1))$

Assumptions

Assume that we can find disjoint sets $D_{(i:0)}$ and $D_{(i:1)}$ outside BASE and a vector $r_{c\searrow i}$ with integer coefficients s.t., for all p in BASE,

$p \in BASE_{(i:0)} \Rightarrow$ there exists p' in $D_{(i:0)}$ and an integer m s.t.

$$p = p' - m*r_{c\searrow i}$$

and

$p \in BASE_{(i:1)} \Rightarrow$ there exists p' in $D_{(i:1)}$ and an integer m s.t.

$$p = p' - m*r_{c\searrow i}$$

Assume further that for all p' in $D_{(i:0)} \cup D_{(i:1)}$, there exists M s.t. $1 < m < M$ $\Leftrightarrow (p' - m*r_{c\searrow i} \in BASE)$

Note that the set is the domain of the edge computation $CONTROL_{(i:1)}$ and is outside BASE, which is a base for DATA, DATA', CONTROL''' and INTERIOR (to be defined later).

Definitions (continued)

Let $CONTROL_{(i:1)}$ be s.t.

$In(CONTROL_{(i:1)}) = \emptyset$

$Out(CONTROL_{(i:1)}) = \{<c_i, p> : p \in D_{(i:0)} \cup D_{(i:1)}\}$

$Rel(CONTROL_{(i:1)})v \Leftrightarrow$

$(p \in D_{(i:0)} \Rightarrow v(<c_i, p>) = 0$ and

$p \in D_{(i:1)} \Rightarrow v(<c_i, p>) = 1)$

and $CONTROL_{(i:2)}$ be s.t.

$\mathrm{In}(\mathrm{CONTROL}_{(i:2)}) \quad = \quad \{<c_i, p> : p \in D_{(i:0)} \cup D_{(i:1)}\}$

$\mathrm{Out}(\mathrm{CONTROL}_{(i:2)}) \quad = \quad \{<c_i, p> : p \in \mathrm{BASE}\}$

$\mathrm{Rel}(\mathrm{CONTROL}_{(i:2)})v \quad \Leftrightarrow$

$(p \in \mathrm{BASE} \quad \Rightarrow \; v(<c_i, p>) = v(<c_i, p+r_{c\searrow i}>))$

Let $R\_CP_{(i)}$ be $Id_{\{c\searrow i\}\times BASE}$ and $Varset\_CP_{(i)}$ be $\{c_i\}\times(D_{(i:0)} \cup D_{(i:1)})$.

$\mathrm{CONTROL''} \;:=\; \|_{i \in \{1...n\}} \mathrm{CONTROL}_{(i:1)}$

$\mathrm{CONTROL'''} \;:=\; \|_{i \in \{1...n\}} \mathrm{CONTROL}_{(i:2)}$

$R \;:=\; Id_{Vars(\mathrm{CONTROL'})\setminus V)}$

$\mathrm{EDGE} \;:=\; \mathrm{CONTROL''}$

$\mathrm{INTERIOR} \;:=\; \mathrm{CONTROL'''} \,\|\, \mathrm{DATA'}$

$V \;:=\; \bigcup_{i \in Nat(n)} Varset\_CP_{(i)}$

Comment

For a discussion of the roles of $\mathrm{CONTROL}_{(i:1)}$, $\mathrm{CONTROL}_{(i:2)}$, CONTROL'' and CONTROL''', see section 4.3 (starting on page 94). These computations, along with EDGE and INTERIOR, appear in Figure 4.11 on page 98. The renaming function $R\_CP_{(i)}$ and variable set $Varset\_CP_{(i)}$ are used to prove that $\mathrm{CONTROL}_{(i:1)} \,\|\, \mathrm{CONTROL}_{(i:2)}$ implements $\mathrm{CONTROL}_{(i)}$ (Lemma 35) and the renaming function R and the variable set V are used to prove that EDGE || INTERIOR implements CONTROL' || DATA' (Theorem 12). (CONTROL' and DATA' are defined on page 198.)

Assumptions (continued)

Comment

If the following statement holds then all the computations which appear in the proofs in this Appendix are well-defined.

CONTROL'', CONTROL''',
CONTROL'' || CONTROL''', CONTROL''' || DATA',
INTERIOR, EDGE\V,
EDGE || INTERIOR, CONTROL' || DATA',
((CONTROL'' || CONTROL''')\V) || DATA',
(EDGE || INTERIOR)\V, $\text{CONTROL}_{(i:1)}$,
$\text{CONTROL}_{(i:2)}$, $(\text{CONTROL}_{(i:1)} \parallel \text{CONTROL}_{(i:2)})$,
$(\text{CONTROL}_{(i:1)} \parallel \text{CONTROL}_{(i:2)})\backslash\text{Varset\_CP}_{(i)}$,
$(\parallel_{i \in \{1...n\}}(\text{CONTROL}_{(i:1)} \parallel \text{CONTROL}_{(i:2)})) \parallel \text{DATA}'$,
$((\parallel_{i \in \{1...n\}}(\text{CONTROL}_{(i:1)} \parallel \text{CONTROL}_{(i:2)})) \parallel \text{DATA}')\backslash\text{V}$
and $(\parallel_{i \in \{1...n\}}(\text{CONTROL}_{(i:1)} \parallel \text{CONTROL}_{(i:2)}))\backslash\text{V} \parallel \text{DATA}'$

are well-defined and,

for all k in {1...n},
$(\text{CONTROL}_{(k:1)} \parallel \text{CONTROL}_{(k:2)})$,
$(\parallel_{i \in \{1...k-1\}}(\text{CONTROL}_{(i:1)} \parallel \text{CONTROL}_{(i:2)}))$,
$(\text{CONTROL}_{(k:1)} \parallel \text{CONTROL}_{(k:2)}) \parallel$
$(\parallel_{i \in \{1...k-1\}}(\text{CONTROL}_{(i:1)} \parallel \text{CONTROL}_{(i:2)}))$,

Now for $\mathrm{Rel}(\mathrm{DATA}_{(i)})$

$\mathrm{Rel}(\mathrm{DATA}_{(i)})v \quad \Leftrightarrow$

$\Leftrightarrow \mathrm{Rel}(\|_{p\in \mathrm{BASE}}(\mathrm{DATA\_M}_{(i-1)} \circledR \mathrm{R\_DP}_{(i)} \| \mathrm{PIPE\_M}_{(i)}) \circledR \mathrm{R\_DATA}_{(i : p)})v$

$\Leftrightarrow \mathrm{Rel}(\|_{p\in \mathrm{BASE}}(\mathrm{DATA\_M}_{(i-1)} \circledR (\mathrm{R\_DATA}_{(i : p)}|_{\mathrm{Vars(DATA\_M↘(i-1))}}$
$[<a_i, \Delta_i> \rightarrow <z_i, p>])$
$\| (\mathrm{PIPE\_M}_{(i)} \circledR \mathrm{R\_DATA}_{(i : p)}|_{\mathrm{Vars(PIPE\_M(i))}}))v$

$\Leftrightarrow$ For all p in BASE,
$(\mathrm{Rel}(\mathrm{DATA\_M}_{(i-1)} \circledR (\mathrm{R\_DATA}_{(i : p)}|_{\mathrm{Vars(DATA\_M↘(i-1))}}$
$[<a_i, \Delta_i> \rightarrow <z_i, p>])))$
$v|_{\mathrm{Vars(DATA\_M↘(i-1)} \circledR \mathrm{(R\_DATA↘(i : p)}/\mathrm{Vars(DATA\_M↘(i-1))[<a↘i, \Delta↘i> \rightarrow <z↘i, p>]))}}$
and
$(\mathrm{Rel}(\mathrm{PIPE\_M}_{(i)} \circledR (\mathrm{R\_DATA}_{(i : p)}|_{\mathrm{Vars(PIPE\_M↘(i))}})))$
$v|_{\mathrm{Vars(PIPE\_M↘(i)} \circledR \mathrm{(R\_DATA↘(i : p)}/\mathrm{Vars(PIPE\_M↘(i))))}}$

by definition of composition and the definition of the variables of a renamed computation

$\Leftrightarrow$ For all p in BASE,
$(\mathrm{Rel}(\mathrm{DATA\_M}_{(i-1)} \circledR (\mathrm{R\_DATA}_{(i : p)}|_{\mathrm{Vars(DATA\_M↘(i-1))}}$
$[<a_i, \Delta_i> \rightarrow <z_i, p>])))$
$v|_{\mathrm{Vars(DATA\_M↘(i-1)} \circledR \mathrm{(R\_DATA↘(i : p)}/\mathrm{Vars(DATA\_M↘(i-1))[<a↘i, \Delta↘i> \rightarrow <z↘i, p>]))}}$
and
$v(<z_i, p>) = v(<c_i, p>) * v(<z_i, p+r_i>) + \bar{v}(<c_i, p>) * v(<a_i, p>)$
)

by definition of $\mathrm{PIPE\_M}_{(i)}$

<u>Expressions for In(CONTROL↘(i) || DATA↘(i)).</u>

Out(CONTROL$\searrow$(i) || DATA$\searrow$(i)) and Rel(CONTROL$\searrow$(i) || DATA$\searrow$(i))

Let us now expand $\mathrm{CONTROL}_{(i)}$.

$\mathrm{In}(\mathrm{CONTROL}_{(i)}) = \emptyset$

$\mathrm{Out}(\mathrm{CONTROL}_{(i)}) = \{<c_i, p> : p \in \mathrm{BASE}\}$

$\mathrm{Rel}(\mathrm{CONTROL}_{(i)})v \Leftrightarrow$ For all p in BASE, $(v(<c_i, p>) = 1 \Leftrightarrow p \neq \Delta_i(p))$ and $(v(<c_i, p>) = 0 \Leftrightarrow p = \Delta_i(p))$

$\mathrm{In}(\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)}) = \mathrm{In}(\mathrm{DATA}_{(i)}) - \{<c_i, p> : p \in \mathrm{BASE}\}$

by definition of composition and $\mathrm{CONTROL}_{(i)}$

$$= \bigcup_{p \in \mathrm{BASE}} \mathrm{ran}(\mathrm{R\_DATA}_{(i\,:\,p)}|_{\mathrm{In}(\mathrm{DATA\_M}\searrow(i-1))}[<a_i, \Delta_i> \rightarrow <z_i, p>])$$
$$\cup \bigcup_{p \in \mathrm{BASE}} \{<z_i, p+r_i>, <a_i, p>\}$$
$$- \left(\bigcup_{p \in \mathrm{BASE}} \mathrm{ran}(\mathrm{R\_DATA}_{(i\,:\,p)}|_{\mathrm{Out}(\mathrm{DATA\_M}\searrow(i-1))}) \cup \{<z_i, p> : p \in \mathrm{BASE}\}\right)$$

rewriting $\mathrm{In}(\mathrm{DATA}_{(i)})$ and simplifying, using the fact that $<a_i, \Delta_i> \in \mathrm{In}(\mathrm{DATA\_M}_{(i-1)})$ and $<a_i, \Delta_i> \notin \mathrm{Out}(\mathrm{DATA\_M}_{(i-1)})$

$\mathrm{Out}(\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)}) =$

$\mathrm{Out}(\mathrm{DATA}_{(i)}) \cup \mathrm{Out}(\mathrm{CONTROL}_{(i)})$

$$= \bigcup_{p \in \mathrm{BASE}} \mathrm{ran}(\mathrm{R\_DATA}_{(i\,:\,p)}|_{\mathrm{Out}(\mathrm{DATA\_M}\searrow(i-1))})$$
$$\cup \{<z_i, p> : p \in \mathrm{BASE}\}$$
$$\cup \{<c_i, p> : p \in \mathrm{BASE}\}$$

rewriting $\mathrm{Out}(\mathrm{DATA}_{(i)})$, $\mathrm{Out}(\mathrm{CONTROL}_{(i)})$ and simplifying

$\mathrm{Rel}(\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)}) \Leftrightarrow$

For all p in BASE,

$(\mathrm{Rel}(\mathrm{DATA\_M}_{(i-1)}) ® (\mathrm{R\_DATA}_{(i\,:\,p)}|_{\mathrm{Vars}(\mathrm{DATA\_M}\searrow(i-1))}$

$[\langle a_i, \Delta_i\rangle \rightarrow \langle z_i, p\rangle])))$

$v|_{\mathrm{Vars}(\mathrm{DATA\_M}\searrow(i-1)} ® (\mathrm{R\_DATA}\searrow(i\,:\,p))/\mathrm{Vars}(\mathrm{DATA\_M}\searrow(i-1))[\langle a\searrow i, \Delta\searrow i\rangle$
$\rightarrow \langle z\searrow i, p\rangle]))$

and

$v(\langle z_i, p\rangle) = v(\langle c_i, p\rangle) * v(\langle z_i, p+r_i\rangle) + \bar{v}(\langle c_i, p\rangle) * v(\langle a_i, p\rangle)$

)

and, for all p in BASE,

$(v(\langle c_i, p\rangle) = 1 \Leftrightarrow p \neq \Delta_i(p))$ and $(v(\langle c_i, p\rangle) = 0 \Leftrightarrow p = \Delta_i(p))$

using rewriting of $\mathrm{Rel}(\mathrm{DATA}_{(i)})$ and the definition of $\mathrm{Rel}(\mathrm{CONTROL}_{(i)})$

<u>The core of the proof</u>

It is necessary and sufficient to show that

$$\mathrm{Out}(((\mathrm{CONTROL}_{(i)} \,||\, \mathrm{DATA}_{(i)})\backslash\mathrm{Varset}_{(i)}) ® \mathrm{R}_{(i)}) = \mathrm{Out}(\mathrm{DATA}_{(i-1)}) \quad \text{(iv)}$$

$$\mathrm{In}(((\mathrm{CONTROL}_{(i)} \,||\, \mathrm{DATA}_{(i)})\backslash\mathrm{Varset}_{(i)}) ® \mathrm{R}_{(i)}) = \mathrm{In}(\mathrm{DATA}_{(i-1)}) \quad \text{(v)}$$

$$\mathrm{Rel}(((\mathrm{CONTROL}_{(i)} \,||\, \mathrm{DATA}_{(i)})\backslash\mathrm{Varset}_{(i)}) ® \mathrm{R}_{(i)}) \Leftrightarrow \mathrm{Rel}(\mathrm{DATA}_{(i-1)}) \quad \text{(vi)}$$

<u>Proof of (iv)</u>

$\mathrm{Out}(((\mathrm{CONTROL}_{(i)} \,||\, \mathrm{DATA}_{(i)})\backslash\mathrm{Varset}_{(i)}) ® \mathrm{R}_{(i)})$

$= \mathrm{Out}((\mathrm{CONTROL}_{(i)} \,||\, \mathrm{DATA}_{(i)})\backslash\mathrm{Varset}_{(i)})$

$= \bigcup_{p \in \mathrm{BASE}} \mathrm{ran}(\mathrm{R\_DATA}_{(i\,:\,p)}|_{\mathrm{Out}(\mathrm{DATA\_M}\searrow(i-1))})$

$\cup \{\langle z_i, p\rangle : p \in \mathrm{BASE}\}$

$\cup \{\langle c_i, p\rangle : p \in \mathrm{BASE}\}$

$$- \ \mathrm{Varset}_{(i)}$$

$$= \ \mathrm{Out}(\mathrm{DATA}_{(i-1)})$$

<u>Proof of (v)</u>

$$\mathrm{In}(((\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)})\backslash\mathrm{Varset}_{(i)}) \circledR \mathrm{R}_{(i)})$$

$$= \ \mathrm{In}((\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)})\backslash\mathrm{Varset}_{(i)})$$

trivially from the definition of $R_{(i)}$ on page 198

$$= \ \bigcup_{p \in \mathrm{BASE}} \mathrm{ran}(\mathrm{R\_DATA}_{(i:p)}|_{\mathrm{In}(\mathrm{DATA\_M}\searrow(i-1))}[<a_i, \Delta_i> \rightarrow <z_i, p>])$$

$$\cup \ \bigcup_{p \in \mathrm{BASE}} \{<z_i, p+r_i>, <a_i, p>\}$$

$$- \ (\bigcup_{p \in \mathrm{BASE}} \mathrm{ran}(\mathrm{R\_DATA}_{(i:p)}|_{\mathrm{Out}(\mathrm{DATA\_M}\searrow(i-1))})$$

$$\cup \{<z_i, p> : \ p \in \mathrm{BASE}\})$$

$$- \ \mathrm{Varset}_{(i)}$$

from definition of $\mathrm{Varset}_{(i)}$ on page 198

$$= \ \bigcup_{p \in \mathrm{BASE}} \mathrm{ran}(\mathrm{R\_DATA}_{(i:p)}|_{\mathrm{In}(\mathrm{DATA\_M}\searrow(i-1)) - \{<a\searrow i, \Delta\searrow i>\}})$$

$$\cup \ \bigcup_{p \in \mathrm{BASE}} \{<a_i, p>\}$$

$$- \ \bigcup_{p \in \mathrm{BASE}} \mathrm{ran}(\mathrm{R\_DATA}_{(i:p)}|_{\mathrm{Out}(\mathrm{DATA\_M}\searrow(i-1))})$$

$$- \ \{<a_i, p> : p \in \mathrm{BASE} \text{ and } <a_i, p> \notin \mathrm{In}(\mathrm{DATA}_{(i-1)})\}$$

From (iii) on page 199, we may deduce that $\Delta_i(p) \in \mathrm{BASE}$ for all p in BASE, so $\{<a_i, \Delta_i(p)> : p \in \mathrm{BASE}\} \subseteq \bigcup_{p \in \mathrm{BASE}} \{<a_i, p>\}$; therefore we know that the above expression equals

$$\bigcup_{p \in \mathrm{BASE}} \mathrm{ran}(\mathrm{R\_DATA}_{(i:p)}|_{\mathrm{In}(\mathrm{DATA\_M}\searrow(i-1))})$$

$$\cup \ \bigcup_{p \in \mathrm{BASE}} \{<a_i, p>\}$$

$$- \bigcup_{p \in BASE} \mathrm{ran}(R\_DATA_{(i\,:\,p)}|_{Out(DATA\_M\searrow(i-1))})$$

$$- \{<a_i, p> : p \in BASE \text{ and } <a_i, p> \notin In(DATA_{(i-1)})\}$$

$$= \bigcup_{p \in BASE} \mathrm{ran}(R\_DATA_{(i\,:\,p)}|_{In(DATA\_M\searrow(i-1))})$$

$$- \bigcup_{p \in BASE} \mathrm{ran}(R\_DATA_{(i\,:\,p)}|_{Out(DATA\_M\searrow(i-1))})$$

by Lemma 15 on page 179 with A equal to

$$\bigcup_{p \in BASE} \mathrm{ran}(R\_DATA_{(i\,:\,p)}|_{In(DATA\_M\searrow(i-1))})$$

and B equal to $\bigcup_{p \in BASE} \{<a_i, p>\}$, since A - B will then be

$$\{<a_i, p> : p \in BASE \text{ and } <a_i, p> \notin In(DATA_{(i-1)})\}$$

$$= In(DATA_{(i-1)})$$

<u>Proof of (vi)</u>

$Rel(((CONTROL_{(i)} \parallel DATA_{(i)})\backslash Varset_{(i)}) \circledR R_{(i)})$

$\Leftrightarrow$ $Rel((CONTROL_{(i)} \parallel DATA_{(i)})\backslash Varset_{(i)})\, v$

$\Leftrightarrow$ For all v',

$Rel(CONTROL_{(i)} \parallel DATA_{(i)})v' \Rightarrow$

$(v'|_{In((CONTROL\searrow(i) \parallel DATA\searrow(i))\backslash Varset\searrow(i))}$

$= v|_{In((CONTROL\searrow(i) \parallel DATA\searrow(i))\backslash Varset\searrow(i))}$

$\Rightarrow v'|_{Out((CONTROL\searrow(i) \parallel DATA\searrow(i))\backslash Varset\searrow(i))}$

$= v|_{Out((CONTROL\searrow(i) \parallel DATA\searrow(i))\backslash Varset\searrow(i))})$

by definition of hiding

We want to show that this is equivalent to $Rel(DATA_{(i-1)})v$. Now

$Rel(DATA_{(i-1)})v \Leftrightarrow$

For all p, $Rel(DATA\_M_{(i-1)} \circledR R\_DATA_{(i-1\,:\,p)})v|_{Vars(DATA\_M\searrow(i-1) \circledR R\_DATA\searrow(p))}$

