A Case Study in Transformational Design of Concurrent Systems*

Ernst-Rüdiger Olderog   Stephan Rössig

FB Informatik, Universität Oldenburg
Postfach 2503, 2900 Oldenburg, Germany†

Abstract. We explain a transformational approach to the design and verification of communicating concurrent systems. The transformations start form specifications that combine trace-based with state-based assertional reasoning about the desired communication behaviour, and yield concurrent implementations. We illustrate our approach by a case study proving correctness of implementations of safe and regular registers allowing concurrent writing and reading phases, originally due to Lamport.

1 Introduction

For concurrent systems a variety of specification formalisms have been developed, among them Temporal Logic [MP91], iterative programs like action systems [Bac90] or UNITY programs [CM88], input/output automata [LT89], and process algebra [Mil89, BW90]. However, it remains a difficult task to design correct implementations starting from such specifications. It is here that we wish to make a contribution.

We are developing a novel transformational approach to the design of communicating concurrent systems. Our work originates from the ESPRIT Basic Research Action "ProCoS". ProCoS stands for "Provably Correct Systems" and is a wide-spectrum verification project where embedded communicating systems are studied at various levels of abstraction ranging from requirements' capture over specification language and programming language down to the machine language [Bjø89].

We use a specification language SL that combines trace-based with state-based assertional reasoning. The *trace part* specifies in a modular fashion in which order communications on the channels may occur. To this end, regular expressions over channel alphabets are used. In the trace part we build on ideas of pure process algebra with uninterpreted action symbols. Of course in any realistic application one has also to reason about values that are communicated. In SL the communication values are specified with the help of a *state part* which consists of state variables and communication assertions describing when a channel is enabled for communication and what the effect of such a communication is. The state part corresponds to an iterative program in the style of action systems or UNITY extended by communication through explicit message passing.

The specification language SL is not as high-level as temporal logic can be, but it has the advantage that it allows us to formulate transformation rules for the stepwise design of implementations. In the ProCoS project we have developed a set of transformation rules that is complete for transforming a large class of specifications into sequential occam-like programs [ORSS92]. In this paper we present further transformation rules that enable us to derive distributed concurrent systems with components communicating by synchronous message passing.

Our work on transformational design is in the tradition of the work originated by Burstall and Darlington and pursued fur-

*This research was partially supported by the CEC with the ESPRIT Basic Research Project No. 7071 ProCoS II and by the German Ministry of Research and Technologie (BMFT) as part of the project KORSO (Korrekte Software) under grant No. 01 IS 203 N.

†{olderog,roessig}@informatik.uni-oldenburg.de

ther to practical application in projects like CIP (standing for Computer-aided Intuition-guided Programming) [Bau87] and PROSPECTRA (standing for PROgram development by SPECification and TRAnsformation) [Kri89]. While these approaches were concerned with conventional sequential programs, we study here concurrency and communication.

Central to our approach is the concept of a *mixed term* [Old91b], i.e. a construct that mixes programming and specification constructs. Mixed terms are well suited to express intermediate stages of a design where some implementation details are fixed and others are still open. Mixed terms arise naturally as a formalization of the method of stepwise refinement originally advocated by Dijkstra and Wirth. They appear also in the refinement calculi of [Mor90, Bac90], but these calculi deal with sequential or iterative programs without explicit communication.

In this paper we illustrate our approach by a case study that is concerned with one of the basic assumptions of many distributed algorithms, viz. the correct interprocess communication. In his article [Lam86], Lamport analyzes interprocess communication through registers that can be accessed by writers and readers in a possibly concurrent, i.e. overlapping fashion. The assumptions that distributed algorithms make about interprocess communication is mirrored by the values that a reader of the register may obtain in case of an overlapping writing phase. Lamport defines three classes of registers called safe, regular and atomic where safe registers are the weakest and atomic are the strongest class. The main contribution of [Lam86] are several constructions of stronger register types from weaker ones together with correctness proofs in a specific formalism. The topic of concurrent registers has excited quite some interest in the literature on distributed algorithms. A good overview can be found in [LG89].

In this paper we specify safe and regular registers in the language SL and systematically derive one of Lamport's concurrent implementations using our transformational approach.

2 Specifications

In this section we use the example of registers to provide an introduction to the specification language SL. As in [Lam86] we consider registers that can store a value of some value set V and that are shared by one writer and possibly several readers. We begin with the case of only one reader. Following [LG89] such a register can be modelled as a system communicating through directed channels with its environment consisting of a writer and a reader as shown in Figure 1. The writer initiates a

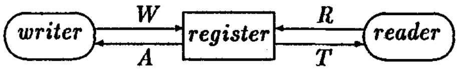

Figure1: Register as communicating system

writing phase by sending a value from the set V along the input channel W. This phase ends when a corresponding acknowledgement signal is output on channel A. Conversely, the reader initiates a *reading phase* by sending a signal along the input channel R. This phase ends when a value from the set V is returned along the output channel T.

It remains to be specified what value is returned at the end of a reading phase. For a reading phase that does not overlap with any writing phase there is only one correct value to be returned, viz. the most recently written one. However, it is not clear what should happen in the case of concurrent, i.e. overlapping reading and writing phases. Therefore Lamport distinguishes three classes of registers called safe, regular and atomic [Lam86].

For a *safe* register, *any* value of the value set V may be returned. For a *regular* register, either the value *before* or *after* the overlapping write must be returned. More generally, a read that overlaps with several writes, one of the values before or after these writes must be returned. For an *atomic* register, overlapping reads and writes must have the same effect as if they occur in some non-overlapping order. We shall consider here only safe and regular registers.

2.1 Safe Registers

Let us first explain how to specify a safe register in SL. An SL specification describes a communicating system using several parts.

Interface. This part lists the communication channels of the system with their direction (input or output) and value type. For the register the interface is given by

```
input W of V
output A of signal
input R of signal
output T of V.
```

Trace Part. This part specifies the sequencing constraints on the interface channels whereas the communicated values are ignored. This is done by stating one or more *trace assertions*, each one consisting of an alphabet, i.e. a subset of the interface channels, and a regular expression over these channels. The regular expression describes the sequencing constraints on the channels mentioned in the alphabet. By stating several such trace assertions, we can specify different aspects of the intended system behaviour in a modular fashion.

For the register the trace part is given by

trace W, A in $pref(W.A)^*$
trace R, T in $pref(R.T)^*$.

The first trace assertion concerns the writer. It states that communications on the channels W and A should occur in alternating order starting with W. In other words, at each moment the trace of channels W and A should be a prefix of some word in the regular language $(W.A)^*$. The second trace assertion states a similar requirement for the reader.

