Logic of Trace Languages (Extended Abstract)

Alexander Rabinovich
IBM Research Division
T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Abstract

Usually, laws established in process calculi have the format of equations and/or inequations between process-terms. Though such set of laws captures important properties of the underlying algebra it cannot reveal some basic *logical* properties of the algebra. From the logical point of view, the consequence relation associated with an algebra is much more fundamental than the set of laws valid in it. That is why in this paper our main concern is about consequence relation which provides the answer to questions in the following format (the formalization is in terms of sequents): what terms are equal under the assumption that some other pairs of terms are equal. We compare two algebras: algebra of linear trace languages and algebra of relations. The fundamental operations in trace algebra are *synchronization* (parallel composition) of two trace languages, *nondeterministic choice* and *hiding* of a port in a language. The corresponding operations in relational algebra are *join, union* and *projection.* We show that these algebras have the same laws, i.e. two terms have the same meaning in all trace interpretations iff they have the same meaning in all relational interpretations. Moreover, we show that these algebras have the same consequence relations. We embed both algebras into first order logic and through this embedding obtain sound and complete proof systems for reasoning about the consequence relations in these algebras.

1. Introduction

1.1. Goals and Contribution

Synchronization, nondeterministic choice and hiding are fundamental operations in concurrency. The main goal of this paper is to investigate the algebra of these operations in a framework which is more general than that which is traditionally used in process algebras. We will deal here with the simplest model of concurrent behavior which does not take into account neither branching nor causality. The behavior of a system is represented by a linear trace language. For this simplest model one can observe that synchronization (notations $||$), nondeterministic choice (notations $+$) and hiding obey laws similar to well known logical laws for conjunction, disjunction and existential quantifier. For example:

- Synchronization is idempotent, commutative and associative, i.e. $L||L = L,\ L_1||L_2 = L_2||L_1,\ (L_1||L_2)||L_3 = L_1||(L_2||L_3)$.
- Distributivity of synchronization over nondeterministic choice: $L_1||(L_2 + L_3) = (L_1||L_2) + (L_1||L_3)$.

Besides of being interesting observations such facts are also useful for applications. For example, the proof of the generalized Kahn principle in [10] and proofs of modularity for nets of processes in [11] are based on such observations.

The set of laws valid in an algebra captures many algebraic properties. But this set is not enough to reveal some basic logical properties of an algebra. For example, in the algebra of processes the equality of process Pr_1 and of process *hide* x *in* Pr_2 implies that the processes Pr_1 and *hide* x *in* $Pr_1||Pr_2$ are also equal. But this cannot be derived from the set of laws which are valid in this algebra. From the logical point of view, the notion of consequence relation (see [1]) associated with an algebra is much more fundamental than the set of laws which are valid in it. Very roughly speaking, the consequence relation of an algebra captures the pattern of reasoning appropriate for the algebra. Knowledge of the consequence relations provides the answer to such questions: what terms are equal under the assumption that some other pairs of terms are equal.

Consequence relation is formalized through valid sequents. A sequent is an expression of the form $A_1, \cdots A_n \rightarrow B_1 \cdots B_k$, where $A_i,\ B_j$ are formulas. A sequent is valid in an algebra iff whenever its antecedents formulas $A_1, \cdots A_n$ hold at least one of its succedents B_j holds. The consequence relation associated with an algebra is just the set of all sequents valid in the algebra.

Terms are used as notations for processes; as formulas one usually takes equations between terms, and in cases when there is a partial order on the elements of the algebra it is more convenient to work with inequations between terms.

As mentioned above, we consider only a subset of the usual repertoire of operations; in particular, prefix operation and recursion are omitted. Nevertheless, even synchronization and hiding are enough for many applications, in particular, for net based models of concurrency. Actually, applications for net models provided the initial motivation for this research.

Usually the behavior of nets is described in an operational way but in all cases we know the behavior of a net can be obtained from the behavior of its components using two operations: *synchronization* (notations $||$) and *hiding* (a port).

When a semantical model and a specification language are chosen, the central question is to characterize when two systems are equivalent. Often in net theory, some components of the nets are underspecified, but one knows that two nets at certain nodes contain the same component. Therefore, from schematological perspective, the question of equivalence is modified as follows:

characterize when two nets are equivalent for arbitrary interpretations of their components.

This leads to the next question:

assume that we consider only interpretations under which net N_1 is equivalent to N_1', N_2 is equivalent to N_2', ... N_k is equivalent to N_k'. Can we conclude from this that N is equivalent to N' under these particular interpretations?