$\Leftrightarrow$ For all p, Rel(DATA_M$_{(i-1)}$)(v$|_{\text{Vars(DATA_M↘(i-1)}}$ ® R_DATA↘(p))$^{\bullet}$R_DATA$_{(i-1\,:\,p)}$

We will divide the proof of (vi) into "$\Rightarrow$" and "$\Leftarrow$", but first we will prove

Rel(CONTROL$_{(i)}$ || DATA$_{(i)}$)v' $\Rightarrow$ Rel(DATA$_{(i-1)}$)v'$|_{\text{Vars(DATA↘(i-1))}}$ (vii)

<u>Proof of (vii)</u>

Now

Rel(CONTROL$_{(i)}$ || DATA$_{(i)}$)v $\Leftrightarrow$

For all p in BASE,

(Rel(DATA_M$_{(i-1)}$ ® (R_DATA$_{(i\,:\,p)}$$|_{\text{Vars(DATA_M↘(i-1))}}$

[$<a_i, \Delta_i> \rightarrow <z_i, p>$])))

v$|_{\text{Vars(DATA_M↘(i-1)}}$ ® (R_DATA↘(i : p)$|$Vars(DATA_M↘(i-1))[<a↘i, Δ↘i> → <z↘i, p>]))

and

$v(<z_i, p>) = v(<c_i, p>) * v(<z_i, p+r_i>) + \bar{v}(<c_i, p>) * v(<a_i, p>)$

)

and, for all p in BASE,

$(v(<c_i, p>) = 1 \Leftrightarrow p \neq \Delta_i(p))$ and $(v(<c_i, p>) = 0 \Leftrightarrow p = \Delta_i(p))$

from previous work

The last two subclauses of the R.H.S. imply that

for all p in BASE,

$(p \neq \Delta_i(p) \Rightarrow v(<z_i, p>) = v(<z_i, p+r_i>))$

and $(p = \Delta_i(p) \Rightarrow v(<z_i, p>) = v(<a_i, p>))$

which implies that, for all p in BASE, $v(<z_i, p>) = v(<a_i, \Delta_i(p)>)$

by Lemma 31 and (iii) on page 199

So L.H.S. of (vi) $\Rightarrow$

(For all p in BASE,

(Rel(DATA_M$_{(i-1)}$ ® (R_DATA$_{(i\,:\,p)}$|Vars(DATA_M↘(i-1))[<a$_i$, Δ_i> $\rightarrow$ <z$_i$, p>])))

v'|Vars(DATA_M↘(i-1) ® (R_DATA↘(i : p))/Vars(DATA_M↘(i-1))[<a↘i, Δ↘i> $\rightarrow$ <z↘i, p>]))

and, for all p in BASE,

v'(<z$_i$, p>) = v'(<a$_i$, Δ_i(p)>)

)

$\Rightarrow$

(For all p in BASE,

(Rel(DATA_M$_{(i-1)}$)

(v'|Vars(DATA_M↘(i-1) ® (R_DATA↘(i : p))/Vars(DATA_M↘(i-1))[<a↘i, Δ↘i> $\rightarrow$ <z↘i, p>]))$\bullet$(R_DATA$_{(i\,:\,p)}$|Vars(DATA_M↘(i-1))[<a$_i$, Δ_i> $\rightarrow$ <z$_i$, p>])

and, for all p in BASE,

v'(<z$_i$, p>) = v'(<a$_i$, Δ_i(p)>)

)

by definition of renaming

$\Rightarrow$

(For all p in BASE,

(Rel(DATA_M$_{(i-1)}$)

(v'|Vars(DATA_M↘(i-1) ® (R_DATA↘(i : p))/Vars(DATA_M↘(i-1))))

$\bullet$(R_DATA$_{(i\,:\,p)}$|Vars(DATA_M↘(i-1)))

)

using the fact that for all p in BASE,

v'(<z$_i$, p>) = v'(<a$_i$, Δ_i(p)>)

$\Leftrightarrow$ $\mathrm{Rel}(\mathrm{DATA}_{(i-1)})v'|_{\mathrm{Vars}(\mathrm{DATA}\searrow(i-1))}$

by definition of $\mathrm{DATA}_{(i-1)}$ on page 197

and definition of re-naming

$\underline{\Rightarrow}$

...follows from (vii), the fact that

$\mathrm{Vars}(\mathrm{DATA}_{(i-1)}) \subseteq \mathrm{Vars}(\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)})$

and

Lemma 33 on page 205

$\underline{\Leftarrow}$

...follows directly from (vii) and the fact that

$\mathrm{Vars}(\mathrm{DATA}_{(i-1)}) \subseteq \mathrm{Vars}(\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)})$

and

$\mathrm{In}(\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)})|_{\mathrm{Vars}(\mathrm{DATA}\searrow(i-1))} = \mathrm{In}(\mathrm{DATA}_{(i-1)})$

and

$\mathrm{Out}(\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)})|_{\mathrm{Vars}(\mathrm{DATA}\searrow(i-1))} = \mathrm{Out}(\mathrm{DATA}_{(i-1)})$

and

Lemma 34 on page 205

<u>Theorem 3</u>

There is no other way to pipeline the dependencies < ox, p $\longrightarrow$ A'.p> and <oy, p $\longrightarrow$ A'.p> (see page 134) i.e.

If

r, a vector with integer components is such that

for all p in $\mathrm{BASE}_{(\mathrm{QR})}$, there exists a positive integer m s.t.

$$p \;=\; A'.p - m^{*}r$$

then

$$r = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}$$

<u>Proof</u>

Let the components of r be d, e, and f.

Assume the hypothesis, that, for all $\begin{bmatrix} i \\ j \\ k \end{bmatrix}$ in $BASE_{(QR)}$, there exists a positive integer m s.t.

$$(A' - I).\begin{bmatrix} i \\ j \\ k \end{bmatrix} = m^*r$$

i.e.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} i \\ j \\ k \end{bmatrix} = m^*r$$

i.e.

$$\begin{bmatrix} 0 \\ k-j \\ 0 \end{bmatrix} = m^* \begin{bmatrix} d \\ e \\ f \end{bmatrix}$$

Now let p equal $\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$. This is in $BASE_{(QR)}$ so we know that

0 = m*d, -1 = m*e and 0 = m*f

This implies that

m = 1, d = 0, e = -1 and f = 0

Theorem 4

CONTROL' || DATA' simulates CONTROL || DATA

Proof

...using Theorem 1.

By Theorem 2 on page 206,

$\text{CONTROL}_{(i)} \parallel \text{DATA}_{(i)}$ simulates $\text{DATA}_{(i-1)}$ w.r.t. $<\text{Varset}_{(i)}, R_{(i)}>$, for all i s.t. $1 < i \leq n$

If we can prove that, for all k s.t. $1 < k \leq n$

$$\left(\bigcup_{i \in \text{Nat}(k-1)} \text{Vars}(\text{CONTROL}_{(i)})\right) \cap \text{Varset}_{(k)} = \emptyset \qquad \text{(viii)}$$

then all the hypotheses, and therefore the conclusion of Theorem 1 will hold and Theorem 4 will be proven; but (viii) is true because the only control variables in $\text{Varset}_{(k)\text{DATA}}$ are of class c_k and $\bigcup_{i \in \text{Nat}(k-1)} \text{Vars}(\text{CONTROL}_{(i)})$ consists solely of control variables in classes $c_1 \ldots c_k$.

Appendix E : Propositions relating to control-pipelining

The main result of this section is Theorem 12, which states that under certain conditions control-pipelining preserves behaviour. A key result is Lemma 35 which states sufficient conditions for the pipelining of each control-variable-class to be valid. Most of the other propositions (i.e. Theorem 7 to Theorem 11) in this section prove the validity of pipelining the particular control-variable-classes in the convolution and QR-factorisation examples, assuming the well-definedness of certain computations.

The definitions and assumptions made at the start of the previous appendices are assumed to hold for this one. The following ones also hold:

Definitions

$\mathrm{BASE}_{(i:0)} := \{p : p = \Delta_i(p)\}$

$\mathrm{BASE}_{(i:1)} := \{p : p \neq \Delta_i(p)\}$

(Consequently the definition of $\mathrm{CONTROL}_{(i)}$ on on page 198 may be rewritten:

$\mathrm{In}(\mathrm{CONTROL}_{(i)}) = \emptyset$

$\mathrm{Out}(\mathrm{CONTROL}_{(i)}) = \{<c_i, p> : p \in \mathrm{BASE}\}$

$\mathrm{Rel}(\mathrm{CONTROL}_{(i)})v \Leftrightarrow$

For all p in BASE,

$(p \in \mathrm{BASE}_{(i:0)} \Rightarrow v(<c_i, p>) = 0$ and

$p \in \mathrm{BASE}_{(i:0)} \Rightarrow v(<c_i, p>) = 1))$

Assumptions

Assume that we can find disjoint sets $D_{(i:0)}$ and $D_{(i:1)}$ outside BASE and a vector $r_{c\searrow i}$ with integer coefficients s.t., for all p in BASE,

$p \in BASE_{(i:0)} \Rightarrow$ there exists p' in $D_{(i:0)}$ and an integer m s.t.

$$p = p' - m^* r_{c\searrow i}$$

and

$p \in BASE_{(i:1)} \Rightarrow$ there exists p' in $D_{(i:1)}$ and an integer m s.t.

$$p = p' - m^* r_{c\searrow i}$$

Assume further that for all p' in $D_{(i:0)} \cup D_{(i:1)}$, there exists M s.t. $1 < m < M$ $\Leftrightarrow (p' - m^* r_{c\searrow i} \in BASE)$

Note that the set is the domain of the edge computation $CONTROL_{(i:1)}$ and is outside BASE, which is a base for DATA, DATA', CONTROL''' and INTERIOR (to be defined later).

Definitions (continued)

Let $CONTROL_{(i:1)}$ be s.t.

$In(CONTROL_{(i:1)}) = \emptyset$

$Out(CONTROL_{(i:1)}) = \{<c_i, p> : p \in D_{(i:0)} \cup D_{(i:1)}\}$

$Rel(CONTROL_{(i:1)})v \Leftrightarrow$

$(p \in D_{(i:0)} \Rightarrow v(<c_i, p>) = 0$ and

$p \in D_{(i:1)} \Rightarrow v(<c_i, p>) = 1)$

and $CONTROL_{(i:2)}$ be s.t.

$$\mathrm{In}(\mathrm{CONTROL}_{(i:2)}) = \{<c_i, p> : p \in D_{(i:0)} \cup D_{(i:1)}\}$$

$$\mathrm{Out}(\mathrm{CONTROL}_{(i:2)}) = \{<c_i, p> : p \in \mathrm{BASE}\}$$

$$\mathrm{Rel}(\mathrm{CONTROL}_{(i:2)})v \Leftrightarrow$$

$$(p \in \mathrm{BASE} \Rightarrow v(<c_i, p>) = v(<c_i, p+r_{c\searrow i}>))$$

Let $\mathrm{R\_CP}_{(i)}$ be $\mathrm{Id}_{\{c\searrow i\}\times \mathrm{BASE}}$ and $\mathrm{Varset\_CP}_{(i)}$ be $\{c_i\}\times(D_{(i:0)} \cup D_{(i:1)})$.

$$\begin{array}{lll}
\mathrm{CONTROL''} & := & \|_{i \in \{1...n\}} \mathrm{CONTROL}_{(i:1)} \\
\mathrm{CONTROL'''} & := & \|_{i \in \{1...n\}} \mathrm{CONTROL}_{(i:2)} \\
\mathrm{R} & := & \mathrm{Id}_{\mathrm{Vars(CONTROL')}\setminus \mathrm{V}} \\
\mathrm{EDGE} & := & \mathrm{CONTROL''} \\
\mathrm{INTERIOR} & := & \mathrm{CONTROL'''} \| \mathrm{DATA'} \\
\mathrm{V} & := & \bigcup_{i \in \mathrm{Nat}(n)} \mathrm{Varset\_CP}_{(i)}
\end{array}$$

Comment

For a discussion of the roles of $\mathrm{CONTROL}_{(i:1)}$, $\mathrm{CONTROL}_{(i:2)}$, CONTROL'' and CONTROL''', see section 4.3 (starting on page 94). These computations, along with EDGE and INTERIOR, appear in Figure 4.11 on page 98. The renaming function $\mathrm{R\_CP}_{(i)}$ and variable set $\mathrm{Varset\_CP}_{(i)}$ are used to prove that $\mathrm{CONTROL}_{(i:1)} \| \mathrm{CONTROL}_{(i:2)}$ implements $\mathrm{CONTROL}_{(i)}$ (Lemma 35) and the renaming function R and the variable set V are used to prove that EDGE || INTERIOR implements CONTROL' || DATA' (Theorem 12). (CONTROL' and DATA' are defined on page 198.)

Assumptions (continued)

Comment

If the following statement holds then all the computations which appear in the proofs in this Appendix are well-defined.

CONTROL'', CONTROL''',

CONTROL'' || CONTROL''', CONTROL''' || DATA',

INTERIOR, EDGE\V,

EDGE || INTERIOR, CONTROL' || DATA',

((CONTROL'' || CONTROL''')\V) || DATA',

(EDGE || INTERIOR)\V, $\text{CONTROL}_{(i:1)}$,

$\text{CONTROL}_{(i:2)}$, $(\text{CONTROL}_{(i:1)} \parallel \text{CONTROL}_{(i:2)})$,

$(\text{CONTROL}_{(i:1)} \parallel \text{CONTROL}_{(i:2)})\backslash\text{Varset\_CP}_{(i)}$,

$(\parallel_{i \in \{1...n\}}(\text{CONTROL}_{(i:1)} \parallel \text{CONTROL}_{(i:2)})) \parallel \text{DATA'}$,

$((\parallel_{i \in \{1...n\}}(\text{CONTROL}_{(i:1)} \parallel \text{CONTROL}_{(i:2)})) \parallel \text{DATA'})\backslash\text{V}$

and $(\parallel_{i \in \{1...n\}}(\text{CONTROL}_{(i:1)} \parallel \text{CONTROL}_{(i:2)}))\backslash\text{V} \parallel \text{DATA'}$

are well-defined and,

for all k in {1...n},

$(\text{CONTROL}_{(k:1)} \parallel \text{CONTROL}_{(k:2)})$,

$(\parallel_{i \in \{1...k-1\}}(\text{CONTROL}_{(i:1)} \parallel \text{CONTROL}_{(i:2)}))$,

$(\text{CONTROL}_{(k:1)} \parallel \text{CONTROL}_{(k:2)}) \parallel$

$(\parallel_{i \in \{1...k-1\}}(\text{CONTROL}_{(i:1)} \parallel \text{CONTROL}_{(i:2)}))$,

$(\|_{i \in \{1...k-1\}}(\text{CONTROL}_{(i:1)} \| \text{CONTROL}_{(i:2)})) \| \text{CONTROL}_{(k)}$,

$((\|_{i \in \{1...k-1\}}(\text{CONTROL}_{(i:1)} \| \text{CONTROL}_{(i:2)})) \|$

$(\text{CONTROL}_{(k:1)} \| \text{CONTROL}_{(k:2)}))\backslash V_k$

and $((\|_{i \in \{1...k-1\}}(\text{CONTROL}_{(i:1)} \| \text{CONTROL}_{(i:2)})) \|$

$((\text{CONTROL}_{(k:1)} \| \text{CONTROL}_{(k:2)})\backslash V_k))$

are well-defined.

Theorem 5

If CONTROL'' || CONTROL''' simulates CONTROL' w.r.t. <Varset, R> then

EDGE || INTERIOR simulates CONTROL' || DATA'

(Refer to the diagrams on page 74 and page 110.)

Proof

... from Lemma 2 on page 171.

Lemma 35

$\text{CONTROL}_{(i\ :\ 1)} \| \text{CONTROL}_{(i\ :\ 2)}$ simulates $\text{CONTROL}_{(i)}$ w.r.t. $<\text{Varset\_CP}_{(i)}, \text{R\_CP}_{(i)}>$

(definitions at the beginning of this appendix)

<u>Proof</u>

$$\mathrm{Out}(\mathrm{CONTROL}_{(i:1)} \parallel \mathrm{CONTROL}_{(i:2)}) = \{<c_i, p> \mid p \in \mathrm{BASE}\} \cup \{<c_i, p> \mid p \in D_{(i:0)} \cup D_{(i:1)}\}$$

$$\mathrm{In}(\mathrm{CONTROL}_{(i:1)} \parallel \mathrm{CONTROL}_{(i:2)}) = \emptyset$$

since $\mathrm{Out}(\mathrm{CONTROL}_{(i:1)}) = \mathrm{In}(\mathrm{CONTROL}_{(i:2)})$

and $\mathrm{In}(\mathrm{CONTROL}_{(i:1)}) = \emptyset$

$\mathrm{Rel}(\mathrm{CONTROL}_{(i:1)} \parallel \mathrm{CONTROL}_{(i:2)})v \iff$

$p \in D_{(i:0)} \Rightarrow v(<c_i, p>) = 0$ and

$p \in D_{(i:1)} \Rightarrow v(<c_i, p>) = 1)$ and

$p \in \mathrm{BASE} \Rightarrow v(<c_i, p>) = v(<c_i, p+r_{c\searrow i}>)$ (ix)

We easily have the results

$\mathrm{Out}((\mathrm{CONTROL}_{(i:1)} \parallel \mathrm{CONTROL}_{(i:2)})\backslash \mathrm{Varset\_CP}_{(i)}) = \mathrm{Out}(\mathrm{CONTROL}_{(i)})$

$\mathrm{In}((\mathrm{CONTROL}_{(i:1)} \parallel \mathrm{CONTROL}_{(i:2)})\backslash \mathrm{Varset\_CP}_{(i)}) = \mathrm{In}(\mathrm{CONTROL}_{(i)})$

Similarly to what was done on page 218, we can use Lemma 33 and Lemma 34 to prove that

$\mathrm{Rel}((\mathrm{CONTROL}_{(i:1)} \parallel \mathrm{CONTROL}_{(i:2)})\backslash \mathrm{Varset\_CP}_{(i)}) \iff \mathrm{Rel}(\mathrm{CONTROL}_{(i)})v$

all we need to do is prove that

$\mathrm{Rel}(\mathrm{CONTROL}_{(i:1)} \parallel \mathrm{CONTROL}_{(i:2)})v$

$\Rightarrow \mathrm{Rel}(\mathrm{CONTROL}_{(i)})v|_{\{<c\searrow i, p> : p \in \mathrm{BASE}\}}$

Now, assuming the L.H.S. of this statement, we need to prove the R. H. S., i.e. that, for all p in BASE,

$p \in \mathrm{BASE}_{(i:0)} \Rightarrow v(<c_i, p>) = 0$ (x)

and

$p \in BASE_{(i:1)} \Rightarrow v(<c_i, p>) = 1$ (xi)

Let us consider (x). It is sufficient to prove that, for all p' in $D_{(i:0)}$, and all integral m,

$p' - m*r_{c\searrow i} \in BASE \Rightarrow v(<c_i, p' - m*r_{c\searrow i}>) = 0$ (we may deduce this from the assumptions starting on page 221)

Proof of (x)

...by induction on m, the inductive hypothesis being, "$p' - m*r_{c\searrow i} \in BASE \Rightarrow v(<c_i, p'-(m-1)*r_{c\searrow i}>) = 0$"

Base case: m=1

If $p' - m*r_{c\searrow i} \in BASE$, then we know, by (ix), that

$v(<c_i, p'-r_{c\searrow i}>) = v(<c_i, p'>)$

but

$v(<c_i, p'>) = 0$

Inductive case: m > 1

If $p' - m*r_{c\searrow i} \in BASE$ then

$v(<c_i, p'-m*r_{c\searrow i}>) = v(<c_i, p'-m*r_{c\searrow i}+r_{c\searrow i}>) = v(<c_i, p'-(m-1)*r_{c\searrow i}>)$

but m must be less that M, so m-1 must be, so $p - (m-1)r_{c\searrow i} \in BASE$, by the last of the assumptions starting on page 221 so, by the inductive hypothesis,

$v(<c_i, p'-(m-1)*r_{c\searrow i}>) = 0$

(xi) can be proved in an exactly parallel manner.

Comment

Theorem 5, Theorem 6 and Theorem 8 are very similar and assert the validity of the pipelining for the control dependencies in the convolution example. Theorem 9, Theorem 10 and Theorem 11 do the same for the QR-factorisation example. All six theorems are simple applications of Lemma 35.

Theorem 6

(As well as defining certain computations, the statement of this theorem contains a list of well-definedness conditions.)

