Development of a Distributed Min/Max Component

Max Fuchs, Ketil Stølen
Fakultät für Informatik
Technische Universität München
D-80290 München, Arcisstr. 21
e-mail: fuchs,stoelen@informatik.tu-muenchen.de

Abstract

We introduce a specification technique and a refinement calculus for networks of components communicating asynchronously via unbounded FIFO channels. Specifications are formulated in a relational style. The given refinement rules allow modular system development in a step-wise, top-down manner. We employ the proposed formalism to specify and develop a so-called Min/Max Component. First an overview of the whole design process is given. Then certain steps of the development are described in detail.

1 Introduction

Focus [BDD$^+$93] is a general framework, in the tradition of [Kah74], [Kel78], for the formal specification and development of distributed systems. A system is modeled by a network of components working concurrently, and communicating asynchronously via unbounded FIFO channels. A number of reasoning styles and techniques are supported. Focus provides mathematical formalisms which support the formulation of highly abstract, not necessarily executable specifications with a clear semantics. Moreover, Focus offers powerful refinement calculi which allow distributed systems to be developed in the same style as the methods presented in [Jon90], [Bac88], [Mor90] allow for the development of sequential programs. Finally, Focus is modular in the meaning that design decisions can be checked at the point where they are taken, that component specifications can be developed in isolation, and that already completed developments can be reused in new program developments.

This paper presents a new style of reasoning inside the Focus framework [BS94], [BS95]. The objective of this paper is to explain how the proposed formalism can be employed in practical system design. It is shown how an abstract requirement specification can be refined into a concrete implementation using compositional refinement techniques.

Section 2 introduces the underlying formalism. In Section 3 it is explained what we mean by a specification. Moreover, a number of composition operators are defined and some simple refinement rules are formulated. The development of the Min/Max Component is the subject of Section 4. Section 5 gives a brief conclusion.

2 Underlying Formalism

N denotes the set of natural numbers, N^+ denotes $\mathsf{N} \setminus \{0\}$. For any set S, $\wp(S)$ denotes the set of all nonempty subsets of S. We assume the availability of the usual logical operators and the standard set operators including min and max for sets of natural numbers. As usual, $\Rightarrow$ binds weaker than $\wedge, \vee, \neg$ which again bind weaker than all other operators and function symbols.

A stream is a finite or infinite sequence of actions. It models the history of a communication channel, i.e. it represents the sequence of messages sent along the channel. Given a set of actions D, D^* denotes the set of all finite streams generated from D; D^∞ denotes the set of all infinite streams generated from D, and D^ω denotes $D^* \cup D^\infty$.

If $d \in D$, $r, s \in D^\omega$ and $j \in \mathsf{N}$, then:

- ϵ denotes the empty stream;
- $\#r$ denotes the length of r, i.e. ∞ if r is infinite, and the number of elements in r otherwise;
- $\mathsf{dom}.r$ denotes N^+ if $\#r = \infty$, and $\{1, 2, \ldots, \#r\}$ otherwise;
- $r[j]$ denotes the j'th element of r if $j \in \mathsf{dom}.r$;
- $\mathsf{rng}.r$ denotes $\{r[j] | j \in \mathsf{dom}.r\}$;
- $r|_j$ denotes the prefix of r of length j if $j < \#r$, and r otherwise;
- $d \,\&\, s$ denotes the result of appending d to s;
- $r \sqsubseteq s$ denotes that r is a prefix of s.

A formula P is a safety formula iff it is prefix-closed and admissible, i.e. whenever it holds for a stream tuple s, then it also holds for any prefix of s, and whenever it holds for each element of a chain, then it also holds for the least upper bound of the chain. $\mathsf{sft}(P)$ holds iff P is a safety formula.

For formulas we need a substitution operator. Given a variable a and term t, then $P[{}^a_t]$ denotes the result of substituting t for every free occurrence of a in P. The operator is generalized in an obvious way in the case that a and t are lists.

3 Specification and Refinement

A specification of a component with n input channels and m output channels is an expression of the form

$$\mathsf{spec}\ \mathrm{S} :: i : I \rhd o : O \equiv R$$

S is the specification's name; i is a list of n input identifiers (with corresponding types I); o is a list of m output identifiers (with corresponding types O); R is a formula with the elements of i and o as its only free variables.

