A Continuous Algebraic Semantics of CSP

Liu Zhiming (刘志明)

(*Institute of Software , Academia Sinica*)

Received August 11 , 1987 ; revised November 11 , 1987 .

Abstract

This is an attempt to use continuous algebras to describe the semantics of CSP — continuity being used to solve recursive definitions of processes as infinite objects . By so doing , we combine the algebraic specifications of abstract data types with CSP to make up a new language , which is recommended as a promising candidate of specification language for designing and developing communicating systems .

1 . Introduction

As is known , the CSP (Communicating Sequential Processes) language , developed in [6] is good for protocol specification of communicating systems . But it is not powerful enough in specifying the external behaviour of the systems . The kinds of behaviour are generally classified as fairness , liveness , and safety , etc . Temporal logic is thus introduced to fill this vacancy[13] , and a combined language of temporal logic and CSP is given in [12] . However , the data worlds over which the systems are established could neither be specified in the pure CSP , nor in the combined language .

Nowadays , the algebraic specifications of abstract data types are popular for describing data structures and the semantics of sequential programming languages[1,5] . In [7,10] , the combination of algebraic specifications of abstract data types and CCS [8] is investigated . But the resulting languages are just as expressive as CCS , except that they could handle data types . Also in [8] , it is claimed that it may be possible to define the fairness of distributed systems if we combine CCS with temporal logic . This problem has been solved perfectly in [12] with the combining of CSP and temporal logic . Our work presented hereafter has made the progress of strengthening CSP to tackle the problem of data types , and we have made one first step toward the specification of fairness as mentioned in [8] . And now we are working on another subject about defining temporal logic with continuous algebras given in this paper , after that we will get a combined language of algebraic specifications , temporal logic and CSP .

In Section 2 , we give the continuous and non-continuous specifications that define the continuous and non-continuous algebras as the semantics domains of the communicating systems . We assume the readers have the relevant knowledge of the concepts of algebras and continuous algebras , e . g . , in [1 , 2 , 5] . Meanwhile in this paper , we make some generalizations of the concepts of continuous algebras and continuous algebraic specifications appeared in [2 , 3] . Firstly , some functions may be non-continuous at one or more arguments , actually no continuity is involved in these arguments . So recursive definitions at these arguments cannot be made infinitely . Secondly , a continuous algebraic specification may have equations and inequalities as

axioms . The concepts concerning the continuity of homomorphisms between algebras are generalized in the similar way. We make this explanation precise with the following definitions.

Definition 1 . *An ordered signature* $SIG=(NS+CS, OP)$ *consists of* $NS+CS$, *the set of sorts , where NS is the set of non-continuous sorts and CS is the set of continuous sorts ; and a* $(NS+CS)^*(NS+CS)$- *indexed family of sets , and OP, the family of operation symbols .*

Definition 2 . *Given* $SIG=(NS+CS)$ *as a signature , a SIG-algebra* $A=(NS_A+CS_A, OP_A, \leqslant_A)$ *is given by three families* $S_A=NS_A+CS_A=(A_s)_{s\in NS+CS}$, $OP_A=(N_A)_{N\in OP}$, $\leqslant_A=(\leqslant_{S_A})_{s\in CS}$ *where*

1 . A_s *are sets for all* $s\in NS+CS$;

2 . $N_A: A_{s_1}\times\cdots\times A_{s_n}\to A_s$ *are functions for all* $N: s_1\times\cdots\times s_n\to s$ *in* $OP, n\geqslant 0$;

3 . $\leqslant_{S_A}\subseteq A_s\times A_s$ *are partial orderings for all* $s\in CS$;

4 . *If* $(a_1,\cdots,a_n)\in A_{s_1}\times\cdots\times A_{s_n}$, $(a_1',\cdots,a_n')\in A_{s_1}\times\cdots\times A_{s_n}$ *such that* $a_{ij}\leqslant_A a_{ij}'$ *for* $s_{ij}\in CS$, $a_{ij}=a_{ij}'$ *for* $s_{ij}\in NS$, *then*

$$N_A(a_1,\cdots,a_n)\leqslant_A N_A(a_1',\cdots,a_n')$$

for all $N_A: A_{s_1}\times\cdots\times A_{s_n}\to A_s$ *such that* $s\in CS$.

