(*)

ON THE PARAMETERIZED ALGEBRAIC SPECIFICATION OF CONCURRENT SYSTEMS

E.ASTESIANO⁽¹⁾, G.F.MASCARI⁽²⁾, G.REGGIO⁽¹⁾, M.WIRSING⁽³⁾

(1) Istituto di Matematica, Università di Genova, Italy

(2) Collaboratore esterno, Istituto Applicazioni del Calcolo
C.N.R. Roma, Italy

(3) Fakultät für Mathematik und Informatik, Universität Passau, FRG

Abstract. A technique for specifying concurrent systems is shown, that uses the algebraic specification language ASL. A system is algebraically specified as a transition system and a concurrent system is the result of composing systems by three basic operations: synchronization, parallel composition and monitoring. These operations are schematically described using the parameterization concept of ASL and they are in the same time examples for the power of ASL since they cannot be formally specified in other specification languages. Each particular synchronization, parallel composition or monitoring is defined by instantiating on appropriate parameters a unique specification, which produces a transition system out of an input transition system. By combining the three operations we obtain a formal support for a methodology of hierarchical and modular specification of concurrent systems. Moreover it is shown that combining tools for defining semantics in ASL with the above parameterized schema provides a standard way for giving to a concurrent system a variety of semantics depending on observability constraints.

Introduction. It is well known that abstraction is a major problem in specification and to cope with that various brands of abstract data type techniques have been developed, now generalized to allow specifications of programming languages (cf. e.g. [BW1]). When the task is the specification of concurrent systems there is a need of combining a good level of abstraction with the use of some specific structures, serving both as a guide in specifying and as a tool for deriving properties.

The main aim of this paper is precisely to connect an abstract data type approach to a view of concurrency based on some very fundamental parameterized operations and structures. In a sense we try to lay a bridge between the purely translational approach (specify a system by translating into a basic language/model, like CCS or CSP) and the approach which considers a concurrent system simply as an abstract data type [BW2]. Note that our approach is more semantics than syntax oriented; we can see many points of contacts with the aims of Mosses' ASA theory [Mo].

Basically we view a concurrent system as a transition system obtained by composing other transition systems, representing the component processes and we decompose the description of a system into three basic operations of composition: synchronization, parallel composition and monitoring. Our approach presents three main technical features.

(*) Work partially supported by CNR ITALY PFI-CNET, by a grant MPI and by the ESPRIT project METEOR.

- We use the algebraic specification language ASL [W], which provides tools for the description of parameterized abstract structures, not available in other specification languages: apart from specifications also other types of (high-level) objects such as signatures or sets of formulas can be passed as parameters. That way we define static structures (states of systems), dynamic structures (transition systems), composition of systems (the operations above).
- In particular we can define specific synchronization, parallel composition or monitoring mechanisms as instantiations on appropriate parameters of a unique operation (hierarchical construction) which derives a system out of an input system.
- Every specification has a precise mathematical semantics, defined by ASL. Due to the special form of the axioms in our hierarchical construction we can show that the algebraic semantics corresponds to an abstract operational semantics. Moreover, by using the ASL "observe" construct, we show how to give a variety of semantics depending on what we want to observe in a system; that is essential in concurrency, where now it is recognized that no single semantic equivalence can capture the variety of meanings while preserving a sufficient level of abstraction. .

Our hierarchical and parameterized approach is the formal support for a precise methodology of modular specification. For example, we can show that the methodology underlying the SMO LCS approach [AMMR],[AR] developed in project CNET can be expressed by describing systems via a specification consisting of the nested instantiations of our composition operations. In section 1 we present an ASL description of hierarchical transition systems; in section 2 we define the basic concurrent schemata for synchronization, parallel composition and monitoring; in section 3 we discuss semantic issues.

1. HIERARCHICAL TRANSITION SYSTEMS.

The concurrent schemata of the following section are based on an operational viewpoint of communicating processes, which is formally expressed by describing processes with the help of (hierarchically structural) transition systems. Therefore we describe "hierarchical transition systems".

Note that we use partial algebra semantics, with minimally defined models; for these and other basic algebraic notions see [BW2]. For a description of ASL we refer to [SW] or [W], although we hope it will be possible to understand the principal ideas of our schemata with the help of the informal explanations of ASL in this section.

1.1 Transition systems. Usually, a transition system such as a term rewriting system is understood as a binary relation between terms, the so-called "single step transition relation", and this relation is defined by a set of rewriting rules. For the application to concurrency this notion is extended in two ways (cf. e.g. [P1]). Instead of a binary relation a ternary one is used to express the rewriting of terms (here called "states") as well as to indicate further information such as output or synchronization of data (here called "flags"); moreover instead of simple rewriting rules, conditional axioms are used where the premises of the axioms express applicability conditions.

Algebraically we say that a conditional transition system is based on two specifications STATE and FLAG of data structures with the sorts state and flag. Then a ternary boolean function symbol

$$.\overset{f}{\Longrightarrow}. : \text{state} \times \text{flag} \times \text{state} \longrightarrow \text{bool}$$

is defined by axioms E_T of the form $\text{cond} \supset s \overset{f}{\Longrightarrow} s' = \text{true}$ where cond is an applicability condition, consisting usually of a conjunction of equations.

Usually we write $s \overset{f}{\Longrightarrow} s'$ instead of $s \overset{f}{\Longrightarrow} s' = \text{true}$.

In an algebraic style this can be presented as the following specification $\overline{\text{TS}}$:

$$\begin{array}{l} \text{enrich } \text{BOOL} + \text{STATE} + \text{FLAG} \text{ by} \\ \text{opns } .\overset{f}{\Longrightarrow}. : \text{state} \times \text{flag} \times \text{state} \longrightarrow \text{bool} \\ \text{axioms } E_T \end{array}$$

In the body of $\overline{\text{TS}}$ we use the ASL-operator "+" which builds the sum of two specifications $T+T'$. Formally the signature of $T+T'$ is the union Σ_+ of the signatures of T and T' and a Σ_+ -algebra is a model of $T+T'$ iff its restriction to the signature of T is a model of T and also its restriction to the signature of T' is a model of T' . The other specification operator enrich T by sorts S opns F axioms E corresponds to the enrich-operation of CLEAR and can be derived using "+". The signature of this expression is the signature of T extended by the sorts in S and the function symbols in F . An algebra of this extended signature is a model if it satisfies the axioms E and its restriction to the signature of T is a model of T .

More generally, in ASL we get a function which has the specifications STATE and FLAG and a set of formulas (the conditional axioms) as parameters and which yields a specification (of the transition system) as result. In the following ASL-specification TS of a schema for conditional transition systems the reserved word specfunc expresses that a function with a specification as result is defined. Parameters are written similarly to typed λ -calculus: " $\lambda_m x:b.$ " indicates that x is a formal parameter of mode (or type) m which is subject to the parameter restriction b . There we have a restriction of the third parameter, the set of formulas E_T , which is an application of a

function from the domain formulas of sets of formulas into booleans (boolfunc) saying that every element of E_T must have the syntactic form (*).

$$(*) \quad \text{cond} \supset (s \overset{f}{\Longrightarrow} s') = \text{true}$$

Moreover in the following by writing $T\langle s \rangle$ (where T is a specification expression and s a sort) we mean that s is considered the main sort of T . Similarly specfunc $F \equiv \lambda \text{spec } T.e$ is an abbreviation of specfunc $F \equiv \lambda \text{spec } T, \text{sort } t:t \in \text{sorts}(T).e$ and, correspondingly, the instantiations $F(A)$ and $F(A\langle s \rangle)$ are abbreviations of $F(A,a)$ and $F(A,s)$.