It is assumed that i and o are disjoint and without repetitions. i and o name the n input and m output channels, respectively. In R each such identifier represents a stream modelling the complete communication history of the channel named by the identifier. Thus R characterizes the relationship between the complete communication histories of the input channels and the complete communication histories of the output channels. For any specification with name S we refer to its corresponding formula as R_{S}.

The operator $\otimes$ can be used to compose two specifications by connecting any output channel of the former to an identically named input channel of the latter, and by connecting any output channel of the latter to an identically named input channel of the former. For example, if $\otimes$ is used to compose the specifications S_1 and S_2 with respectively $(i, x)/(o, y)$ and $(y, r)/(x, s)$ as input/output identifiers, then the output channels denoted by y of S_1 are connected to the identically named input channels of S_2, and the output channels denoted by x of S_2 are connected to the identically named input channels of S_1, as indicated in Figure 1. The composite specification has $(i, r)/(o, s)$ as input/output identifiers. Thus the identifiers of the lists x and y are now hidden in the sense that they represent local channels.

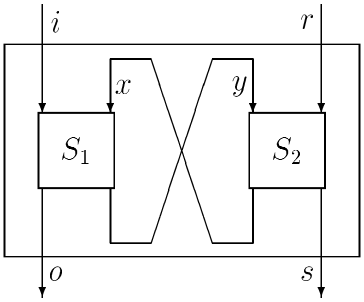

Figure 1: $S_1 \otimes S_2$.

When using $\otimes$ to build networks of specifications one will often experience that the operator needed is not $\otimes$, but a slight modification of $\otimes$, where for example there are no input channels corresponding to r, no output channels corresponding to o, or the channels represented by x are not hidden. Instead of introducing a new operator (and a new refinement rule) for each possible variation, we overload and use $\otimes$ for all of them, with two exceptions. To increase the readability, we use $\|$ instead of $\otimes$ when there are no feedback channels, i.e. no channels corresponding to x and y in Figure 1, and ; instead of $\otimes$ in the case of sequential composition, i.e. when there are no channels corresponding to o, r and x. Whenever $\otimes$ is used it will always be clear from the context which version is the intended one. We will refer to ;, $\|$ and $\otimes$ as sequential composition, parallel composition and mutual feedback, respectively.

A specification S_2 refines another specification S_1, written

$$S_1 \leadsto S_2$$

iff any behavior allowed by S_2 is also an allowed behavior of S_1. Given a requirement specification *Spec*, the goal of a system development is to construct a network of components A such that $\mathit{Spec} \leadsto A$ holds. The refinement relation $\leadsto$ is reflexive, transitive and a congruence w.r.t. the composition operators. Hence, $\leadsto$ allows compositional system development: once a specification is decomposed into a network of subspecifications, each of these subspecifications can be further refined in isolation. For a more formal treatment of specification, composition and refinement, see [BS94], [BS95].

The next step is to explain how refinements can be proved correct. Here we will present 4 rules altogether. For more rules see [FS93]. All rules should be understood as follows: whenever each premise is valid, then the conclusion is valid. Thus, there is no binding between the input/output observables of two different premises.

The first three rules are easy to understand:

Rule 1 :

$$\frac{\begin{array}{l} S_1 \rightsquigarrow S_2 \\ S_2 \rightsquigarrow S_3 \end{array}}{S_1 \rightsquigarrow S_3}$$

Rule 2 :

$$\frac{S_1 \rightsquigarrow S_2}{S \rightsquigarrow S(S_2/S_1)}$$

Rule 3 :

$$\frac{R_{S_2} \Rightarrow R_{S_1}}{S_1 \rightsquigarrow S_2}$$

Rule 1 and 2 state that $\rightsquigarrow$ is transitive and a congruence. $S(S_2/S_1)$ denotes the result of substituting S_2 for one occurrence of S_1 in the network of specifications S. Rule 3 is a traditional consequence rule. It is assumed that the two specifications have the same input/output identifiers.