If $BASE_{(CONV)}$, $CONTROL_{(CONV)(1:\ 1)}$, $CONTROL_{(CONV)(1:\ 2)}$ and $CONTROL_{(CONV)(1)}$ are defined as follows:

$$BASE_{(CONV)} := \{ \begin{bmatrix} i \\ j \end{bmatrix} \ i \geq 0, j \geq 0 \text{ and } j \leq 3\text{-}i\}$$

$In(CONTROL_{(CONV)(1)}) := \emptyset$

$Out(CONTROL_{(CONV)(1)}) := \{<c_y, p> : p \in BASE_{(CONV)}\}$

$Rel(CONTROL_{(CONV)(1)})v \Leftrightarrow$

$$(p \in \{ \begin{bmatrix} i \\ 0 \end{bmatrix} : o \leq i \leq 3\} \Rightarrow v(<c_y, p>) = 0$$

and

$$p \in (BASE_{(CONV)} - \{ \begin{bmatrix} i \\ 0 \end{bmatrix} : 0 \leq i \leq 3\}) \Rightarrow v(<c_y, p>) = 1)$$

$In(CONTROL_{(CONV)(1:1)}) := \emptyset$

$Out(CONTROL_{(CONV)(1:1)}) := \{<c_y, \begin{bmatrix} -1 \\ j \end{bmatrix}> : 0 \leq j \leq 3\}$

Rel(CONTROL$_{(\text{CONV})(1:1)}$) $\Leftrightarrow$

$$(p \in \{ \begin{bmatrix}-1\\1\end{bmatrix}, \begin{bmatrix}-1\\2\end{bmatrix}, \begin{bmatrix}-1\\3\end{bmatrix} \} \Rightarrow v(<c_y, p>) = 1$$

and

$$p \in \{ \begin{bmatrix}-1\\0\end{bmatrix} \} \Rightarrow v(<c_y, p>) = 0)$$

In(CONTROL$_{(\text{CONV})(1:2)}$) :=

$$\{<c_y, p + \begin{bmatrix}-1\\0\end{bmatrix}> : p \in \text{BASE}_{(\text{CONV})}\} - \{<c_y, p> : p \in \text{BASE}_{(\text{CONV})}\}$$

Out(CONTROL$_{(\text{CONV})(1:2)}$) := $\{<c_y, p> : p \in \text{BASE}_{(\text{CONV})}\}$

Rel(CONTROL$_{(\text{CONV})(1:2)}$)v $\Leftrightarrow$

$$p \in \text{BASE}_{(\text{CONV})} \Rightarrow v(<c_y, p>) = v(<c_y, p + \begin{bmatrix}-1\\0\end{bmatrix}>)$$

and if

CONTROL$_{(\text{CONV})(1:1)}$,

CONTROL$_{(\text{CONV})(1:2)}$,

(CONTROL$_{(\text{CONV})(1:1)}$ || CONTROL$_{(\text{CONV})(1:2)}$)

and

(CONTROL$_{(\text{CONV})(1:1)}$ || CONTROL$_{(\text{CONV})(1:1)}$)\Varset_CP$_{(\text{CONV})(1)}$

are well-defined, where Varset_CP$_{(\text{CONV})(1)}$ is $\{c_y\} \times (D_{(\text{CONV})(1:0)} \cup D_{(\text{CONV})(1:1)})$ and where

$$D_{(\text{CONV})(1:0)} := \{ \begin{bmatrix}-1\\0\end{bmatrix} \}$$

$$D_{(\text{CONV})(1:1)} := \{ \begin{bmatrix}-1\\1\end{bmatrix}, \begin{bmatrix}-1\\2\end{bmatrix}, \begin{bmatrix}-1\\3\end{bmatrix} \}$$

then

$\mathrm{CONTROL}_{(\mathrm{CONV})(1:\,1)} \parallel \mathrm{CONTROL}_{(\mathrm{CONV})(1:\,2)}$

simulates $\mathrm{CONTROL}_{(\mathrm{CONV})(1)}$

<u>Proof</u>

...by Lemma 35 with

$$r_{c\searrow i} := \begin{bmatrix} -1 \\ 0 \end{bmatrix} \quad (= r_{(\mathrm{CONV})c\searrow 1})$$

We can see from the definitions of $\mathrm{CONTROL}_{(\mathrm{CONV})(1:\,1)}$ and $\mathrm{CONTROL}_{(\mathrm{CONV})(1:\,2)}$ that

$$\mathrm{BASE}_{(\mathrm{CONV})(1:0)} = \{ \begin{bmatrix} i \\ 0 \end{bmatrix} : 0 \le i \le 3 \}$$

$$\mathrm{BASE}_{(\mathrm{CONV})(1:1)} = \mathrm{BASE}_{(\mathrm{CONV})} - \mathrm{BASE}_{(\mathrm{CONV})(1:0)}$$

These two sets are disjoint, and cover $\mathrm{BASE}_{(\mathrm{CONV})}$. We just need to prove the assumptions starting on page 221 for $\mathrm{BASE}_{(i:0)}$ equal to $\mathrm{BASE}_{(\mathrm{CONV})(i:0)}$ etc.

$p \in \mathrm{BASE}_{(\mathrm{CONV})(1:0)}$

$\Rightarrow$ there exists $p' \in D_{(\mathrm{CONV})(1:0)}$ and integer m s.t. $p = p' - m * r_{(\mathrm{CONV})c\searrow 1}$ (xii)

$p \in \mathrm{BASE}_{(\mathrm{CONV})(1:1)}$

$\Rightarrow$ there exists $p' \in D_{(\mathrm{CONV})(1:1)}$ and integer m s.t. $p = p' - m * r_{(\mathrm{CONV})c\searrow 1}$

(xiii)

and

for all p' in $\mathrm{BASE}_{(\mathrm{CONV})(1:0)} \cup \mathrm{BASE}_{(\mathrm{CONV})(1:1)}$, there exists M' s.t.

$$1 \leq m' \leq M' \Leftrightarrow p' - m' * r_{(\mathrm{CONV})c\searrow 1} \in \mathrm{BASE}_{(\mathrm{CONV})} \qquad \text{(xiv)}$$

<u>Proof of (xii)</u>

If $p = \begin{bmatrix} (i) \\ (j) \end{bmatrix}$ then let $p' = \begin{bmatrix} -1 \\ j \end{bmatrix}$ and m = i + 1. If $p \in \mathrm{BASE}_{(\mathrm{CONV})(1:0)}$ then j = 0 and so $p' \in \mathrm{BASE}_{(\mathrm{CONV})(1:0)}$

<u>Proof of (xiii)</u>

If $p = \begin{bmatrix} (i) \\ (j) \end{bmatrix}$ then let $p' = \begin{bmatrix} -1 \\ j \end{bmatrix}$ and m = i + 1. If $p \in \mathrm{BASE}_{(\mathrm{CONV})(1:1)}$ then $1 \leq j \leq 3$ from the definition of $\mathrm{BASE}_{(\mathrm{CONV})}$ on page 228 and so $p' \in \mathrm{BASE}_{(\mathrm{CONV})(1:1)}$.

<u>Proof of (xiv)</u>

$p' = \begin{bmatrix} -1 \\ j \end{bmatrix}$ where $0 \leq j \leq 3$

Let M' = 4 - j. Then for all m, $1 \leq m' \leq M' \Rightarrow p' - m' * \begin{bmatrix} -1 \\ 0 \end{bmatrix} \in \mathrm{BASE}_{(\mathrm{CONV})}$.

<u>Theorem 7</u>

<u>Comment</u>
Theorem 7 is essentially the same as Theorem 6. It states that the pipelining of c_y in the convolution example is valid.

Let $\mathrm{CONTROL}_{(\mathrm{CONV})(2:1)}$, $\mathrm{CONTROL}_{(\mathrm{CONV})(2:2)}$ and $\mathrm{CONTROL}_{(\mathrm{CONV})(2)}$ be defined as $\text{CONTROL}_{(\text{CONV})(1:\ 1)}$, $\text{CONTROL}_{(\text{CONV})(1:\ 2)}$ and $\text{CONTROL}_{(\text{CONV})(1)}$ respectively, but with c_y replaced by c_x and $r_{(\text{CONV})c\searrow 1}$ replaced by $r_{(\text{CONV})c\searrow 2}$ in the definitions.

If

$\text{CONTROL}_{(\text{CONV})(2:\ 1)}$,

$\text{CONTROL}_{(\text{CONV})(2:\ 2)}$,

$(\text{CONTROL}_{(\text{CONV})(2:\ 1)} \parallel \text{CONTROL}_{(\text{CONV})(2:\ 2)})$

and

$(\text{CONTROL}_{(\text{CONV})(2:\ 1)} \parallel \text{CONTROL}_{(\text{CONV})(2:\ 1)})\backslash\text{Varset\_CP}_{(\text{CONV})(2)}$

are well-defined, where $\text{Varset\_CP}_{(\text{CONV})(2)}$ is $\{c_x\}\times(D_{(\text{CONV})(2\,:\,0)} \cup D_{(\text{CONV})(2\,:\,1)})$ and where

$$D_{(\text{CONV})(2\,:\,0)} := \{ \begin{bmatrix}-1\\0\end{bmatrix} \}$$

$$D_{(\text{CONV})(2\,:\,1)} := \{ \begin{bmatrix}-1\\1\end{bmatrix}, \begin{bmatrix}-1\\2\end{bmatrix}, \begin{bmatrix}-1\\3\end{bmatrix} \}$$

then

$\text{CONTROL}_{(\text{CONV})(2:\ 1)} \parallel \text{CONTROL}_{(\text{CONV})(2:\ 2)}$

simulates $\text{CONTROL}_{(\text{CONV})(2)}$

<u>Proof</u>

Replace c_y by c_x in proof of Theorem 6, and the first occurrence of 1 in each subscript by 2.

<u>Theorem 8</u>

<u>Comment</u>

Theorem 8 is similar as Theorem 6. It states that the pipelining of c_w in the convolution example is valid.

Let $\text{CONTROL}_{(\text{CONV})(3:\,1)}$, $\text{CONTROL}_{(\text{CONV})(3:\,2)}$ and $\text{CONTROL}_{(\text{CONV})(3)}$ be defined as $\text{CONTROL}_{(\text{CONV})(1:\,1)}$, $\text{CONTROL}_{(\text{CONV})(1:\,2)}$ and $\text{CONTROL}_{(\text{CONV})(1)}$ respectively, but with c_y replaced by c_w $r_{(\text{CONV})c\searrow 1}$, replaced by $r_{(\text{CONV})c\searrow 3}$ in the definitions and $\begin{bmatrix}(s)\\(t)\end{bmatrix}$ replaced by $\begin{bmatrix}(t)\\(s)\end{bmatrix}$ (i.e. all column vectors inverted).

If

$\text{CONTROL}_{(\text{CONV})(3:\,1)}$,

$\text{CONTROL}_{(\text{CONV})(3:\,2)}$,

$(\text{CONTROL}_{(\text{CONV})(3:\,1)} \parallel \text{CONTROL}_{(\text{CONV})(3:\,2)})$

and

$(\text{CONTROL}_{(\text{CONV})(3:\,1)} \parallel \text{CONTROL}_{(\text{CONV})(3:\,1)})\backslash\text{Varset\_CP}_{(\text{CONV})(3)}$

are well-defined, where $\text{Varset\_CP}_{(\text{CONV})(3)}$ is $\{c_w\}\times(D_{(\text{CONV})(3\,:\,0)} \cup D_{(\text{CONV})(3\,:\,1)})$ and where

$$D_{(\text{CONV})(3\,:\,0)} := \{ \begin{bmatrix}0\\-1\end{bmatrix} \}$$

$$D_{(\text{CONV})(3\,:\,1)} := \{ \begin{bmatrix}1\\-1\end{bmatrix}, \begin{bmatrix}2\\-1\end{bmatrix}, \begin{bmatrix}3\\-1\end{bmatrix} \}$$

then

$\text{CONTROL}_{(\text{CONV})(3:\,1)} \parallel \text{CONTROL}_{(\text{CONV})(3:\,2)}$

simulates $\text{CONTROL}_{(\text{CONV})(2)}$

$\text{CONTROL}_{(\text{CONV})(3:\,1)} \parallel \text{CONTROL}_{(\text{CONV})(3:\,2)}$

simulates $\mathrm{CONTROL}_{(\mathrm{CONV})(3)}$

Proof

Replace c_y by c_w and $\begin{bmatrix}(s)\\(t)\end{bmatrix}$ by $\begin{bmatrix}(t)\\(s)\end{bmatrix}$ (i.e. invert all column vectors) in proof of Theorem 6, and the first occurrence of 1 in each subscript by 3.

Theorem 9

Comment

Theorem 9 states that the pipelining of cont in the QR-factorisation example is valid. Its statement and proof follow the pattern for Theorem 6, with the minor difference that $D_{(\mathrm{QR})(1)}$ is defined explicitly (on page 234) whereas $D_{(\mathrm{CONV})(1)}$ is not. It can therefore be clearly seen how $D_{(\mathrm{QR})(1:0)}$ and $D_{(\mathrm{QR})(1:1)}$ are constructed (page 234), whereas $D_{(\mathrm{CONV})(1:0)}$ and $D_{(\mathrm{CONV})(1:1)}$ are seemingly plucked from nowhere (page 229).

Let $D_{(\mathrm{QR})(1)}$, $D_{(\mathrm{QR})(1:0)}$, $D_{(\mathrm{QR})(1:1)}$, $\mathrm{BASE}_{(\mathrm{QR})(1:0)}$, $\mathrm{BASE}_{(\mathrm{QR})(1:1)}$, $\mathrm{CONTROL}_{(\mathrm{QR})(1:1)}$, $\mathrm{CONTROL}_{(\mathrm{QR})(1:2)}$ and $\mathrm{CONTROL}_{(\mathrm{QR})(1)}$

be defined as follows:

$$D_{(\mathrm{QR})(1)} := \{(p + \begin{bmatrix}0\\-1\\0\end{bmatrix}) : p \in \mathrm{BASE}_{(\mathrm{QR})}\} - \mathrm{BASE}_{(\mathrm{QR})}$$

$$D_{(\mathrm{QR})(1:0)} := D_{(\mathrm{QR})(1)} \cap \{ \begin{bmatrix}i\\j\\k\end{bmatrix} : i = M\}$$

$$D_{(QR)(1:1)} := D_{(QR)(1)} \cap \{ \begin{bmatrix} i \\ j \\ k \end{bmatrix} : i = M \}$$

$$BASE_{(QR)(1:0)} := BASE_{(QR)} \cap \{ \begin{bmatrix} i \\ j \\ k \end{bmatrix} : i = M \}$$

$$BASE_{(QR)(1:1)} := BASE_{(QR)} \cap \{ \begin{bmatrix} i \\ j \\ k \end{bmatrix} : i \neq M \}$$

$In(CONTROL_{(QR)(1)}) := \emptyset$

$Out(CONTROL_{(QR)(1)}) := \{<cont, p> : p \in BASE_{(QR)}\}$

$Rel(CONTROL_{(QR)(1)})v \Leftrightarrow$

$(p \in BASE_{(QR)(1:0)} \Rightarrow v(<cont, p>) = 0$

and

$p \in BASE_{(QR)(1:1)} \Rightarrow v(<cont, p>) = 1)$

$In(CONTROL_{(QR)(1:1)}) := \emptyset$

$Out(CONTROL_{(QR)(1:1)}) := \{<cont, p> : p \in D_{(QR)(1)}\}$

$Rel(CONTROL_{(QR)(1:1)})v \Leftrightarrow$

$(p \in D_{(QR)(1:0)} \Rightarrow v(<cont, p>) = 1$

and

$p \in D_{(QR)(1:1)} \Rightarrow v(<cont, p>) = 0)$

$In(CONTROL_{(QR)(1:2)}) := \{<cont, p> : p \in D_{(QR)(1)}\}$

$Out(CONTROL_{(QR)(1:2)}) := \{<cont, p> : p \in BASE_{(QR)}\}$

$Rel(CONTROL_{(QR)(1:2)})v \Leftrightarrow$

$$p \in BASE_{(QR)} \Rightarrow v(<cont, p>) = v(<cont, p + \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}>)$$

and if

$CONTROL_{(QR)(1:\,1)}$,

$CONTROL_{(QR)(1:\,2)}$,

$(CONTROL_{(QR)(1:\,1)} \parallel CONTROL_{(QR)(1:\,2)})$

and

$(CONTROL_{(QR)(1:\,1)} \parallel CONTROL_{(QR)(1:\,1)})\backslash Varset\_CP_{(QR)(1)}$

are well-defined, where $Varset\_CP_{(QR)(1)}$ is $\{cont\}\times(D_{(QR)(1)})$

then

$CONTROL_{(QR)(1:\,1)} \parallel CONTROL_{(QR)(1:\,2)}$

simulates $CONTROL_{(QR)}$

Proof

...using Lemma 35 with

$$r_i \quad := \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} \quad (= r_{(QR)c\searrow 1})$$

We just need to prove that

$p \in BASE_{(QR)(1\,:\,0)}$

$\Rightarrow$ there exists $p' \in D_{(QR)(1\,:\,0)}$ and integer m s.t. $p = p' - m^*r_{(QR)c\searrow 1}$ (xv)

$p \in BASE_{(QR)(1:1)}$

$\Rightarrow$ there exists $p' \in D_{(QR)(1:1)}$ and integer m s.t. $p = p' - m^*r_{(QR)c\searrow 1}$ (xvi)

for all p' in $BASE_{(QR)(1:0)} \cup BASE_{(QR)(1:1)}$, there exists M s.t.

$$1 \leq m \leq M \Leftrightarrow p' - m^*r_{(QR)c\searrow 1} \in BASE_{(QR)} \qquad \text{(xvii)}$$

Note that

$$D_{(QR)(1)} = \left\{ \begin{bmatrix} i \\ j \\ k \end{bmatrix} : k \in \{1...M\text{-}1\}, j \in \{k\text{-}1...M\text{-}1\} \text{ and } i \in \{k+1...M\} \right\}$$

$$\cap \left\{ \begin{bmatrix} i \\ j \\ k \end{bmatrix} : k \notin \{1...M\text{-}1\}, j \notin \{k...M\} \text{ or } i \notin \{k+1...M\} \right\}$$

$$= \left\{ \begin{bmatrix} i \\ j \\ k \end{bmatrix} : k \in \{1...M\text{-}1\}, j = k\text{-}1 \text{ and } i \in \{k+1...M\} \right\}$$

<u>Proof of (xv)</u>

If $p = \begin{bmatrix} i \\ j \\ k \end{bmatrix}$ then let p' equal $\begin{bmatrix} i \\ k-1 \\ k \end{bmatrix}$ and m equal j-k+1.

If

$p \in D_{(QR)(1:0)}$ then $i = M$ and so $p' \in D_{(QR)(1:0)}$

<u>Proof of (xvi)</u>

If $p = \begin{bmatrix} i \\ j \\ k \end{bmatrix}$ then let p' equal $\begin{bmatrix} i \\ k-1 \\ k \end{bmatrix}$ and m equal j-k+1.

If

$p \in D_{(QR)(1:1)}$ then $i \neq M$ and so $p' \in D_{(QR)(1:1)}$

Proof of (xiv)

$$p' = \begin{bmatrix} i \\ k-1 \\ k \end{bmatrix}, \text{ where } k \in \{1 \ldots M\text{-}1\} \text{ and } i \in \{k+1 \ldots M\}.$$

Let $M' = M\text{-}k\text{+}1$. Then for all m', $1 < m < M'$, $p' - m' * \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} \in BASE_{(QR)}$

Theorem 10

Comment

Theorem 10 is similar as Theorem 9. It states that the pipelining of oy in the QR-factorisation example is valid.