$$\begin{array}{l} \text{specfunc } \text{TS} \equiv \lambda \text{spec } \text{STATE}, \text{FLAG}, \\ \text{formulas } E_T : E_T \text{ is "cond } \supset (s \overset{f}{\Longrightarrow} s') = \text{true}" . \overline{\text{TS}} \end{array}$$

where boolfunc $\text{is "cond } \supset (s \overset{f}{\Longrightarrow} s') = \text{true}" \equiv \lambda \text{formulas } E$.

$$\begin{array}{l} \forall \text{formula } e : e \in E \text{ implies } \text{is } \supset (e) \text{ and } \text{is} = (\text{son}(2,e)) \text{ and} \\ \text{root}(\text{son}(1,\text{son}(2,e))) = .\overset{f}{\Longrightarrow}. : \text{state} \times \text{flag} \times \text{state} \longrightarrow \text{bool} \text{ and} \\ \text{root}(\text{son}(2,\text{son}(2,e))) = \text{true} : \longrightarrow \text{bool} \end{array}$$

For any schema ψ of formulas (above ψ is the schema (*)) the function $\text{is } \psi$ analyses whether every formula in a set of formulas has the syntactic form ψ

by means of the tree-like operators *root*, *son* and *isa* (where $\text{isa}(e)=\text{true}$ iff $\text{root}(e)=\alpha$). In the following the exact definition is left to the reader. The specification function *TS* provides a syntactic schema for transition systems. No semantic properties are required. By instantiating *TS* with actual parameters we get a specification satisfying the requirements for signature and form of axioms.

Example CL. We illustrate the definitions by a small concurrent language CL. For a more significant application of the method to a real project see [AMRZ]. A program of CL consists of a set of processes. The processes can cooperate by means of a CSP-like communication mechanism (statements ? and !) and by means of a shared memory assigning to a local variable the content of a shared variable and viceversa (statements read and write). Moreover a process can create other processes (by the statement start) and there is a construct (or) for nondeterministic choice. A process can either terminate normally, when it has completed the execution of its body, or fail, when it tries to communicate with a terminated process. The program execution terminates when all processes are terminated (either normally or failing). To every program will be given an input-output semantics (see section 3.3) where the outputs are the states of the shared memory. For lack of space we only give the fragments of the full specification which are related to the basic concepts of the paper; it should be clear that all omitted details can be easily defined in ASL. We start with the basic transition system PBTS which specifies the behaviour of the processes:

$$\text{spec PBTS} \equiv \text{TS}(\text{PS}, \text{PFLAG}, \text{PACT})[\rightarrow / \Rightarrow].$$

$T[\rightarrow / \Rightarrow]$ denotes the renaming of the operation symbol " $\Rightarrow$ " by " $\rightarrow$ " in the signature of *T* and all models of *T* (formally this is an application of the specification operation *derive*; see e.g. [SW]).

The states of PBTS (specification *PS*) are triples of process identifiers (PID), statements (specification *STAT* with the zeroary function symbol Λ for the null statement) and local memories (MEM), plus a special element *nil*, needed for specifying creation and termination of processes. The local memory MEM is represented by finite maps from variable identifiers (VID) to values (specification *VAL* with the zeroary function symbol vl_0), the parametric specification MAP is given in the appendix.

$$\begin{aligned} \text{spec PS} &\equiv \text{enrich PROD}(\text{PID}, \text{STAT}, \text{MEM}) [\text{ps/prod}] \text{ by} \\ &\quad \text{opns nil} : \longrightarrow \text{ps} \\ \text{spec MEM} &\equiv \text{MAP}(\text{VID}, \text{VAL}) [\text{mem/map}] \end{aligned}$$

The flags of PBTS (specification *PFLAG*) are in correspondence with the concurrent statements of CL, exception made for τ , which corresponds to the sequential statements (Milner's internal action).

The set of formulas describing the process actions *PACT* is the union of *CACT* (concurrent actions) and *SACT* (sequential actions). We do not specify *SACT*, whose formulas have the form:

$$\text{cond } \supset \langle p, \text{seqstat}, m \rangle \xrightarrow{\tau} \langle p, \Lambda, m' \rangle.$$

Note that a complete specification of the sequential statements can be given separately from *CACT*. We now give the specification of *CACT* that will be essential for the rest of the example.

Formula C2 completely defines the global nondeterministic choice statement, for the operator "or" is commutative ($\text{st}_1 \text{ or } \text{st}_2 = \text{st}_2 \text{ or } \text{st}_1$). Note that a process in a

state $\langle p, \text{st}, m \rangle$, where *st* is an input statement (? or read) can perform an infinite number of actions, one for every possible value which can be received (formulas C3 and C5). "empty" and " $.[./.]$ " are function symbols of the parametric specification MAP defining respectively the map with empty domain and the usual substitution operation. Formula C7 denotes the capability of a process of creating a new process named p_1 , with local variables $v_1, \dots, v_n$ (with initial value vl_0) and the body given

by the statement st. Formula C8 defines the capability of being created, which is represented by a transition of the null process (nil) in the initial state of the created process and is denoted by the flag new(...).

formulas CACT $\equiv$

$$\begin{aligned}
 \{C1 & \langle p, st, m \rangle \xrightarrow{f} \langle p, \Lambda, m' \rangle \supset \langle p, st; st_1, m \rangle \xrightarrow{f} \langle p, st_1, m' \rangle, \\
 C2 & \langle p, st_1, m \rangle \xrightarrow{f} \langle p, st'_1, m' \rangle \supset \langle p, st_1 \text{ or } st_2, m \rangle \xrightarrow{f} \langle p, st'_1, m' \rangle, \\
 C3 & \langle p, v?p', m \rangle \xrightarrow{?(p, v_1, p')} \langle p, \Lambda, m[v_1/v] \rangle, \\
 C4 & \langle p, v!p', m \rangle \xrightarrow{!(p, m(v), p')} \langle p, \Lambda, m \rangle, \\
 C5 & \langle p, \text{read}(v, sv), m \rangle \xrightarrow{\text{read}(v_1, sv)} \langle p, \Lambda, m[v_1/v] \rangle, \\
 C6 & \langle p, \text{write}(v, sv), m \rangle \xrightarrow{\text{write}(m(v), sv)} \langle p, \Lambda, m \rangle \} \cup \\
 & \quad \text{start}(p_1, \{v_1, \dots, v_n\}, st) \\
 \{C7 & \langle p, \text{start}(p_1, \{v_1, \dots, v_n\}, st), m \rangle \xrightarrow{\text{new}(p, \{v_1, \dots, v_n\}, st)} \langle p, \Lambda, m \rangle, \\
 C8 & \text{nil} \xrightarrow{} \langle p, st, \text{empty}[v_1/v_1] \dots [v_1/v_n] \rangle \mid n \geq 0 \} \cup \\
 \{C9 & \langle p, \Lambda, m \rangle \xrightarrow{\text{end}(p)} \text{nil} \} \cup \\
 \{C10 & \langle p, st, m \rangle \xrightarrow{\text{fail}(p, p')} \text{nil} \mid \text{term } st \in \{v?p', v!p'\} \}
 \end{aligned}$$

1.2 Hierarchical systems. A hierarchical transition system refines the concept of transition system defining a new transition system HTS based on a "basic transition system" BTS with sorts bstate (basic states), bflag (basic flags) and transition operation $\xrightarrow{\quad}$. HT is based on specification STATE and FLAG and its relationship

to BTS is expressed in the axioms E_H of form $\text{cond} \wedge \bigwedge_j (bs_j \xrightarrow{bt_j} bs'_j) \supset s \xrightarrow{f} s'$.

Thus HT is simply the sum of the basic transition system BTS and of another transition system i.e. of an appropriate instantiation of TS:

$$BTS + TS(\text{STATE}, \text{FLAG}, E_H).$$

Similarly to the specification of transition systems, a schema for hierarchical transition systems is specified as a function with a specification as result. It has as parameters the basic transition system BTS and the parameters STATE, FLAG, E_H for the new system.