If S_1 and S_2 have lists of input/output identifiers as in Figure 1, then the rule for mutual feedback looks as follows:

Rule 4 :

$$\frac{\begin{array}{l} \mathsf{sft}(I_1) \wedge \mathsf{sft}(I_2) \\ I_1\genfrac{[}{]}{0pt}{}{x}{\epsilon} \wedge I_2\genfrac{[}{]}{0pt}{}{y}{\epsilon} \\ I_1 \wedge R_{S_1} \Rightarrow I_2 \\ I_2 \wedge R_{S_2} \Rightarrow I_1 \\ I_1 \wedge R_{S_1} \wedge I_2 \wedge R_{S_2} \Rightarrow R_S \end{array}}{S \rightsquigarrow S_1 \otimes S_2}$$

Recall that $\mathsf{sft}(P)$ holds if P is a safety formula. I_1 and I_2 are formulas with the elements of i, r, x and i, r, y as their only free variables (see Figure 1), respectively. This rule is closely related to the while-rule of Hoare-logic. I_1 and I_2 can be thought of as invariants. The first, third and fourth premise imply that when the invariants hold after n computation steps then the invariants also hold after $n+1$ computation steps. (Note that since our specifications only constrain the behavior for infinite inputs, this does not follow without the first premise, i.e. without the fact that I_1 and I_2 are safety formulas.) By induction on n, the second premise then implies that the invariants hold after any finite number of computation steps, in which case the first premise can be used to infer that the invariants hold for any computable fixpoint. The conclusion can now be deduced from premise five. See [BS94], [SDW95] for a more detailed discussion.

4 Design of a Min/Max Component

We want to specify and formally develop a component with two input channels ia and ib, and two output channels mn and mx, as shown in Figure 2.

For each natural number the component reads from one of its input channels, it is required to output the minimum and the maximum received so-far along mn and mx, respectively. There are no constraints on the order in which the component switches from processing

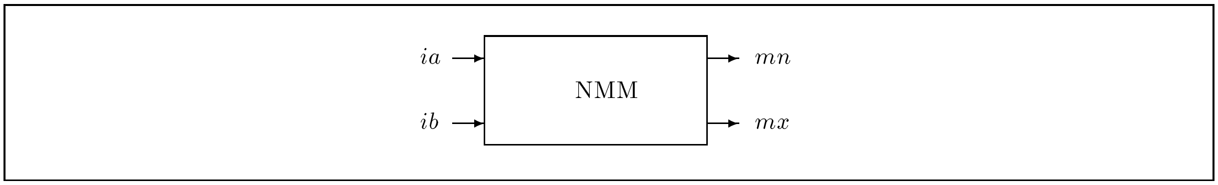

Figure 2: Min/Max Component.

inputs received on *ia* to processing inputs received on *ib*, and back again. However, it is required that all input messages eventually are read and processed. We will refer to this component as NMM (for Nondeterministic Min/Max).

To allow for an implementation where each channel is refined by a tuple of channels all of type Bit, we restrict the natural numbers received on the input channels to be less than 2^{BW}, where BW is a constant representing the bit width.

The development is conducted in a step-wise fashion:

- First the component NMM is formally specificed in Focus.
- This specification is then decomposed into the network of four component specifications pictured in Figure 4.
- Then the FILTER components of Figure 4 are decomposed into networks of two components in accordance with Figure 5. At the end of this step we have a network consisting of six component specifications and eleven channels.
- Rules for interface refinement are then used to replace each of these eleven channels carrying naturals with BW channels of type Bit.
- We then take advantage of the interface refinement of the previous step and conduct six structural decompositions. The resulting network is pictured in Figure 3. The dashed box containing the BFM specifications refines the FM specification of Figure 4, the dashed box containing the BCY specifications refines the COPY specification of Figure 4, and so on.
- Each of these Bit-level specifications are then transformed into a certain state-machine oriented form.
- The resulting specification is translated into the specification language SDL.

Because of the space-constraints only the steps under the three first "bullets" are shown below. For more details we refer to [FS93][1].