Recall that the behavior of a net is obtained from the behavior of its components through synchronization and hiding. Hence, the first question may be reduced to the question of equivalence of two terms constructed by these operations. The second question is reduced to the question of equality of two terms under the assumption that some other pairs of terms are equal, i.e. to the validity of an appropriate sequent.

We provide a complete and sound calculus for the consequence relation associated with linear trace algebra. It turns out that there is nothing incidental in the observation about similarity between synchronization, choice and hiding on the one hand and conjunction, disjunction and existential quantifier on the other hand. Roughly speaking our main theorem states that their corresponding consequence relations are the same.

Before proceeding to more detailed survey of the paper let us contrast it with other works in the field. Unlike others we investigate the simplest model of concurrency which ignores branching and causalities. On the other hand, despite the fact that in logic the consequence relation plays much more important and fundamental role than the set of valid sentences, to our knowledge, this is the first time that the consequence relation is investigated for a model of concurrency. More technical, but still important, are the following two points: (1) Though synchronization (parallel composition) is one of the most fundamental operations in concurrency, when it comes to axiomatization it is usually eliminated through expansion theorem [5, 7] and it is reduced to other operations such as prefixing, choice, left and right merge, etc. In such axiomatizations the properties of synchronization are disguised. (2) Unlike in this paper, completeness theorems which appear in the literature are very weak. Namely, they state that an (in)equation between *closed* terms is provable iff it is valid. Only in the works of Milner [8, 9] and Groote [4] versions of completeness theorem for open terms are considered.

1.2. Structure of the Paper

We compare two algebras: algebra of linear trace languages and algebra of relations. The fundamental operations in the trace algebra are synchronization (parallel composition) of two trace languages, nondeterministic choice and hiding of a port in a language. The corresponding operations in relational algebra are join, union and projection. We show that these algebras have the same laws, i.e., two terms have the same meaning in all trace interpretations iff they have the same meaning in all relational interpretations. Moreover, we show that these algebras have the same consequence relations.

Let us describe now the structure of the paper, some results and underlying ideas.

Section 2 describes two algebras: (1) algebra of relations with operations join, union and projection and (2) algebra of trace languages with operations synchronization, choice and hiding.

A linear trace language consists of the sets of all runs of a concurrent system. Each trace language L is equipped with the alphabet of actions which are available in L. Runs of L use actions from this alphabet but not necessarily all of them.

Section 3 begins with the syntactical definition of our language $\mathcal{L}$ which has three syntactical categories: typed terms, formulas and sequents. Two semantics for $\mathcal{L}$ are provided. The first one is relational semantics which assigns *port relations* to $\mathcal{L}$-terms. Ports of such relations are the same as attributes for database relations; the type of a term is the set of ports (attributes) of the corresponding relation in this semantics. The second semantics assigns trace languages to terms; and the type of a term provides information about the alphabet of the corresponding trace language.

Terms of $\mathcal{L}$ are constructed from typed variables by three constructors which are interpreted in relational semantics as join, union and projection; in trace semantics these constructors are interpreted as synchronization, choice and hiding.

As it usually happens in programming languages semantics, inequational reasoning is more simple than the equational one. This is one of the reasons that formulas of $\mathcal{L}$ are inequations between terms. A formula $t_1 \subseteq t_2$ is valid in one of the semantics above if the meaning of t_1 is a subset of the meaning of t_2. Hence, in a well formed formula $t_1 \subseteq t_2$ terms t_1 and t_2 have the same type.

Recall that a sequent is an expression of the form $A_1, \cdots A_n \rightarrow B_1 \cdots B_k$, and it holds in a semantics iff whenever all of its antecedent formulas $A_1, \cdots A_n$ hold at least one of its succedent formula B_j holds. The consequence relation associated with an algebra is just the set of all sequents which hold in the algebra.

Here is the main theorem of the paper

Theorem 1: *A single conclusion sequent $A_1, \cdots A_n \rightarrow B$ is valid in trace algebra iff it is valid in relational algebra.*

The fact that trace validity of a sequent implies its relational validity is more simple and we give a sketch of the proof in section 4.

Section 5 describes an embedding of $\mathcal{L}$ into a proper subset of first order logic. This embedding transforms relationally valid sequents into first order valid sequents. Moreover, sequents which are not valid relationally are transformed into sequents which are not valid in first order logic. This transformation gives us a way to construct a complete proof system for relational semantics from any complete proof system of first order logic.

In section 6 we describe the plan of the proof that relational validity of a sequent implies its trace validity; we point to difficulties which arise here. In this way together with section 4 we accomplish the proof of theorem 1.1. Section 7 contains concluding remarks and states some open problems.