The informal semantics of this part of an SL specification is that the described behaviour must satisfy the sequencing constraints of all trace assertions simultaneously. The trace part of an SL specification corresponds to *path expressions* in the sense of [CH74] or to a regular fragment of *trace logic* in the sense of [Zwi89] and [Old91a].

State Part. This part describes what the exact values are that can be exchanged over the interface channels. To this end, this part may introduce local state variables. These variables constitute the state space of the specification and are used in so-called communication assertions specifying the link between values and channels. However, these variables need not appear in an implementation of the specified system.

For the register we use the following state variables:

```
var v of V
var m of bool
var c of bool.
```

The variable v represents the current *value* of the register. The boolean variable m stands for write *modus* and expresses whether the register is currently engaged in a writing phase. The boolean variable c indicates whether a reading phase has to return the *correct* value, i.e. the one currently stored in v.

A *communication assertion* for a channel ch is of the form

com ch write $\overline{w}$ read $\overline{r}$ when wh then th

where $\overline{w}$ and $\overline{r}$ are disjoint lists of state variables, the list $\overline{w}$ of *write* variables and the list $\overline{r}$ of *read* variables, and two predicates, the *when* or *enable* predicate wh describing when a channel is enabled for communication and the *then* or *effect* predicate th describing the communication value and the effect of this communication on the state variables.

The enable predicate may only use variables from $\overline{w}$ and $\overline{r}$. The effect predicate may additionally use primed versions of the write variables and the distinguished variable @ch obtained by prefixing the channel name ch by the symbol @. As in the specification language Z [Spi89], a primed variable x' refers to the value of the variable x at the moment of termination. The read variables are not changed. The variable @ch refers to the communication value on the channel ch.

If one of the variable lists is empty or one of the predicates is true, these components are omitted from the communication assertion. In general, there can be several communication assertions for the same channel. We require that the set of all write variables in these assertions is disjoint from the set of all read variables.

For the channels W, A, R, T of the register we state the following communication assertions.

```
com W write v, m, c
        then m' ∧ v' = @W ∧ ¬c'
```

asserts that each communication on channel W updates the state variables v, m and c as follows: the write modus m is set, the value v of the register becomes the current communication value $@W$, and c is set to `false` to indicate for a possibly overlapping reading phase that an arbitrary value may be returned.

Simpler is the communication assertion for channel A:

```
com A write m then ¬m'
```

just asserts that a communication on A switches off the write modus. For channel R,

```
com R write c read m then c' = ¬m
```

asserts that in c it is recorded whether the register is currently outside a write modus. For channel T,

```
com T read c, v then c ⇒ @T = v
```

asserts that when c is set the communication value on channel T has to be the correct value as given by v.

Putting these parts together, we arrive at the SL specification of a safe register for one reader and value set V shown in Figure 2.

Informally, this SL specification describes the set of all traces of communications along the channels W, A, R and T that satisfy the constraints given by the trace assertions and communication assertions simultaneously. Communications are denoted by pairs (ch, k) where ch is a channel name and

```
spec input W of V
     output A of signal
     input R of signal
     output T of V
     trace W, A in pref(W.A)*
     trace R, T in pref(R.T)*
     var m of bool
     var c of bool
     var v of V
     com W write v, m, c
             then m' ∧ v' = @W ∧ ¬c'
     com A write m then ¬m'
     com R write c read m
             then c' = ¬m
     com T read c, v then c ⇒ @T = v
end
```

Figure2: Specification `1-reader-V-safe`

k is the communication value. Communications on channels ch of type `signal` will be simply denoted by the channel name ch itself. For example, for V={1,2,3,4,5} the communication trace

$$tr = (W,3).A.R.(T,3).R.(W,5).A.(T,4).R.(T,5)$$

satisfies `1-reader-V-safe`. Note that within the second reading phase a writing phase occurs which sets the state variable c to `false`. Hence at the end of this reading phase an arbitrary value from V may be output on channel T. Here we have chosen the value 4. On the other hand, the last reading phase ends with c evaluating to `true` and thus outputs the most recently written value, which is 5.

The formal semantics of the specification language SL is defined in a *predicative style* in [Old91b] and [ORSS92] and is beyond the scope of this paper. We mention only that in this semantics each SL specification S is identified with a pair $\Delta : P$ where Δ is the interface of S and P is a predicate describing the behaviour specified by S in terms of communication traces, ready sets and some other ingredients that are not important here.

A *ready set* is a set of channels that are ready for communication. The formal semantics of an SL specification S requires that each communication trace of a system satisfying S should be ready on all channels that may occur next according to the trace and communication assertions of S. For example, after the above trace tr the specification `1-reader-V-safe` requires that a register should be ready for communication on the channels W and R. In particular, a register may not refuse to interact with its writer or reader after the trace tr.

In principle, the trace part of specification `1-reader-V-safe` can be eliminated in favour of an extended state part. However, this would result in a specification that is more difficult to understand. In general, we strive to express the data independent aspects of a system behaviour in the trace part.

For each given n, the above specification can be extended to one specifiying a safe register with n readers using communication channels R_i and T_i for $i \in 1..n$ (Figure 3).

```
spec input $W$ of $V$
     output $A$ of signal
     input $R_1, ..., R_n$ of signal
     output $T_1, ..., T_n$ of $V$
     trace $W, A$ in $pref(W.A)^*$
$i\in 1..n$: trace $R_i, T_i$ in $pref(R_i.T_i)^*$
     var $m$ of bool
     var $c_1, ..., c_n$ of bool
     var $v$ of $V$
     com $W$ write $v, m, c_1, ..., c_n$
       then $m' \wedge v' = @W \wedge \bigwedge_{i=1}^{n} \neg c_i'$
     com $A$ write $m$ then $\neg m'$
$i\in 1..n$: com $R_i$ write $c_i$ read $m$
          then $c_i' = \neg m$
$i\in 1..n$: com $T_i$ read $c_i, v$ then $c_i \Rightarrow @T_i = v$
end
```

Figure3: Specification `n-reader-V-safe`

Since each of the readers can have different overlappings with writing phases, we introduce separate state variables c_i to record whether at the end of a reading phase the reader i should get the correct value of the register as stored in variable v. This is specified in the communication assertion for channel T_i.