If srt is a sort then $\langle \text{srt} \rangle$ indicates the trivial specification in which there is just one sort and no operations nor axioms. An expression of the form specfunct $F \equiv \lambda \text{spec } G: X \dots$ where X is a specification indicates that in every instantiation of F the actual parameter instantiating G must satisfy the specification X.

$$\begin{aligned}
 \text{specfunct HTS} & \equiv \lambda \text{spec } BTS : TS(\langle \text{bstate} \rangle, \langle \text{bflag} \rangle, \emptyset) \{ \xrightarrow{\quad} / \xRightarrow{\quad} \}, \\
 & \quad \text{spec STATE, FLAG,} \\
 \text{formulas } E_H & : E_H \text{ is "cond} \wedge \bigwedge_{j=1}^n bs_j \xrightarrow{bf_j} bs'_j \supset s \xrightarrow{f} s' ". HT
 \end{aligned}$$

2. CONCURRENT SCHEMATA.

In this section transition system schemata for concurrent processes are established w.r.t. three different semantic features:

- synchronization
- parallelism (without synchronization)
- monitoring.

These schemata will all be special hierarchical transition systems; i.e. particular instantiations of HTS. To get a description of some particular parallel programming language (or system) one can combine them in various ways using the operators of ASL, in particular the function application.

2.1 Synchronization. A synchronous transition system $\overline{STS}$ can be seen as a hierarchical transition system and thus as an instantiation ("makro-expansion") of the schema HTS where:

- states of the basic system are combined to multisets (mset stands for multiset) of states, that, together with an information part, will form the states SST of the resulting system according to some conditions expressed in E_{SST} ;
- the specification SYNC of the flags of the new system is done in two steps: the set of the possible flags of the new system is given as a parameter SFLAG, then the admissible new flags are filtered out by a function

$$iss : sflag \times mset(flag) \times inf \longrightarrow bool$$
 denoting that an sflag is the (nondeterministic) result of the synchronization of a multiset of flags;
- for the transformation of the information in a transition from one state to another we need a function

$$sit : sflag \times inf \longrightarrow inf$$
 which gives the new information part;
- the axioms E_{SYNC} define the functions iss and sit;
- with iss and sit the axioms for a synchronized system are given by a fixed set of axioms E_S .

In the following specification these properties are formally expressed. In contrast to the specification of transition systems the synchronization schema involves semantic properties such as the axiom set E_S specifying the result of a synchronization and the multiset structure of the first component of STATE. To describe the admissible states as pairs of multisets of basic states and an information part the parameterized specifications MSET of finite multisets and PRODCOND of pairs of objects are used which are defined in the appendix.

$$\underline{spec} \overline{STS} \equiv HTS(BTS, SST, SYNC \langle sflag \rangle, E_S)$$

where $SST = PRODCOND(MSET(\langle \langle bstate \rangle \rangle), INF, E_{SST}) [sstate/prod]$

$$SYNC = \underline{enrich} MSET(\langle \langle bflag \rangle \rangle) + SFLAG + INF \text{ by}$$

$$\underline{opns} \quad \begin{array}{ll} iss : sflag \times mset(bflag) \times inf & \longrightarrow bool \\ sit : sflag \times inf & \longrightarrow inf \end{array}$$

$$\underline{axioms} \quad E_{SYNC}$$

$$E_S = \{ iss(sf, bf_1 | \dots | bf_n, i) = true \mid \bigwedge_{j=1}^n bs_j \xrightarrow{bf_j} bs'_j \} \supset$$

$$\langle bs_1 | \dots | bs_n, i \rangle \xrightarrow{sf} \langle bs'_1 | \dots | bs'_n, sit(sf, i) \rangle \mid n \geq 1$$

Note that in the resulting system $s \xrightarrow{f} s'$ iff f is the result of a synchronization, and also that, for $n=1$, also actions (or action capabilities) of a unique agent can be seen as a result of a synchronization. Creation of processes can be handled at this step, by synchronizing the creating process with a null state process evolving into the initial state of the new process (see the example in what follows). A synchronized action can influence the information.

By considering the unspecified identifiers of STS as formal parameters one gets the synchronization schema STS:

$$\underline{spec} \text{funct } STS \equiv \lambda \underline{spec} BTS : TS(\langle \langle bstate \rangle \rangle, \langle \langle bflag \rangle \rangle, \emptyset) [\longrightarrow / \Longrightarrow],$$

$$\underline{spec} INF, SFLAG,$$

$$\underline{formulas} E_{SST}, E_{SYNC} . \overline{STS}$$

Example CL. The synchronous interactions between the processes of a CL program are described by CLSTS.

$$\underline{spec} CLSTS \equiv STS(PBTS, CLINF, PFLAG, E_{CLSST}, E_{CLSYNC})$$

The information needed for synchronization is described as a cartesian product of shared memory states (SMEM) and sets of process identifiers (TERM, representing the names of the terminated processes),

$$\text{spec CLINF} \equiv \text{PROD}(\text{SMEM}, \text{TERM})[\text{inf/prod}] .$$

The shared memory states are defined as finite mappings from shared variable identifiers (SVID) to values (VAL):

$$\text{spec SMEM} \equiv \text{MAP}(\text{SVID}, \text{VAL})[\text{smem/map}] .$$

$$\text{spec TERM} \equiv \text{SET}(\text{PID})[\text{term/set(pid)}]$$

The flags of the synchronous actions are the same as for process actions (the actual parameter for SFLAG is PFLAG).

The axioms E_{CLSST} ensure that a process is not contemporaneously present and terminated and that nil is the neutral element for the parallel composition of processes.

$$\text{formulas } E_{\text{CLSST}} \equiv \{ \neg D(\langle \langle p, \text{st}, m \rangle \mid \text{mps}, \langle \text{sm}, \{p\} \cup \text{tr} \rangle \rangle), \langle \text{nil} \rangle \mid \text{mps}, i \rangle = \langle \text{mps}, i \rangle \} .$$

The synchronous actions are defined by the following set of axioms.

$$\begin{aligned} \text{formulas } E_{\text{CLSYNC}} \equiv & \{ S1 \text{ iss}(\tau, \text{mfs}, i) = \text{true} \mid \text{term mfs} \in \{ \tau, \{ (p, \text{vl}, p_1) \} \{ \{ (p_1, \text{vl}, p) \}, \\ & \{ \text{start}(p, \text{vs}, \text{st}) \} \{ \text{new}(p, \text{vs}, \text{st}) \} \} \} \cup \\ & \{ S2 \text{ sit}(\tau, i) = i, \\ & S3 \text{ iss}(\text{read}(\text{vl}, \text{sv}), \{ \text{read}(\text{vl}, \text{sv}) \}, \langle \text{sm}, \text{tr} \rangle) = \langle \text{sm}(\text{sv} = \text{vl}) , \\ & S4 \text{ sit}(\text{read}(\text{vl}, \text{sv}), i) = i, \\ & S5 \text{ iss}(\text{write}(\text{vl}, \text{sv}), \{ \text{write}(\text{vl}, \text{sv}) \}, i) = \text{true}, \\ & S6 \text{ sit}(\text{write}(\text{vl}, \text{sv}), \langle \text{sm}, \text{tr} \rangle) = \langle \text{sm}[\text{vl}/\text{sv}], \text{tr} \rangle, \\ & S7 \text{ iss}(\text{end}(p), \{ \text{end}(p) \}, i) = \text{true}, \\ & S8 \text{ sit}(\text{end}(p), \langle \text{sm}, \text{tr} \rangle) = \langle \text{sm}, \text{tr} \cup \{p\} \rangle, \\ & S9 \text{ iss}(\text{end}(p), \{ \text{fail}(p, p_1) \}, \langle \text{sm}, \text{tr} \rangle) = \{ p_1 \in \text{tr} \} \} . \end{aligned}$$

The axiom schema S1 expresses handshaking communication, creation of new processes and the fact that every internal process action is understood as synchronous action. The process action capabilities of form $\text{read}(\text{vl}, \text{sv})$ are accepted as synchronous actions only if the value of sv in the actual state of the shared memory is vl (S3). The synchronous actions with flags $\text{write}(\text{vl}, \text{sv})$ update the value of sv transforming the first component of the information part (S6). When a process terminates (normally or by failing) its name is added to the second component of the information part (S8). The process capabilities of the form $\text{fail}(p, p_1)$ become synchronous actions only when the process p_1 is terminated (S9).