Let $D_{(QR)(2)}$, $D_{(QR)(2:0)}$, $D_{(QR)(2:1)}$, $BASE_{(QR)(2:0)}$, $BASE_{(QR)(2:1)}$, $CONTROL_{(QR)(2:1)}$, $CONTROL_{(QR)(2:2)}$ and $CONTROL_{(QR)(2)}$

be defined as follows:

$$D_{(QR)(2)} := \{(p + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}) : p \in BASE_{(QR)}\} - BASE_{(QR)}$$

so

$$D_{(QR)(2)} = \{ \begin{bmatrix} i \\ j \\ k \end{bmatrix} : k \in \{1 \ldots M\text{-}1\}, j \in \{k \ldots M\} \text{ and } i \in \{k+2 \ldots M+1\}\}$$

$$\cap \; \{ \begin{bmatrix} i \\ j \\ k \end{bmatrix} : k \notin \{1...M-1\}, j \notin \{k...M\} \text{ or } i \notin \{k+1...M\} \}$$

$$= \; \{ \begin{bmatrix} i \\ j \\ k \end{bmatrix} : k \in \{1...M-1\}, j \in \{k...M\} \text{ and } i = M+1 \}$$

$$D_{(QR)(2:0)} := D_{(QR)(2)} \cap \{ \begin{bmatrix} i \\ j \\ k \end{bmatrix} : j = k \}$$

$$D_{(QR)(2:1)} := D_{(QR)(2)} \cap \{ \begin{bmatrix} i \\ j \\ k \end{bmatrix} : j \neq k \}$$

$$BASE_{(QR)(2:0)} := BASE_{(QR)} \cap \{ \begin{bmatrix} i \\ j \\ k \end{bmatrix} : j = k \}$$

$$BASE_{(QR)(2:1)} := BASE_{(QR)} \cap \{ \begin{bmatrix} i \\ j \\ k \end{bmatrix} : j \neq k \}$$

$\mathrm{In}(\mathrm{CONTROL}_{(QR)(2)}) := \emptyset$

$\mathrm{Out}(\mathrm{CONTROL}_{(QR)(2)}) := \{<ox, p> : p \in BASE_{(QR)}\}$

$\mathrm{Rel}(\mathrm{CONTROL}_{(QR)(2)})v \Leftrightarrow$

$(p \in BASE_{(QR)(2:0)} \Rightarrow v(<ox, p>) = 0$

and

$p \in BASE_{(QR)(2:1)} \Rightarrow v(<ox, p>) = 1)$

$\mathrm{In}(\mathrm{CONTROL}_{(QR)(2:1)}) := \emptyset$

$\mathrm{Out}(\mathrm{CONTROL}_{(QR)(2:1)}) := \{<ox, p> : p \in D_{(QR)(2)}\}$

$\mathrm{Rel}(\mathrm{CONTROL}_{(QR)(2:1)})\, v \Leftrightarrow$

$$(p \in D_{(QR)(2:0)} \quad \Rightarrow \quad v(<ox, p>) = 1$$

and

$$p \in D_{(QR)(2:1)} \quad \Rightarrow \quad v(<ox, p>) = 0)$$

$$\mathrm{In}(\mathrm{CONTROL}_{(QR)(2:2)}) \quad := \quad \{<ox, p> : p \in D_{(QR)(2)}\}$$

$$\mathrm{Out}(\mathrm{CONTROL}_{(QR)(2:2)}) \quad := \quad \{<ox, p> : p \in \mathrm{BASE}_{(QR)}\}$$

$$\mathrm{Rel}(\mathrm{CONTROL}_{(QR)(2:2)})v \quad \Leftrightarrow$$

$$p \in \mathrm{BASE}_{(QR)} \Rightarrow v(<ox, p>) = v(<ox, p + \begin{bmatrix}1\\0\\0\end{bmatrix}>)$$

and if

$\mathrm{CONTROL}_{(QR)(2:\,1)}$,

$\mathrm{CONTROL}_{(QR)(2:\,2)}$,

$(\mathrm{CONTROL}_{(QR)(2:\,1)} \parallel \mathrm{CONTROL}_{(QR)(2:\,2)})$

and

$(\mathrm{CONTROL}_{(QR)(2:\,1)} \parallel \mathrm{CONTROL}_{(QR)(2:\,1)}) \backslash \mathrm{Varset\_CP}_{(QR)(2)}$

are well-defined, where $\mathrm{Varset\_CP}_{(QR)(2)}$ is $\{ox\} \times (D_{(QR)(2)})$

then

$\mathrm{CONTROL}_{(QR)(2:\,1)} \parallel \mathrm{CONTROL}_{(QR)(2:\,2)}$

simulates $\mathrm{CONTROL}_{(QR)}$

Proof

We parallel the proof of Theorem 9 and use Lemma 35 with

$$r_i \quad = \quad \begin{bmatrix}1\\0\\0\end{bmatrix} \quad (= r_{(QR)c\searrow 2})$$

We just need to prove that

$p \in BASE_{(QR)(2:0)}$

$\Rightarrow$ there exists $p' \in D_{(QR)(2:0)}$ and integer m s.t. $p = p' - m^* r_{(QR)c\searrow 2}$(xviii)

$p \in BASE_{(QR)(2:1)}$

$\Rightarrow$ there exists $p' \in D_{(QR)(2:1)}$ and integer m s.t. $p = p' - m^* r_{(QR)c\searrow 2}$ (xix)

for all p' in $BASE_{(QR)(2:0)} \cup BASE_{(QR)(2:1)}$, there exists M s.t.

$$1 \leq m \leq M \Leftrightarrow p' - m^* r_{(QR)c\searrow 2} \in BASE_{(QR)} \tag{xx}$$

<u>Proof of (xv)</u>

If $p = \begin{bmatrix}i\\j\\k\end{bmatrix}$ then let p' equal $\begin{bmatrix}M+1\\j\\k\end{bmatrix}$ and m equal M+1-i.

If

$p \in D_{(QR)(2:0)}$ then $j = k$ and so $p' \in D_{(QR)(2:0)}$

<u>Proof of (xvi)</u>

If $p = \begin{bmatrix}i\\j\\k\end{bmatrix}$ then let p' equal $\begin{bmatrix}i\\k-1\\k\end{bmatrix}$ and m equal j-k+1.

If

$p \in D_{(QR)(2:1)}$ then $j \neq k$ and so $p' \in D_{(QR)(2:1)}$

Proof of (xiv)

$p' = \begin{bmatrix} i \\ k-1 \\ k \end{bmatrix}$, where $k \in \{1...M\text{-}1\}$ and $j \in \{k...M\}$.

Let $M' = M\text{-}k$. Then for all m', $1 < m < M'$, $p' - m'^{*}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \in BASE_{(QR)}$

Theorem 11

Comment

Theorem 11 is similar as Theorem 10. It states that the pipelining of oy in the QR-factorisation example is valid.

Let $r_{(QR)c\searrow 3}$, $D_{(QR)(3)}$, $D_{(QR)(3:0)}$, $D_{(QR)(3:1)}$, $BASE_{(QR)(3:0)}$, $BASE_{(QR)(3:1)}$, $CONTROL_{(QR)(3:1)}$, $CONTROL_{(QR)(3:2)}$ and $CONTROL_{(QR)(3)}$ be defined as their counterparts in Theorem 10, but with "oy" replacing "ox", and the first 2 in the subscripts replaced by a 3.

If

$CONTROL_{(QR)(3:1)}$,

$CONTROL_{(QR)(3:2)}$,

$(CONTROL_{(QR)(3:1)} \parallel CONTROL_{(QR)(3:2)})$

and

$(CONTROL_{(QR)(3:1)} \parallel CONTROL_{(QR)(3:1)}) \backslash Varset\_CP_{(QR)(3)}$

are well-defined, where $Varset\_CP_{(QR)(3)}$ is $\{oy\} \times (D_{(QR)(3)})$

then

$\mathrm{CONTROL}_{(QR)(3:\,1)} \parallel \mathrm{CONTROL}_{(QR)(3:\,2)}$

simulates $\mathrm{CONTROL}_{(QR)(3)}$

Proof

As for Theorem 10, with "oy" replacing "ox" and the first 2 in the subscripts replaced by a 3.

Theorem 12

EDGE || INTERIOR simulates CONTROL' || DATA'

Proof

Consider the following three claims:

We know fromLemma 35 that, for all i in {1...n}, $\mathrm{CONTROL}_{(i:1)} \parallel \mathrm{CONTROL}_{(i:2)}$ simulates $\mathrm{CONTROL}_{(i)}$ w.r.t. <Varset_CP$_{(i)}$, R_CP$_{(i)}$>. By Theorem 2, the required result follows from the following three:

$$\|_{i \in \{1...n\}}(\mathrm{CONTROL}_{(i:1)} \parallel \mathrm{CONTROL}_{(i:2)}) = \mathrm{CONTROL''} \parallel \mathrm{CONTROL'''} \qquad \text{(xxi)}$$

$$\|_{i \in \{1...n\}}(\mathrm{CONTROL}_{(i:1)} \parallel \mathrm{CONTROL}_{(i:2)}) \text{ simulates } \|_{i \in \{1...n\}}\mathrm{CONTROL}_{(i)} \quad \text{w.r.t.} <V, R> \qquad \text{(xxii)}$$

$$V \cap \mathrm{Vars(DATA')} = \emptyset \qquad \text{(xxiii)}$$

These statements together imply Theorem 12. To see this, the following reasoning may be used:

(xxi) and (xxii) imply that CONTROL'' $\|$ CONTROL''' simulates $\|_{i \in \{1...n\}}\text{CONTROL}_{(i)}$ w.r.t. <V, R>; now

$$\|_{i \in \{1...n\}}\text{CONTROL}_{(i)} = \text{CONTROL}'$$

so CONTROL'' $\|$ CONTROL''' implements (CONTROL'' $\|$ CONTROL''') $\|$ DATA' by Lemma 22 on page 184, if (xxiii) holds. Now

$$\begin{aligned} \text{CONTROL}''' &= \text{EDGE} \\ \text{CONTROL}''' \,\|\, \text{DATA}' &= \text{INTERIOR} \end{aligned}$$

so, by Lemma 2 on page 171,

$$(\text{CONTROL}'' \,\|\, \text{CONTROL}''') \,\|\, \text{DATA}' = \text{EDGE} \,\|\, \text{INTERIOR}$$

and so the result follows. It is therefore sufficient to prove (xxi), (xxii) and (xxiii)

Proof of (xxi)

The R.H.S. is well-defined and $\text{CONTROL}_{(i:1)} \,\|\, \text{CONTROL}_{(i:2)}$ is well-defined for all i in {1...n}. It is trivial to prove that In(L.H.S.) = In(R.H.S.), Out(L.H.S.) = Out(R.H.S.) and Rel(L.H.S.)v $\Leftrightarrow$ Rel(R.H.S.)v so the L.H.S. is well-defined and L.H.S. = R.H.S.

Proof of (xxii)

We will prove, by induction on k, that, for all k in {1...n},

$\|_{i \in \{1...k\}}(\text{CONTROL}_{(i:1)} \,\|\, \text{CONTROL}_{(i:2)})$ simulates

$\|_{i \in \{1...k\}}\text{CONTROL}_{(i)}$

w.r.t.$<V_k, \text{Id}_{\text{Vars}(\|\searrow(i \in \{1..k\})\text{CONTROL}\searrow(i)) - V\searrow k}>$ where

$$V_k = \bigcup_{i \in \text{Nat}(k)} \text{Varset\_CP}_{(i)}$$

(The statement after “for all” reduces to (xxii) when k = n, since R and V are defined as at the beginning of this appendix and CONTROL’ is defined as on page 198.) The inductive hypothesis is:

“$\|_{i \in \{1..k-1\}}(\mathrm{CONTROL}_{(i:1)} \| \mathrm{CONTROL}_{(i:2)})$ simulates
$\|_{i \in \{1..k-1\}}\mathrm{CONTROL}_{(i)}$
w.r.t.$<V_{k-1}, Id_{Vars(\|\searrow(i \in \{1...k-1\})CONTROL\searrow(i)) - V\searrow k-1}>$”

<u>Base case: k=1</u>

Trivial.

<u>Inductive case</u>

Assume the unquantified statement of the theorem is true for k-1.
Let B’ equal $\|_{i \in \{1..k-1\}}(\mathrm{CONTROL}_{(i:1)} \| \mathrm{CONTROL}_{(i:2)})$
and let B equal $\|_{i \in \{1...k-1\}}\mathrm{CONTROL}_{(i)}$
then, by the inductive hypothesis, B’ simulates B
w.r.t. $<V_{k-1}, Id_{Vars(\|\searrow(i \in \{1..k-1\})CONTROL\searrow(i)) - V\searrow k-1}>$

Now

$\mathrm{Varset\_CP}_{(k)} \cap \mathrm{Vars}(B’) \quad = \quad \emptyset$

since the control variable-classes are all distinct

and

$V_k \cap \mathrm{Vars}(\mathrm{CONTROL}_{(k)}) \quad = \quad \emptyset$

since $(D_{(i\,:\,0)} \cup D_{(i\,:\,1)}) \cap \mathrm{BASE} = \emptyset$

and, by Lemma 35 on page 225,
$(\mathrm{CONTROL}_{(k:1)} \| \mathrm{CONTROL}_{(k:2)}) \| B’$ simulates $\mathrm{CONTROL}_{(k)} \| B’$
w.r.t. $<\mathrm{Varset\_CP}_{(k)}, \mathrm{R\_CP}_{(k)})>$

so by Lemma 22 on page 184 with

R_2	equal to	$Id_{Vars(B') \cup Vars(CONTROL\searrow(k))}$
B	equal to	B'
A	equal to	$CONTROL_{(k)}$
A'	equal to	$(CONTROL_{(k:1)} \| CONTROL_{(k:2)})$
R_1	equal to	$R\_CP_{(k)}$

and

Varset equal to $Varset\_CP_{(k)}$

$CONTROL_{(k)} \| B'$ simulates $CONTROL_{(k)} \| B$

w.r.t.

$<V_k, Id_{Vars(B) \cup Vars(CONTROL(k))}>$

so, by Lemma 27 and Lemma 3,

$\|_{i \in \{1..k\}}(CONTROL_{(i:1)} \| CONTROL_{(i:2)})$ simulates

$\|_{i \in \{1..k\}}CONTROL_{(i)}$

w.r.t.

$<V, Id_{Vars(B) \cup Vars(CONTROL(k)) - V}>$

Proof of (xxiii)

This follows from the fact that all the variables of DATA' are within BASE and all the variables in V are outside it.

Appendix F : Propositions relating to scheduling and allocation

There are only two propositions in this section. Theorem 13 states that scheduling-and-allocation is behaviour-preserving. Theorem 14 implies that the chosen scheduling matrix [-1, 1, 2] for the QR-factorisation example is in some sense minimal (see subsection 5.2.1 on page 135).

The definitions and assumptions made at the start of the previous appendices are assumed to hold for this one. The following ones also hold:

Assumptions

There exists an invertible affine function **Im** which satisfies the following conditions:

The uniform dependencies of DATA are time-consistent with **Im**. (xxiv)

The vectors r_i, where $1 \leq i \leq n$, are time-consistent with **Im**. (xxv)

The vectors $r_{c\searrow i}$, where $1 \leq i \leq n$, are time-consistent with **Im** (xxvi)

Definitions

Depvecs: C $\rightarrow$ (the set of dependency vectors of C)

RENAME: <Varclass, p> $\rightarrow$ <Varclass, **Im**(p)>

EDGE' := EDGE ® $\text{RENAME}|_{\text{Vars(EDGE)}}$

INTERIOR' := INTERIOR ® RENAME$|_{\text{Vars(INTERIOR)}}$

Theorem 13

EDGE' || INTERIOR' is well-defined and simulates EDGE || INTERIOR

Proof

... direct from Lemma 21.

Theorem 14

(Recall from Chapter 4 that $\mathbf{Im}_t(p) := \mathbf{Im}(p)\downarrow_1$ and $\mathbf{Im}_t(p) = A_t.p + \mathbf{b}_t$.)

If $A_t = [\alpha, \beta, \gamma]$, α, β and γ are integral and

$$\text{Depvecs(DATA')} = \{ \begin{bmatrix}-1\\0\\-1\end{bmatrix}, \begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\-1\end{bmatrix}, \begin{bmatrix}0\\-1\\0\end{bmatrix} \}$$

then $|\alpha| \geq 1$, $|\beta| \geq 1$ and $|\gamma| \geq 2$.

Proof

All the vectors in Depvecs(DATA') must be time-consistent with **Im** (see section 3.4 (starting on page 65)) so $\alpha < 0$, $\beta > 0$, $\gamma > 0$ and $\alpha + \gamma > 0$. Therefore $\alpha \leq 1$, $\beta \geq 1$, $\gamma \geq 1$. But $\gamma > -a \geq 1$ so $\gamma \geq 2$.

Appendix G : Propositions relating to the whole design process

The main propositions in this section are Theorem 15, Theorem 20, andTheorem 27, which are the three conditions the output design must satisfy. They are proved using the subsidiary results of this section and the main results of the previous three sections. The diagram on page 110 may serve as a reminder of the roles of the various computations mentioned in this section.

The definitions and assumptions made at the start of the previous appendices are assumed to hold for this one. The following ones also hold:

Definitions

Let $M_{c\searrow i}$ be defined as follows:

$In(M_{c\searrow i}) \quad = \quad \{<c_i, p \to p + r_{c\searrow i}>\}$

$Out(M_{c\searrow i}) \quad = \quad \{<c_i, p \to p>\}$

$Rel(M_{c\searrow i})v \quad \Leftrightarrow$

$(p \in BASE \quad \Rightarrow \quad v(<c_i, p \to p>) = v(<c_i, p \to p + r_{c\searrow i}>))$

$ALG \quad := \quad CONTROL \parallel DATA$

$Varclasses(C) := \{varc : \text{there exists } p \text{ s.t. } <varc, p> \in C\}$

$D_{TOTAL} \quad := \quad BASE \cup \bigcup_{i \in Nat(n)} (D_{(i:0)} \cup D_{(i:1)})$

$CONTROL_{(i:1)(p)}$ is s.t.

$In(CONTROL_{(i:1)(p)}) \quad = \quad \emptyset$

$Out(CONTROL_{(i:1)(p)}) \quad = \quad \{<c_i, p>\}$

$Rel(CONTROL_{(i:1)(p)})v \quad \Leftrightarrow \quad (\quad p \in D_{(i:0)} \quad \Rightarrow \quad v(<c_i, p>) = 0$ and

$p \in D_{(i:1)} \Rightarrow v(<c_i, p>) = 1)$

if $p \in D_{(i:0)} \cup D_{(i:1)}$

and is the null computation if $p \in D_{TOTAL} - D_{(i:0)} \cup D_{(i:1)}$

$CONTROL_{(i:2)(p)}$ is s.t.

$In(CONTROL_{(i:2)(p)}) = v(<c_i, p>)$

$Out(CONTROL_{(i:2)(p)}) = \{<c_i, p>\}$

$Rel(CONTROL_{(i:2)(p)})v \Leftrightarrow (\quad p \in D_{(i:0)} \Rightarrow v(<c_i, p>) = 0$ and

$p \in D_{(i:1)} \Rightarrow v(<c_i, p>) = v(<c_i,$

$p+r_{c \searrow i}>))$

if $p \in$ BASE

and is the null computation if $p \in D_{TOTAL}$ - BASE

$CONTROL'''_{(p)} := ||_{i \in \{1...n\}} CONTROL_{(i:2)(p)}$

$EDGE_{(p)} := ||_{i \in \{1...n\}} CONTROL_{(i:1)(p)}$

DATA' is defined to be s.t.

$DATA' = ||_{p \in BASE} DATA'_{(p)}$ and

$DATA'_{(p)}$ is null when $p \in D_{TOTAL}$ - BASE

$INTERIOR_{(p)} = CONTROL'''_{(p)} \, || \, DATA'_{(p)}$

Let the interior of an embedded computation C_i, be

$Dom(C_i) - Edge(C_i)$

"the interior of C_i" may be written "$Int(C_i)$".

Assumptions

$\mathrm{CONTROL}_{(i:2)(p)} \parallel \mathrm{DATA'}_{(p)}$ is well-defined for all p in D_{TOTAL} and

$\parallel_{p\in D\setminus TOTAL}(\mathrm{CONTROL}_{(i:2)(p)} \parallel \mathrm{DATA'}_{(p)})$ is well-defined;

$\mathrm{EDGE}_{(p)}$ is well-defined for all p, and

$\parallel_{p\in D\setminus TOTAL}\mathrm{EDGE}_{(p)}$ is well-defined;

$\mathrm{EDGE}_{(p)} \parallel \mathrm{INTERIOR}_{(p)}$ is well-defined and

$\parallel_{p\in D\setminus TOTAL}(\mathrm{EDGE}_{(p)} \parallel \mathrm{INTERIOR}_{(p)})$ is well-defined;

$\mathrm{CONTROL'''}_{(p)}$ is well-defined and

$\parallel_{p\in D\setminus TOTAL}\ \mathrm{CONTROL'''}_{(p)}$ is well-defined;

$\parallel_{i\in\{1...n\}}(M_{c\setminus i})$ is well-defined;

$(\parallel_{i\in\{1...n\}}(M_{c\setminus i}) \parallel \mathrm{DATA\_M}_{(n)})$ is well-defined.