Definition 3 . *Let* $SIG=(NS+CS, OP)$ *be a signature ,* $X=(X_s)_{s\in NS+CS}$ *be a family of variables , a SIG-equation is a triple* (X, t, t') *with* $t, t'\in T_{OP,s}(X)$, *the terms of some sort s in NS , and we denote it as* $t=t'$; *a SIG- inequality is* $[X, t, t']$ *with* $t, t'\in T_{OP,s}(X)$, *the terms of some sort s in CS , and we denote it as* $t\leqslant t'$.

Remark : We also use equations of continuous sorts , i .e ., $t=t'$ for terms t and t' of some sort s in CS, this is equivalent to the two inequalities $t\leqslant t'$ and $t'\leqslant t$.

Definition 4 . *An ordered specification* $SPEC=(NS+CS, OP, E)$ *consists of an ordered signature* $SIG=(NS+CS, OP)$ *and a set of equations and inequalities E as axioms .*

After making these generalizations , all the results about continuous algebraic theory given in [2 , 3] still properly fit our purpose with or without slight modifications and we use the ω-complete algebras as the models of the specifications .

Section 3 defines the CSP notations in our specifications and in Section 4 the semantics of the algebraic specifications is defined . Also in Section 4 , the consistency of the algebraic semantics with the model for temporal semantics of CSP [12] is presented .

A distributed system in this paper is regarded as a network of processes connected by named channels . Just as in [13] , every channel is owned by two processes , one at each end . For each end of a channel ,and at any time , one and only one of the following events could take place , this end being closed , requesting (to communicate with the other end), rejecting all communications along this channel , and passing a message , i .e ., the communication over this channel succeeds .

2 . Specifications

The behaviour of a process is viewed as sets of histories of events taken place at

each end of its channels . Thus the specification of processes depends on how we specify the sequences of events . This implies that , when we specify the semantics of a process , we should take the specification of the sequences of events as a subspecification , or we can say a parameter . The specification of the sequences of events is defined in turn by employing the specification of events as a parameter . The type " event " might deal with how the message over channels is administered . Therefore , it is necessary for us to specify the message before others .

The message and events need not be defined as continuous algebras , i .e . no continuous sorts in their specifications . We allow parameters appear therein to be non-continuous , and some functions can be non-continuous at one or more arguments .

In the following , we write NSpec and CSpec to denote a non-continuous and a continuous specification respectively .

For simplicity , we allow the use of the concepts of subsort and disjoint union of sorts s_1 , s_2 , $\cdots$, s_n in later discussion . The interpretation of s_1 being a subsort of s (denoted as $s_1 \subset s$)is that the carrier set of s_1 is contained in that of s . The interpretation of the disjoint union , $s_1 + s_2$, is the disjoint union of the carrier sets of s_1 and s_2 . Actually , the concept of subsort is equivalent to the local definition in specifications proposed in [4] , and we can replace the disjoint union , $s_1 + s_2$, by one sort s and two functions $\text{make}_1 : s_1 \to s$, $\text{make}_2 : s_2 \to s$. We also allow the use of local definitions just as [4] .

2 .1. *The Specification of Message*

The algebraic approaches for specifying abstract data types have been well studied at present [1,5]. All of these approaches could be directly applied in the specification of message .

Because a message is composed of fundamental data , we should make it clear what the data are . The type 'Data 'has the primitive sorts of m_1 , m_2 , $\cdots$, m_n . These are the data types permitted in the channels . 'Data ' is defined as

NSpec .Data =

sorts : $m_1 , \cdots , m_n$;

functs : $f_1 , \cdots , f_i$;

axioms : $e_1 , \cdots , e_j$;

End-of-Spec .

We assume that the type 'BOOL' of truth values of { tt , ff } is a primitive specification upon which the NSpec . Data is established as a hierarchical specification . The concepts about hierarchical specification can be found in [11] .