2.2 Parallel composition. Parallel transition systems $\overline{\text{PTS}}$ are hierarchical transition systems, similar to synchronized systems, but with some important differences:

- the states and the flags are the same as in the basic system: the states consist of a product of an information part and of a multiset of some component states; this is expressed by a parameter restriction on the basic transition system BTS;
- the transitions of the system include those of the basic system, hence the basic transition is also called " $\Rightarrow$ ". The basic transitions are augmented by a binary composition expressed by the semigroup operation $\parallel$ on flags, and an axiom E_p specifying the particular transition resulting from the parallel composition;
- a function pit represents the transformation of the information resulting from composing the two component transformations.

The basic intuitive idea behind $\overline{\text{PTS}}$ is that the actions are composed to form a new action, corresponding to a parallel execution of the component actions. The formulas

E_{PAR} specify the function pit and the definedness of the composition operation.

$$\text{spec } \overline{\text{PTS}} \equiv \text{HTS}(\text{BTS}, \langle \text{state} \rangle, \text{PAR} \langle \text{flag} \rangle, E_p)$$

where

$\text{PAR} = \text{sorts } \text{flag}, \text{inf}$
 $\text{opns } // : \text{flag} \times \text{flag} \longrightarrow \text{flag}$
 $\text{pit} : \text{flag} \times \text{inf} \longrightarrow \text{inf}$
 $\text{axioms } \{f_1 // f_2 = f_2 // f_1, (f_1 // f_2) // f_3 = f_1 // (f_2 // f_3)\} \cup E_{\text{PAR}} \text{ and}$
 $E_P = \{D(f_1 // f_2) \wedge (\bigwedge_{j=1}^2 \langle \text{ms}_j, i \rangle \xrightarrow{f_j} \langle \text{ms}'_j, i_j \rangle) \supset$
 $\quad \langle \text{ms}_1 | \text{ms}_2, i \rangle \xrightarrow{f_1 // f_2} \langle \text{ms}'_1 | \text{ms}'_2, \text{pit}(f_1 // f_2, i) \rangle\}$

Similarly to synchronization we get the following schema for parallel composition:

$\text{specfunc } \text{PTS} \equiv$
 $\lambda \text{ spec } \text{BTS} : \text{TS}(\text{PRODCOND}(\text{MSET}(\langle \text{state} \rangle), \langle \text{inf} \rangle, \emptyset) [\text{bstate/prod}], \langle \text{flag} \rangle, \emptyset),$
 $\text{formulas } E_{\text{PAR}} . \text{PTS}$

Note that for an instantiation of PTS the specification of FLAG (which is a subspecification of BTS) must admit models which are not term generated models, since otherwise the introduction of the function symbol $//$ could cause inconsistencies in PTS.

Example CL. The parallel composition between the synchronous actions of the CL processes is defined by the formulas E_{CLPAR} .

$\text{spec } \text{CLPTS} \equiv \text{PTS}(\text{CLSTS}, E_{\text{CLPAR}}).$

$\text{formulas } E_{\text{CLPAR}} \equiv$

$\{P1 \ D(\text{write}(v1, sv) // \text{write}(v1', sv')) \leftrightarrow (sv \neq sv')\} \cup$
 $\{P2 \ D(f) \supset D(f // f') \mid \text{term } f' \in \{\text{read}(v1, sv), \text{end}(p), \tau\}\} \cup$
 $\{P3 \ \text{sit}(f, i) = i' \supset \text{pit}(f, i) = i',$
 $P4 \ \text{pit}(f_1 // f_2, i) = \text{pit}(f_1, \text{pit}(f_2, i))\}.$

P1 and P2 forbid two contemporaneous writings on the same shared variable while allow contemporaneous readings. It can be easily proved that $\text{pit}(f_1, \text{pit}(f_2, i))$ is equal to $\text{pit}(f_2, \text{pit}(f_1, i))$ whenever $D(f_1 // f_2)$ is true, so the specification is consistent.

Different assumptions on the mutual exclusion of the accesses to the shared variables can be expressed modularly by changing P1 and P2. For example

$\{P'_1 \ D(f_1 // f_2) \leftrightarrow (sv \neq sv') \mid \text{term } f_1 \in \{\text{write}(v1, sv), \text{read}(v1, sv)\},$
 $\text{term } f_2 \in \{\text{write}(v1', sv'), \text{read}(v1', sv')\}\} \cup$
 $\{P'_2 \ D(f) \supset D(f // f') \mid \text{term } f' \in \{\text{end}(p), \tau\}\}$

forbid contemporaneous accesses to a single shared variable.

2.3 Monitoring. Monitored systems arise from a basic transition system by restricting the possible transitions to those that are allowed by the monitor. The restriction is achieved by using (in a specification MON) the function

$\text{mon} : \text{bflag} \times \text{bstate} \times \text{minf} \longrightarrow \text{mflag}$

which depends also on some information in MINF. This information is updated in a transition by the function

$\text{mit} : \text{bflag} \times \text{minf} \longrightarrow \text{minf}.$

A basic transition with flag f , starting from a basic state bs is allowed by the monitor according to the monitor information mi , when $\text{mon}(f, bs, mi)$ is defined; the resulting value is in MFLAG and represents the interaction of that transition with the external world (see E_M).

The states, MSTATE , of a monitored system are specified by a conditional product of the basic states of BTS and a monitor information part MINF.

We usually assume as further constraint (which is not specified in the following) that mon , for any fixed f and mi , depends only on the multiset of all flags of the

transitions starting in a state.

spec $\overline{MTS} \equiv HTS(BTS, MSTATE, MON\langle mflag \rangle, E_M)$
where

$MSTATE = \text{PRODCOND}(\langle \text{bstate} \rangle, MINF, E_{MST}) [mstate/prod],$

$MON = \text{enrich } BTS + MFLAG \text{ by}$

opns $\text{mon} : bflag \times bstate \times minf \longrightarrow mflag$

$\text{mit} : bflag \times minf \longrightarrow minf$

axioms E_{MON}

and

$E_M = \{ \text{mon}(bf, \langle \bar{s}, i \rangle, mi) = mf \wedge \langle s, i \rangle \xrightarrow{bf} \langle s', i' \rangle \wedge s \subseteq \bar{s} \supset$
 $\langle \langle \bar{s}, i \rangle, mi \rangle \xrightarrow{mf} \langle \langle \bar{s} - s \rangle | s', i' \rangle, \text{mit}(bf, mi) \rangle \}$

Intuitively monitoring can express some kind of centralized control (e.g. interleaving mode vs. parallel mode, fairness constraints and so on) or the result of some global external observation (cutting capabilities needing synchronization when the system is isolated, converting flags to some external behaviour).

By abstracting from unspecified identifiers we get the following schema:

specfunct $\overline{MTS} \equiv$

$\lambda \text{ spec } BTS : TS(\text{PRODCOND}(\text{MSET}(\langle \text{state} \rangle), \langle \text{in} \rangle, \emptyset) [bstate/prod], \langle \text{flag} \rangle, \emptyset) [\rightarrow / \Rightarrow],$

spec $MINF, MFLAG,$

formulas $E_{MST}, E_{MON}, \overline{MTS}$

Example CL. We specify an interleaving mode for the CL processes; only one action resulting from a synchronization is allowed at each step.

spec $CLMTS \equiv MTS(CLPTS, NULL, NULL, E_{CLMST}, E_{CLMON})$

No monitor informations are needed; so the actual parameter for MINF is the specification NULL, which has just one sort, one zeroary function symbol, no axioms and such that every model has just one element (that can be obtained by using the construct reachable of ASL).