Theorem 15

EDGE' || INTERIOR' simulates ALG

Proof

...follows directly from Theorem 13 on page 248, Theorem 12 on page 243, Theorem 4 on page 220 and Lemma 27 on page 194.

Theorem 16

Recall the definition of "Edge" on page 52.

Let R(<var, p>) equal <var, I(p)> where I is invertible and R is defined, say, over Varclasses $\times$ D. Let computation C have domain D; then

$$\mathrm{Edge}(C \circledR R) = \mathrm{Ran}(\Gamma|_{\mathrm{Edge}(C)})$$

Proof

$$<var, p'> \in \mathrm{Out}(C \circledR R) \Leftrightarrow <var, \Gamma^{-1}(p')> \in \mathrm{Out}(C)$$

so

$$\begin{aligned} p' \in \mathrm{Edge}(C \circledR R) &\Leftrightarrow \text{for all var in Varclasses } <var, p'> \notin \mathrm{Out}(C \circledR R) \\ &\Leftrightarrow \text{for all var in Varclasses } <var, \Gamma^{-1}(p')> \notin \mathrm{Out}(C) \\ &\Leftrightarrow \Gamma^{-1}(p') \in \mathrm{Edge}(C) \\ &\Leftrightarrow p' \in \mathrm{Ran}(\Gamma|_{\mathrm{Edge}(C)}) \end{aligned}$$

Theorem 17

If $(\|_{i \in \{1...n\}} C_i)$is well-defined, then

$$\mathrm{Edge}(\|_{i \in \{1...n\}} C_i) = \mathrm{Dom}(\|_{i \in \{1...n\}} C_i) - \bigcup_{i \in \mathrm{Nat}(n)} \mathrm{Int}(C_i)$$

Proof

$p \in \mathrm{Edge}(\|_{i \in \{1...n\}} C_i)$

$\Leftrightarrow p \in \mathrm{Dom}(\|_{i \in \{1...n\}} C_i)$ and, for all var, $<var, p> \notin \mathrm{Out}(\|_{i \in \{1...n\}} C_i)$

$\Leftrightarrow p \in \mathrm{Dom}(\|_{i \in \{1...n\}} C_i)$ and, for all var, $1 \le i \le n \Rightarrow <var, p> \notin \mathrm{Out}(C_i)$

by definition of composition

$\Leftrightarrow p \in \mathrm{Dom}(\|_{i \in \{1...n\}} C_i)$ and $1 \le i \le n \Rightarrow p \notin \mathrm{Int}(C_i)$

by definition of "Int" on page 250

$\Leftrightarrow p \in \mathrm{Dom}(\|_{i \in \{1...n\}} C_i)$ and $p \notin \bigcup_{i \in \mathrm{Nat}(n)} \mathrm{Int}(C_i)$

$\Leftrightarrow p \in \mathrm{Dom}(\|_{i \in \{1...n\}} C_i) - \bigcup_{i \in \mathrm{Nat}(n)} \mathrm{Int}(C_i)$

Theorem 18

$DATA_1$ is a recurrence over BASE.

Proof

DATA' = $DATA_{(n)}$; from the definition of $DATA_{(k)}$ on page 197, it is sufficient to prove that $DATA\_M_{(n)}$ is of the right form to be a mould for a recurrence over BASE. It is sufficient to prove therefore that for all k in {1...n} $DATA\_M_{(k)}$ is of the right form to be a mould for a recurrence over BASE. We will prove this by induction on k with the inductive hypothesis, "$DATA\_M_{(k)}$ is of the right form to be a mould for a recurrence over BASE".

Base case

$DATA\_M_{(1)}$ is of the required form because DATA is a recurrence over BASE.

Inductive case

$\mathrm{Out}(DATA\_M_{(k)}) \quad =$

$\mathrm{Out}(DATA\_M_{(k-1)} \circledR RENAME_{(k-1)}) \cup \mathrm{Out}(PIPE\_M_{(k-1)})$

$= \quad (\mathrm{ran}(RENAME_{(k-1)}) |_{\mathrm{Out}(DATA\_M \searrow (k-1))}) \cup \mathrm{Out}(PIPE\_M_{(k-1)})$

by definition of renaming

$= \quad \mathrm{Out}(DATA\_M_{(k-1)}) \cup \{<z_k, Id_{BASE}>\}$

by definition of $RENAME_{(k-1)}$ and $PIPE\_M_{(k-1)}$

$\subseteq \quad (Varclasses \cup \{z_k\}) \times \{Id_{BASE}\}$

by the inductive hypothesis

$\mathrm{In}(DATA\_M_{(k)}) \quad =$

$\text{In}(\text{DATA\_M}_{(k-1)} \circledR \text{RENAME}_{(k-1)}) \cup \text{In}(\text{PIPE\_M}_{(k-1)}) - \text{Out}(\text{DATA\_M}_{(k)})$

$= (\text{In}(\text{DATA\_M}_{(k-1)}) - \{<a_{k-1}, \Delta_{k-1}>\})$

$\cup \{<z_k, \text{Id}_{\text{BASE}}>, <c_k, \text{Id}_{\text{BASE}}>, <z_k, p \rightarrow p+r_k>, <c_k, \text{Id}_{\text{BASE}}>\}$

$- \{<z_k, \text{Id}_{\text{BASE}}>\}$

by definition of $\text{RENAME}_{(k-1)}$, $\text{PIPE\_M}_{(k-1)}$ and $\text{DATA\_M}_{(k)}$

$= (\text{In}(\text{DATA\_M}_{(k-1)}) - \{<a_{k-1}, \Delta_{k-1}>\})$

$\cup \{<c_k, \text{Id}_{\text{BASE}}>, <z_k, p \rightarrow p+r_k>, <c_k, \text{Id}_{\text{BASE}}>\}$

which is of the required form, since $\text{In}(\text{DATA\_M}_{(k-1)})$ is a set of the required form (by the inductive hypothesis).

Theorem 19

(Recall the definition of "edge-computation" from page 53.)

EDGE is an edge-computation of INTERIOR.

Proof

Since $\text{Vars}(\text{EDGE}) = \bigcup_{i \in \text{Nat}(n)} \text{Vars}(\text{CONTROL}_{(i:1)})$, it is sufficient to prove that

$1 \leq i \leq n \quad \Rightarrow \text{Dom}(\text{CONTROL}_{(i:1)}) \subseteq \text{Edge}(\text{INTERIOR})$

by Theorem 17 on page 252

$\text{Edge}(\text{INTERIOR}) =$

$\text{Dom}(\text{INTERIOR}) - (\text{Int}(\text{DATA}') \cup (\bigcup_{i \in \text{Nat}(n)} \text{Int}(\text{CONTROL}_{(i:2)})))$

Now, for all i,

$$\text{Int}(\text{CONTROL}_{(i:2)}) = \text{BASE}$$

Also, since $\text{DATA}_{(k)}$ is a recurrence over BASE, $\text{Int}(\text{DATA}_{(k)}) \subseteq \text{BASE}$, since

$$\text{Out}(\text{DATA\_M}_{(k)}) \subseteq \text{Varclasses} \cup \{z_k\}\times\{\text{Id}_{\text{BASE}}\}$$

(so $\text{Out}(\text{DATA}_{(k)}) \subseteq \text{Varclasses} \cup \{z_k\}\times\{\text{BASE}\}$)

so

$$\text{Int}(\text{DATA'}) \cup (\bigcup_{i \in \text{Nat}(n)} \text{Int}(\text{CONTROL}_{(i:2)})) = \text{BASE}$$

and

$$\text{Edge}(\text{INTERIOR}) = \text{Dom}(\text{INTERIOR}) - \text{BASE}$$

$$\begin{aligned}\text{Dom}(\text{CONTROL}_{(i:1)}) &= D_{(i:0)} \cup D_{(i:1)} \\ &\subseteq \text{Dom}(\text{CONTROL}_{(i:2)}) \\ &\subseteq \text{Dom}(\text{INTERIOR})\end{aligned}$$

so it is s.t.p. that, for all i,

$\text{Dom}(\text{CONTROL}_{(i:1)}) \cap \text{BASE} = \emptyset$

but $D_{(i:0)} \cap \text{BASE} = D_{(i:1)} \cap \text{BASE} = \emptyset$ so, for all i,

$\text{Dom}(\text{CONTROL}_{(i:1)}) \cap \text{BASE}$

$= (D_{(i:0)} \cup D_{(i:1)}) \cap \text{BASE}$

$= \emptyset$

Theorem 20

EDGE' is an edge-computation of INTERIOR'.

Proof

$$\text{INTERIOR'} = \text{INTERIOR} \circledR \text{RENAME}$$

$$\text{EDGE'} = \text{EDGE} \circledR \text{RENAME}$$

and Theorem 20 is equivalent to saying that,

for all v,

$$v \in \text{Vars(EDGE')} \Rightarrow v = \text{<var, p>} \text{ for some p in Edge(INTERIOR')}$$

Now

$$\text{Vars(EDGE')} = \text{RENAME}|_{\text{Vars(EDGE)}}$$

and by Theorem 16 on page 251

$$\text{Edge(INTERIOR')} = \text{Ran}(\textbf{Im}|_{\text{Edge(INTERIOR)}})$$

so it is s.t.p. that EDGE is an edge-computation of INTERIOR for, if it is, then

$$\begin{aligned} v \in \text{Vars(EDGE')} &\Rightarrow v = \text{<var, } \textbf{Im}\text{(p)> and } p \in \text{Vars(EDGE)} \\ &\Rightarrow v = \text{<var, } \textbf{Im}\text{(p)> and } p \in \text{Edge(INTERIOR)} \\ &\Rightarrow v = \text{<var, p'> and } p' \in \text{Edge(INTERIOR')} \end{aligned}$$

so the result follows immediately from Theorem 19.

Theorem 21

If C is a UR and Vars(C) = dom(RENAME) then C ® RENAME is a UR.

Proof

$$C = \|_{p \in \text{BASE}}(\text{M} \circledR \text{RENAME}_{(p)})$$

where M is s.t.

$$\text{In(M)} = \{\text{<vn}_i, \Delta_i\text{>} : 1 \leq i \leq n\}$$

and

$$\text{Out(M)} \subseteq \text{Varclasses} \times \{\text{Id}_{\text{BASE}}\}$$

and

$$\text{RENAME}_{(p)}(\text{<vc, fun>})=(\text{<vc, fun(p)>}) \text{ for all <vc, fun> in Vars(M)}$$

where for all relevant i, is uniform ($\Delta_i : p \rightarrow p+r_i$), say. So

$$C \circledR \text{RENAME}=(\|_{p \in \text{BASE}}(\text{M} \circledR \text{RENAME}_{(p)})) \circledR \text{RENAME}$$

$$= \|_{p \in BASE}((M \circledR RENAME_{(p)}) \circledR RENAME|_{Vars(M \circledR RENAME\searrow(p))})$$

by Lemma 21 on page 182

$$= \|_{p \in BASE}(M \circledR (RENAME|_{Vars(M \circledR RENAME\searrow(p))} {}^{\bullet} RENAME_{(p)}))$$

by Lemma 26 on page 193

We want to show that this is equal to

$$\|_{p' \in BASE'}(M' \circledR RENAME'_{(p')})$$

(where $RENAME'_{(p')}(<vc, fun>) = (<vc, fun(p)>)$ for all $<vc, fun>$ in $Vars(M')$) for some suitable mould, M', and base, BASE'. Assume that

$\Delta : p \rightarrow p+r$

and let Δ' be s.t.

$\Delta' : p' \rightarrow p' + A.r$

(recalling from page 105 and page 68 that $\mathbf{Im} : p \rightarrow A.p + \mathbf{b}$)

Then let M' equal $M \circledR RENAME'$ where

$RENAME'(<vn, \Delta>)=<vn, \Delta'>$ for all $<vn, \Delta>$ in $Vars(M)$

Claim

$M \circledR (RENAME|_{Vars(M \circledR RENAME\searrow(p))} {}^{\bullet} RENAME_{(p)})$

$= \quad M' \circledR RENAME'_{(p')}$, where $p' = \mathbf{Im}(p)$

Proof of claim

$M' \circledR RENAME'_{(p')}$

$= \quad M \circledR RENAME' \circledR RENAME'_{(p')}$

$= \quad M \circledR (RENAME'_{(p')}|_{Vars(M \circledR RENAME')} {}^{\bullet} RENAME')$

by Lemma 26 on page 193

so it is sufficient to prove that

$RENAME|_{Vars(M \circledR RENAME\searrow(p))} {}^{\bullet} RENAME_{(p)}$

$= \quad \text{RENAME'}_{(p')}|_{\text{Vars}(M \circledR \text{RENAME'})} \bullet \text{RENAME'}$

$\text{RENAME}|_{\text{Vars}(M \circledR \text{RENAME} \searrow (p))} \bullet \text{RENAME}_{(p)}(<\text{vn}, \Delta>)$

$= \quad \text{RENAME}(<\text{vn}, \Delta(p)>)$

by definition of $\text{RENAME}_{(p)}$

$= \quad <\text{vn}, \mathbf{Im}(\Delta(p))>$

by definition of RENAME

$= \quad <\text{vn}, A.(\Delta(p)) + \mathbf{b}>$

by definition of **Im**

$= \quad <\text{vn}, A.p + A.r + \mathbf{b}>$ (xxvii)

$\text{RENAME'}_{(p')}|_{\text{Vars}(M \circledR \text{RENAME'})} \bullet \text{RENAME'}(<\text{vn}, \Delta>)$

$= \quad \text{RENAME'}_{(p')}(<\text{vn}, p \rightarrow p + A.r>)$

by definition of RENAME'

$= \quad <\text{vn}, p' + A.r>$

by definition of $\text{RENAME'}_{(p')}$

$= \quad <\text{vn}, A.p + \mathbf{b} + A.r>$ (xxviii)

The R.H.S.s of (xxvii) and (xxviii) are equal and so the claim has been proved. Let BASE' equal {p' : p' = **Im**(p) for some p in BASE} and M' be as in the claim, then the theorem follows directly.

<u>Lemma 36</u>

If, for all j and k, $\text{Out}(B_{j,k}) \subseteq C$, then

$$\bigcup_{k \in K} \Big(\bigcup_{j \in J} \text{In}(B_{j,k}) - \bigcup_{j \in J} \text{Out}(B_{j,k}) \Big) - C \;=\; \bigcup_{k \in K} \Big(\bigcup_{j \in J} \text{In}(B_{j,k}) \Big) - C$$

<u>Proof</u>

Easy.

Lemma 37

If $\|_{j\in J} P_{j,k}$ is well-defined for all k in K and $\|_{k\in K} P_{j,k}$ is well-defined for all j in J and $\|_{j\in J}(\|_{k\in K} P_{j,k})$ is well-defined

then $\|_{k\in K}(\|_{j\in J} P_{j,k})$ is well-defined and equals $\|_{j\in J}(\|_{k\in K} P_{j,k})$.

Proof

That $\mathrm{Out}(\|_{k\in K}(\|_{j\in J} P_{j,k})) = \mathrm{Out}(\|_{j\in J}(\|_{k\in K} P_{j,k}))$ is easily proved.

$$\begin{aligned}
\mathrm{In}(\|_{k\in K}(\|_{j\in J} P_{j,k})) &= \bigcup_{k\in K} \mathrm{In}(\|_{j\in J} P_{j,k}) - \mathrm{Out}(\|_{k\in K}(\|_{j\in J} P_{j,k}))\\
&= \bigcup_{k\in K} (\bigcup_{j\in J} \mathrm{In}(P_{j,k}) - \bigcup_{j\in J} \mathrm{Out}(P_{j,k})) - \mathrm{Out}(\|_{k\in K}(\|_{j\in J} P_{j,k}))\\
&\qquad \text{by definition of composition}\\
&= \bigcup_{k\in K} (\bigcup_{j\in J} \mathrm{In}(P_{j,k})) - \mathrm{Out}(\|_{k\in K}(\|_{j\in J} P_{j,k}))\\
&\qquad \text{by Lemma 36}\\
&= \bigcup_{j\in J} (\bigcup_{k\in K} \mathrm{In}(P_{j,k})) - \mathrm{Out}(\|_{j\in J}(\|_{k\in K} P_{j,k}))\\
&= \mathrm{In}(\|_{j\in J}(\|_{k\in K} P_{j,k}))
\end{aligned}$$

That $\mathrm{Rel}(\|_{k\in K}(\|_{j\in J} P_{j,k})) = \mathrm{Rel}(\|_{j\in J}(\|_{k\in K} P_{j,k}))$ is trivial (by the reverse of a similar sequence). $\|_{k\in K}(\|_{j\in J} P_{j,k})$ is well-defined since $\|_{j\in J}(\|_{k\in K} P_{j,k})$ is.

Theorem 22

Assume that, for all j in J, C_j is a UR over base BASE that $\|_{j\in J} C_j$ is well-defined, that $C_j = \|_{p\in \mathrm{BASE}}(M_j \circledR \mathrm{RENAME}_{(j\,:\,p)})$
where

$$\mathrm{In}(M_j) = \{<vn_{j,i},\ \Delta_{j,i}> : i = 1 \ldots n_j\}$$

$$\mathrm{Out}(M_j) = \mathrm{Varclasses} \times \{\mathrm{Id}_{BASE}\}$$

where

$$\Delta_{j,i} = p \rightarrow p + r_{j,i}$$

for j = 1 and 2

and RENAME(j : p) : $<vn,\ \Delta> \rightarrow <vn,\ \Delta(p)>$ for all $<vn,\ \Delta>$ in Vars(M_j)

$$(\mathrm{dom}(\mathrm{RENAME}_{(j\,:\,p)}) = \mathrm{Vars}(M_j))$$

and assume that $\|_{j\in J}\, M_j$ is well-defined

then

$\|_{j\in J}\, C_j$ is a UR over BASE.

Proof

$$(\|_{j\in\{1..n\}}\, C_j) = \|_{j\in\{1..n\}}\, \|_{p\in BASE}(M_j \circledR \mathrm{RENAME}_{(j\,:\,p)})$$

$$\|_{p\in BASE}(\|_{j\in\{1...n\}}\, (M_j \circledR \mathrm{RENAME}_{(j\,:\,p)})$$

by Lemma 37 on page 259

$$\|_{p\in BASE}((\|_{j\in\{1...n\}}\, M_j) \circledR \mathrm{RENAME}_{(p)})$$

where $\mathrm{dom}(\mathrm{RENAME}_{(p)}) = \mathrm{Vars}(\|_{j\in\{1...n\}}\, M_j)$

and $\mathrm{RENAME}_{(p)}|_{\mathrm{Vars}(M_j)} = \mathrm{RENAME}_{(j\,:\,p)}$

by Lemma 21 on page 182

Now

$$\mathrm{Out}(\|_{j\in\{1...n\}}\, M_j) \subseteq \mathrm{Varclasses} \times \{\mathrm{Id}_{BASE}\}$$

and

$$\mathrm{In}(\|_{j\in\{1...n\}}\, M_j) =$$

$$(\bigcup_{j\in J} \{<vn_{j,i},\ \Delta_{j,i}> : i = 1 \ldots n_j\}) - \mathrm{Out}(\|_{j\in\{1...n\}}\, M_j)$$

which is of the required form for a UR mould, since $\Delta_{j,i}$ is uniform, for all relevant <i, j>.

Theorem 23

If $\{<a_i, \Delta_i> : i = 1...n-1\}$ are the only non-uniform dependencies of DATA_M, then the set of non-uniform dependencies of $DATA\_M_{(k)}$ is $\{<a_i, \Delta_i> : k \leq i \leq n-1\}$

Proof

..by induction on k, the induction hypothesis being "the set of non-uniform dependencies of $DATA\_M_{(k-1)}$ is $\{<a_i, \Delta_i> : k-1 \leq i \leq n-1\}$".

Proof

Base case: k = 1

Trivial

Inductive case

We just need to consider $In(DATA\_M_{(k)})$

$In(DATA\_M_{(k)})$ =
$(In(DATA\_M_{(k-1)}) - \{<a_{k-1}, \Delta_{k-1}>\})$
$\cup \{<c_k, Id_{BASE}>, <z_k, p \rightarrow p + r_k>, <a_k, Id_{BASE}>\}$

The elements which are added are all uniform dependencies, so the set of non-uniform dependencies of $DATA\_M_{(k)}$ is
$\{<a_i, \Delta_i> : k \leq i \leq n-1\}$

Theorem 24

For all i in {1...n}, $CONTROL_{(i:2)}$ is a UR over BASE.