It is reasonable that the data of the message of a system can be manipulated with a sequential programming language , i .e ., some sequential programs can generate the data as their results . Consequently , a channel may pass a result of a sequential program . We specify the sequential programming language as a hierarchical abstract data type with the NSpec . Data as its primitive specification . The type 'Messages ' is defined as

NSpec . Messages =

primitive : NSpec . Data ;

sorts : $s_1, \cdots, s_m$;

functs : $g_1, \cdots, g_k$

axioms : $e_1', \cdots, e_h'$;

End-of- Spec .

For the above two specification , we make some remarks :

i . For each sort of NSpec . Data m , we need a function eq : $m \times m \to$ bool among the functions , $f_1, \cdots, f_i$, and it is to be interpreted as the equality between the terms of sort m .

ii . For simplicity , we assume that all the functions in the specifications are total , and NSpec . Data defines finite data types , i . e .

$$(\text{NSpec . Data } /\equiv)_{m_i}, \quad \text{for } i=1, \cdots, n$$

are finite .

iii . We choose the initial models for the interpretation of NSpec . Data and NSpec . Messages .

2 . 2 . The Specification of Events

Now we define the events as a parameterized abstract data type with the model of NSpec . Messages and the channel names as its parameters . Each event represents an act of a process of some time .

NSpec . Events =

parameters : Messages , Ch ;

sorts : evt ;

functs : **null** : $\to$ evt ; (no event takes place)

all : $\to$ evt ; (any event can take place)

Icld : Ch $\to$ evt ; (the input of a channel closed)

Ocld : Ch $\to$ evt ; (the output of a channel closed)

Ireq : ch $\to$ evt ; (the input of a channel requesting communication)

Oreq : Ch $\to$ evt ; (the output of a channel requesting communication)

Irej : Ch $\to$ evt ; (the input of a channel rejecting communications)

Orej : Ch $\to$ evt ; (the output of a channel rejecting communications)

Pass : Ch $\times (m_1 + \cdots + m_n) \to$ evt ; (**Pass** (c , x) denotes that the value of x is passed on c)

and : evt $\times$ evt $\to$ evt ; (a **and** b means that a and b occur at the same time)

or : evt $\times$ evt $\to$ evt ; (a **or** b means that one of a and b takes place)

In the following , we should have the axioms to characterize the functions we have listed above . According to what we remarked at the end of Section 1 , the pairs of the events , **Icld** (c) and **Ireq** (c) , **Ocld** (c) and **Pass** (c , x) , **Pass** (c , x) and **Pass** (c , y) ($x \neq y$) may not take place at the same time . These are specified among the following axioms .

axioms : **Icld** (c) **and Ireq** (c) = **null** ;

Icld (c) **and Irej** (c) = **null** ;

Icld (c) **and Pass** (c , x) = **null** ;

Ireq (c) **and Irej** (c) = **null** ;
Ireq (c) **and Pass** (c ,x) = **null** ;
Irej (c) **and Pass** (c ,x) = **null** ;
Ocld (c) **and Oreq** (c) = **null** ;
Ocld (c) **and Orej** (c) = **null** ;
Ocld (c) **and Pass** (c ,x) = **null** ;
Oreq (c) **and Orej** (c) = **null** ;
Oreq (c) **and Pass** (c ,x) = **null** ;
Orej (c) **and Pass** (c ,x) = **unll** ;
Pass (c ,x) **and Pass** (c ,y) = if eq (x ,y) = tt then **Pass** (c ,x) else **null** ;
a **and null** = **null** ;
a **and** b = b **and** a ;
a **and** a = a ;
a **and all** = a ;
a **and** (b **and** c) = (a **and** b) **and** c ;
a **or** b = b **or** a ;
a **or all** = **all** ;
a **or null** = a ;
a **or** (b **or** c) = (a **or** b) **or** c ;
a **and** (b **or** c) = (a **and** b) **or** (a **and** c);
a **or** (b **and** c) = (a **or** b) **and** (a **or** c);
a **and** (a **or** b) = a

End-of- Spec .

We can see that these axioms define a class of lattices . From now on , we use **AND** to denote the grand operator for **and** when more than two but finite events take place at the same time , and **OR** to denote the grand operator for **or** when more than two but finite choices of events take place .

Since the data types of 'Data ' are finite , the event **all** can be generated as a term with the use of other operators . We use the operation **all** here for the sake of simplicity .