4.1 Requirement Specification

The requirement specification characterizes the black-box behavior of the Min/Max Component — in other words: the components external behavior. Given that

[1] Can be copied from: http://www4.informatik.tu-muenchen.de/BERICHT

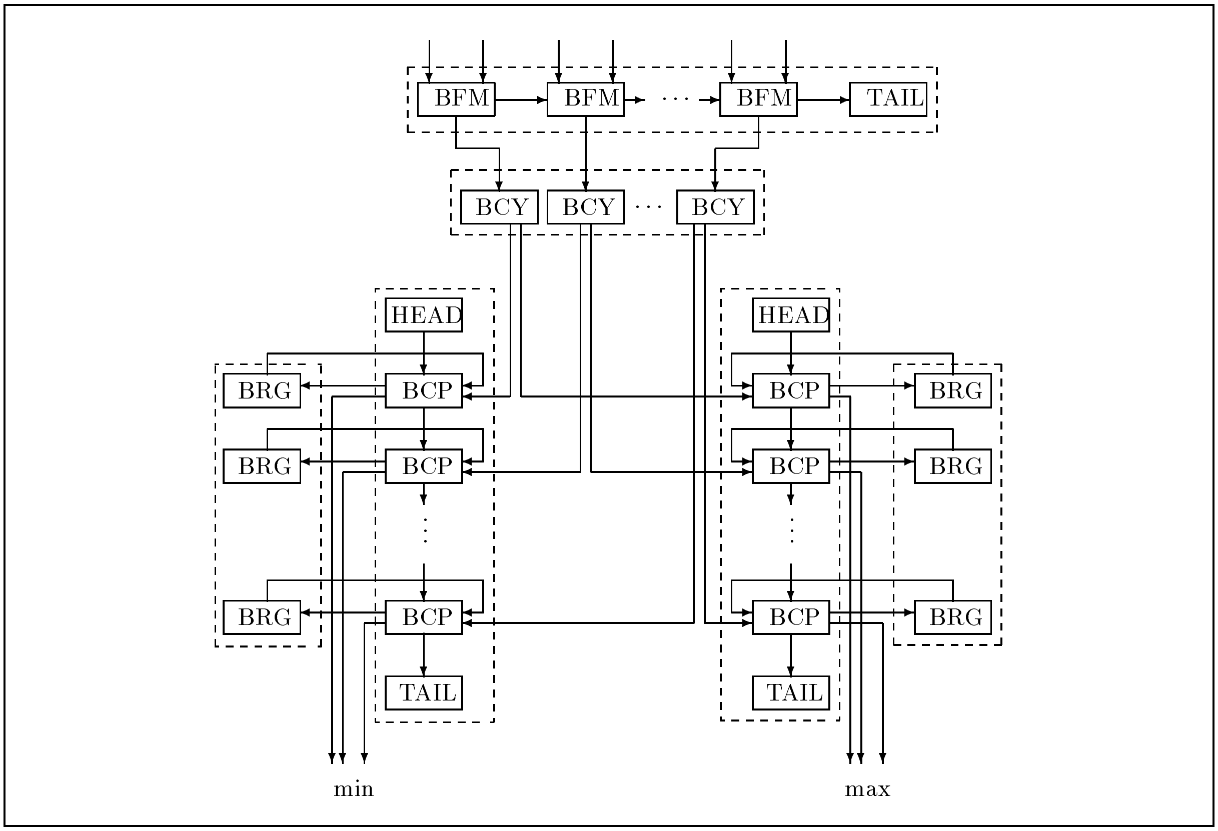

Figure 3: The Bitwise Min/Max Network.

$$Q \overset{\mathsf{def}}{=} \{0, \ldots, 2^{\mathrm{BW}} - 1\}$$

$$read : Q^{\omega} \times \{l, r\}^{\omega} \times \{l, r\} \xrightarrow{c} Q^{\omega}$$

$$read(o \,\&\, op, y \,\&\, hp, x) = \mathsf{if}\ y = x\ \mathsf{then}\ o \,\&\, read(op, hp, x)\ \mathsf{else}\ read(op, hp, x)$$

the Min/Max Component is required to satisfy:

$$\mathsf{spec}\ \mathrm{NMM} :: ia, ib : Q^{\omega} \rhd mn, mx : Q^{\omega} \equiv$$

$$\begin{aligned}
&\exists h \in \{l, r\}^{\omega} : \exists o \in Q^{\omega} : \\
&\quad ia = read(o, h, l) \wedge ib = read(o, h, r) \wedge \\
&\quad \#mx = \#mn = \#ia + \#ib \wedge \\
&\quad \forall j \in \mathsf{dom}.mn : mn[j] = \mathsf{min}(\mathsf{rng}.o|_j) \wedge mx[j] = \mathsf{max}(\mathsf{rng}.o|_j)
\end{aligned}$$

The existentially quantified h is used to model the order in which the input messages are read. The existentially quantified o can be thought of as representing an internal channel in which ia and ib are fairly merged together in accordance with h (this fact is exploited when NMM is decomposed in the next section). The first two conjuncts make sure that the input channels are read fairly. The third conjunct constrains the component to process all its inputs, and the fourth conjunct requires the minimum and the maximum to be output along mn and mx, as described above.