2. Two Algebras

Let J be a non-empty set. Each element of J is called a port. Finite sets of ports are called types. a, b, c will be metavariables which range over ports; $\Sigma_0,\ \Sigma_1, \cdots$ are metavariables which range over types. We describe in this section two algebras: algebra of relations and algebra of trace languages. Objects in these algebras will have a type. In relational algebra ports have the role similar to the role of sorts in many-sorted logic. But unlike relations in many sorted logic, every relation can have at most one argument of a given sort. In trace algebra a type of relation provides an information not only which communications can happen, but also which communications cannot happen.

2.1. Algebra of Relations

Let D be a set. A **port relation** of type Σ over D is a subset of D^{Σ}. Given $x \in D^{\Sigma}$ and $x_1 \in D^{\Sigma_1}$. Assume that Σ_1 is a subset of Σ and for every port p in Σ_1 the equality $x(p) = x_1(p)$ holds. In this case we say that x_1 is the **projection** of x on Σ_1. For a port relation R of type Σ its projection on Σ_1 is the relation of type Σ_1 which consists of the projection of elements of R on Σ_1.

Let R_1 be a relation of type Σ_1 and R_2 be a relation of type Σ_2. The **join** of R_1 and R_2 is the relation of type $\Sigma_1 \cup \Sigma_2$ defined as follows: $x \in R_1 \& R_2$ if the projection of x on Σ_1 is in R_1 and the projection of x on Σ_2 is in R_2.

For a relation R of type Σ the relation $\exists a.R$ has the type $\Sigma - \{a\}$ and it is the projection of R on this type.

For relations $R_1,\ R_2$ of the same type Σ the relation $R_1 \cup R_2$ has type Σ and it is the **union** of R_1 and R_2.

Example 1: *$(a_1 \rightsquigarrow d_1, \ldots, a_n \rightsquigarrow d_n)$ is the notation for the element of $D^{\{a_1,\ldots a_n\}}$ which maps a_i to d_i for $i = 1, \ldots, n$. Consider port relations $R_1,\ R_2,\ R_3$ over set $\{0,\ 1\}$ of types $\{a_1,\ a_2\}$, $\{a_1,\ a_2\}$ and $\{a_1,\ a_3\}$ respectively which are defined as follows:*

$$R_1 = \{(a_1 \rightsquigarrow 0, a_2 \rightsquigarrow 1)\}$$
$$R_2 = \{(a_1 \rightsquigarrow 1, a_2 \rightsquigarrow 0),\ (a_1 \rightsquigarrow 1, a_2 \rightsquigarrow 1)\}$$
$$R_3 = \{(a_1 \rightsquigarrow 1, a_3 \rightsquigarrow 0),\ (a_1 \rightsquigarrow 0, a_3 \rightsquigarrow 1)\}$$

Then

$R_1 \& R_2$ is the empty relation of type $\{a_1,\ a_2\}$;

$$R_1\&R_3 = \{(a_1 \rightsquigarrow 0,\ a_2 \rightsquigarrow 1,\ a_3 \rightsquigarrow 1)\};$$
$$R_2\&R_3 = \{(a_1 \rightsquigarrow 1,\ a_2 \rightsquigarrow 0,\ a_3 \rightsquigarrow 0),\ (a_1 \rightsquigarrow 1,\ a_2 \rightsquigarrow 1,\ a_3 \rightsquigarrow 0)\};$$
$$\exists a_1.R_3 = \{(a_3 \rightsquigarrow 0),\ (a_3 \rightsquigarrow 1)\};$$
$$\exists a_2.R_3 = \{(a_1 \rightsquigarrow 1, a_3 \rightsquigarrow 0),\ (a_1 \rightsquigarrow 0, a_3 \rightsquigarrow 1)\};.$$
$$\exists a_2.R_2 = \{(a_1 \rightsquigarrow 1)\}.$$

Remark. In database terminology (see [6]) ports are called attributes; types are called relational schemes; join is denoted by $\bowtie$; projection of R on Σ is denoted by $\pi_\Sigma(R)$.

2.2. Algebra of Trace Languages

Let T^* denotes the set of all strings over alphabet T. As usual the concatenation of strings $s_1,\ s_2 \in T^*$ will be denoted by $s_1 s_2$; we say that s_1 is a prefix of s_2 if there is a string s_3 such that $s_1 s_3 = s_2$. A (linear) trace language over alphabet T is a pair $< S,\ T >$, where S is a subset of T^*. For a language L its alphabet is denoted by $alph(L)$ and its strings denoted by $trace(L)$.