2.2 Regular Registers

Let us now specify the behaviour of a regular register for n readers and the value set V in the language SL. We can reuse a large part of the specification `n-reader-V-safe` above. Only the specification of the value returned at the end of a reading phase need to be changed. The idea is here to replace the boolean variables c_i by *set valued* variables C_i which at each moment represent the set of values from V that may be returned. Thus we declare

$$\texttt{var } C_1, ..., C_n \texttt{ of } V - set$$

and change the communication assertion for T_i to

$$\texttt{com } T_i \texttt{ read } C_i \texttt{ then } @T_i \in C_i.$$

It remains to be specified how to update the state variables C_i. Since by the regularity condition a reading phase overlapping a writing phase should either return the value before or after the write, we keep track of the *old* value of the register with every write. Thus we introduce the state variable

$$\texttt{var } old \texttt{ of } V$$

and use the following communication assertions for channels R_i and W:

```
com $R_i$ write $C_i$ read $m, old, v$
     then $(\neg m \Rightarrow C_i' = \{v\})$
          $\wedge\ (m \Rightarrow C_i' = \{v, old\})$

com $W$ write $v, m, C_1, ..., C_n, old$
     then $m' \wedge v' = @W \wedge old' = v$
       $\wedge \bigwedge_{i=1}^{n} C_i' = C_i \cup \{@W\}$ .
```

Thus at the start of a reading phase through R_i the set of correct values depends on whether the register is in a write modus. If not, only the current value is the correct one. Otherwise the current and the old value are both correct. If during the reading phase new writing phases are initiated by a communication on channel W, each time the set of correct values is enlarged by the newly written value $@W$.

```
spec input $W$ of $V$
       output $A$ of signal
       input $R_1, ..., R_n$ of signal
       output $T_1, ..., T_n$ of $V$
       trace $W, A$ in $pref(W.A)^*$
$i \in 1..n$: trace $R_i, T_i$ in $pref(R_i.T_i)^*$
       var $m$ of bool
       var $C_1, ..., C_n$ of $V - set$
       var $v$ of $V$
       var $old$ of $V$
       com $W$ write $v, m, C_1, ..., C_n, old$
          then $m' \wedge v' = @W \wedge old' = v$
               $\wedge \bigwedge_{i=1}^{n} C_i' = C_i \cup \{@W\}$
       com $A$ write $m$ then $\neg m'$
$i \in 1..n$: com $R_i$ write $C_i$ read $m, old, v$
          then $(\neg m \Rightarrow C_i' = \{v\})$
               $\wedge (m \Rightarrow C_i' = \{v, old\})$
$i \in 1..n$: com $T_i$ read $C_i$ then $@T_i \in C_i$
end
```

Figure4: Specification n-reader-V-regular

Altogether we obtain the specification shown by Figure 4. Examples of communication traces satisfying n-reader-V-regular are

$$(W,3).A.R.(T,3).R.(W,5).A.(W,1).A.(T,k)$$

where $k \in \{1,3,5\}$. After each of these traces the register is ready to engage in communications on channels W and R.

3 Transformational Approach

The standard setting for a transformational approach is that specifications are transformed stepwise into programs. For example, our aim in ProCoS is to transform specifications of the language SL into programs of an occam-like programming language PL. In our present study we do not aim at occam-like programs but wish to show how to construct complex registers from simpler ones.

Such a construction can be conveniently expressed in the language MIX of *mixed terms*. MIX comprises n-ary programming operators OP that can be applied to specifications or other mixed terms $S_1, ..., S_n$ yielding a mixed term OP$[S_1, ..., S_n]$. In general, MIX serves to express the intermediate stages of a transformational design from SL to PL and thus contains SL and PL as proper subsets. Here MIX is used as a language for expressing implementations of registers.

Under the predicative semantics described in [Old91b] and [ORSS92], specifications, programs and mixed terms are all identified with so-called *systems*. These are pairs $\Delta : P$ where Δ is an interface and P is a predicate describing the communication behaviour on the interface channels in Δ. Logical implication and equivalence on predicates are lifted to systems as follows:

- system implication: $\Delta_1 : P_1 \equiv> \Delta_2 : P_2$ if $\Delta_1 = \Delta_2$ and $\models P_1 \Rightarrow P_2$,
- system equivalence: $\Delta_1 : P_1 \equiv \Delta_2 : P_2$ if $\Delta_1 = \Delta_2$ and $\models P_1 \Leftrightarrow P_2$.

We also write $\Delta_2 : P_2 <\equiv \Delta_1 : P_1$ instead of $\Delta_1 : P_1 \equiv> \Delta_2 : P_2$. Under the predicative semantics, system implication models the *satisfaction* or *implementation* or *refinement* relation. Thus a program or mixed term Q *is correct w.r.t.* or *satisfies* or *implements* or *refines* a specification S iff $Q \equiv> S$ holds. Note that system equivalence is a special case of refinement.

In the transformational approach a design of a program or mixed term Q from a specification S is a sequence

$$S \equiv R_1 <\equiv ... <\equiv R_n \equiv Q$$

of system implications between mixed terms $R_1, ..., R_n$ where R_1 is the given specification S and R_n is the desired result Q. The transitivity of the relation $\equiv>$ ensures the desired correctness result $Q \equiv> S$.

Each of the implications $R_i <\equiv R_{i+1}$ in the design sequence is generated by an application of a transformation rule. We distinguish two classes of transformation rules. Rules preserving system equivalence $\equiv$ do not modify the system behaviour but only its syntactic representation. By contrast, *implementation* or *strengthening* rules relate systems of different behaviour by $\equiv>$. An application of such a rule represents an irreversible design decision: nondeterminism within the behaviour may be removed or an over-specification is obtained.

3.1 Transforming the State Part

As a first contact with our transformational approach we present four groups of transformation rules dealing exclusively with the state part of a specification. As an application we shall then formally derive (the intuitively clear statement) that regular registers refine safe registers.

Specification format. It is convenient to extend the specification format by introducing *invariant declarations* of the form

$$\texttt{inv}\ p$$

where p is a predicate such that all free variables are declared within the specification considered. Such a declaration postulates that p holds in the initial state and after each communication.

Thus an SL specification can be represented as a tuple spec Δ TA Va CA I end where the components are as follows:

Δ	– a set of interface channels,
TA	– a set of trace assertions,
Va	– a set of variable declarations,
CA	– a set of communication assertions,
I	– a set of invariants.

Conjunction. These transformations reveal the conjunctive nature of communication assertions and invariants; their application always yield equivalent specifications.