Every state of CLPTS becomes a state of CLMTS:

formulas $E_{CLMST} \equiv \{ D(sst) \supset D(\langle sst, null \rangle) \}.$

Since we are interested in an input-output semantics of CL we do not care about the flags of the system CLMTS; then the actual parameter for MFLAG is NULL.

formulas $E_{CLMON} \equiv$

$\{ M1 \text{ mon}(f, sst, null) = null \mid \text{term } f \in \{ \tau, \text{read}(vl, sv), \text{write}(vl, sv), \text{end}(p) \} \} \cup$

$\{ M2 \text{ mit}(f, null) = null \}.$

These formulas define the wanted interleaving mode. Note that $\text{mon}(f, f_2, \dots)$ is undefined in minimally defined models (then the contemporaneous execution of several synchronous actions is prohibited).

Due to the modularity of our methodology, by choosing a different axiom schema M1 we could define different modes for the CL processes. For example, for a parallel mode where all parallel actions are allowed, we use the axiom

$M1' \text{ mon}(f, sst, null) = null;$

while for a parallel mode, where, when a process fails, the program execution stops, we use the axiom

$M1'' (\nexists ps \in \text{emps } ps \xrightarrow{\text{fail}(p, p')} nil) \supset \text{mon}(f, \langle \text{mps}, i \rangle, null) = null.$

2.4 Iterated hierarchical systems. Finally we give two examples illustrating how the previous specifications can be combined. Structured Linear Concurrent Systems SLCS can be obtained by instantiating PTS with STS.

spec $\overline{SLCS} \equiv PTS(STS(BTS, INF, SFLAG, E_{SST}, E_{SYNC}), E_{PAR})$

By λ -abstraction we get

$$\text{specfunc } \text{SLCS} \equiv \lambda \text{spec } \text{BTS} : \text{TS}(\langle \text{bstate} \rangle, \langle \text{bflag} \rangle, \emptyset) [\longrightarrow / \Longrightarrow], \\ \text{spec } \text{INF}, \text{SFLAG}, \\ \text{formulas } E_{\text{SST}}, E_{\text{SYNC}}, E_{\text{PAR}} . \overline{\text{SLCS}}$$

Structured Monitored Linear Concurrent Systems SMoLCS are obtained by instantiating MTS with SLCS (see [AMMR],[AR]) and then by applying λ -abstraction.

$$\text{specfunc } \text{SMoLCS} \equiv \lambda \text{spec } \text{BTS} : \text{TS}(\langle \text{bstate} \rangle, \langle \text{bflag} \rangle, \emptyset) [\longrightarrow / \Longrightarrow], \\ \text{spec } \text{INF}, \text{SFLAG}, \text{MINF}, \text{MFLAG}, \\ \text{formulas } E_{\text{SST}}, E_{\text{SYNC}}, E_{\text{PAR}}, E_{\text{MST}}, E_{\text{MON}} . \\ \text{MTS}(\text{SLCS}(\text{BTS}, \text{INF}, \text{SFLAG}, E_{\text{SST}}, E_{\text{SYNC}}, E_{\text{PAR}}) [\longrightarrow / \Longrightarrow], \\ \text{MINF}, \text{MFLAG}, E_{\text{MST}}, E_{\text{MON}})$$

Example CL. CLMTS is an example of SMoLCS system; indeed

$$\text{CLMTS} = \text{SMoLCS}(\text{PBTS}, \text{CLINF}, \text{PFLAG}, \text{NULL}, \text{NULL}, E_{\text{CLSST}}, E_{\text{CLSYNC}}, E_{\text{CLPAR}}, E_{\text{CLMST}}, E_{\text{CLMON}}).$$

3. SEMANTICS.

In this paragraph we consider three kinds of semantics for transition systems by analysing the hierarchical specifications $\overline{\text{TS}}$ (see 1.1), $\overline{\text{TREE-SEM}}$ and $\overline{\text{OBS-SEM}}$.

The structure of the models of hierarchical specifications has been studied in the literature (cf. e.g. [BW2],[WPPDB]) and we will apply these results in this section. For notions of the theory of partial abstract types such as "minimally defined models", "hierarchy-persistence", "partial completeness", "monomorphic type" we refer to [BW1] and [WPPDB].

3.1 Operational semantics. The specification $\overline{\text{TS}}$ permits already to consider an operational semantics in which intermediate states and flags are observable.

In order to analyse the semantics of the specification of the transition system $\overline{\text{TS}}$ of the previous section we fix specifications STATE , FLAG and the set of formulas E_{T} and make the following technical assumptions.

- The specification FLAG is monomorphic, that is all the models of FLAG are isomorphic.
- The axioms of STATE are positive conditional formulas (see [BW2]) of the form :

$$\bigwedge_{i=1}^n D(p_i) \wedge \bigwedge_{j=1}^k (D(r_j) \wedge r_j = q_j) \supset \bigwedge_{h=1}^m e_h$$

where e_h is of the form $D(t)$ or $t_1 = t_2$; then STATE has an initial model I_{STATE} [BW2].

- The preconditions "cond" in any axiom $\text{cond} \supset t \xrightarrow{f} t'$ are of the form $\bigwedge_i (D(u_i) \wedge u_i = v_i)$ where u_i is of sort bool or flag .

Then we can consider $\overline{\text{TS}}$ as hierarchical specification with primitive sorts bool and flag and the sorts of STATE as non-primitive sorts.

If A is a specification, then $\text{Sig}(A)$ indicates the signature of A ; if Σ is a signature, srt and srt' are sorts, then $W_{\Sigma} \upharpoonright_{\text{srt}}$ and $W_{\Sigma}(\{\text{srt}' \ x\}) \upharpoonright_{\text{srt}}$ indicate the set of the ground terms on Σ of sort srt and the set of the terms on Σ of sort srt with at most one variable x of sort srt' .

Proposition 1.

There exists an initial model $I_{\overline{TS}}$ of $\overline{TS}$ such that for every $s, s' \in W_{\text{Sig}(\overline{TS})} \mid \text{state}$:

- (1) $I_{\overline{TS}} \models s=s'$ iff $\text{STATE} \vdash s=s'$ or $(\overline{TS} \not\vdash D(s) \text{ and } \overline{TS} \not\vdash D(s'))$,
- (2) $I_{\overline{TS}} \models D(s)$ iff $\overline{TS} \vdash D(s)$,
- (3) $I_{\overline{TS}} \models s \xrightarrow{f} s'$ iff $\overline{TS} \vdash s \xrightarrow{f} s'$,
- (4) $I_{\overline{TS}} \models \neg D(s \xrightarrow{f} s')$ iff $\overline{TS} \not\vdash s \xrightarrow{f} s'$.

There exists a weakly terminal model $Z_{\overline{TS}}$ of $\overline{TS}$ such that

for every $s, s' \in W_{\text{Sig}(\overline{TS})} \mid \text{state}$:

- (1) $Z_{\overline{TS}} \models s=s'$ iff (for every $c \in W_{\text{Sig}(\overline{TS})}(\{\text{state } x\}) \mid \text{state}, \overline{s} \in W_{\text{Sig}(\overline{TS})} \mid \text{state}$,

$$\begin{array}{c} f \in W_{\text{Sig}(\overline{TS})} \mid \text{flag} : \\ \overline{TS} \vdash c[s] \xrightarrow{f} \overline{s} \text{ iff } \overline{TS} \vdash c[s'] \xrightarrow{f} \overline{s} \\ \text{or } (\overline{TS} \not\vdash D(s) \text{ and } \overline{TS} \not\vdash D(s')) \end{array}$$
),
- (2) $Z_{\overline{TS}} \models D(s)$ iff $\overline{TS} \vdash D(s)$,
- (3) $Z_{\overline{TS}} \models s \xrightarrow{f} s'$ iff $\overline{TS} \vdash s \xrightarrow{f} s'$,
- (4) $Z_{\overline{TS}} \models \neg D(s \xrightarrow{f} s')$ iff $\overline{TS} \not\vdash s \xrightarrow{f} s'$.