Below we will consider trace languages over structured alphabets. Such an alphabet is the cartesian product of set Σ of ports and set D of values. Notation $< a,\ d >$ is used for elements of an alphabet $\Sigma \times D$. (Intuitevely an action $< a,\ d >$ corresponds to the transmission of the value d over the port a.) If a language L has an alphabet $\Sigma \times D$ then we say that L has type Σ. Assertion that L has type Σ will be written as $L :\ \Sigma$. We consider the following operations on languages:

Synchronization ($\|$).

$$alph(L_1\|L_2) = alph(L_1) \cup alph(L_2)$$
$$s \in trace(L_1\|L_2) \text{ iff for } i = 1, 2$$
$$s|alph(L_i) \in trace(L_i)$$

where $s|A$ is the notation for the string one gets from s by deleting all events which are not in A.

Hiding. *hide* p in L results in the language over $alph(L) - \{p \times D\}$; its strings are obtained from the strings of L through the deletion of all occurrences of actions on the port p.

Choice. For languages L_1 and L_2 of the same type Σ, language $L_1 + L_2$ has type Σ; the language $L_1 + L_2$ consists of all the strings which belong to at least one of the languages $L_1,\ L_2$.

Whenever trace languages will occur in the same context we will always assume that their alphabet is over the same set of values D. Under this assumption, it is clear that if $L_1 :\ \Sigma_1$ and $L_1 :\ \Sigma_2$ then $L_1\|L_2 :\ \Sigma_1 \cup \Sigma_2$ and *hide* a in $L :\ \Sigma - \{a\}$.

Example 2: *Consider languages* L_1, L_2, L_3 *of types* $\{a,\ b\}$, $\{a,\ c\}$, $\{b,\ c\}$ *respectively which contain the following traces:*

$$trace(L_1) = \{< a, 1 >< a, 0 >< b, 0 >,\ < a, 0 >< a, 1 >< b, 1 >\}$$
$$trace(L_2) = \{< a, 1 >< a, 0 >< c, 0 >\}$$
$$trace(L_3) = \{< c, 1 >< b, 1 >\}$$

Below are the traces of some languages resulting by applying synchronization and hiding to L_1, L_2 *and* L_3.

$L_1||L_2 = \{< a, 1 >< a, 0 >< b, 0 >< c, 0 >,\ < a, 1 >< a, 0 >< c, 0 >< b, 0 >\}$

$L_1||L_3 = \{< a, 0 >< a, 1 >< c, 1 >< b, 1 >,\ < a, 0 >< c, 1 >< a, 1 >< b, 1 >,$
$< c, 1 >< a, 0 >< a, 1 >< b, 1 >\}$

$L_2||L_3$ *is the empty language of type* $\{a,\ b,\ c\}$

$hide\ a\ in\ L_1 : \{b\}$ *and* $trace(hide\ a\ in\ L_1) = \{< b, 0 >,\ < b, 1 >\}$

$hide\ b\ in\ L_1 : \{a\}$ *and* $trace(hide\ b\ in\ L_1) = \{< a, 0 >< a, 1 >,\ < a, 1 >< a, 0 >\}$

$hide\ c\ in\ L_1 = L_1$

Remark: The role of alphabet. The type of language is very important in the definition of synchronization. For languages $L_1,\ L_2$ with same set of strings the languages $L_1||L_3$ and $L_2||L_3$ usually have not only different types but also different sets of strings as in the following

Example 3: *Consider languages* L_1, L_2 *and* L_3 *over unary set* D *of values.* $a,\ b,\ c$ *will be used below as ports and as elements of alphabet* $\Sigma \times D$. *Assume that* L_1 *and* L_2 *contain only one string* ab, *and have respectively types* $\{a,\ b,\ c\}$ *and* $\{a,\ b\}$. *Let* L_3 *be of type* $\{a,\ c\}$ *and contain only one string* ac. *Then* $L_1||L_3$ *and* $L_2||L_3$ *have the same type* $\{a,\ b,\ c\}$; $L_1||L_3$ *contains no strings, but* $L_2||L_3$ *contains two strings* abc *and* acb.

3. Syntax and Semantics of $\mathcal{L}$

In 3.1 we define language $\mathcal{L}$ which has three syntactical categories: typed terms, formulas and sequents. Two semantics for $\mathcal{L}$ are provided. The first one is relational semantics (section 3.2) which assigns port relations to $\mathcal{L}$-terms. The second semantics (section 3.3) assigns trace languages to terms.