T 3.1 (*conjunction of communication assertions*) Two communication assertions

$$\texttt{com}\ ch\ \texttt{write}\ \overline{w}_i\ \texttt{read}\ \overline{r}_i\ \texttt{when}\ wh_i\ \texttt{then}\ th_i$$

($i \in \{1,2\}$) for the same channel ch are equivalent to a single one:

$$\texttt{com}\ ch\ \texttt{write}\ \overline{w}_1 \cup \overline{w}_2\ \texttt{read}\ \overline{r}_1 \cup \overline{r}_2\ \texttt{when}\ wh_1 \wedge wh_2\ \texttt{then}\ th_1 \wedge th_2.$$

■

The conjunction of all communication assertions for a channel ch within CA yields the *unique communication assertion* for ch:

$$\texttt{com}\ ch\ \texttt{write}\ \overline{w}_{ch}\ \texttt{read}\ \overline{r}_{ch}\ \texttt{when}\ wh_{ch}\ \texttt{then}\ th_{ch}.$$

T 3.2 (*conjunction of invariants*) Two invariant declarations inv p_1 and inv p_2 are equivalent to a single one: inv $p_1 \wedge p_2$. ■

In the following we denote by $\bigwedge I$ the conjunction of all invariant predicates. If I is empty, we put $\bigwedge I = \textbf{true}$.

Strengthening. These transformations strengthen the system behaviour by restricting the initial state, the state space or the effect of a communication. They either remove some nondeterminism or lead to over-specification. It requires creativity to find the right degree of strengthening within the design process.

T 3.3 (*initialization*) Any variable declaration var x of ty_x may be extended to an initialized declaration var x of ty_x init e thereby defining e as initial value of x. ■

T 3.4 (*invariant strengthening*) An invariant predicate p may be replaced by any predicate q over free variables Va such that $q \Rightarrow p$ holds. ■

Introducing an invariant declaration in a specification without invariants is included as the special case of strengthening inv true.

T 3.5 (*effect strengthening*) An effect predicate p may be replaced by any predicate q such that $q \wedge \bigwedge I \Rightarrow p$ holds and its free variables agree with the read and write list. ■

Modifying specification components. These transformations describe the interaction of several specification components. Their application always yield equivalent specifications.

T 3.6 (*communication assertion modifications*) An enable or effect predicate p may be replaced by any predicate q with $\bigwedge I \Rightarrow (p \Leftrightarrow q)$ as far as the static semantics conditions are not violated. ∎

Thus dependent on the specification invariant a single enable or effect predicate can be strengthened or weakened without changing the behaviour.

The counterpart of this transformation allows to modify an invariant dependent on the initial state and CA. To this end we need the notion of stability. A predicate q with free variables in Va is *stable for a channel ch* if the unique communication assertion of ch guarantees the following: q holds in all termination states of communications on ch that start in a state satisfying q. Formally, we have

$$stable(q, ch)$$
$$\Leftrightarrow_{df}$$
$$wh_{ch} \wedge th_{ch} \wedge q \Rightarrow q[\overline{w}'_{ch}/\overline{w}_{ch}]$$

where the substitution $q[\overline{w}'_{ch}/\overline{w}_{ch}]$ replaces the write variables $\overline{w}_{ch}$ by their primed versions. Note that disjointness of $\overline{w}_{ch}$ and $free(q)$ guarantees $stable(q, ch)$.

T 3.7 (*invariant modifications*) Let q be a predicate which holds in the initial state and is stable for all channels in Δ. Then any invariant predicate p may be weakened to the implication $q \Rightarrow p$ or strengthened to the conjunction $q \wedge p$. ∎

Removing a declaration **inv** p is done by weakening to $p \Rightarrow p$.

Local variables. Here we consider how read and write lists of a communication assertions and the set of local variable declarations can be changed by equivalence transformations.

T 3.8 (*read list modification*) Any variable $x \in Va$ may be added to the read list of a communication assertion of channel ch provided x does not occur in the write list $\overline{w}_{ch}$. A variable occurring free neither in the enable nor in the effect predicate of a communication assertion may be removed from its read list.
∎

The following two rules are corollaries of a quite complex equivalence transformation dealing with the combined modification of variable declarations, invariants and communication assertions. They allow us to add and remove variable declarations together with modifications of communication assertions.

T 3.9 (*state space extension*) The state space can be extended by declaration of a new local variable. In addition the write lists of any communication assertions may be extended by this variable. ∎

T 3.10 (*removing write only variables*) A variable declaration `var` x `of` ty_x[`init` e] may be removed from a specification if x does not occur free in any invariant, enable or effect predicate. In that case x must be removed from the variable lists of the communication assertions and their effect predicates must be changed such that

$$th^{new}_{ch} \quad \Leftrightarrow \quad \exists x' \bullet th^{old}_{ch}$$

holds for all channels ch where x appears in the write list. ∎

3.2 Regular Implements Safe

From their informal description it seems obvious that a regular register implements a safe one. Here we will prove this relation formally for the SL specifications given in Section 2. Since both specifications agree on their interface and trace parts, we need to relate only their state parts. To this end, we shall apply the above transformation rules and massage the specification `n-reader-V-safe` until specification `n-reader-V-regular` is obtained. We proceed in three steps:

1. The communication assertions of T_i are modified to the pattern of the regular register specification. Therefore the state space is extended by set-valued variables $C_1, ..., C_n$ and the invariant $\bigwedge_{i=1}^{n} c_i \Rightarrow (C_i = \{v\})$ is introduced.
2. By appropriate initialization and effect strengthening the invariant of Step 1 is made redundant. This allows to remove the same and afterwards all variables $c_1, ..., c_n$.

3. The variable *old* is added and channels W and R_i are strengthened to achieve the regular specification pattern.

In the following we give a detailed account of this refinement by referring to the numbers of applied transformation rules. Starting point is the specification `n-reader-V-safe`.

Step 1 We extend the internal state space by new local variables $C_1, ..., C_n$ where write accesses are restricted to channels W and $R_1, ..., R_n$ [T3.9]:

```
var $C_1, ..., C_n$ of $V - set$
com $W$ write $C_1, ..., C_n$
com $R_i$ write $C_i$.
```

The values of $C_1, ..., C_n$ are related to those of $c_1, ..., c_n$ by the following invariant [T3.4]:

$$\texttt{inv} \bigwedge_{i=1}^{n} (c_i \Rightarrow C_i = \{v\}).$$

Thus whenever T_i has to return the correct register value then variable C_i holds value $\{v\}$. The effect predicates of channels R_i are now strengthened [T3.5] to $@T_i \in C_i$ based on implication

$$@T_i \in C_i \land \bigwedge_{i=1}^{n} (c_i \Rightarrow C_i = \{v\}) \Rightarrow (c_i \Rightarrow @T_i = v).$$

Accordingly the read lists are modified [T3.8] by appending C_i and removing c_i. Thus in total the old communication assertions of $T_1, ..., T_n$ are replaced by the following ones for each $i \in 1..n$:

```
com $T_i$ read $C_i$ then $@T_i \in C_i$.
```

Step 2 We strengthen the system behaviour in such a way that the invariant becomes redundant and thus may be removed. Firstly, all variables c_i are initialized with `false` [T3.3]:

```
var $c_1, ..., c_n$ of bool init false.
```

Therefore the invariant holds initially. Secondly, the effect predicates of channels R_i are strengthened [T3.5]:

```
com $R_i$ write $C_i, c_i$ read $m, v$
    then $c_i' = \neg m \land (\neg m \Rightarrow C_i' = \{v\})$.
```

Since any communication on W assigns `false` to all c_i, its termination state satisfies the invariant. All remaining channels $A, T_1, ..., T_n$ do not write variables occurring in the invariant predicate. Thus the invariant holds and its declaration can be removed [T3.7].