This specification is clearly nondeterministic since the order in which the inputs are read is not determined, i.e. h is not fixed. One might think that the third conjunct is a consequence of the fourth. However, this is not the case. Without the third conjunct, the specification is for example also satisfied by a component, which as soon as it inputs a 0, outputs infinitely many 0's along mn.

4.2 Structural Refinement of NMM

A structural refinement replaces a component specification by a network of component specifications without changing the external interface. NMM is decomposed into four component specifications, as shown in Figure 4:

- FM, which fairly merges the two input streams represented by ia and ib into an output stream represented by o;
- COPY, which, as its name indicates, sends copies of the input received on o along ri and le (for right and left);
- two specifications $\mathrm{FILTER}_{(\mathsf{min},\mathsf{ub})}$ and $\mathrm{FILTER}_{(\mathsf{max},\mathsf{lb})}$, where $\mathsf{ub} = 2^{\mathrm{BW}}-1$ and $\mathsf{lb} = 0$, characterizing respectively a Min and a Max component.

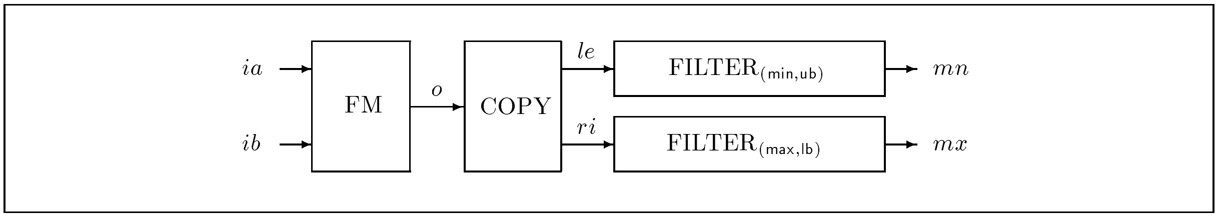

Figure 4: First Decomposition of NMM.

The first one, FM, can be specified as follows:

$$\mathsf{spec}\ \mathrm{FM} :: ia, ib : Q^\omega \rhd o : Q^\omega \equiv \exists h \in \{l, r\}^\omega : ia = read(o, h, l) \wedge ib = read(o, h, r)$$

The second component specification, COPY, is completely trivial:

$$\mathsf{spec}\ \mathrm{COPY} :: o : Q^\omega \rhd le, ri : Q^\omega \equiv le = ri = o$$

The other two can be seen as instances of a parameterized specification, which we call FILTER:

$$\mathsf{spec}\ \mathrm{FILTER} :: ((m : \wp(\mathsf{N}) \to \mathsf{N}) \times init : Q) \times nw : Q^\omega \rhd out : Q^\omega \equiv$$

$$\#out = \#nw \wedge \forall j \in \mathsf{dom}.out : out[j] = m(\mathsf{rng}.nw|_j \cup \{init\})$$

FILTER has, in addition to the input observable nw and the output observable out, two parameters, namely a function m and a natural number $init$. The parameters m and $init$ are instantiated with min and ub in the specification characterizing the Min component, and with max and lb in the specification characterizing the Max component. The first conjunct in the specification of FILTER restricts the number of output messages to be equal to the number of input messages. The second conjunct makes sure that the j'th output message is correctly chosen (modulo m) between the j first input messages and $init$.

The correctness of this decomposition, i.e. that

$$\mathrm{NMM} \rightsquigarrow \mathrm{FM}\,;\mathrm{COPY}\,;(\mathrm{FILTER}_{(\mathsf{min},\mathsf{ub})} \parallel \mathrm{FILTER}_{(\mathsf{max},\mathsf{lb})}) \qquad (1)$$

follows from Rule 4 and straightforward predicate calculus.