Terms of $\mathcal{L}$ are constructed from typed variables by three constructors which are interpreted in relational semantics as join, union and projection; in trace semantics these constructors are interpreted as synchronization, choice and hiding.

3.1. Syntax

Definition 1: *The alphabet of $\mathcal{L}$ consists of the following sets of symbols:*

- *Ports.*
- *An infinite list of variables $X_1^\Sigma,\ X_2^\Sigma, \ldots$ for each type Σ.*
- *Constructors: $\wedge,\ \exists,\ \subseteq,\ \vee$.*
- *Auxiliary symbols: '(', ')', '$\rightarrow$' and ','.*

Below we define inductively terms and their types. Metavariables $t_0,\ t_1 \cdots$ range over terms; metavariables $\Sigma_0,\ \Sigma_1 \cdots$ range over types. An assertion that t has a type Σ will be written as $t:\ \Sigma$.

Definition 2: *Terms and their types.*

- *Variable X^Σ is a term of type Σ.*
- *If t_1 and t_2 are terms of types Σ_1 and Σ_2 respectively, then $(t_1 \wedge t_2)$ is a term of type $\Sigma_1 \cup \Sigma_2$.*
- *If t_1 and t_2 are terms of type Σ, then $(t_1 \vee t_2)$ is a term of type Σ.*
- *If t is a term of type Σ then $\exists a.t$ is a term of type $\Sigma - \{a\}$.*

A formula is an expression of the form $t_1 \subseteq t_2$, where $t_1,\ t_2$ are terms of the same type. $A,\ B,\ C \cdots$ will be metavariables ranging over formulas.

A sequent is an expression of the form $A_1,\ A_2, \cdots A_n \rightarrow B_1, \cdots B_m$, where $A_1,\ A_2, \cdots A_n, B_1, \cdots B_m$ are formulas. $A_1,\ A_2, \cdots A_n$ are the antecedents of the sequent and $B_1, \cdots B_m$ are the succedents of the sequent.

3.2. Relational Semantics of $\mathcal{L}$ terms

A relational environment η is a map from the variables to port relations which respects types, i.e. if X is of type Σ then $\eta(X)$ is a subset of D^Σ. For a term t of type Σ its relational semantics in an environment η (notation - $[|t|]_\eta^{rel}$) is a subset of D^Σ defined as follows:

Definition 3: *(Relational semantics for $\mathcal{L}$-terms)*

- *If t is a variable X then $[|t|]_\eta^{rel} = \eta(X)$*
- *$[|t_1 \wedge t_2|]_\eta^{rel}$ is $[|t_1|]_\eta^{rel} \& [|t_2|]_\eta^{rel}$.*
- *$[|\exists a.t|]_\eta^{rel}$ is $\exists a.[|t|]_\eta^{rel}$.*
- *$[|t_1 \vee t_2|]_\eta^{rel}$ is $[|t_1|]_\eta^{rel} \cup [|t_2|]_\eta^{rel}$.*

3.3. Trace Semantics of $\mathcal{L}$ terms

A trace environment η is a map from the variables to trace languages which respects types, i.e. if X is of type Σ then $\eta(X)$ is a trace language of type Σ. For a term t of type Σ its trace semantics in an environment η (notation - $[|t|]_\eta^{trace}$) is a trace language of type Σ defined as follows:

Definition 4: *(Trace semantics for $\mathcal{L}$-terms)*

- *If t is a variable X then $[|t|]_\eta^{trace} = \eta(X)$*
- $[|t_1 \wedge t_2|]_\eta^{trace}$ *is* $[|t_1|]_\eta^{trace} \,||\, [|t_2|]_\eta^{trace}$.
- $[|\exists a.t|]_\eta^{trace}$ *is hide a in* $[|t|]_\eta^{trace}$
- $[|t_1 \vee t_2|]_\eta^{trace}$ *is* $[|t_1|]_\eta^{trace} + [|t_2|]_\eta^{trace}$.

3.4. Semantics of Formulas and Sequents

Definition 5: *Let A be a formula $t_1 \subseteq t_2$.*

- *We say that a relational environment η satisfies the formula A if the relation $[|t_1|]_\eta^{rel}$is a subset of the relation $[|t_2|]_\eta^{rel}$. Similarly, we say that a trace environment η satisfies A if the trace language $[|t_1|]_\eta^{trace}$is a subset of the trace language $[|t_2|]_\eta^{trace}$.*
- *A formula A is relationally (trace) valid if every relational (trace) environment satisfies it.*
- *An environment satisfies a sequent if it satisfies one of the succedents of the sequent or if it does not satisfy one of the antecedents of the sequent.*
- *A sequent is relational (trace) valid if every relational (trace) environment satisfies it.*

From the above definition it follows that for both relational and trace semantics an environment satisfies a formula A iff it satisfies the sequent $\rightarrow A$.