This leads to a specification without read accesses to variables $c_1, ..., c_n$. By transformation [T3.10] their declarations are removed and the communication assertions of W and $R_1, ..., R_n$ are modified using the logical equivalences $\exists c_1', ..., c_n' \bullet \bigwedge_{i=1}^{n} \neg c_i' \Leftrightarrow$ **true** and $\exists c_i' \bullet c_i' = \neg m \Leftrightarrow$ **true**. We thus obtain the following specification:

```
spec ... interface and trace assertions ...
     var $m$ of bool
     var $C_1, ..., C_n$ of $V - set$
     var $v$ of $V$
     com $W$ write $v, m, C_1, ..., C_n$
             then $m' \land v' = @W$
     com $A$ write $m$ then $\neg m'$
$i \in 1..n$: com $R_i$ write $C_i$ read $m, v$
             then $\neg m \Rightarrow C_i' = \{v\}$
$i \in 1..n$: com $T_i$ read $C_i$ then $@T_i \in C_i$
end
```

Step 3 We introduce the local variable *old* of type V and allow write access to it by W [T3.9]:

```
var $old$ of $V$
com $W$ write $old$.
```

Strengthening W by conjunct $\bigwedge_{i=1}^{n} C_i' = C_i \cup \{@W\} \land old' = v$ [T3.5] and extending the read lists of all R_i by *old* [T3.8] together with strengthening their effects by conjuncts $m \Rightarrow C_i' = \{v, old\}$ [T3.5] delivers the target specification `n-reader-V-regular`.

4 Concurrent Implementations

In this section we study the implementation of a specification as a system of concurrently working subsystems synchronized via internal communication. When designing such a system one first decides on its architecture, i.e. which tasks should be performed concurrently and how subsystems should communicate. The transformational refinement process is then guided by these decisions.

As an example we consider the implementation of an n-reader-V-safe register using n copies $X_1, ..., X_n$ of 1-reader-V-safe registers and an auxiliary write process WP due to [Lam86, LG89]. The architecture of this implementation is shown in Figure 5. Thus

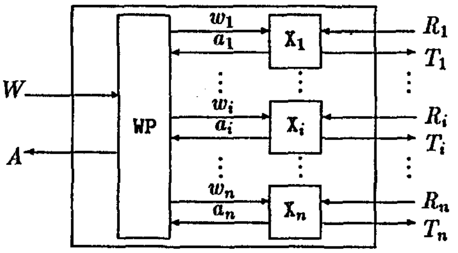

Figure5: Implementation of n-reader-V-safe

each of the n readers can communicate directly via the channels R_i and T_i with a private single reader register X_i. By contrast, the writer communicates via W and A with the auxiliary process WP which is linked with the single reader registers X_i via the internal channels w_i and a_i. The idea is that a write access to the n-reader register is implemented in several stages. After having received a new value by communication on W the process WP transmits this value via potentially parallel internal writes to all X_i. The external acknowledge on A is offered as soon as all internal writes have indicated their termination by the acknowledge events a_i.

We now present a formal transformational design of this implementation consisting of the following steps:

- Local channels are declared and their global sequencing is constrained.
- The state space is extended to cover the state spaces of the n single reader register. The behaviour is strengthened to achieve the effects of the 1-reader-V-safe specification.
- The whole specification is decomposed into the subsystems WP and $X_1, ..., X_n$.

4.1 Local Channels

A communication on a local channel is independent from and invisible to the environment. It may be performed as soon as its enable predicate holds in the current internal state and the extended trace satisfies the sequencing constraints of all trace assertions. Thus in contrast to external channels there is no synchronization with the environment.

Interface channels $ch_1, ..., ch_k$ of a specification S are localized applying to S the declaration operator CHAN with parameters $ch_1, ..., ch_k$:

$$S_1 = \texttt{CHAN}\ ch_1, ..., ch_k\ S.$$

CHAN is one of the operators of the language MIX so that S_1 is a mixed term. The semantics of CHAN implies that the system S_1 avoids engaging in unboundedly many communications on the local channels $ch_1, ..., ch_k$. Thus CHAN is a so-called "angelic" operator which is difficult to implement.

To avoid non-implementability we additionally use the operator HIDE from MIX:

$$S_2 = \texttt{CHAN}\ ch_1, ..., ch_k\ \texttt{HIDE}\ ch_1, ..., ch_k\ S.$$

Systems S_1 and S_2 behave the same as long as S does not allow unbounded communication on $ch_1, ..., ch_k$. But in contrast to S_1 unbounded communication on these channels leads to divergence of S_2. For a more detailed analysis see [ORSS92].

Here we present a rule dealing with the combined effect of introducing local channels with hiding.

T 4.1 (*introducing local channels*)
Let S = spec Δ TA Va CA I end be a specification and $ch_1, ..., ch_k$ channel names not in Δ. Then S is equivalent to any mixed term

$$T = \texttt{CHAN}\ ch_1, ..., ch_k\ \texttt{HIDE}\ ch_1, ..., ch_k\ \texttt{spec}\ \Delta_T\ TA_T\ Va\ CA\ I\ \texttt{end}$$

where the interface Δ_T is given by[1]

Δ input ch_1 of ty_{ch_1}, ...input ch_k of ty_{ch_k} output ch_1 of ty_{ch_1}, ...output ch_k of ty_{ch_k}

[1]For technical reasons the local channels are declared with both directions.

and the trace part TA_T satisfies the following conditions:

1. it prevents unbounded communication on the new channels,
2. its projection onto the old channels allows exactly the same traces as TA,
3. for each prefix t of one of its traces and for each trace assertion $ta \in TA_T$ the intersection of all extensions of t with the alphabet of ta contains at most one of the new channels.

■

Condition 1 implies that T describes a divergence free system. Conditions 2 and 3 imply that semantically S and T describe the same traces and ready sets. An application of this rule requires to find a right extension of the trace part meeting both these conditions and the overall development idea.