Proof idea. From the form of the axioms and our assumption it is easy to see that $\overline{TS}$ is hierarchy consistent and partially complete (cf. e.g. [BW2]). Then Proposition 1 and corollary 1 of [BW2] ensure the existence of initial algebras with properties (1)-(3).

Proposition 2 and corollary 4 of [BW2] ensure the existence of weakly terminal algebras with properties (1)-(3). Property (4) follows from the form of the axioms

E_T which implies that $(s \xrightarrow{f} s') = \text{false}$ can never be derived. $\square$

Hence the provable equality between states depends only on the axioms of states and not on those of the transition system. The weakly terminal models represent an operational equality, since two states s and s' are equal in $Z_{\overline{TS}}$ iff all transitions involving s are also possible for s' and viceversa.

3.2. Tree semantics. We generalize now the "single step" transition relation $\Rightarrow$ to its tree closure that is we consider the unordered trees of possible finite derivations from a given state.

First we define unordered finite trees with arcs labelled by flags and leaves labelled by states. The specification TRFLST is based on FLAG and STATE and defines finite unordered trees using a specification $\text{MSET}(\langle\langle \text{trflst} \rangle\rangle)$ (cf. appendix) of multisets over some (auxiliary) sort trflst . The terms of sorts $\text{mset}(\text{trflst})$ and trflst are built by mutual recursive use of the operations on these sorts.

spec $\text{TRFLST} =$

reachable

enrich $\text{FLAG} + \text{STATE} + \text{MSET}(\langle\langle \text{trflst} \rangle\rangle)$

by opns $\text{tr}: \text{state} \rightarrow \text{trflst}$

$\text{.o.}: \text{flag } x \text{ mset}(\text{trflst}) \rightarrow \text{trflst}$

$\text{sel1}: \text{trflst} \rightarrow \text{flag}, \text{sel2}: \text{trflst} \rightarrow \text{mset}(\text{trflst})$

axioms $\{\text{sel1}(f \circ \text{mst}) = f, \text{sel2}(f \circ \text{mst}) = \text{mst}, D(\text{tr}(s))\}$

The ASL-construct reachable T restricts the models of T to the term generated ones.

The trees of flags and states are represented by terms of sort $\text{mset}(\text{trflst})$, precisely: trees of depth 0 by terms of form $\{\text{tr}(s)\}$, while trees of depth greater than 0

are represented by terms of form $\{f_1 \circ mst_1\} | \dots | \{f_n \circ mst_n\}$ where for $i=1, \dots, n$

In the following we will write t instead of $\{t\}$.

By considering STATE and FLAG as parameters and NAT as primitive one easily proves that TRFLST is persistent wrt these specifications. Moreover, if they are monomorphic, then TRFLST is monomorphic.

By the specification TREE-SEM we give the tree semantics of a transition system $\overline{\text{TS}}$ by introducing a multistep relation $\sqsupset$ which associates to every state a family of (unordered) trees in TREEFLST.

$$\begin{aligned} \text{spec } \overline{\text{TREE-SEM}} = & \\ & \text{enrich } \overline{\text{TS}} + \text{TREEFLST} \text{ by} \\ & \text{ops } . \sqsupset . : \text{state} \times \text{mset}(\text{trflst}) \longrightarrow \text{bool} \\ & \text{axioms } \{s \sqsupset \text{tr}(s)\} \cup \\ & \quad \left\{ \bigwedge_{i=1}^n (s \xrightarrow{f_i} s_i \wedge s_i \sqsupset mst_i) \supset \right. \\ & \quad \left. s \sqsupset f_1 \circ mst_1 | \dots | f_n \circ mst_n \mid n \geq 1 \right\}. \end{aligned}$$

Since TREEFLST is a persistent extension of $\overline{\text{TS}}$ and the new axioms of TREE-SEM define just $\sqsupset$ in a partially complete way, TREE-SEM is a persistent extension of $\overline{\text{TS}}$ and we get the following properties (by a proof similar to the one of prop. 1, (2)-(3)).

Proposition 2.

There exists an initial algebra $I_{\overline{\text{TREE-SEM}}}$ and a weakly terminal algebra $Z_{\overline{\text{TREE-SEM}}}$

which both satisfy the following properties:

for $A \in \{I_{\overline{\text{TREE-SEM}}}, Z_{\overline{\text{TREE-SEM}}}\}$ and for all ground terms s, s' of sort state and mst of sort $\text{mset}(\text{trflst})$:

- (1) $A \models D(s)$ iff $\overline{\text{TS}} \vdash D(s)$,
- (2) $A \models s \xrightarrow{f} s'$ iff $\overline{\text{TS}} \vdash s \xrightarrow{f} s'$,
- (3) $A \models s \sqsupset mst$ iff $\overline{\text{TREE-SEM}} \vdash s \sqsupset mst$.

Therefore the initial and weakly terminal algebras of TREE-SEM are just extensions of those of $\overline{\text{TS}}$: TREE-SEM is not more abstract than $\overline{\text{TS}}$.

3.3 Observational semantics. For the (mathematical) observational semantics of concurrent transition systems several abstractions are possible: one may be interested in observing the input-output relation of a program or the stream of communications the program is capable to produce. In the following we formalize input-output semantics and strong equivalence semantics (in the sense of Milner [M1],[M2]) for transition systems in the framework of ASL, as a particular case of a general observational semantics of transition systems, by means of the observe ASL-construct. This operator enlarges the class of the models of a specification T to all those algebras which behave like a model of T on some fixed set of observable terms; more formally, for any fixed set W of terms and for any specification T , the semantics of "observe T by W " is the set of all (classes of isomorphic) algebras A on the signature of T , s.t. there exists an algebra A_0 , which is a model of T and is W -equivalent to A , i.e. such that there exist surjective assignments $v: X \rightarrow A$ and $v_0: X \rightarrow A_0$.

such that for all $t, t' \in W$

$$A, v \models t=t' \Leftrightarrow A_o, v_o \models t=t' \quad \text{and} \quad A, v \models D(t) \Leftrightarrow A_o, v_o \models D(t) .$$

To get the specification $\overline{\text{OBS-SEM}}$ of the observational semantics of a transition system $\overline{\text{TS}}$ we enrich first the specification $\overline{\text{TREESEM}}$ (of derivation trees of $\overline{\text{TS}}$) and a specification OBS of the observable part by a partial function obsmap (defining the observable part of a tree by means of a set of formulas E_{SEM}) and by a transition

relation $\overline{\text{OBS}}$ (defining the observable transitions). Then we forget all sorts and operation symbols not involving $\overline{\text{OBS}}$ by means of the derive-construct (in particular we forget flags and the transition relations $\Rightarrow$ and $\Rightarrow$ which is expressed by an injection $\text{inSig}(\dots)$ of the observable signature into the signature of $\overline{\text{TREE-SEM}} + \text{OBS}$) and abstract from these algebras by means of observe

spec $\text{OBS-SEM} \equiv \text{observe derive}$

$$\begin{array}{l} \text{enrich } \overline{\text{TREE SEM}} + \text{OBS by} \\ \quad \text{opns } \text{obsmap} : \text{mset}(\text{trflst}) \longrightarrow \text{obs} \\ \quad \quad \quad \cdot \overline{\text{OBS}} : \text{state} \times \text{obs} \longrightarrow \text{bool} \\ \text{axioms } E_{\text{SEM}} \cup \\ \quad \{ (D(\text{obsmap}(\text{mst})) \wedge s \sqsubseteq \text{mst}) \supset s \overline{\text{OBS}} \text{obsmap}(\text{mst}) \} \end{array}$$

by $\text{inSig}(\text{Sig}(\text{STATE}) \cup \text{Sig}(\text{OBS}) \cup \{ \overline{\text{OBS}} \})$ by $\{ s \overline{\text{OBS}} x \mid s \in W_{\text{Sig}(\text{STATE})} \mid \text{state} \wedge x \in X_{\text{obs}} \}$

In the following we assume that the enrichment of $\overline{\text{TREE-SEM}} + \text{OBS}$ is hierarchy-persistent. In the minimally defined models of OBS-SEM a state is defined only if it is already defined in STATE and two states are equivalent if they cannot be distinguished using the relation $\overline{\text{OBS}}$.