Proof

$In(CONTROL_{(i:2)}) = \{<c_i, p> : p \in D_{(i:0)} \cup D_{(i:1)}\}$

$Out(CONTROL_{(i:2)}) = \{<c_i, p> : p \in BASE\}$

$Rel(CONTROL_{(i:2)})v \Leftrightarrow$

$(p \in BASE \Rightarrow v(<c_i, p>) = v(<c_i, p+r_{c\searrow i}>))$

$CONTROL_{(i:2)} = \|_{p \in BASE}(M_{c\searrow i} \circledR RENAME_{(i:2:p)})$

where $RENAME_{(i:2:p)}$: $<vn, \Delta> \rightarrow <vn, \Delta(p)>$

and $dom(RENAME_{(i:2:p)}) = Vars(M_{c\searrow i})$

and it is a UR.

Theorem 25

CONTROL''' is a U.R. over BASE.

Proof

This follows directly from Theorem 22 and Theorem 24.

Theorem 26

DATA' is a uniform recurrence over BASE.

Proof

We know from Theorem 18 on page 253 that DATA' is a recurrence over BASE. By Theorem 23,

DATA' (= $\mathrm{DATA}_{(n)}$) is uniform,

since the set of non-uniform dependencies in $\mathrm{DATA\_M}_{(n)}$

$= \{<a_i, \Delta_i> : n \le i \le n-1\}$

$= \emptyset$

Theorem 27

INTERIOR' is a uniform recurrence.

Proof

This follows from Theorem 21, Theorem 22, Theorem 25 and Theorem 26.

Theorem 28

The dependency vectors of DATA' are time-consistent with **Im**.

Proof

Claim

The dependency vectors of $DATA_{(k)}$ (which correspond to the uniform dependencies of $DATA_{(k)}$) are time-consistent with **Im**.

Proof

...by induction on k, the induction hypothesis being, "The dependency vectors of $DATA_{(k)}$ are time-consistent with **Im.**"

Base case: k = 1

Recalling that $DATA_{(1)}$ = DATA, the result follows by assumption (xxiv) on page 247.

Inductive case

$In(DATA\_M_{(k)}) = \{<a_i, \Delta_i> : k \leq i \leq n-1\}$ by Theorem 23

so the set of dependency vectors of $DATA_{(k)}$ is the set of dependency vectors of $DATA_{(k-1)}$ (which by the inductive hypothesis are time-consistent with **Im**) united with $\{\underline{0}, r_k\}$, the vectors of which are time-consistent with **Im** by (xxv) on page 247.

So Theorem 28 is proved by the claim and noting that DATA' = $DATA_{(n)}$

Lemma 38

Assuming the definitions and hypotheses of Theorem 22, $Depvecs(||_{j\in J} C_j) \subseteq \bigcup_{j\in J} Depvecs(C_j)$.

Proof

$\mathrm{In}(\|_{j\in J} M_j) \quad \subseteq \quad \bigcup_{j\in J} \mathrm{In}(M_j)$

so

$b \in \mathrm{Depvecs}(\|_{j\in J} C_j)$

$\Leftrightarrow \; <vn, p \rightarrow p + b> \in \mathrm{In}(\|_{j\in J} M_j)$

by the definition of C_j

$\Rightarrow \; <vn, p \rightarrow p + b> \in \mathrm{In}(M_j)$ for some j in $\{1...n\}$

$\Leftrightarrow \; b \in \mathrm{Depvecs}(C_j)$ for some j in $\{1...n\}$

$\Leftrightarrow \; b \in \bigcup_{j\in J} \mathrm{Depvecs}(C_j)$

Theorem 29

The dependency vectors of CONTROL''' are time-consistent with **Im**.

Proof

Because Lemma 38 holds, it is s.t.p. that, for all i, the dependency vectors of $\mathrm{CONTROL}_{(i:2)}$ are time-consistent with **Im**.

Now

$\mathrm{CONTROL}_{(i:2)} = \|_{p\in \mathrm{BASE}}(M_{c\searrow i} ® \mathrm{RENAME}_{(i:2:p)})$

where

$\mathrm{In}(M_{c\searrow i}) \quad = \quad \{<c_i, p \rightarrow p + r_{c\searrow i}>\}$

$\mathrm{Out}(M_{c\searrow i}) \quad = \quad \{<c_i, p \rightarrow p>\}$

and

$\mathrm{Rel}(M_{c\searrow i})v \quad \Leftrightarrow$

$(p \in \mathrm{BASE} \quad \Rightarrow \; v(<c_i, p \rightarrow p>) = v(<c_i, p \rightarrow p + r_{c\searrow i}>))$

so the set of dependency vectors of $\mathrm{CONTROL}_{(i:2)}$ is $\{r_{c\searrow i}\}$ and we already know by (xxvi) on page 247 that is time-consistent with **Im**.

Lemma 39

If, for all i in some set I, C_i is an embedded computation s.t.

$\mathrm{Vars}(C_i) \subseteq \mathrm{Varclasses}_i \times D_i$

$C_i = \|_{p \in D \searrow i} C_{(i:p)}$

and

$\mathrm{Out}(C_{(i:p)}) \subseteq \mathrm{Varclasses}_i \times \{p\}$

and if $\|_{i \in I \searrow p} C_{(i:p)}$ and $\|_{p \in D}(\|_{i \in I \searrow p} C_{(i:p)})$ are well-defined (where $I_p = \{i : p \in D_i\}$), then $\|_{i \in I} C_i$ is an embedded computation satisfying with (2) and (1) on page 51, with

Varclasses	equal to	$\bigcup_{j \in I} \mathrm{Varclasses}_i$
D	equal to	$\bigcup_{j \in I} D_i$
and		
$C_{(p)}$	equal to	$\|_{i \in I \searrow p} C_{(i:p)}$

Proof

It is easy to see that (1) holds for C equal to $\|_{i \in I} C_i$. We know $\mathrm{Out}(\|_{i \in I \searrow p} C_{(i:p)}) \subseteq \bigcup_{i \in I_p} \mathrm{Out}(C_{(i:p)}) \subseteq \mathrm{Varclasses} \times \{p\}$; so to prove (2) it is s.t.p. that $\|_{i \in I} C_i = \|_{p \in D} C_{(p)}$.

Now

$$\|_{i \in I} C_i = \|_{i \in I}(\|_{p \in D \searrow i} C_{(i:p)})$$

$$= \|_{i \in I}(\|_{p \in D} C_{(i:p)})$$

where $C_{(i:p)}$ is the computation without variables, if $i \in D - D_i$

$$= \|_{p \in D}(\|_{i \in I} C_{(i:p)})$$

by Lemma 37 and well-definedness assumptions of this theorem

$= \|_{p \in D}(\|_{i \in I \searrow p}\, C_{(i\,:\,p)})$

Lemma 40

If dom(R) = Vars(C) and R : <var, p> $\rightarrow$ <var, f(p)>, where R is 1-to-1 from D to a Euclidean space D' and C is an embedded computation then C ® R is an embedded computation.

Proof

Let (2) and (1) on page 51 hold for C.

Let

Varclasses' equal Varclasses

and

D' equal $\{f(p) : p \in D\}$

Then Vars(C ® R) $\subseteq$ Varclasses' $\times$ D'
so (1) holds when C is replaced by C ® R and when Varclasses and C are replaced by Varclasses' and D' respectively. (2) can be deduced as follows:

From Lemma 21,

$(\|_{p \in D} C_{(p)}) ® R \quad = \quad \|_{p \in D}(C_{(p)} ® R|_{Vars(C\searrow(p))})$

and we know Out($C_{(p)}$ ® $R|_{Vars(C\searrow(p))}$) $\subseteq$ Varclasses $\times$ $\{f(p)\}$ so let C'(f(p)) be $C_{(p)}$ ® $R|_{Vars(C\searrow(p))}$. (This is unambiguous since f is 1-to-1.) Then $(\|_{p \in D} C_{(p)})$ ® R $=$ $\|_{p' \in D'} C'_{(p')}$ and so (2) holds.

Theorem 30

$CONTROL_{(i:1)}$ is an embedded computation with Varclasses equal to $\{c_i\}$, D equal to D_{TOTAL} and $C_{(p)}$ equal to $CONTROL_{(i:1)(p)}$.

Proof

(1) and (2) hold on page 51.

Theorem 31

EDGE is an embedded computation with

Varclasses	equal to	$\bigcup_{i \in Nat(n)}$ Varclasses($CONTROL_{(i:1)}$)
D	equal to	D_{TOTAL}
and		
$C_{(p)}$	equal to	$EDGE_{(p)}$

Proof

...directly from Theorem 30 and Lemma 39 with D_i equal to D_{TOTAL} and $C_{(i:p)}$ equal to $CONTROL_{(i:1)(p)}$

Theorem 32

For all i, $CONTROL_{(i:2)}$ is an embedded computation with

Varclasses	equal to	Varclasses($CONTROL_{(i:2)}$), which equals $\{c_i\}$
D	equal to	D_{TOTAL}
and		
$C_{(p)}$	equal to	$CONTROL_{(i:2)(p)}$

Proof

(1) and (2) hold on page 51.

Theorem 33

CONTROL''' is an embedded computation with

Varclasses	equal to	$\bigcup_{i \in Nat(n)}$ Varclasses($CONTROL_{(i:2)}$)
D	equal to	D_{TOTAL}
and		
$C_{(p)}$	equal to	$\|_{i \in \{1..n\}} CONTROL_{(i:2)(p)}$

Proof

...from Theorem 32 and Lemma 39.

Theorem 34

INTERIOR is an embedded computation with

Varclasses	equal to	Varclasses(CONTROL''') $\cup$ Varclasses(DATA')
D	equal to	D_{TOTAL}
and		
$C_{(p)}$	equal to	$INTERIOR_{(p)}$

Proof

...immediately from assumptions at the beginning of the chapter and Lemma 39.

Theorem 35

EDGE || INTERIOR is an embedded computation with

Varclasses	equal to	
		Varclasses(CONTROL'') $\cup$ Varclasses(INTERIOR)
D	equal to	D_{TOTAL}
and		
$C_{(p)}$	equal to	$EDGE_{(p)}$ || $INTERIOR_{(p)}$

Proof

...immediately from assumptions at the beginning of the chapter, Theorem 34 and Lemma 39.

Theorem 36

EDGE' || INTERIOR' is an embedded computation

Proof

...by Theorem 35 and Lemma 40.

Theorem 37

EDGE' || INTERIOR' is a space-time network. (It is obviously an embedded computation.)

Proof

EDGE' || INTERIOR' = (EDGE || INTERIOR)® RENAME

so, by the discussion starting on page 66, it is sufficient to prove that all the dependency vectors of INTERIOR are time-consistent with **Im**. But, by Lemma 38 and the fact that EDGE has no inputs, it is sufficient to prove that all the dependency vectors of CONTROL''' and all the dependency vectors of DATA' are time-consistent with **Im**. This follows directly from Theorem 28 and Theorem 29.

Since EDGE' has no inputs, it is sufficient to prove that INTERIOR is a space-time network. Because Lemma 38 holds, it is sufficient to prove that all dependency vectors of INTERIOR are time-consistent with **Im**. (see 3.4 on page 65). To prove this it is sufficient to prove that all the dependency vectors of CONTROL''' and all the dependency vectors of DATA' are time-consistent with **Im**. This follows from Theorem 28 and Theorem 29.

Appendix H : Proof of some of the well-definedness assumptions

In this appendix we will prove that the following computations are well-defined when $1 < i \leq n$.

$DATA\_M_{(i)}$ and $DATA_{(i)}$ (Theorem 38)

$CONTROL_{(i)}$ for i in {1...n} (Theorem 39)

$CONTROL_{(i)} \| DATA_{(i)}$ for i in {1...n} (Theorem 40)

$\|_{j \in \{1...i\}} CONTROL_{(j)}$

(Theorem 41)

$Varset_{(i)} \cap Vars(\|_{j \in \{1...i-1\}} CONTROL_{(j)}) = \emptyset$

(Theorem 42)

$(\|_{j \in \{1...i-1\}} CONTROL_{(j)}) \| (CONTROL_{(i)} \| DATA_{(i)})$,

(Theorem 43)

$((\|_{j \in \{1...i-1\}} CONTROL_{(j)}) \| (CONTROL_{(i)} \| DATA_{(i)})) \backslash Varset_{(i)}$,

(Theorem 44)

$(CONTROL_{(i)} \| DATA_{(i)}) \backslash Varset_{(i)}$, (Theorem 45)

$(\|_{j \in \{1...i-1\}} CONTROL_{(j)}) \| ((CONTROL_{(i)} \| DATA_{(i)}) \backslash Varset_{(i)})$

(Theorem 46)

$(\|_{j \in \{1...i\}} CONTROL_{(j)}) \| DATA_{(i)})$

(Theorem 47)

The well-definedness of these computations, stated in the assumptions on page 199, is required for the proof of Theorem 1 on page 200 and Theorem 2 on page 206.

The definitions and assumptions made in Appendix B, Appendix C and Appendix D will be assumed to hold.

Definitions

Let var and var' be distinct variables in Vars(C); var' depends on var relative to C if, for some valuations v' and v,

Rel(C)v' and Rel(C)v and $v|_{In(C) - var} = v'|_{In(C) - var}$ but $v(var') \neq v'(var')$

In other words, it is possible to affect the value of var' changing the value of just var, keeping the values of the other inputs constant.

Obviously, no variable depends on any output variable and no input variable depends on any other variable.

Let C_i be a computation when $i \in \{1...n\}$.

Let TotVars be $\bigcup_{i \in Nat(n)} Vars(C_i)$.

$$In(C_{(C)(i:p)}) := \emptyset$$

$$Out(C_{(C)(i:p)}) := \{<c_i, p>\}$$

$$Rel(C_{(C)(i:p)})v \Leftrightarrow \begin{array}{l} (v(<c_i, p>) = 1 \Leftrightarrow p \neq \Delta_i(p) \\ \text{and} \\ v(<c_i, p>) = 0 \Leftrightarrow p = \Delta_i(p)) \end{array}$$

$$C_{(D)(i:p)} := DATA\_M_{(i)}®R\_DATA_{(i:p)}$$

$$C_{(CD)(i:p)} := C_{(C)(i:p)} \parallel C_{(D)(i:p)}$$

Assumptions

Assume that there exists and integer function t on BASE s.t.

$t(p) \leq t(p') \Rightarrow \mathrm{In}(C_{(1:p)}) \cap \mathrm{Out}(C_{(1:p')}) = \emptyset$

and

r_i is s.t., for all p in BASE,

$t(p+r_i) < t(p)$

(In fact, if the assumptions of Appendix F on page 247 hold, then we may take t(p) to be $\mathbf{Im}_t(p)$.)

z_i is not in Varclasses($DATA\_M_{(i-1)}$) for all i in {1...n}.

Lemma 41

Let H be a non-empty subset of TotVars. (Note that H must therefore be finite.) If there is no sequence var_1 ... var_m s.t.

$var_1 = var_m$ and, for all j s.t. $1 < j < m$, there exists an i in {1...n} s.t. var_j depends on var_{j-1} relative to C_i

then

there exists var in H s.t. var doesn't depend on any other variable in H.

Proof

If every variable of H is dependent on some other relative to C_i for some i in {1...n}, then it is possible to construct an infinite sequence var_1, var_2, var_3 ... s.t.

var_i depends on var_{i-1} for all i s.t. $i \geq 2$. If $\text{var}_i = \text{var}_j$ for any i and j then there would be a loop which contradicts the aforementioned property. So there is an infinite chain of distinct variables, which contradicts H being finite.

Lemma 42

Preamble

Lemma 42 states that, if $\{C_i : i \in \{1...n\}\}$ has no dependency loops, its is possible to build up, one element at a time, from a given valuation (v_{in}) on $\text{In}(\|_{j \in \{1...n\}})$, a valuation ($v_k$) for which $\text{Rel}(\|_{j \in \{1...n\}})\text{val}_k$ holds.

Statement

If there is no sequence $\text{var}_1 \ldots \text{var}_m$ s.t.

$\text{var}_1 = \text{var}_m$ and, for all j s.t. $1 < j < m$, there exists an i in $\{1...n\}$ s.t. var_j depends on var_{j-1} relative to C_i

then

for all valuations v_{in} on $\text{In}(\|_{i \in \{1...n\}} C_i)$, there exists a chain

$\text{val}_1, \text{val}_2, \text{val}_3 \ldots \text{val}_k$, where $k = |\text{TotVars}| - |\text{dom}(v_{in})|$, $\text{val}_1 = v_{in}$,

val_k is a valuation on TotVars and, for all m from 1 to k inclusive,

$$\text{val}_m \subseteq \text{val}_k \qquad \text{(xxix)}$$

and

for all i in $\{1...n\}$, there exists v_i on $\text{Vars}(C_i)$ s.t. $\text{Rel}(C_i)v_i$

and $\mathrm{val}_m|_{\mathrm{Vars}(C\searrow i)\,\cap\,\mathrm{dom}(\mathrm{val}\searrow m)} \subseteq v_i$ (xxx)

Note that $\mathrm{Rel}(\|_{i \in \{1...n\}}C_i)\mathrm{val}_k$ follows from this conclusion. (Note also that v_i may vary with m.)

Proof

By induction on j, with the following inductive hypothesis:

"If $j \leq k$ then,

for all m less than j,

$\mathrm{val}_m \subseteq \mathrm{val}_j$ (xxxi)

for all m less than j,

val_m exists which satisfies (xxx) with val_m substituted for val_j (xxxii)

and

for all m less than j and for all var in $\mathrm{dom}(\mathrm{val}_m)$ and var' in TotVars - $\mathrm{dom}(\mathrm{val}_m)$,

var does not depend on var' relative to C_i for any i in {1...n} (xxxiii)

and

for all m less than j,

$|\mathrm{dom}(\mathrm{val}_m)| = |\mathrm{dom}(v_{in})| + m - 1$" (xxxiv)

Note that the inductive hypothesis implies (xxix) and (xxx) when j = k.

Base case

$\mathrm{val}_1 = v_{in}$. So

(xxxi) holds since $j=1$;

(xxxii) holds by Lemma 5 since $(\mathrm{dom}(v_{in}) \cap \mathrm{Vars}(C_i)) \subseteq \mathrm{In}(C_i)$

(xxxiii) holds since $var \in \mathrm{dom}(v_{in})$ implies that, for all i in $\{1...n\}$, $var \notin \mathrm{Out}(C_i)$; so for no i does var depend on another variable relative to C_i.

(xxxiv) holds trivially.

<u>Inductive case</u>

Assume that $j \leq k$. (The case where $j > k$ is trivial.) We assume that the inductive hypothesis holds with j replaced by j-1; so there exists val_{j-1} satisfying (xxxiii), (xxxii), (xxxiii), and (xxxiv) with j-1 substituted for j. We will now construct a val_j which satisfies (xxxiii), (xxxii), (xxxiii), and (xxxiv). Let $\{v_i' : i \in \{1...n\}\}$ be s.t., for all i in $\{1...n\}$,

$$val_{j-1}|_{\mathrm{Vars}(C{\searrow}i) \cap \mathrm{dom}(val{\searrow}(j-1))} \subseteq v_i'$$

(we know we can do this since, by the inductive hypothesis, (xxxii) holds with (j-1) substituted for j). Let H equal TotVars - $\mathrm{dom}(val_{j-1})$. $H \neq \emptyset$ since $j-1 < k$ so by Lemma 41, we may choose var_j in H s.t. var_j is not dependent on any other element of H relative to C_i for any i in $\{1...n\}$. Let g be s.t. var_j is in $\mathrm{Out}(C_g)$. Define v_g to be v_g'. Define v_i (where $i \neq g$) as follows:

<u>Case 1</u> $var_j \notin \mathrm{Vars}(C_i)$

$$v_i := v_i'$$

<u>Case 2</u> $var_j \in \mathrm{Vars}(C_i)$

In this case, $var_j \in \mathrm{In}(C_i)$, since $\mathrm{Out}(C_g) \cap \mathrm{Out}(C_i) = \emptyset$.

So let v_i be the extension by of $v_i'|_{\mathrm{In}(C{\searrow}i)}[var_j \rightarrow v_g(var)]$ s.t. $\mathrm{Rel}(C_i)v_i$; let

val_j be $val_{j-1} \cup <var_j, v_g(var)>$.

We will now show that (xxxiii), (xxxii), (xxxiii), and (xxxiv) hold.