In our example we introduce the local channels $w_1, ..., w_n, a_1, ..., a_n$. As mentioned above communications on w_i and a_i are always enclosed by a preceding W and a finishing A communication. Hence the application conditions 1–3 of [T4.1] are satisfied. No restrictions are required between write and acknowledge channels of different single readers. Thus we replace the specification `n-reader-V-safe` by the following mixed term:

```
CHAN $w_1, ..., w_n, a_1, ..., a_n$
HIDE $w_1, ..., w_n, a_1, ..., a_n$
spec input $w_1, ..., w_n$ of $V$
     output $w_1, ..., w_n$ of $V$
     input $a_1, ..., a_n$ of signal
     output $a_1, ..., a_n$ of signal
   ... all n-reader-V-safe components ...
$i \in 1..n$:
  trace $W, A, w_i, a_i$ in $pref(W.w_i.a_i.A)^*$
end
```

4.2 Trace Assertions and Invariants

Now we aim at the behaviour of the single reader registers. This requires an extended reasoning about modifications of the state part where in addition to the rules presented in 3.1 also the trace part is taken into account.

The following rule provides a generalization of the invariant reasoning based on [T3.6] and [T3.7]. It checks whether a predicate q holds whenever the system may engage in a communication on channel ch and allows us to modify its communication assertion appropriately. We say that a channel ch^* establishes q if it holds in the terminating state of each ch^* communication:

$$establish(q, ch^*)$$
$$\Leftrightarrow_{df}$$
$$wh_{ch^*} \wedge th_{ch^*} \Rightarrow q[\overline{w}'_{ch^*}/\overline{w}_{ch^*}].$$

T 4.2 (*effect modifications under trace assertions*) Let `trace` $ch_1, ..., ch_k$ `in` re be a trace assertion and q be a predicate which is stable for all channels of $\Delta \backslash \{ch_1, ..., ch_k\}$. Let $ch \in \{ch_1, ..., ch_k\}$ be a channel such that in every word of the regular language of re each occurrence of ch is preceded by a channel ch^* establishing q. If further all intermediate channels between ch^* and ch are stable for q then the effect th_{ch} may be weakened to $q \Rightarrow th_{ch}$ or strengthened to $q \wedge th_{ch}$ without changing the behaviour. ■

For the moment the design process proceeds by the same technique shown in detail in 3.2: the state space is changed and communication assertions are strengthened; invariants are introduced to modify communication assertions and are removed afterwards.

At first we introduce new local variables $m_i, \ell c_i, v_i$ which shall correspond to the local variables m, c, v of an `1-reader-V-safe` register specification. Write and read access to these new variables is restricted as follows:

```
var $m_1, ..., m_n, \ell c_1, ..., \ell c_n$ of bool
var $v_1, ..., v_n$ of $V$
com $w_i$ write $v_i, m_i, \ell c_i$
com $a_i$ write $m_i$
com $R_i$ write $\ell c_i$ read $m_i$
com $T_i$ read $\ell c_i, v_i$.
```

Then the effect predicates of theses channels are strengthened. The additional restrictions are motivated by the corresponding predicates in `1-reader-V-safe`.

```
com $w_i$ write $v_i, m_i, \ell c_i$ read $v$
      then $m_i' \wedge v_i' = v \wedge \neg \ell c_i'$
com $a_i$ write $m_i$ then $\neg m_i'$
com $R_i$ write $c_i, \ell c_i$ read $m, m_i$
      then $c_i' = \neg m \wedge \ell c_i' = \neg m_i$
com $T_i$ read $c_i, v, \ell c_i, v_i$
  then $(c_i \Rightarrow @T_i = v) \wedge (\ell c_i \Rightarrow @T_i = v_i)$
```

In the following steps the effect predicates are modified such that they become independent from variables m and $c_1, ..., c_n$. This is done by iterated application of the invariant technique shown in the previous example. New invariants are introduced and some effects are modified under them. After that effect predicates of other channels are strengthened to make the invariant redundant.

Firstly we deal with channels T_i whose effect predicates are simplified under the following invariant:

$$\texttt{inv} \bigwedge_{i=1}^{n} (c_i \Rightarrow \ell c_i \wedge v_i = v).$$

Thus communication assertions of T_i can be replaced by the following ones:

```
com $T_i$ read $\ell c_i, v_i$ then $\ell c_i \Rightarrow @T_i = v_i$.
```

Consideration of the single effect predicates shows that this invariant is established by any communication on W, since all c_i are set to `false`, and is a stable property of channels T_i, a_i, A. The initialization of variables $c_1, ..., c_n$ by `false` makes it also valid in the initial state. The effect predicates of channels w_i and R_i are strengthened by the conjuncts $\neg c_i$ and $\neg m \Rightarrow (\neg m_i \wedge v_i = v)$, respectively, and thus the invariant becomes redundant.

```
var $c_1, ..., c_n$ of bool init false
com $w_i$ write $v_i, m_i, \ell c_i$ read $v, c_i$
      then $m_i' \wedge v_i' = v \wedge \neg \ell c_i' \wedge \neg c_i$
com $R_i$ write $c_i, \ell c_i$ read $m, m_i, v_i, v$
      then $c_i' = \neg m \wedge \ell c_i' = \neg m_i \wedge$
          $(\neg m \Rightarrow \neg m_i \wedge v_i = v)$
```

Next all read accesses to variables c_i are removed using the invariant $m \Rightarrow \bigwedge_{i=1}^{n} \neg c_i$. It allows us to strengthen the effects of all w_i by replacing conjuncts $\neg c_i$ with m:

```
com $w_i$ write $v_i, m_i, \ell c_i$ read $v, m$
      then $m_i' \wedge v_i' = v \wedge \neg \ell c_i' \wedge m$.
```

The invariant predicate used here holds in the initial state and is stable for all channels. Thus it can be removed without any further changes.