Proposition 3.

Let OBS be a partially complete specification which is monomorphic w.r.t. fixed isomorphism classes of primitive models and assume that the enrichment of $\overline{\text{TREE-SEM}} + \text{OBS}$ is hierarchy-persistent. Then $\overline{\text{OBS-SEM}}$ has an initial algebra $I_{\overline{\text{OBS-SEM}}}$ and a weakly terminal algebra $Z_{\overline{\text{OBS-SEM}}}$ with the following properties.

Let $A \in \{ I_{\overline{\text{OBS-SEM}}}, Z_{\overline{\text{OBS-SEM}}} \}$, s and o be ground terms respectively of sort state and obs :

- (1) $A \models D(s)$ iff $\text{STATE} \models D(s)$,
- (2) $A \models s \overline{\text{OBS}} o$ iff $\overline{\text{OBS-SEM}} \models s \overline{\text{OBS}} o$.

Moreover the equality in $I_{\overline{\text{OBS-SEM}}}$ and $Z_{\overline{\text{OBS-SEM}}}$ is defined for states s and s' with

$\overline{\text{OBS-SEM}} \models D(s) \wedge D(s')$ as follows:

$I_{\overline{\text{OBS-SEM}}} \models s=s'$ iff s and s' are syntactically the same term,

$Z_{\overline{\text{OBS-SEM}}} \models s=s'$ iff for all ground terms o of sort obs and all contexts $c[x] \in W_{\text{Sig}(\overline{\text{OBS-SEM}})}(\{\text{state } x\}) \mid \text{state} :$

$$\overline{\text{OBS-SEM}} \models c[s] \overline{\text{OBS}} o \text{ iff } \overline{\text{OBS-SEM}} \models c[s'] \overline{\text{OBS}} o .$$

Proof idea. $\overline{\text{OBS-SEM}}$ can be considered as an equational specification over BOOL where the only axioms are of the form $s \overline{\text{OBS}} o$ such that $s \overline{\text{OBS}} o$ is an axiom iff $s \overline{\text{OBS}} o$ holds in all models of the enrichment of $\overline{\text{TREE-SEM}} + \text{OBS}$. Then everything follows from the theorems in [BW2].

□

By considering $\overline{\text{TS}}$, OBS and E_{SEM} as parameter we get the following schema:

$$\text{specfunc } \overline{\text{OBS-SEM}} \equiv \lambda \text{ spec } \overline{\text{TS}}, \text{OBS, formulas } E_{\text{SEM}} . \overline{\text{OBS-SEM}}$$

Different semantics of transition systems can be obtained by instantiating the parameterized specification OBS-SEM with different sets of formulas E_{SEM} .

The specifications IO-SEM and SE-SEM define respectively the input-output and strong equivalence ([M1],[M2]) semantics.

Input-output semantics. For the IO-semantics we need the notion of final (or normal) state and of observable result of a final state which will be represented by the operations

$$\begin{array}{ll} \text{normal} : \text{state} & \longrightarrow \text{bool} \\ \text{res} : \text{state} & \longrightarrow \text{result} \end{array}$$

where result is some observable sort.

specfunc IO-SEM $\equiv \lambda \text{spec } T : \text{TS}(\ll \text{state} \gg, \ll \text{flag} \gg, \emptyset),$

spec RESULT : C .

OBS-SEM($T, \text{SET}(\text{RESULT}), E_{IO}$)

where C is the requirement specification

sorts state, result, bool

opns normal : state $\longrightarrow$ bool

res : state $\longrightarrow$ result

axioms normal(s) $\supset D(\text{res}(s))$

and where

$E_{IO} = \{ \text{normal}(s) = \text{true} \supset \text{obsmap}(\text{tr}(s)) = \{ \text{res}(s) \},$
 $\text{normal}(s) = \text{false} \supset \text{obsmap}(\text{tr}(s)) = \emptyset,$

$s \xrightarrow{f} s' \supset \text{normal}(s) = \text{false} \} \cup$

$\{ \text{obsmap}(f_1 \circ \text{mst}_1 \mid \dots \mid f_n \circ \text{mst}_n) = \text{obsmap}(\text{mst}_1) \cup \dots \cup \text{obsmap}(\text{mst}_n) \mid n \geq 1 \}$

Then in any weakly terminal model Z_{IO} of an instantiation $\overline{\text{IO-SEM}}$, two defined states s and s' are equal iff they lead to the same result in any context c :

$Z_{IO} \models s = s'$ iff for all ground terms r of sort result

for all contexts $c [x] \in W_{\text{Sig}(\overline{\text{IO-SEM}})}(\{ \text{state } x \}) \mid_{\text{state}} :$

$\overline{\text{IO-SEM}} \models c [s] \text{ obs } r$ iff $\overline{\text{IO-SEM}} \models c [s'] \text{ obs } r$.

Termination is ignored in this specification. It could be added e.g. by introducing a function stop : state $\longrightarrow$ bool.

Example CL. To define an IO semantics for the language CL we consider the transition system CLMTS (defined in section 2.3). First to every program of CL a state of CLMTS is associated by a "representation" function R defined by:

$R(\langle \text{program shared var } sv_1, \dots, sv_n \text{ proc}_1 \parallel \dots \parallel \text{proc}_m \text{ end program} \rangle =$

$\langle \langle \text{PR}(\text{proc}_1) \mid \dots \mid \text{PR}(\text{proc}_m), \langle \text{empty}, [v_1 / sv_1] \dots [v_1 / sv_n], \emptyset \rangle, \text{null} \rangle$

$\text{PR}(\langle \text{process } p \text{ var } v_1, \dots, v_k \text{ stat end process} \rangle =$

$\langle p, \text{stat}, \text{empty } [v_1 / v_1] \dots [v_1 / v_k] \rangle.$

Then we get an IO semantics for CLMTS by instantiating the parameterized specification IO-SEM with CLMTS and a specification CLRES of admissible results. These are chosen to be the final states of the shared memory: CLRES is SMEM + CLMTS enriched by operations "normal" and "res" defined by normal($\langle \langle \text{mps}, i \rangle, \text{null} \rangle$) = (mps = $\emptyset$) and res($\langle \langle \text{mps}, \langle \text{sm}, \text{tr} \rangle \rangle, \text{null} \rangle$) = sm.

Then we define two programs prog_1 and prog_2 of CL to be equivalent iff $R(\text{prog}_1)$ and

$R(\text{prog}_2)$ coincide in the weakly terminal models of IO-SEM(CLMTS, CLRES). That means

that prog_1 and prog_2 are equivalent iff for every context $c [x]$ of sort state

$c [R(\text{prog}_1)]$ and $c [R(\text{prog}_2)]$ yield the same results. But the form of $R(\text{prog}_1)$ implies

that proper contexts $c [x]$ of $R(\text{prog}_1)$ do not exist. Hence prog_1 and prog_2 are equivalent

iff $R(\text{prog}_1)$ and $R(\text{prog}_2)$ yield the same set of results.

Strong equivalence semantics. For the specification of strong equivalence semantics we need the specification TREEFL of trees of flags which are the same as trees of flags and states in TREEFLST (where states are the leaves of unordered trees) except that now flags are the labels of leaves and ares.