Proof of (xxxiii)

This is trivial since $val_{j-1} \subseteq val_j$.

Proof of (xxxii)

It is s.t.p. that, for all i in {1...n},

$val_j \mid_{(Vars(C_i) \cap dom(val_j))} \subseteq v_i$

i.e., for all i in {1...n} and for all var in $(Vars(C_i) \cap dom(val_j))$,

$val_j(var) \quad = \quad v_i(var)$

By the inductive hypothesis we know that this holds when j is replaced by j-1 and v_i is replaced by v_i'. Now

$Vars(C_i) \cap dom(val_j) = \quad (Vars(C_i) \cap dom(val_{j-1})) \cup \{var_j\}$

and we know that, for all i in {1...n},

$val_j(var_j) = v_i(var_j)$

by definitions of val_j and v_i. So it is s.t.p. that for all i in {1...n}, var in $Vars(C_i) \cap dom(val_{j-1})$ implies

$val_j(var) \quad = \quad v_{j-1}(var)$ (xxxv)

and

$v_i(var) \quad = \quad v_i'(var)$ (xxxvi)

since we know that

$val_{j-1}(var) = \quad v_i'(var)$

(xxxv) is true by definition of val_j; (xxxvi) is trivially true if $var_j \notin Vars(C_i)$ or if $i = g$. Let's assume $var_j \in Vars(C_i)$ and $i \neq g$. So $var_j \in In(C_i)$ since

$Out(C_g) \cap Out(C_i) \quad = \quad \emptyset$

var does not depend on var_j (by the inductive hypothesis (with j replaced by j-1 in (xxxiii)) since $var \in dom(val_{(j-1)})$ and $var_j \in TotVars - dom(val_{(m)})$) and $Rel(C_i)v_i$ and $Rel(C_i)v_i'$ hold and

$$v_i|_{In(C_i)-var_j} = v_i'|_{In(C_i)-var_j}$$

so

$$v_i(var) = v_i'(var)$$

<u>Proof of (xxxiii)</u>

Assume $var \in dom(val_j)$ and $var' \in TotVars - dom(val_j)$. It is s.t.p. that var doesn't depend on var'. Well *either* $var \in dom(val_{j-1})$, in which case, since $TotVars - dom(val_j) \subseteq TotVars - dom(val_{j-1})$, var doesn't depend on var' relative to C_i (by the inductive hypothesis) *or* $var = var_j$ and var doesn't depend on var' relative to C_i because of the way we chose var_j.

<u>Proof of (xxxiv)</u>

(xxxiv) follows since

$$\begin{aligned} |dom(val_j)| &= |dom(val_{j-1})| + |\{var_j\}| \\ |dom(val_{j-1})| + 1 &= |dom(v_{in})| + (m-1) + 1 \\ &= |dom(v_{in})| + m - 1 \end{aligned}$$

◇

<u>Lemma 43</u>

If $\{C_i : i \in \{1...n\}\}$ is s.t. there is no sequence $var_1 \ldots var_m$ s.t. var_j depends on var_{j-1} (relative to C_i for some i in $\{1...n\}$) for all j s.t. $1 < j < m$ and $var_1 = var_m$ then $\|_{i \in \{1...n\}} C_i$ is well-defined.

Proof

It is sufficient to prove that for all v_{in} on $In(||_{i \in \{1...n\}}C_i)$, there exists a unique v_{out} s.t. $Rel(||_{i \in \{1...n\}}C_i)(v_{in} \cup v_{out})$.

Let us choose arbitrary v_{in}.

If var depends on var' relative to some C_i, then let us say var ➤ var'. This may be extended by transitivity. The full extension will be a partial ordering since var ➤ var' and var' ➤ var would together imply that var, var', var is a sequence which is assumed, in the assumption of the lemma, not to exist.

By Lemma 42, there exists v_{out} s.t. $Rel(||_{i \in \{1...n\}}C_i)v_{out}$ by letting v_{out} equal v_k. We now simply need to prove its uniqueness. Assume that there exists v_{out}' s.t. $v_{out}' \neq v_{out}$ but $Rel(||_{i \in \{1...n\}}C_i)(v_{in} \cup v_{out})$. So for some i in {1...n} $v_{out}'|_{Vars(C\searrow i)} \neq v_{out}|_{Vars(C\searrow i)}$ but $Rel(C_i)v_{out}'$ and $Rel(C_i)v_{out}$ so the well-definedness of C_i is contradicted.

Lemma 44

R is 1-to-1 and C®R is well-defined implies that

(var depends on var' relative to C) $\Leftrightarrow$

(R(var) depends on R(var') relative to C®R)

Proof

L.H.S. $\Rightarrow$

for some v and v', Rel(C)v' and Rel(C)v and

$v|_{In(C)\text{-}var} = v'|_{In(C)\text{-}var}$

but

$v(var') \neq v'(var')$

$\Rightarrow$

for some v and v' (the same ones) $Rel(C®R)v' \bullet R^{-1}$ and $Rel(C®R)v \bullet R^{-1}$ and

$v \bullet R^{-1}|_{In(C®R) - R(var)} \quad = \quad v' \bullet R^{-1}|_{In(C®R) - R(var)}$

but

$v \bullet R^{-1}(R(var')) \qquad \neq \quad v' \bullet R^{-1}(R(var))$

by definition of renaming

$\Rightarrow$ R.H.S.

We can prove that R.H.S. $\Rightarrow$ L.H.S. by a similar argument.

<u>Lemma 45</u>

If R is a function on Vars(C) s.t. $R|_{Out(C)}$ is 1-to-1 then is C®R well-defined.

<u>Proof</u>

(cf. proof of Lemma 4)

It is s.t.p. Lemma 45 for renaming functions R for which $R|_{Out(C)} = Id|_{Out(C)}$ since every other is the (functional) composition of such a function with a 1-to-1 renaming function (which is the identity on In(C)), and can then be proved using Lemma 4 and Lemma 25. By definition of what it means for C®R to be well-defined, it is s.t.p. that

(for all v and v', Rel(C®R)v and Rel(C®R)v'

$v'|_{In(C®R)} \quad = \quad v|_{In(C®R)}$

$$v'|_{Out(C®R)} \quad = \quad v|_{Out(C®R)}) \qquad \text{(xxxvii)}$$

and, for all valuations v_{in} on In(C®R), there exists v_{out} s.t. Rel(C®R)$v_{in} \cup v_{out}$

(xxxviii)

<u>proof of (xxxvii)</u>

As for (i) ofLemma 4

<u>proof of (xxxviii)</u>

Let v_{in}' equal v_{in}•R. Then there exists v_{out}' s.t. Rel(C)$v_{in}' \cup v_{out}'$ so Rel(C®R)($v_{in} \cup v_{out}'$•R$|_{Out(C®R)}{}^{-1}$), by definition of renaming.

Let v_{out} equal v_{out}'•R$|_{Out(C®R)}{}^{-1}$.

<u>Lemma 46</u>

If there exists an integer function t on BASE s.t.

$$t(p) < t(p') \quad \Rightarrow \quad Out(C_{(p')}) \cap In(C_{(p)}) \;=\; \emptyset$$

then

$||_{p \in BASE}C_{(p)}$ is well-defined.

<u>Proof</u>

...by Lemma 43

We will assume that the precondition of Lemma 43 doesn't hold for the set $\{C_{p\searrow i} : i \in \{1...n\}\}$, and derive a contradiction. The inverse of the precondition of Lemma 43 is equivalent to the statement that there exist var_1, ... var_n s.t. var_j depends on $var_{(j-1)mod(m-1)}$ relative to $C_{(p\searrow j)}$ for some p_j in BASE. So

$$\mathrm{Out}(C_{p\searrow((j-1)\mathrm{mod}(m-1))}) \cap \mathrm{In}(C_{(p\searrow j)}) \neq \emptyset$$

since $var_{j-1} \in \mathrm{Out}(C_{(p\searrow(j-1))})$ and $var_{j-1} \in \mathrm{In}(C_{(p\searrow j)})$

so

$$t(p_{((j-1)\mathrm{mod}(m-1))}) < t(p_j) \qquad \text{for all } j \text{ in } \{1...m\}$$

so

$$t(p_j) < t(p_j)$$

...a contradiction

So the precondition of Lemma 43 holds and the lemma may be applied to deduce that $\|_{p \in BASE} C_{(p)}$ is well-defined.

<u>Theorem 38</u>

For all i in {1...n}, $DATA\_M_{(i)}$ and $DATA_{(i)}$ are well-defined.

<u>Proof</u>

By induction on i using the inductive hypothesis,

"$DATA\_M_{(i)}$ and $DATA_{(i)}$ are well-defined and $R\_DATA_{(i\ :\ p)}|_{Out(DATA\_M\searrow(i))}$ is 1-to-1 and there exists and integer function t on BASE s.t.

$$t(p) \leq t(p') \Rightarrow \mathrm{In}(C_{(D)(i\,:\,p)}) \cap \mathrm{Out}(C_{(D)(i\,:\,p')}) = \emptyset\text{"}$$

<u>Base case</u>

$DATA_{(1)}$ (which equals DATA) and $DATA\_M_{(1)}$ are well-defined. DATA is an embedded computation so the outputs of $DATA\_M_{(1)}$ are all of the form $<var, Id_{BASE}>$, so $R\_DATA_{(1\ :\ p)}|_{Out(DATA\_M\searrow(1))}$ is 1-to-1. From the assumptions at the beginning of the appendix, starting on page 274, we know that there exists an integer function t on BASE s.t.

$$t(p) \leq t(p') \Rightarrow In(C_{(D)(1:p)}) \cap Out(C_{(D)(1:p')}) = \emptyset$$

Inductive case

Let us examine the definition of Rel(PIPE_M$_{(i)}$) on page 197. *, + and - are well-defined functions, so PIPE_M$_{(i)}$ is a well-defined computation. Let us assume the inductive hypothesis with i replaced by (i-1). It is s.t.p. that

DATA_M$_{(i-1)}$®R_DP$_{(i)}$ is well-defined (xxxix)

and that

PIPE_M$_{(i)}$ and DATA_M$_{(i-1)}$®R_DP$_{(i)}$ satisfy the condition for Lemma 43 (xl)

(i.e. where n = 2, PIPE_M$_{(i)}$ and C_2 = DATA_M$_{(i-1)}$®R_DP$_{(i)}$. If the condition for Lemma 43 is satisfied then DATA_M$_{(i-1)}$®R_DP$_{(i)}$ || PIPE_M$_{(i)}$ will be proven well-defined.)

and

(DATA_M$_{(i-1)}$®R_DP$_{(i)}$ || PIPE_M$_{(i)}$)®R_DATA$_{(i:p)}$ is well-defined (xli)

and

$||_{p \in BASE}$(DATA_M$_{(i-1)}$®R_DP$_{(i)}$ || PIPE_M$_{(i)}$)®R_DATA$_{(i:p)}$ is well-defined (xlii)

and

there exists an integer function t on BASE s.t.

$$t(p) \leq t(p') \Rightarrow In(C_{(D)(i:p)}) \cap Out(C_{(D)(i:p')}) = \emptyset$$

(xliii)

Proof of (xxxix)

This is easily proved by Lemma 4.

Proof of (xl)

It is s.t.p. an absence of dependency loops. The only variables of $DATA\_M_{(i-1)}\circledR R\_DP_{(i)} \parallel PIPE\_M_{(i)}$ which can possibly participate in a dependency loop are those in $(In(DATA\_M_{(i-1)}\circledR R\_DP_{(i)}) \cap Out(PIPE\_M_{(i)})) \cup (In(PIPE\_M_{(i)}) \cap Out(DATA\_M_{(i-1)}\circledR R\_DP_{(i)}))$, which equals $\{<a_i, Id_{BASE}>, <z_i, Id_{BASE}>\}$ (if $r_i \neq 0$). It is s.t.p. that $<a_i, Id_{BASE}>$ doesn't depend on $<z_i, Id_{BASE}>$ relative to $DATA\_M_{(i-1)}\circledR R\_DP_{(i)}$ (of which $<a_i, Id_{BASE}>$ is an output and $<z_i, Id_{BASE}>$ is an input) since then no dependency loop can be formed. By Lemma 44, it is s.t.p. that $<a_i, Id_{BASE}>$ doesn't depend on $<a_i, \Delta_i>$ relative to $DATA\_M_{(i-1)}$. This can be proved by induction, using the inductive hypothesis, "$j < i-1 \Rightarrow <a_i, Id_{BASE}>$ doesn't depend on $<a_i, Id_{BASE}>$ relative to $DATA\_M_{(j)}$" and Lemma 44. We need to assume, however that $<a_i, Id_{BASE}>$ doesn't depend on $<a_i, \Delta_i>$ relative to $DATA\_M_{(1)}$.

Proof of (xli)

$Out((DATA\_M_{(i-1)}\circledR R\_DP_{(i)}) \parallel PIPE\_M_{(i)})$

$= Out(DATA\_M_{(i-1)}\circledR R\_DP_{(i)}) \cup Out(PIPE\_M_{(i)})$

by definition of composition

$= Out(DATA\_M_{(i-1)}) \cup \{<z_i, Id_{BASE}>\}$

by definition of $R\_DP_{(i)}$ and $PIPE\_M_{(i)}$

$R\_DATA_{(i\,:\,p)}|_{Out((DATA\_M\searrow(i-1)\circledR R\_DP\searrow(i)) \parallel PIPE\_M\searrow(i))}$

$= R\_DATA_{(i\,:\,p)}|_{Out(DATA\_M\searrow(i-1))}[<z_i, Id_{BASE}> \rightarrow <z_i, p>]$

by the above re-writing

$$= \text{R\_DATA}_{((i-1):p)}|_{\text{Out}((\text{DATA\_M}\searrow(i-1))}[<z_i, \text{Id}_{\text{BASE}}> \rightarrow <z_i, p>]$$

This is 1-to-1, since by the inductive hypothesis $\text{R\_DATA}_{((i-1):p)}|_{\text{Out}((\text{DATA\_M}\searrow(i-1))}$ is 1-to-1, and z_i is not in Varclasses($\text{DATA\_M}_{(i-1)}$), by the assumption at the beginning of these appendices. Therefore, by Lemma 45,

$$(\text{DATA\_M}_{(i-1)}®\text{R\_DP}_{(i)} \parallel \text{PIPE\_M}_{(i)})®\text{R\_DATA}_{(i:p)}$$

is well-defined.

Proof of (xlii) and (xliii)

If we can prove (xliii), then (xlii) follows by Lemma 46 and the definition of $C_{(D)(i:p)}$ on page 273.

Assume that (xliii) holds with i replaced by i-1, and assume that

$$t(p) \quad < \quad t(p')$$

We will show that

$$\text{In}(C_{(D)(i:p)}) \cap \text{Out}(C_{(D)(i:p')}) \quad = \quad \emptyset \qquad \text{(xliv)}$$

We know

$$\text{In}(C_{(D)(i-1:p)}) \cap \text{Out}(C_{(D)(i-1:p')}) \quad = \quad \emptyset$$

Now if we can prove that

$$\text{In}(C_{(D)(i:p)}) \subseteq \text{In}(C_{(D)(i-1:p)}) \cup \{<c_i, p>, <z_i, p+r_i>\} \qquad \text{(xlv)}$$

and

$$\text{Out}(C_{(D)(i:p')}) \subseteq \text{Out}(C_{(D)(i-1:p')}) \cup \{<z_i, p'>\} \qquad \text{(xlvi)}$$

then since, by the assumptions at the start of this appendix, r_i is such that

$$t(p + r_i) \quad < \quad t(p) \text{ for all } p \text{ in BASE}$$

then

$$p' \neq p + r_i$$

so (xliv) holds.

<u>Proof of (xlvi)</u>

$$\begin{aligned}
\mathrm{Out}(C_{(D)(i:p')}) \quad = \quad & \\
& \mathrm{Out}((DATA\_M_{(i-1)} \circledR R\_DP_{(i)} \parallel PIPE\_M_{(i)}) \circledR R\_DATA_{(i:p)}) \\
& \qquad \text{by definition} \\
= \quad & \mathrm{ran}(R\_DATA_{(i:p')}|_{\mathrm{Out}((DATA\_M\searrow(i-1) \circledR R\_DP\searrow(i)) \parallel PIPE\_M\searrow(i))}) \\
& \qquad \text{by definition of renaming} \\
= \quad & \mathrm{ran}(R\_DATA_{(i:p)}|_{\mathrm{Out}(DATA\_M\searrow(i-1))}[<z_i, Id_{BASE}> \to <z_i, p>]) \\
& \qquad \text{by proof of (xli)} \\
= \quad & \mathrm{Out}(C_{(D)(i-1:p')}) \cup \{<z_i, p'>\} \\
& \qquad \text{by definition of } \mathrm{Out}(C_{(D)(i-1:p')})
\end{aligned}$$

<u>Proof of (xlv)</u>

$$\begin{aligned}
\mathrm{In}(C_{(D)(i:p)}) = {} & \mathrm{ran}(R\_DATA_{(i:p)}|_{\mathrm{In}(DATA\_M\searrow(i))}) - \mathrm{Out}(C_{(D)(i:p)}) \\
= \quad & \mathrm{ran}(R\_DATA_{(i:p)}|_{\mathrm{In}(DATA\_M\searrow(i))}) - (\mathrm{Out}(C_{(D)(i-1:p)}) \cup \{<z_i, p>\}) \\
& \qquad \text{by similar proof to that of (xlvi)}
\end{aligned}$$

$$\begin{aligned}
\mathrm{In}(DATA\_M_{(i)}) \quad \subseteq \quad & \mathrm{ran}(R\_DP_{(i)}|_{\mathrm{In}(DATA\_M\searrow(i-1))}) \\
& \cup \{<c_i, Id_{BASE}>, <z_i, p \to p+r_i>, <a_i, Id_{BASE}>\} \\
& - \mathrm{Out}(DATA\_M_{(i-1)} \circledR R\_DP_{(i)}) \cup \mathrm{Out}(PIPE\_M_{(i)}) \\
& \qquad \text{by definition of composition and } PIPE\_M_{(i)} \\
\subseteq \quad & \mathrm{ran}(R\_DP_{(i)}|_{\mathrm{In}(DATA\_M\searrow(i-1))}) \\
& \cup \{<c_i, Id_{BASE}>, <z_i, p \to p+r_i>\}
\end{aligned}$$

since $<a_i, Id_{BASE}> \in Out(DATA\_M_{(i-1)} ® R\_DP_{(i)})$

So

$$
\begin{aligned}
In(C_{(D)(i:p)}) \subseteq\ & ran(R\_DATA_{(i:p)}) \rfloor (ran(R\_DP_{(i)}|_{In(DATA\_M\searrow(i-1))}) \\
& \quad \cup \{<c_i, Id_{BASE}>, <z_i, p \rightarrow p+r_i>\})) \\
& \quad - (Out(C_{(D)(i-1:p)}) \cup \{<z_i, p>\}) \\
\subseteq\ & ran(R\_DATA_{(i:p)}) \rfloor \\
& (In(DATA\_M_{(i-1)}) \\
& \quad \cup \{<c_i, Id_{BASE}>, <z_i, p \rightarrow p+r_i>, <z_i, Id_{BASE}>\}) \\
& \quad - (Out(C_{(D)(i-1:p)}) \cup \{<z_i, p>\}) \\
& \qquad \text{by definition of } R\_DP_{(i)} \\
=\ & ran(R\_DATA_{(i-1:p)}) \rfloor In(DATA\_M_{(i-1)})) \\
& \quad \cup \{<c_i, p>, <z_i, p+r_i>, <z_i, p>\}) \\
& \quad - (Out(C_{(D)(i-1:p)}) \cup \{<z_i, p>\}) \\
& \qquad \text{by Lemma 6} \\
\subseteq\ & (In(C_{(D)(i-1:p)}) \cup Out(C_{(D)(i:p)}) \\
& \qquad \cup \{<z_i, p>\} \cup \{<c_i, p>, <z_i, p+r_i>\}) \\
& \qquad - \quad (Out(C_{(D)(i-1:p)}) \cup \{<z_i, p>\}) \\
& \qquad \text{by definition of renaming} \\
=\ & In(C_{(D)(i-1:p)}) \cup \{<c_i, p>, <z_i, p+r_i>\}
\end{aligned}
$$

<u>Theorem 39</u>

$CONTROL_{(i)}$ is well-defined for all i in {1...n}

<u>Proof</u>

It is s.t.p. that $Rel(CONTROL_{(i)})$ is functional i.e. for all valuations v_{in} on $In(CONTROL_{(i)})$, there exists a v_{out} s.t.