4.3 Structural Refinement of FILTER

The FILTER specification can be decomposed into two component specifications, REG and CP, as shown in Figure 5.

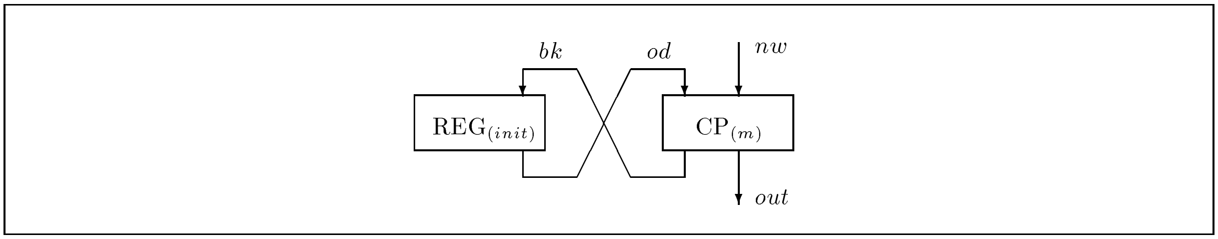

Figure 5: Decomposition of FILTER.

REG can be interpreted as specifying a register storing the last number received on bk. Its initial value is $init$. Thus, REG outputs what it receives on bk prefixed with $init$, i.e. the initial value of the register:

$$\mathsf{spec}\ \mathrm{REG} :: init : Q \times bk : Q^\omega \rhd od : Q^\omega \equiv od = init \,\&\, bk$$

CP, on the other hand, compares a number received on nw with the corresponding number received on od. Depending on m, one of these numbers is chosen and output on both bk and out.

$$\mathsf{spec}\ \mathrm{CP} :: (m : \wp(\mathsf{N}) \to \mathsf{N}) \times od, nw : Q^\omega \rhd bk, out : Q^\omega \equiv$$

$$bk = out \wedge \#out = \mathsf{min}(\{\#od, \#nw\}) \wedge \forall j \in \mathsf{dom}.out : out[j] = m(\{od[j], nw[j]\})$$

The first conjunct requires a message to be output along bk iff it is output along out. The second conjunct restricts any implementation to output exactly one message along out for each pair of messages it receives on its two input channels. The third conjunct makes sure that the correct number modulo m is chosen.

To prove that this decomposition is correct, it must be shown that

$$\mathrm{FILTER}_{(m,init)} \rightsquigarrow \mathrm{REG}_{(init)} \otimes \mathrm{CP}_{(m)} \qquad (2)$$

Let

$$I_1 \overset{\mathsf{def}}{=} \forall j \in \mathsf{dom}.bk : bk[j] = m(\mathsf{rng}.nw|_j \cup \{init\}),$$

$$I_2 \overset{\mathsf{def}}{=} \forall j \in \mathsf{dom}.od : od[j] = m(\mathsf{rng}.nw|_{j-1} \cup \{init\}).$$

It is easy to prove that I_1 and I_2 are safety formulas. Thus Rule 4 implies that it is enough to show that

$$I_1(m, init)[^{bk}_{\epsilon}] \wedge I_2(m, init)[^{od}_{\epsilon}],$$

$$I_1(m, init) \wedge R_{\mathrm{REG}(init)} \Rightarrow I_2(m, init),$$

$$I_2(m, init) \wedge R_{\mathrm{CP}(m)} \Rightarrow I_1(m, init),$$

$$I_1(m, init) \wedge R_{\mathrm{REG}(init)} \wedge I_2(m, init) \wedge R_{\mathrm{CP}(m)} \Rightarrow R_{\mathrm{FILTER}(m,init)},$$

which follows by straightforward predicate calculus.

From (1), instantiations of (2), Rules 1 and 2 we can deduce

$$\mathrm{NMM} \rightsquigarrow \mathrm{FM}\,;\mathrm{COPY}\,;((\mathrm{REG}_{(\mathsf{ub})} \otimes \mathrm{CP}_{(\mathsf{min})}) \parallel (\mathrm{REG}_{(\mathsf{lb})} \otimes \mathrm{CP}_{(\mathsf{max})})) \qquad (3)$$

5 Conclusions

A relational style for the specification and refinement of nondeterministic Kahn-networks has been introduced. As a running example we have chosen a simple Min/Max component.