Then the local variables $c_1, ..., c_n$ are removed and we obtain the following reduced state part:

```
var $m, m_1, ..., m_n, \ell c_1, ..., \ell c_n$ of bool
var $v, v_1, ..., v_n$ of $V$
com $W$ write $v, m$ then $m' \wedge v' = @W$
com $A$ write $m$ then $\neg m'$
com $w_i$ write $v_i, m_i, \ell c_i$ read $v, m$
      then $m_i' \wedge v_i' = v \wedge \neg \ell c_i' \wedge m$
com $a_i$ write $m_i$ then $\neg m_i'$
com $R_i$ write $\ell c_i$ read $m, m_i, v_i, v$
      then $\ell c_i' = \neg m_i \wedge$
          $(\neg m \Rightarrow \neg m_i \wedge v_i = v)$
com $T_i$ read $\ell c_i, v_i$ then $\ell c_i \Rightarrow @T_i = v_i$.
```

Now we pursue the elimination of variable m. To remove the read accesses to variable m the newly introduced rule [T4.2] is applied. Each communication on W assigns `true` to m and each one on A sets m to `false` and no other communication modifies the value of m. Thus predicate m = `true` is established by W and is stable for all channels but A. We conclude from trace assertion

```
trace $A, W, a_i, w_i$ in $pref(W.w_i.a_i.A)^*$
```

that each w_i communication is preceded by an W communication and there cannot occur an A communication between W and the following w_i. Thus the effect th_{w_i} can be weakened and the communication assertions of all w_i are replaced by

```
com $w_i$ write $v_i, m_i, \ell c_i$ read $v$
      then $m_i' \wedge v_i' = v \wedge \neg \ell c_i'$.
```

To deal with the occurrence of m in the effect of channels R_i the behaviour is strengthened by

$$\texttt{inv} \ \neg m \Rightarrow \bigwedge_{i=1}^{n} (\neg m_i \wedge v_i = v).$$

Under this invariant the communication assertions of all R_i are modified to

$$\texttt{com } R_i \texttt{ write } \ell c_i \texttt{ read } m, m_i, v_i, v \\ \texttt{then } \ell c_i' = \neg m_i.$$

The initialization

$$\texttt{var } m \texttt{ of bool init true}$$

delivers the validity of the above invariant predicate in the initial state. The channels a_i, R_i, T_i do not write any variable of the invariant while the effect predicates of W and w_i always establish this invariant. Thus only channel A must be strengthened to achieve the redundancy of the invariant:

$$\texttt{com } A \texttt{ write } m \\ \texttt{read } m_1, ..., m_n, v_1, ..., v_n, v \\ \texttt{then } \neg m' \wedge \bigwedge_{i=1}^{n} (\neg m_i \wedge v_i = v).$$

Based on the sequencing constraints given by $pref(W.w_i.a_i.A)$ we eliminate by multiple application of [T4.2] all conjuncts $\neg m_i$ and $v_i = v$. The former ones are established by communications a_i and are stable for all channels R_i and T_i. Thus th_A can be weakened by removing conjunct $\bigwedge_{i=1}^{n} \neg m_i$. The other conjuncts $v_i = v$ are established by the w_i's and are stable for all R_i, T_i and a_i. Thus $\bigwedge_{i=1}^{n} v_i = v$ can also be removed. Moreover, since there is no more a read access to m, this local variable can be removed and the communication assertions of W and A are simplified to

$$\texttt{com } W \texttt{ write } v \texttt{ then } v' = @W \\ \texttt{com } A\,.$$

4.3 Parallel Decomposition

A major goal of the definition of SL and MIX was to support the development of concurrent implementations. The result is a parallel decomposition rule based on the n-ary synchronization operator SYN of MIX. According to this rule, interface components, invariants, trace assertions and communication assertions may be divided over the subspecifications in an arbitrary fashion. Only each local variable declaration has to occur in exactly one S_i.

T 4.3 (*parallel decomposition*)
Let $S = \texttt{spec } \Delta\ TA\ Va\ CA\ I \texttt{ end}$ and $S_i = \texttt{spec } \Delta_i\ TA_i\ Va_i\ CA_i\ I_i \texttt{ end}$ for $i \in 1..n$ be specifications where $\Delta = \bigcup_{i=1}^{n} \Delta_i$, $TA = \bigcup_{i=1}^{n} TA_i$, $Va = \biguplus_{i=1}^{n} Va_i$[2], $CA = \bigcup_{i=1}^{n} CA_i$ and $I = \bigcup_{i=1}^{n} I_i$. Then

$$S \quad \equiv \quad \texttt{SYN}[\ S_1, \cdots, S_n\ ]\,,$$

i.e. S is equivalent to a mixed term where synchronization is applied to all S_i. ∎

We remark that TA enforces synchronization of all communications that appear on channels in more than one of the local trace assertions TA_i. The disjointness condition on the local variables Va_i reflects distributed concurrency. Thus a parallel decomposition must be prepared by rearranging specification components to achieve disjointness. To this end, we shall use the transformations [T3.1] and [T3.2].

Semantically the meaning of TA is a regular language over the set of all channels. Thus modifications of the set of all trace assertions do not change the specified system behaviour as long as the same language is described. The trace merging algorithm [RS91, ORSS92] provides a transformation to join several trace assertions into a single one. A special case of trace merging is given by the following rule.

T 4.4 (*trace projection*)
Let $\texttt{trace } ch_1, ..., ch_k \texttt{ in } re$ be a trace assertion within TA and let $ch_{l_1}, ..ch_{l_m}$ be a subset of its alphabet. Let $\widetilde{re}$ be a regular expression equivalent to re where all occurrences of names $\{ch_1, ..., ch_k\} \setminus \{ch_{l_1}, ..ch_{l_m}\}$ are replaced by nil[3]. Then the addition of the projected trace assertion

$$\texttt{trace } ch_{l_1}, ..ch_{l_m} \texttt{ in } \widetilde{re}$$

to TA does not change the behaviour. ∎

Let us now consider the example. The intended architecture (cf. Figure 5) determines the allocation of the interface components and variable declarations to the subsystems. As mentioned above, each single reader

[2] Union of pairwise disjoint sets.

[3] The constant nil denotes the regular language consisting of the empty word.

register X_i shall get the variables $m_i, \ell c_i, v_i$ and therefore variable v must be placed in WP. Since the write list of a communication assertion w_i consists of variables $m_i, \ell c_i, v_i$ and v, we must achieve a conjunctive form of the effect predicate such that the former three and the latter one variable are not accessed in the same conjunct. Restricting the communication value $@w_i$ to v by [T3.5] gives the effect predicate

$$m_i' \land \neg \ell c_i' \land v_i' = @w_i \land @w_i = v.$$

Then by [T3.1] the unique communication assertion is replaced by the following two:

```
com $w_i$ read $v$ then $@w_i = v$
com $w_i$ write $v_i, m_i, \ell c_i$
        then $m_i' \land \neg \ell c_i' \land v_i' = @w_i$.
```