4. Trace Validity Implies Relational Validity

Theorem 2: *If a sequent is trace valid then it is relationally valid.*

To prove that every trace valid sequent is relationally valid we define in the full paper an injective map ν from port relations to trace languages, which increases with respect to inclusion. We then show that ν is a homomorphism i.e.

1. $\nu(R_1 \& R_2) = \nu(R_1) || \nu(R_2)$

2. $\nu(\exists a.R) = hide\ a\ in\ \nu(R)$

3. $\nu(R_1 \cup R_2) = \nu(R_1) + \nu(R_2)$

The fact that ν is an injective and an increasing homomorphism implies

Claim 3: *Let η be a relational environment and η' be trace environment defined as follows: $\eta'(X) = \nu(\eta(X))$. Then*

1. $\nu([|t|]^{rel}_{\eta}) = [|t|]^{trace}_{\eta'}$
2. $x \in [|t|]^{rel}_{\eta}$ *iff* $\nu(x) \subseteq [|t|]^{trace}_{\eta'}$
3. *η satisfies formula A in relational semantics iff η' satisfies formula A in trace semantics.*
4. *η satisfies sequent S in relational semantics iff η' satisfies sequent S in trace semantics.*
5. *If a sequent S is valid in trace semantics then it is valid in relational semantics.*

Claim 4.2 (5) implies theorem 4.1.

5. Embedding of $\mathcal{L}$ into First Order Logic

Relational semantics of $\mathcal{L}$ is tightly connected with semantics of first order logic. Below we give an embedding (transformation) of terms and formulas of $\mathcal{L}$ into first order formulas. In this transformation, port names will play a role of individual variables and to each variable X_i of $\mathcal{L}$ there corresponds a predicate variable R_i in the first order logic. To a term of type $\Sigma = \{a_1, \cdots a_n\}$ there corresponds a formula with the free variables $\{a_1, \cdots a_n\}$. The embedding is straightforward. Here is

Definition 6: *(*Transformation*) Fix a bijection ψ between variables of $\mathcal{L}$ and predicate variable such that if $X : \Sigma$ then $\psi(X)$ has arity $|\Sigma|$. Assume that the set of $\mathcal{L}$-ports is the same as the set of first order individual variables and fix a linear ordering $\prec$ on this set.*

- *For each variable X_i of type $\Sigma = \{a_1, \prec a_2, \cdots \prec a_n\}$ there corresponds an atomic formula $R_i(a_1, \cdots, a_n)$, where $R_i = \psi(X_i)$*
- *A term is transformed into first order formula by replacing all variables by the corresponding atomic formulae.*
- *A formula $t_1 \subseteq t_2$ is transformed into first order formula by transforming t_i into first order formulae and replacing $\subseteq$ by implication $\Rightarrow$.*

Example 4: $\exists c.\ (X_1^{\{a,\ b\}} \wedge X_2^{\{a,\ c\}}) \subseteq (\exists c.\ X_2^{\{a,\ c\}}) \wedge X_1^{\{a,\ b\}}$ *is transformed into* $\exists c.\ (R_1(a,\ b) \wedge R_2(a,\ c)) \Rightarrow (\exists c.\ R_2(a,\ c)) \wedge R_1(a,\ b)$

Now given a relational environment η. Consider an interpretation $\overline{\eta}$ of first order predicate defined as follows: $\overline{\eta}(R_i)(d_1, \cdots, d_n) = true$ iff the function which maps a_j into d_j for $j = 1 \cdots n$ is in $\eta(X_i)$

Claim 4: *Let t be a term and η be a relational environment. Let $\overline{t}$ and $\overline{\eta}$ be the corresponding first order formula and first order interpretation. Assume that t is of type $\Sigma = \{a_1, \cdots a_n\}$ and ρ is any first order valuation which maps a_j to $d_j \in D$ for $j = 1, \cdots n$. Then ρ satisfies $\overline{t}$ in the interpretation $\overline{\eta}$ iff the element of D^{Σ} which maps a_j to $d_j \in D$ for $j = 1, \cdots n$ is in $[|t|]^{rel}_{\eta}$.*

From the claim it follows

Corollary 5: *(Faithfulness of the transformation)*

1. *A relational environment η satisfies an $\mathcal{L}$ formula t iff the transformation of t holds in the interpretation $\overline{\eta}$.*
2. *A relational environment η satisfies a sequent S iff the sequent obtained by universal closure of the formulas which correspond to formulas of S holds in the interpretation $\overline{\eta}$.*

Remark Complete proof systems for relational semantics. Claim 5.2.2 reduces relational validity of $\mathcal{L}$ to the validity of first order formulas. Therefore, from any complete proof system G for first order logic a procedure can be constructed which accepts relationally valid $\mathcal{L}$-sequents. Here is a description of such a procedure.