Given a tree of flags and states t in TREE-SEM, t^- indicates the associated tree of flags (i.e. without its leaves). Then we observe t^- from a state s

$$s \text{ [obs]} t^- \text{ iff } s \sqsupset t.$$

$$\text{spec SE-SEM} \equiv \text{OBS-SEM}(\overline{\text{TS}}, \text{TREEFL}, \mathbb{F}_{\text{SE}})$$

where

$$\begin{aligned} E_{\text{SE}} = \{ & \text{obsmap}(f_1 \circ \text{tr}(s_1)) \mid \dots \mid f_m \circ \text{tr}(s_m) \mid f_{m+1} \circ \text{mst}_{m+1} \mid \dots \mid f_{m+n} \circ \text{mst}_{m+n} \} = \\ & \text{tr}(f_1) \mid \dots \mid \text{tr}(f_n) \mid f_{m+1} \circ \text{obsmap}(\text{mst}_{m+1}) \mid \dots \mid f_{n+m} \circ \text{obsmap}(\text{mst}_{m+n}) \mid n+m \geq 1 \}. \end{aligned}$$

As expected in the weakly terminal models Z_{SE} of the specification SE-SEM, defined

states s and s' are equal iff in all contexts they accept the same communication trees.

$$\begin{aligned} Z_{\text{SE}} \models s=s' \text{ iff } & \text{for all ground terms mtf of sort mset('trfl'),} \\ & \text{for all contexts } c[x] \in W_{\text{Sig}(\text{SE-SEM})}(\{\text{state } x\}) \mid \text{state:} \\ & \text{SE-SEM} \vdash c[s] \text{ [obs]} \text{ mtf iff SE-SEM} \vdash c[s'] \text{ [obs]} \text{ mtf.} \end{aligned}$$

Concluding remarks. In this paper we have mainly intended to provide a precise formal support for a methodology of specification of concurrent systems, supporting abstraction and modularity. Some work has already been done for giving properties of richer subtheories, for example relating different execution modes (interleaving vs. parallel); but much work has still to be done in that direction.

An interesting topic could be to expose the exact relationship between our approach, the SOS methodology [P], Mosses' ASA's [Mo] and the algebraic approach to concurrency in [BW1]. Here we give only some hints.

In the SOS methodology transition systems are specified using derivation rules which correspond to axioms of the form

$$(o) \quad \bigwedge_i s_i \xrightarrow{f_i} s'_i = \text{true} \supset s \xrightarrow{f} s' = \text{true}.$$

One or more such derivations are associated to every construct of a programming language yielding a term model which is an initial model for the axioms (o) in our sense. The main methodological difference to our approach seems to be that Plotkin's approach is more oriented to the syntax of programming languages and that no semantic equations between concurrent constructs are postulated.

In that way our approach resembles more to Mosses' Abstract Semantic Algebras [Mo] where equational algebraic specifications are associated to many concepts of programming languages. Methodologically Mosses does not specify transition systems but gives equations which hold in denotational semantics; as technical consequence the underlying semantics of specifications are initial models of continuous algebras.

From the technical point of view our approach is strongly related to the algebraic specification of CSP in [BW1]. We have applied the techniques of [BW1] (such as transition systems, partial abstract types, the analysis of their models and the consideration of different abstractions from operational semantics) to the SMO LCS methodology of operational specification of concurrent systems. This was made possible by the use of the specification language ASL in which our general specification schemata of concurrent transition systems as well as the transition from operational semantics to more abstract mathematical semantics could be formulated.

REFERENCES

- [AMMR] E.Astesiano, F.Mazzanti, U.Montanari, G.Reggio: Design and formal specification of a family of distributed architectures. Quaderni CNET n. 88, ETS Pisa, 1983.
- [AMRZ] E.Astesiano, F.Mazzanti, G.Reggio, E.Zucca: Formal specification of a concurrent architecture in a real project. To appear in Proc. of ICS'85 ACM (International Computing Symposium), 1985.
- [AR] E.Astesiano, G.Reggio: A unifying viewpoint for the constructive specification of cooperation, concurrency and distribution. Quaderni CNET n.115, ETS PISA, 1983.
- [BaWo] F.L.Bauer, H.Wossner: Algorithmic language and program development, Springer Verlag Berlin, 1982.
- [BW1] M.Broy, M.Wirsing: On the algebraic specification of finitary infinite communicating sequential processes. In Proc. IFIP TC2 Working Conference on "Formal Description of Programming Concepts II", Garmisch 1982.
- [BW2] M.Broy, M.Wirsing: Partial abstract types. Acta Informatica 18, 47-64, 1982.
- [BW3] M.Broy, M.Wirsing: Generalized heterogeneous algebras and partial interpretations. In Proc. CAAP 83, Lecture Notes in Computer Science n. 159, Springer Verlag Berlin, 1983.
- [M1] R.Milner: A calculus of communicating systems. Lecture Notes in Computer Science n.92, Springer Verlag Berlin, 1980.
- [M2] R.Milner: Calculi for synchrony and asynchrony. TCS 25, 267-310, 1983.
- [Mo] P.Mosses: A basic abstract semantic algebra. In Proc. Semantics of Types, Lecture Notes in Computer Science, Springer Verlag, Berlin, 1984.
- [P] G.Plotkin: A structural approach to operational semantics. Lecture notes, Aarhus University, 1981.
- [P1] G.Plotkin: An operational semantics for CSP. In Proc. IFIP TC2 Working Conference on "Formal Description of Programming Concepts II", Garmisch 1982.
- [SW] D.T.Sannella, M.Wirsing: A kernel language for algebraic specification and implementation. In Proc. Int. Conf. on Foundations of Computation Theory, Borgholm Sweden, Lecture Notes in Computer Science n.158, 1983.
- [W] M.Wirsing: Structured algebraic specifications: a kernel language. Habilitation thesis, Technische University Munchen, 1983.
- [WPPDB] M.Wirsing, P.Pepper, H.Partsch, W.Dosch, M.Broy: On the hierarchies of abstract types. Acta Informatica 20, 1-33, 1983.

Appendix.

specfunct PRODCOND $\equiv \lambda \text{spec } A, B, \text{formulas COND} .$
 reachable enrich A+B by sorts prod
 opns <.,.> : a x b $\rightarrow$ prod
 axioms COND

specfunct MAP $\equiv \lambda \text{spec } \text{IND}, \text{VAL} .$
 reachable enrich IND+VAL by
 sort map opns empty: $\rightarrow$ map axioms{i#j $\supset$ m[v/j] (i) = m(i),
 .(.) : map x ind $\rightarrow$ val m[v/i] (i) = v,
 .[./.] : map x val x ind $\rightarrow$ map $\neg$ D(empty(i)) }

specfunct MSET $\equiv \lambda \text{spec ELEM.}$

enrich ELEM+NAT by

sort mset(elem)

opns {.: elem $\rightarrow$ mset(elem)

.--, .|. : mset(elem) $\times$ mset(elem) $\rightarrow$ mset(elem)

card: mset(elem) $\times$ elem $\rightarrow$ nat, $\emptyset$: $\rightarrow$ mset(elem)

axioms { ms|(ms'|ms'') = (ms|ms') |ms'', ms|ms' = ms'|ms,

card({e},e) = 1,

$e \neq e' \supset \text{card}(\{e\},e') = 0$,

card(ms|ms',e) = card(ms,e) + card(ms',e),

ms-(ms'|ms'') = (ms-ms') - ms'',

(ms|{e}) - {e} = ms,

$e \neq e' \supset (\text{ms}| \{e\}) - \{e'\} = (\text{ms}-\{e'\}) | \{e\}$,

$\emptyset | \text{ms} = \text{ms}$, $\text{ms} - \emptyset = \text{ms}$, $\emptyset - \text{ms} = \emptyset$, $\text{card}(\emptyset,e) = 0$ }