Now we proceed to the definition of the sequences of events in order to specify the histories of the processes .

2 .3. The Specification of the Sequences of Events

What we actually need are infinite sequences of events . So we employ continuous specification .

CSpec .Sequence-of -events =

parameter : Events ;
csorts : seq ;
functs : ε :→ seq ;
T :→ seq ;
app : evt × seq → seq ;
axioms : **app** (a , **T**) = **T** :
app (**null** , s) = **T** ;
$\varepsilon \leqslant s$

End-of -Spec .

In this specification , the constant **T** means that the unfair event , **null** . occurs in the sequence . Note that $s \leqslant \mathbf{T}$ for all s of sort seq .

For simplicity , we write a^s for **app** (a ,s). If a_1 , a_2 ,$\cdots$ a_n ,$\cdots$ is an infinite sequence of events such that no **null** occurs in it , we have

$$a_1\hat{}\varepsilon \leqslant a_1\hat{}\, a_2\hat{}\,\varepsilon \leqslant \cdots \leqslant a_1\hat{}\, a_2\hat{}\cdots\hat{}\, a_n\hat{}\,\varepsilon \leqslant \cdots .$$

The least upper bound of this increasing sequence , which we denote as

$$a_1 a_2 \cdots a_n \cdots \quad ,$$

is an infinite term of sort seq .

The next subsection will define well formed process expressions as special sets of sequences of events . From the intuitive view , a sequence of events is actually a set of sequences of events of the form a **and** b , i .e ., no occurrence of **or** in it . This will be made clearer in Section 4 .

2 .4. The Specification of Sets of Sequences

We define the sets of sequences of events with which we then define the semantics of a process expression in CSP. The semantics is defined as a set of sequences of events that take place during the execution of the process . The interpretation of the specification is required to be continuous .

CSpec .Set-of - sequences =
parameters : Events , Sequences- of - events ;
csorts : set ;
functs : **empty** : → set ;
all : → set ;
insert : seq × set → set ;
set - app : evt × set → set ;
⊔ : set × set → set ;
⊓ : set × set → set ;
axioms : **empty** ⩽ *s* (**empty** is the least element);
s ⩽ **All** (**All** is the greatest element);
insert (**T** , s) = s ;
insert (sq , **insert** (sq ′,s)) = **insert** (sq ′, **insert** (sq ,s));
insert (sq , **insert** (sq ,s)) = **insert** (sq ,s);
empty ⊔ s = s ;
All ⊓ s = s ; All ⊔ s = All ;
s ⊔ s ′ = s ′⊔ s ;
insert (sq ,s) ⊔ s ′ = **insert** (sq ,s ⊔ s ′);
set-app (a , **empty**) = **empty** ;
set-app (a , **insert** (sq ,s)) = **insert** (**app** (a , sp), **set- app** (a ,s)) ;
set- app (a **or** b ,s) = **set-app** (a ,s) ⊔ **set-app** (b ,s);
empty ⊓ s = **empty** ;
s ⊓ s = s ;
s ⊓ s ′ = s ′⊓ s ;
set-app (a ,s) ⊓ **insert** (ε , **empty**) = **empty** ;
set-app (a ,s) ⊓ **set-app** (b ,s ′) = **set -app** (a **and** b , s ⊓ s ′);
s ⊔ (s ′⊓ s ″) = (s ⊔ s ′) ⊓ (s ⊔ s ″);
s ⊓ (s ′⊔ s ″) = (s ⊓ s ′) ⊔ (s ⊓ s ″);
s ⊓ (s ⊔ s ′) = s ;
s ⩽ **insert** (sq ,s);
s ⩽ s ⊔ s ′;
s ⊓ s ′ ⩽ s

End- of - Spec .

Note that from $s \sqcap s = s$, $s \sqcup (s' \sqcap s'') = (s \sqcup s') \sqcap (s \sqcup s'')$ and $s = s \sqcap (s \sqcup s')$, we can obtain $s = s \sqcup (s \sqcap s')$, and one can show that $s \leqslant s'$ iff $s \sqcap s' = s$ iff $s \sqcup s' = s'$. We observe that the operators $\sqcup$ and $\sqcap$ are intuitively consistent with the operators for the least upper bound and the greatest lower bound. Thus the same symbols are used for the latter ones.