We have emphasized reasoning about communication — the type of reasoning that normally leads to complicated proofs in design/proof methods for distributed systems. In particular it has been shown that proofs about networks involving feedback can be carried out by formulating invariants in the style of a while-rule of Hoare-logic.

A central question at this point is of course: what happens when we try to apply the same strategy to specifications of a non-trivial complexity? We believe that our technique scales up quite well for the simple reason that we conduct our reasoning at a very abstract level. For example, shared-state concurrency is hard to handle formally because of the very complicated way the different processes are allowed to interfere with each other. In our approach, we still have interference, because the different processes may communicate, but the interference is much more controlled.

The style of reasoning employed in this paper has also been successfully used to develop a non-trivial specification of a production cell [FP95]. For an overview of other case-studies carried out in Focus, see [BFG^{+}94].

Since [FS93] was completed, which is the report on which this paper builds, our approach has been improved in a number of ways. In particular a more elegant semantics can be found in [BS95], and a technique for the translation of Focus specifications into SDL is described in [HS94].

6 Acknowledgements

We would first of all like to thank Manfred Broy who has influenced this work in a number of ways. Detailed comments which led to many improvements have been received from Pierre Collette. Jürgen Kazmeyer and Bernard Schätz have read earlier drafts and provided valuable feedback. Financial support has been received from the Sonderforshungsbereich 342 "Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen".

References

[Bac88] R. J. R. Back. A calculus of refinments for program derivations. *Acta Informatica*, 25:593–624, 1988.

[BDD^{+}93] M. Broy, F. Dederichs, C. Dendorfer, M. Fuchs, T. F. Gritzner, and R. Weber. The design of distributed systems — an introduction to Focus (revised version). Technical Report SFB 342/2/92 A, Technische Universität München, 1993.

[BFG^{+}94] M. Broy, M. Fuchs, T. F. Gritzner, B. Schätz, K. Spies, and K. Stølen. Summary of case studies in Focus - a design method for distributed systems. Technical Report SFB 342/13/94 A, Technische Universität München, 1994.

[BS94] M. Broy and K. Stølen. Specification and refinement of finite dataflow networks — a relational approach. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *Proc. FTRTFT'94, Lecture Notes in Computer Science 863*, pages 247–267, 1994.

[BS95] M. Broy and K. Stølen. A framework for the specification and development of reactive systems. Submitted, January 1995.

[FP95] M. Fuchs and J. Philipps. Formal development of a production cell in Focus – a case study. In C. Lewerenz and T. Lindner, editors, *Formal Development of Reactive Systems: Case Study Production Cell, Lecture Notes in Computer Science 891*, pages 187–200. 1995.

[FS93] M. Fuchs and K. Stølen. Development of a distributed Min/Max Component. Technical Report SFB 342/18/93 A, Technische Universität München, 1993.

[HS94] E. Holz and K. Stølen. An attempt to embed a restricted version of SDL as a target language in Focus. In D. Hogrefe and S. Leue, editors, *Proc. Forte'94*, page ?, 1994. Extended version available as Technical Report SFB 342/11/94 A, Technische Universität München.

[Jon90] C. B. Jones. *Systematic Software Development Using VDM, Second Edition.* Prentice-Hall, 1990.

[Kah74] G. Kahn. The semantics of a simple language for parallel programming. In J.L. Rosenfeld, editor, *Proc. Information Processing 74*, pages 471–475. North-Holland, 1974.

[Kel78] R. M. Keller. Denotational models for parallel programs with indeterminate operators. In E. J. Neuhold, editor, *Proc. Formal Description of Programming Concepts*, pages 337–366. North-Holland, 1978.

[Mor90] C. Morgan. *Programming from Specifications.* Prentice-Hall, 1990.

[SDW95] K. Stølen, F. Dederichs, and R. Weber. Specification and refinement of networks of asynchronously communicating agents using the assumption/commitment paradigm. 1995. To appear in Formal Aspects of Computing. Also available as SFB-report 342/2/93 A.