The specification of a single reader register requires an alternation between write and acknowledge communications. Thus for each $i \in 1..n$ we add by [T4.4] the projections

```
trace $w_i, a_i$ in $pref(w_i.a_i)^*$
```

of trace assertions

```
trace $W, A, w_i, a_i$ in $pref(W.w_i.a_i.A)^*$
```

on channels w_i and a_i to TA. After these preparations the whole specification is decomposed by [T4.3] yielding the following mixed term:

```
CHAN $w_1, ..., w_k, a_1, ..., a_n$
HIDE $w_1, ..., w_n, a_1, ..., a_n$
SYN[ WP, $X_1, ..., X_n$ ]

where
WP =
  spec input $W$ of $V$
       output $A$ of signal
       output $w_1, ..., w_n$ of $V$
       input $a_1, ..., a_n$ of signal
       trace $W, A$ in $pref(W.A)^*$
  $i \in 1..n$ :
    trace $A, W, a_i, w_i$ in $pref(W.w_i.a_i.A)^*$
       var $v$ of $V$
       com $W$ write $v$ then $v' = @W$
       com $w_i$ read $v$ then $@w_i = v$
  end
```

```
$X_i$ = spec input $w_i$ of $V$
          output $a_i$ of signal
          input $R_i$ of signal
          output $T_i$ of $V$
          trace $w_i, a_i$ in $pref(w_i.a_i)^*$
          trace $R_i, T_i$ in $pref(R_i.T_i)^*$
          var $m_i$ of bool
          var $\ell c_i$ of bool
          var $v_i$ of $V$
          com $w_i$ write $v_i, m_i, \ell c_i$
             then $m_i' \land \neg \ell c_i' \land v_i' = @w_i$
          com $a_i$ write $m_i$ then $\neg m_i'$
          com $R_i$ write $\ell c_i$ read $m_i$
                then $\ell c_i' = \neg m_i$
          com $T_i$ read $\ell c_i, v_i$
                then $\ell c_i \Rightarrow @T_i = v_i$
     end
```

Note that each X_i is a copy of the 1-reader-V-safe specification with the interface channels W, A, R, T renamed into w_i, a_i, R_i, T_i. This concludes the construction of an n-reader-V-safe register from n copies of a 1-reader-V-safe register and an auxiliary write process WP.

By applying further transformations from [ORSS92, Ch. 8], we can obtain an occam-like implementation of WP with the following program body:

```
WHILE true SEQ[$W?v$,
              PAR[SEQ[$w_1!v, a_1?$],...,
                  SEQ[$w_n!v, a_n?$] ],
              $A!$].
```

5 Discussion

Our work on transformational design of concurrent systems is close in spirit to the work on UNITY [CM88] and to Back's work on refinement calculus [Bac90]. One of the differences is that UNITY and Back start from iterative programs akin to Dijkstra's do-od loops whereas we start from SL specifications with an explicit treatment of communication. This leads us also to consider a richer class of programming operators and hence transformation rules than previous work.

The case study deals with one of the simpler concurrent implementations of registers originally due to Lamport [Lam86]. Whereas

Lamport gives a correctness proof in some special formalism, we use here a specification formalism combining standard ideas from process algebra, in particular about the semantics of CSP-like communicating processes [Hoa85, Old91a], and from state-based assertional reasoning. Our communication-based specification of the register is inspired by [LG89].

New is our transformational derivation of the concurrent implementation of the n reader safe register. We have applied the same transformational approach to derive also other implementations of register due to [Lam86]: n reader *regular* register implemented by one reader regular registers and by one reader safe registers; safe registers for a finite value set V implemented by safe registers for *binary* values. These transformations rely on further rules dealing with renaming and branching in the flow of control. We have not yet attacked the really difficult constructions for atomic registers. This would be a challenging task for a transformational design.

Acknowledgement. We are grateful to Andrea Sprock who performed an initial study on transformational design of safe and regular registers [Spr92]. This paper is influenced by her work but our proofs are different.

References

[Bac90] R.J.R. Back. Refinement calculus, Part II: Parallel and Reactive Programs. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems - Models, Formalisms, Correctness*, LNCS 430, pages 67–93. Springer-Verlag, 1990.

[Bau87] F.L. Bauer et al. *The Munich Project CIP, Vol. II: The Transformation System CIP-S.* LNCS 292. Springer-Verlag, 1987.

[Bjø89] D. Bjørner et al. A ProCoS project description - ESPRIT BRA 3104. *EATCS Bulletin*, 39:60–73, 1989.

[BW90] J.C.M. Baeten and P. Weijland. *Process Algebra.* Cambridge University Press, 1990.

[CH74] R.H. Campbell and A.N. Habermann. The specification of process synchronisation by path expressions. LNCS 16. Springer-Verlag, 1974.

[CM88] K.M. Chandy and J. Misra. *Parallel Program Design - A Foundation.* Addison-Wesley, 1988.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, London, 1985.

[Kri89] B. Krieg-Brückner. Algebraic specification and functionals for transformational program and meta program development. In J. Diaz and F. Orejas, editors, *Proc. TAPSOFT '89*, LNCS 352. Springer-Verlag, 1989.

[Lam86] L. Lamport. On interprocess communications II. *Distributed Comp.*, 1:86–101, 1986.

[LG89] N.A. Lynch and K.J. Goldman. Distributed algorithms. Technical Report MIT/LCS/RSS 5 6.852 Fall 1988, MIT, 1989.

[LT89] N.A. Lynch and M.R. Tuttle. An introduction to input/ouput automata. Technical Report CWI-Quaterly 2(3), CWI, 1989.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, London, 1989.

[Mor90] C. Morgan. *Programming from Specifictions.* Prentice Hall, London, 1990.

[MP91] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems - Specification.* Springer-Verlag, 1991.

[Old91a] E.-R. Olderog. *Nets, Terms and Formulas: Three Views of Concurrent Processes and Their Relationship.* Cambridge University Press, 1991.

[Old91b] E.-R. Olderog. Towards a Design Calculus for Communicating Programs. In J.C.M. Baeten and J.F. Groote, editors, *Proc. CONCUR '91*, LNCS 527, pages 61–77. Springer-Verlag, 1991. invited paper.

[ORSS92] E.-R. Olderog, S. Rössig, J. Sander, and M. Schenke. ProCoS at Oldenburg: The Interface between Specification Language and occam-like Programming Language. Technical Report Bericht 3/92, Univ. Oldenburg, Fachbereich Informatik, 1992.

[RS91] S. Rössig and M. Schenke. Specification and stepwise development of communicating systems. In S. Prehn and W.J. Toetenel, editors, *VDM'91 Formal Software Development Methods*, LNCS 551, pages 149–163. Springer-Verlag, 1991.

[Spi89] J.M. Spivey. *The Z Notation: A Reference Manual.* Prentice Hall, London, 1989.

[Spr92] A. Sprock. Spezifikation von Registern und Verifikation der Implementation von Registern. Studienarbeit, Univ. Oldenburg, 1992.

[Zwi89] J. Zwiers. *Compositionalty, Concurrency and Partial Correctness - Proof Theories for Networks of Processes and Their Relationship.* LNCS 321. Springer-Verlag, 1989.