Before stepping to the next section, we would like to impose some constraints on the interpretations of the CSpec.Set-of-sequences. This is a parameterized specification with parameters Events and Sequence-of-events. Though the sort seq is continuous in CSpec.Sequence-of-events, all the operators with set as their codomain are non-continuous at the arguments of sort seq. The models of CSpec.Set-of-sequences are free extensions of the models of CSpec.Sequence-of-events. And because we added a greatest element **All** to the sort set, the carrier set of sort set is actually an ω-complete lattice (c.f.[2]) in each model. Therefore, an infinite decreasing sequence of elements of sort set has a greatest lower bound.

3. Defining the CSP Notations with the Specifications

Each process expression is defined recursively as an infinite object of sort set. So the process expressions in CSP are of sort set in CSpec.Set-of-sequences.

Let $X = F(X)$ be an algebraic equation such that F is continuous with respect to X. We use $|X = F(X)|$ to stand for the least solution of the equation, i.e., the least fixed point of F. It can be explicitly written as $\bigsqcup_{i=0} F^i(W)$, where W is the least element of the same sort as X.

In this paper, we are only interested in the syntactic notations of CSP presented in [14]. We define the CSP operations as those of CSpec.Set-of-sequences in the following way.

For $c \in Ch$, $t \in m_1 + m_2 + \cdots + m_n$, x is a variable of some sort m in Data, and $s \in$ set in Set-of-sequences, we have the definitions:

(1) $STOP_{\alpha,\beta} =_{df}$ **insert** ($|X =$ **app** (**Icld** (α) **and Ocld** (β), X)|, **empty**).

Obviously, it meets our intuitive view of STOP closing its channel ends forever. In the above formula,

$$\textbf{Icld}(\alpha) =_{df} \underset{c \in \alpha}{\textbf{AND}} (\textbf{Icld}(c)),\ \textbf{Ocld}(\beta) =_{df} \underset{c \in \beta}{\textbf{AND}} (\textbf{Ocld}(c)),$$

and α and β are finite sets of channel names.

(2) $c!t \rightarrow P_{\alpha,\beta} =_{df}$ **insert** ($|X =$ **app** (**only-Oreq** (c), X)|, **empty**)
$|X =$ **set-app** (**only-Oreg** (c), X) $\sqcup$ **set-app** (**only-Oreq** (c), **set-app** (**only-Pass** (c, t), $P_{\alpha,\beta}$))|.

The process $c!t \rightarrow P_{\alpha,\beta}$ always requests to communicate at the output end of c, or the communication succeeds and then the whole process behaves like the process $P_{\alpha,\beta}$.

(3) $c?x{:}m \rightarrow P_{\alpha,\beta}(x) =_{df}$ **insert** ($|X =$ **app** (**only-Ireq** (c), X)|, **empty**)
$|X =$ **set-app** (**only-Ireq** (c), X) $\sqcup$
set-app (**only-Ireq** (c), $\bigsqcup_{x \in m}$ (**set-app** (**only-pass** (c, x), $P_{\alpha,\beta}(x)$))|.

In (2) and (3), **only - Oreq** is defined as

only - Oreq (c) $=_{df}$ **Oreq** (c) **and** $_{d}$ **AND**$_{c}$ (**Orej** (d)) **and** **AND**$_{d}$ (**Irej** (d)),

only - Ireq and **only- Pass** can be defined similarly .

(4) $P_{\alpha,\beta} \Box Q_{\alpha,\beta} =_{df} P_{\alpha,\beta} \sqcup Q_{\alpha,\beta}$.

(5) $P_{\alpha,\beta} \| Q_{\alpha',\beta'} =_{df} P_{\alpha,\beta} \sqcap Q_{\alpha',\beta'}$.

(6) $P_{\alpha,\beta} | Q_{\alpha,\beta} =_{df} | X =$ **set- app (Irej** (α)**and Orej** (β), X$) \sqcup$ **set - app (Irej** (α) and **Orej** (β), $P_{\alpha,\beta} \sqcup Q_{\alpha,\beta}$)|.