$Rel(CONTROL_{(i)})v_{in} \cup v_{out}$

and, for all valuations v on Vars(CONTROL$_{(i)}$),

$\mathrm{Rel}(\mathrm{CONTROL}_{(i)})v$ and $v|_{\mathrm{In}(\mathrm{CONTROL}\searrow(i))} = v_{in}$

$\Rightarrow$

$v|_{\mathrm{Out}(\mathrm{CONTROL}\searrow(i))} = v_{out}$

From the definition of Rel(CONTROL$_{(i)}$), there exists a unique v s.t. Rel(CONTROL$_{(i)}$)v. In(CONTROL$_{(i)}$) = $\emptyset$ so the above statements hold.

Lemma 47

If there exists a partial order $\succ$ on $\{C_i : i \{1...n\}\}$ such that

$\mathrm{In}(C_i) \cap \mathrm{Out}(C_j) \neq \emptyset \Rightarrow C_i \succ C_j$

then $||_{i \in \{1...n\}} C_i$ is well-defined.

Proof

It is s.t.p. the hypothesis of Lemma 43, assuming the existence of such a partial order. Assume that there exists a path $var_1 ... var_m$ such that var_j depends on var_{j-1} and $var_1 = var_m$. Let var_j be in the output of $C_{i\searrow j}$ so that var_j is in the input of $C_{i\searrow((j+1)\mathrm{mod}(m-1))}$; so

$C_{i\searrow j} \succ C_{i\searrow((j+1)\mathrm{mod}(m-1))}$ for all j in {1...m-1}

so

$C_{i\searrow 1} \succ C_{i\searrow 1}$

...a contradiction.

So there does not exist a path $var_1 ... var_m$ such that var_j depends on var_{j-1} and $var_1 = var_m$; so the hypothesis of Lemma 43 holds.

Theorem 40

$\mathrm{CONTROL}_{(i)} \parallel \mathrm{DATA}_{(i)}$ is well-defined for all i in {1...n}

<u>Proof</u>

This theorem follows from Lemma 47 and Theorem 39 since

$$\mathrm{In}(\mathrm{CONTROL}_{(i)}) \cap \mathrm{Out}(\mathrm{DATA}_{(i)}) \quad = \quad \emptyset$$

<u>Theorem 41</u>

$\parallel_{i \in \{1...n\}}\mathrm{CONTROL}_{(i)}$ is well-defined for all i in {1...n}

<u>Proof</u>

This theorem follows from Lemma 47 and Theorem 39 since

$$\mathrm{In}(\mathrm{CONTROL}_{(i)}) = \emptyset \qquad \text{for all i in } \{1...n\}$$

<u>Theorem 42</u>

$$\mathrm{Varset}_{(i)} \cap \mathrm{Vars}(\parallel_{j \in \{1...i-1\}}\mathrm{CONTROL}_{(j)}) \quad = \quad \emptyset \qquad \text{for all i in } \{2...n\}$$

<u>Proof</u>

$$\mathrm{Varclasses}(\parallel_{j \in \{1...i-1\}}\mathrm{CONTROL}_{(j)}) \quad = \quad \{c_j : 1 < j \leq i-1\}$$

and

$$\{\text{var : there exists p s.t. } <\text{var, p}> \in \text{Varset}\} \quad = \quad \{z_j, c_j, a_i\}$$

Theorem 42 follows.

Theorem 43

$(\|_{j \in \{1...i-1\}}\text{CONTROL}_{(j)})\|(\text{CONTROL}_{(i)}\|\text{DATA}_{(i)})$ is well-defined for all i in $\{2...n\}$.

Proof

$$\text{In}(\|_{j \in \{1...i-1\}}\text{CONTROL}_{(j)}) = \emptyset$$

so

$$\text{In}(\|_{j \in \{1...i-1\}}\text{CONTROL}_{(j)}) \cap \text{Out}(\text{CONTROL}_{(i)} \| \text{DATA}_{(i)}) = \emptyset$$

The theorem follows by Lemma 47.

Lemma 48

If, for all var in Varset and for all var' in Vars(C)-Varset, var' is not dependent on var relative to C, then C\Varset is well-defined.

Proof

Assume C\Varset is not well-defined. So by Lemma 24 there exist v and v' s.t.

Rel(C) v and Rel(C)v'

and

$$v|_{\text{In(C)- Varset}} = v'|_{\text{In(C)- Varset}}$$

but

$$v|_{\text{Out(C)- Varset}} \neq v'|_{\text{Out(C)- Varset}}$$

Let v and v' be such a pair with minimal number of differences on In(C). Let var on In(C) be s.t.

$v(var) \neq v'(var)$

Let v'' be s.t.

$v''|_{In(C) - \{var\}} = v'|_{In(C) - \{var\}}$

and

$v''(var) = v(var)$

and

Rel(C)v''

then

$v''|_{Out(C) - Varset} = v'|_{Out(C) - Varset}$

since no element of Out(C)-Varset depends on var; and, since the number of differences between v'' and v on In(C) is less than between v' and v and

$v''|_{In(C) - Varset} = v|_{In(C) - Varset}$

we know that

$v''|_{Out(C) - Varset} = v|_{Out(C) - Varset}$

so

$v|_{Out(C) - Varset} = v'|_{Out(C) - Varset}$

contradicting the assumption that v and v' are such a pair; so C\Varset must be well-defined.

Lemma 49

If

C_i has var as an input

and

for all var' in Out(C_i), var' doesn't depend on var relative to C_i

and

for all j in {1...n} s.t. $j \neq i$, var $\notin$ Vars(C_j)

and

$\|_{i \in \{1...n\}} C_i$ is well-defined

then

for all var' in $\mathrm{Out}(\|_{i \in \{1...n\}} C_i)$,

var' doesn't depend on var relative to $\|_{i \in \{1...n\}} C_i$

<u>Proof</u>

We know that var $\in \mathrm{In}(\|_{i \in \{1...n\}} C_i)$ since

var $\in \mathrm{In}(C_i)$

and

var $\in \bigcup_{j \neq i} \mathrm{Out}(C_j)$

. Let v and v' be s.t.

$\mathrm{Rel}(\|_{i \in \{1...n\}} C_i)$v

and

$\mathrm{Rel}(\|_{i \in \{1...n\}} C_i)$v'

and

$v \rfloor (\mathrm{In}(\|_{i \in \{1...n\}} C_i)\text{-var}) = v' \rfloor (\mathrm{In}(\|_{i \in \{1...n\}} C_i)\text{-var})$.

Now assume that v(var') $\neq$ v'(var') for some var' in $\mathrm{Out}(\|_{i \in \{1...n\}} C_i)$. Consider v'' defined s.t.

v''(var'') = v(var'') when var'' $\neq$ var

v''(var'') = v'(var'') when var'' = var

Now

$v'' \rfloor (\mathrm{In}(C_i)\text{-var}) \quad = \quad v \rfloor (\mathrm{In}(C_i)\text{-var})$

and

$v'' \rfloor (\mathrm{Out}(C_i)) \quad = \quad v \rfloor (\mathrm{Out}(C_i))$

so

$$Rel(C_i)v''\rceil Vars(C_i)$$

since no output of C_i depends on var.

But

$$v''\rceil In(\|_{i \in \{1...n\}}C_i) \quad = \quad v'\rceil In(\|_{i \in \{1...n\}}C_i)$$

since

$$v\rceil(In(\|_{i \in \{1...n\}}C_i)\text{-}var) = v'\rceil(In(\|_{i \in \{1...n\}}C_i)\text{-}var).$$

so $v'' = v'$ since $\|_{i \in \{1...n\}}C_i$ is well-defined. Hence

$$v(var') \quad = \quad v''(var') \quad = \quad v'(var')$$

...which contradicts the assumption that $v(var') \neq v'(var')$; so $v(var') = v'(var')$ for all var' in $Out(\|_{i \in \{1...n\}}C_i)$. Therefore, for all var' in $Out(\|_{i \in \{1...n\}}C_i)$, var' doesn't depend on var relative to $\|_{i \in \{1...n\}}C_i$.

<u>Lemma 50</u>

$$CONTROL_{(i)}\|DATA_{(i)} \quad = \quad \|_{p \in BASE}C_{(i:p)}$$

<u>Proof</u>

$$CONTROL_{(i)} \quad = \quad \|_{p \in BASE}C_{(C)(i:p)}$$

from the definition of $C_{(C)(i:p)}$ on page 273

$$DATA_{(i)} \quad = \quad \|_{p \in BASE}C_{(D)(i:p)}$$

from the definition of $C_{(D)(i:p)}$ on page 273

Now

$$In(CONTROL_{(i)}\|DATA_{(i)}) \quad = \quad In(\|_{p \in BASE}C_{(CD)(i:p)})$$

$$Out(CONTROL_{(i)}\|DATA_{(i)}) \quad = \quad Out(\|_{p \in BASE}C_{(CD)(i:p)})$$

$$Rel(CONTROL_{(i)}\|DATA_{(i)}) \Leftrightarrow Rel(\|_{p \in BASE}C_{(CD)(i:p)})$$

so it is sufficient to prove that $C_{(CD)(i\,:\,p)}$ is well-defined for all p. This can be done using Lemma 47: we may say

$C_{(D)(i\,:\,p)} \succ C_{(C)(i\,:\,p)}$

since

$In(C_{(C)(i\,:\,p)}) \cap Out(C_{(C)(i\,:\,p)}) \quad = \quad \emptyset$

Lemma 51

Let A be s.t. var $\notin$ Vars(A);

let C be s.t. var $\notin$ Vars(C) and In(C) = $\emptyset$

let B be s.t., for all var' in Out(B) and v, v', valuations on Vars(B || C),

Rel(B || C)v

and

Rel(B || C)v'

and

$v|_{In(B)\,-\,var} = v'|_{In(B)\,-\,var}$ implies v(var') = v'(var')

(Note that the condition involving B is a weaker one than non-dependency of var' on var relative to B, since Rel(B || C)v and Rel(B || C)v' must hold.)

Then for all var' in Out(A || B), var' doesn't depend on var relative to (A || B) || C.

Proof

Assume the contrary to Lemma 51, i.e. that there exist v, v' and var' in Out(A) s.t.

Rel(B || C)v

and

Rel(B || C)v'

and

$v|_{In(B) - var} = v'|_{In(B) - var}$

but $v(var') \neq v'(var')$

Let v'' be a valuation on Vars(B || C) s.t.

$v''(var'') = v(var'')$
(if $var'' \neq var$ and $var'' \in In(B)$ or $var'' \in Vars(C)$)

$v''(var'') = v'(var'')$
(if $var'' = var$)

and

$Rel(B)v''|_{Vars(B)}$

Since $In(C) = \emptyset$, we know that extending v'' to Var(B) from In(B) doesn't interfere with C. so let us do this in such a way that $Rel(B)v''|_{Vars(B)}$ holds; we know byLemma 5 that we can do this.

$Rel(B)v''|_{Vars(B)}$ and $Rel(C)v''|_{Vars(C)}$ and

$v''|_{Vars(B)-var} = v|_{Vars(B)-var}$

so

$v''(var') = v(var')$

by the assumption of the lemma.

We may extend v'' onto Vars(A) by stating that v''(var) = v'(var) for var in Vars(A). Then

$v''|_{In((A || B) || C)} = v'|_{In((A || B) || C)}$

since

$In(C) = \emptyset$,

and

Rel((A || B) || C)v''

and

$v''|_{Vars(B)-var} = v|_{Vars(B)-var} = v'|_{Vars(B)-var}$

and

$v''(var) = v'(var)$

so

$v''(var') = v'(var')$

since ((A || B) || C) is well-defined.

So

$v'(var') = v(var')$

...a contradiction. So Lemma 51 holds.

<u>Theorem 44</u>

$((||_{j \in \{1...i-1\}}CONTROL_{(j)})||(CONTROL_{(i)}||DATA_{(i)}))\backslash Varset_{(i)}$ is well-defined for all i in {2...n}.

<u>Proof</u>

By Lemma 48, it is sufficient to prove that

for all var in $Varset_{(i)}$

and

for all var' in

$Vars((||_{j \in \{1...i-1\}}CONTROL_{(j)})||(CONTROL_{(i)}||DATA_{(i)})) - Varset_{(i)}$

var' is not dependent on var relative to

$(||_{j \in \{1...i-1\}}CONTROL_{(j)})||(CONTROL_{(i)}||DATA_{(i)})$.

Since all the elements of $Varset_{(i)}$ are in

$Out((||_{j \in \{1...i-1\}}CONTROL_{(j)})||(CONTROL_{(i)}||DATA_{(i)}))$

except those in

$\{<z_i, p+r_i> : p \in BASE\} - \{<z_i, p> : p \in BASE\}$

it is sufficient to prove that

for all var in $\{<z_i, p+r_i> : p \in BASE\} - \{<z_i, p> : p \in BASE\}$

and

for all var' in $Out((||_{j \in \{1...i-1\}} CONTROL_{(j)}) || (CONTROL_{(i)} || DATA_{(i)}))$

var' doesn't depend on var relative to $(||_{j \in \{1...i-1\}} CONTROL_{(j)}) || (CONTROL_{(i)} || DATA_{(i)})$

Since var (which equals $<z_i, p+r_i>$, say) $\notin Vars(||_{j \in \{1...i-1\}} CONTROL_{(j)})$, by Lemma 49 it is sufficient to prove that, for all var' in $Out(CONTROL_{(i)} || DATA_{(i)})$, var' doesn't depend on var relative to $CONTROL_{(i)} || DATA_{(i)}$

(xlvii)

We will prove using Lemma 50; by Lemma 49 it is sufficient to prove

$<z_i, p+r_i> \notin Vars(C_{(CD)(i:p')})$ for all p' in BASE s.t. $p' \neq p$ (xlviii)

and

for all var' in $Out(C_{(CD)(i:p)})$, var' doesn't depend on $<z_i, p+r_i>$ (xlix)

Proof of (xlviii)

$$\begin{aligned}
Vars(C_{(D)(i:p')}) &= ran(R\_DATA_{(i:p')} | Vars(DATA\_M^{\setminus}(i))) \\
Vars(DATA\_M_{(i)}) &= Vars(DATA_{(i-1)} \circledR R\_DP_{(i)}) \cup Vars(PIPE\_M_{(i)}) \\
&\subseteq Vars(DATA\_M_{(i-1)}) \cup \{<z_i, Id_{BASE}>\} \\
&\quad \cup \{<c_i, Id_{BASE}>, <z_i, p \rightarrow p+r_i>, \\
&\qquad <z_i, Id_{BASE}>, <a_i, Id_{BASE}>\}
\end{aligned}$$

So

$$\begin{aligned}
Vars(C_{(D)(i:p')}) &\subseteq ran(R\_DATA_{(i:p')} | Vars(DATA\_M^{\setminus}(i-1))) \\
&\quad \cup \{<c_i, p'>, <z_i, p'+r_i>, <z_i, p'>, <a_i, p'>\}
\end{aligned}$$

by Lemma 6 and Lemma 17

So

$$\mathrm{Vars}(C_{(CD)(i\,:\,p')}) \subseteq \mathrm{Vars}(C_{(D)(i\,:\,p')}) \cup \mathrm{Vars}(C_{(C)(i\,:\,p')})$$
$$\subseteq \{<c_i, p>, <z_i, p+r_i>, <z_i, p>, <a_i, p>\}$$
$$\cup\ \mathrm{ran}(R\_DATA_{(i\,:\,p)}|_{\mathrm{Vars}(DATA\_M\searrow(i-1))})$$

(xlviii) follows from the fact that

$p \neq p'$,

$p + r_i \notin BASE$ (so $p + r_i \neq p'$)

and

$z_i \notin \mathrm{Varclasses}(DATA\_M_{(i-1)})$

<u>Proof of (xlix)</u>

<u>Case 1</u> $var' \in \mathrm{Out}(C_{(C)(i\,:\,p)})$

So $var' = <c_i, p>$. Let v and v' be s.t.

$$v|_{\mathrm{In}(C\searrow(CD)(i\,:\,p))\,-\,var} = v'|_{\mathrm{In}(C\searrow(CD)(i\,:\,p))\,-\,var}$$

$$v(<c_i, p>) = 0 \Leftrightarrow p = \Delta_i(p)$$
$$\Leftrightarrow v'(<c_i, p>) = 0$$

$$v(<c_i, p>) = 1 \Leftrightarrow p \neq \Delta_i(p)$$
$$\Leftrightarrow v'(<c_i, p>) = 1$$

(In fact $p = \Delta_i(p)$ since, if not, by assumption at start of Appendix D,

$p + r_i \in \mathrm{Coset}_i(p)$

which implies

$p + r_i \in BASE$

...a contradiction)

So $v(var') = v'(var') = 0$

Case 2

var' $\in$ Out($C_{(D)(i:p)}$)

Now

$C_{(D)(i:p)}$ = (DATA$_{(i-1)}$®R_DP$_{(i)}$®R_DATA$_{(i:p)}$) ||
(PIPE_M$_{(i)}$®R_DATA$_{(i:p)}$)

<z_i, p> $\notin$ (DATA$_{(i-1)}$®R_DP$_{(i)}$®R_DATA$_{(i:p)}$)

and

In($C_{(C)(i:p)}$) = $\emptyset$

So, by Lemma 51 with

A equal to DATA$_{(i-1)}$®R_DP$_{(i)}$®R_DATA$_{(i:p)}$

B equal to PIPE_M$_{(i)}$®R_DATA$_{(i:p)}$

and

C equal to $C_{(C)(i:p)}$,

it is sufficient to prove...

Claim

For all var' in Out(PIPE_M$_{(i)}$®R_DATA$_{(i:p)}$) and v, v' valuations on Vars(PIPE_M$_{(i)}$®R_DATA$_{(i:p)}$ || $C_{(C)(i:p)}$),

if

Rel(PIPE_M$_{(i)}$®R_DATA$_{(i:p)}$ || $C_{(C)(i:p)}$)v

and

Rel(PIPE_M$_{(i)}$®R_DATA$_{(i:p)}$ || $C_{(C)(i:p)}$)v'

and

v$\rfloor$(In(PIPE_M$_{(i)}$®R_DATA$_{(i:p)}$)-<z_i, p + r_i> =
v$\rfloor$(In(PIPE_M$_{(i)}$®R_DATA$_{(i:p)}$)-<z_i, p + r_i>

then

$v(\text{var'}) = v'(\text{var'})$

Proof of claim

var' must be $<z_i, p>$, since this is the only element of PIPE_M$_{(i)}$®R_DATA$_{(i:p)}$. Now

$$\begin{aligned} v(<z_i, p>) &= v(<z_i, p+r_i>)*v(<c_i, p>) + \overline{v}(<c_i, p>)*v(<a_i, p>) \\ &= v(<a_i, p>) \\ &\qquad \text{since } v(<c_i, p>) = 0 \\ &= v'(<a_i, p>) \\ &= v'(<z_i, p+r_i>)*v'(<c_i, p>) + \overline{v'}(<c_i, p>)*v'(<a_i, p>) \\ &\qquad \text{since } v(<c_i, p>) = 0 \\ &= v'(<z_i, p>) \end{aligned}$$

Theorem 45

(CONTROL$_{(i)}$ || DATA$_{(i)}$) \ Varset$_{(i)}$ is well-defined when $1 < i \leq n$

Proof

It is sufficient to prove that, for all var in Varset$_{(i)}$ and for all var' in Out((CONTROL$_{(i)}$ || DATA$_{(i)}$) \ Varset$_{(i)}$), var' doesn't depend on var relative to (CONTROL$_{(i)}$ || DATA$_{(i)}$). This is a corollary of in Theorem 44.

Theorem 46

($||_{j \in \{1...i-1\}}$CONTROL$_{(j)}$) || ((CONTROL$_{(i)}$ || DATA$_{(i)}$)\Varset$_{(i)}$)

when $1 < i \leq n$

Proof

...by Lemma 47 with

$(\|_{j \in \{1...i-1\}} \mathrm{CONTROL}_{(j)}) \succ ((\mathrm{CONTROL}_{(i)} \| \mathrm{DATA}_{(i)}) \backslash \mathrm{Varset}_{(i)})$

Theorem 47

$(\|_{j \in \{1...i\}} \mathrm{CONTROL}_{(j)}) \| \mathrm{DATA}_{(i)})$ when $1 < i \leq n$

Proof

...by Lemma 47 with $(\|_{j \in \{1...i\}} \mathrm{CONTROL}_{(j)}) \succ \mathrm{DATA}_{(i)}$