$Proc_G$(input: an $\mathcal{L}$ sequent $A_1, \ldots A_n \rightarrow B_1, \ldots B_m$).

1. Let first order formulas $\bar{A}_i$, $\bar{B}_j$ be obtained from $\mathcal{L}$ formulas A_i, B_j as in definition 6.
2. if formula $(\bar{A}_1^{\forall} \wedge \bar{A}_2^{\forall} \wedge \cdots \wedge \bar{A}_n^{\forall}) \Rightarrow (\bar{B}_1^{\forall} \vee \bar{B}_2^{\forall} \vee \cdots \vee B_m^{\forall})$ is provable in G then the sequent $A_1, \ldots A_n \rightarrow B_1, \ldots B_m$ is relationally valid.[1].

It is clear that if G is a complete proof system for first order logic then $Proc_G$ accepts exactly relationally valid sequents.

6. Relational Validity Implies Trace Validity

Let us summarize what we proved till now. We know that (1) trace validity implies relational validity; (2) we have complete proof systems for relational validity. Therefore, to show that trace validity coincides with relational validity it is enough to check that the inference rules of a complete proof system for relational semantics are sound for trace semantics. It turns out that the Gentzen sequent calculus [2] is most appropriate for these purposes.

[1] We use notations $C^{\forall}$ for the universal closure of the formula C

To show that a proof system is sound for trace semantics we have to verify that if the premises of the inference rules are trace valid, then their conclusions are trace valid. But here arises a new obstacle. Note that $\mathcal{L}$ was mapped to a proper subset of first order logic. As a result of this it might happen that the premises of an inference rule are images of $\mathcal{L}$-sequents but the conclusion is not the image of any $\mathcal{L}$-sequent. So trace semantics does not assign any meaning to this sequent. One can hope that a proof system has the following *property:*

(*) if the image $\bar{S}$ of an $\mathcal{L}$-sequent is provable then there exists a proof of $\bar{S}$ which contains only the images of $\mathcal{L}$-sequents.

This property would allow to consider only such instances of inference rules in which premises and conclusions are images of $\mathcal{L}$ sequents. The Gentzen sequent calculus has many nice characterizations which guarantee that it is a conservative extension of its subcalculus. This is one of the main reasons of its choice for our proofs. But nevertheless, the property (*) does not hold for the Gentzen sequent calculus.

To overcome these difficulties another language $\mathcal{L}'$ which contains $\mathcal{L}$ is introduced in the full paper. Trace semantics is provided for $\mathcal{L}'$ sequents. For single conclusion sequents the trace semantics of $\mathcal{L}'$ conservatively extends the trace semantics of $\mathcal{L}$. $\mathcal{L}'$ sequents are transformed into a proper subset of first order logic sequents. But now, the property (*) mentioned above holds for $\mathcal{L}'$-sequents and the proofs in the Gentzen calculus. The proof of this appeals to quite deep proof theoretical results and will appear in the full paper.

From Gentzen calculus one can extract a sequent calculus for $\mathcal{L}'$, and verify the soundness of its inference rules for $\mathcal{L}'$.

Since this calculus is complete for relational semantics it follows that every relational valid sequent is provable. Since this calculus is sound for $\mathcal{L}'$ trace semantics it follows that every provable $\mathcal{L}'$ sequent is trace valid. Therefore, every single conclusion $\mathcal{L}$ sequent which is relationally valid is trace valid. Together with theorem 4.1 this completes the proof of equivalence of relational and trace validity for single conclusion $\mathcal{L}$-sequents (theorem 1.1).

7. Concluding Remarks and Some Open Problems

7.1. Prefix Closed Languages (Linear Processes)

We modeled concurrent systems by trace languages. Each string in a language represents a 'complete' run. Sometimes all runs of a concurrent system are considered. In this case semantical objects will be trace languages which have the following prefix closeness property:

> If the trace language L contains a string s then L also contains all prefixes of s.

Below, prefix closed trace languages are called (linear) processes. Processes are closed under synchronization, choice and hiding. One of the natural questions is to characterize the algebra of processes. Since, processes are trace languages every trace valid sequent is process valid. Therefore, according to theorems 1.1 every relationally valid single conclusion sequent is process valid. Hence one can safely apply first order logic to find laws in the process algebra and to reason about process validity of (single conclusion) sequents. Is every process valid sequent relationally valid? The following example shows that this is not the case.

Example 5: *(A process valid sequent which is not relationally valid.)*

$\exists a.\ X^{\{a\}} \subseteq \exists a.\ Y^{\{a\}},\ \exists a.\ Y^{\{a\}} \subseteq \exists a.\ X^{\{a\}} \rightarrow \exists a.\ Y^{\{a\}} \subseteq \exists a.\ (X^{\{a\}} \wedge Y^{\{a\}})$

The sequent is valid for processes because its antecedents hold if either both X and Y are the empty languages of type a or they contain an empty string ϵ. In the first case the languages $\exists a.\ Y^{\{a\}}$ and $\exists a.\ (X^{\{a\}} \wedge Y^{\{a\}})$ are empty. In the second case they consist of the empty string.

The corresponding first order sequent is: $\exists a.\ R_1(a) \Rightarrow \exists a.\ R_2(a)$, $\exists a.\ R_2(a) \Rightarrow \exists a.\ R_1(a) \rightarrow \exists a.\ R_2(a) \Rightarrow \exists a.\ (R_1(a) \wedge R_2(a))$. It is clear that it is not valid.

We would like to know the answer to the following problems:

- Is it possible to add some sequents to the Gentzen calculus such that a single conclusion sequent is process valid iff it is provable?
- Does there exist process valid laws which are not relationally valid?
- Is it decidable whether a single conclusion sequent is process valid?

7.2. Decidability

It is clear that the validity of $\mathcal{L}$ formulas is decidable. We don't know whether validity of $\mathcal{L}$-sequents is decidable.

7.3. Conclusion

We hope that our paper will stimulate the study of consequence relations for more subtle concurrent models. These consequence relations are quite different from our simplest case (for example, almost in all other models synchronization is not idempotent). We believe that knowledge of consequence relations will highlight merits and demerits of concurrent models. Some of these models are based on the same intuition and only different (and not always justified) technical decisions lead to the diversity. The study of the corresponding consequence relations might justify these decisions and even suggest other more natural models.

The previous discussion highlights the way from concurrent models to their consequence relations and to logic. Another challenge is to go in the opposite direction and to provide 'concurrent' models for logics which claim to have a flavor of parallelism. Among such logics one of the most important is linear logic [3].

Acknowledgments

Many ideas of the paper have arisen in numerous discussions with Boris A. Trakhtenbrot. Thanks also to Arnon Avron, Joram Hirshfield and Albert R. Meyer for many stimulating discussions and helpful comments.

References

[1] A. Avron. Simple Consequence Relations. In *Information and Computation*, volume 92, 1991

[2] G. Gentzen. Investigations into logical deduction. In *The collected works of Gerhard Gentzen*, (M. Szabo, Ed.), North-Holland, Amsterdam, 1969.

[3] J.-Y. Girard. Linear Logic. In *Theoretical Computer Science*, 50(1):1-102, 1987.

[4] J. F. Groote. A new strategy for proving ω-completeness with applications in process algebra. In *Proceedings of CONCUR 90*, volume 458 of *Lect. Notes in Computer Science*. Springer Verlag, 1990.

[5] M. Hennessy. *Algebraic Theory of Processes.* MIT Press, 1988.

[6] P. .C. Kanellakis. *Elements of Relational Database Theory.* In *Handbook of Theoretical Computer Science*, North-Holland, 1989.

[7] R. Milner. *Communication and Concurrency.* Prentice-Hall International, Englewood Cliffs, 1989.

[8] R. Milner. A complete proof system for a class of regular behaviors. In *JCSS* , volume 28, pp 439-466, 1984.

[9] R. Milner. A complete axiomatisation for observational Congruence of finite state behaviors. In *Information and Computation*, volume 81, pp 227-247, 1989.

[10] A. Rabinovich and B. A. Trakhtenbrot. Communication among relations. In *International Conference on Automata, Languages and Programming*, volume 443 of *Lect. Notes in Computer Science*. Springer Verlag, 1990.

[11] A. Rabinovich and B. A. Trakhtenbrot. On Nets, Algebras and Modularity. In *International Conference on Theoretical Aspects of Computer Software*, to appear in *Lect. Notes in Computer Science*. Springer Verlag, 1991.