(7) $P_{\alpha,\beta}[\,d/c] =_{df} P_{\alpha,\beta}[d/c]$ (the right hand- side is a set).

$P_{\alpha,\beta}[d/c]$ is the set resulted from substituting d for c in set $P_{\alpha,\beta}$, and finally, the behaviour of the recursively defined process is

(8) $p = F(p) =_{df} \prod_{i=0}^{\infty} F^{i}$(**All**).

The right hand side of (8) is actually the greatest lower bound of

$$\mathbf{All} \geqslant F(\mathbf{All}) \geqslant F^{2}(\mathbf{All}) \geqslant \cdots \geqslant F^{n}(\mathbf{All}) \geqslant \cdots ,$$

and a fixed point of $p = F(p)$.

4. The Semantics of the Algebraic Specifications

Up to now, we have not yet investigated the properties of the model of our specifications, hence we could not support a proving system of CSP safely by the semantics we have defined. This section is meant to deal with this aspect.

As our specification language is an amalgmation of the algebraic specifications and CSP, the verification of communicating systems can be done by the classical techniques developed in the framework of algebraic specifications.

The existence of the initial models of NSpec. Data, NSpec. Messages and NSpec. Events does not invoke serious problems.

Let D be an initial model of NSpec. Data, M be a hierarchical model of NSpec. Messages on D, and we write them as

$$D = (D_{m_1}, \cdots, D_{m_n}, f_{1D}, \cdots, f_{iD})$$

$$M = (D, M_{s_1}, \cdots, M_{s_m}, g_{1M}, \cdots, g_{kM}),$$

then we can construct a free extension of M with respect to NSpec. Events as the following

$$\text{Event} = (M, Ch, Evt, \mathbf{all}_E, \mathbf{null}_E, \mathbf{Icld}_E, \mathbf{Ocld}_E, \mathbf{Ireq}_E, \mathbf{Oreq}_E, \mathbf{Irej}_E, \mathbf{Orej}_E, \mathbf{Pass}_E, \mathbf{and}_E, \mathbf{or}_E),$$

where

$Evt = P([\,Ch \rightarrow M^2 \cup con^2\,])$ and

$con = \{\,req, cld, rej\,\}$,

$[\,Ch \rightarrow M^2 \cup con^2\,]$ is the set of all the functions from the set of channels, Ch, to $M^2 \cup con^2$, and Ch is finite,

$P(S)$ is the power set of a set, S.

We call each set in Evt an **event**.

$\mathbf{null}_E = \phi$, the empty set,
$\mathbf{all}_E = \mathrm{Evt}$,
$\mathbf{Icld}_E(c) = \{f: f(c) = (cld, y),\ y \in con\}$,
$\mathbf{Ocld}_E(c) = \{f: f(c) = (x, cld),\ x \in con\}$,
$\mathbf{Ireq}_E(c) = \{f: f(c) = (req, y),\ y \in con\}$,
$\mathbf{Oreq}_E(c) = \{f: f(c) = (x, req),\ x \in con\}$,
$\mathbf{Irej}_E(c) = \{f: f(c) = (rej, y),\ y \in con\}$,
$\mathbf{Orej}_E(c) = \{f: f(c) = (x, rej),\ x \in con\}$,
$\mathbf{Pass}_E(c, x) = \{f: f(c) = (x, x),\ x \in M\}$,
$E_1\ \mathbf{and}_E\ E_2 = E_1 \cap E_2$,
$E_1\ \mathbf{or}_E\ E_2 = E_1 \cup E_2$.

It is easy to see that Event is a free extension of M with respect to NSpec.Events.

The existence of initial semantics of Cspec.Sequences-of-events is solved by the following theorem.

Theorem 4.1. *For any model of the parameter specification, NSpec.Events, there is a free extension of it with respect to CSpec.Sequence-of-events.*

Proof. Let E be a model of NSpec.Events, then construct

$$\mathrm{Seq}(E) = (E, \mathrm{Seq}_{seq}, \varepsilon, T, \mathbf{app}_{seq}),$$

where, $\mathrm{Seq}_{seq} = \{s \mid s \text{ is a sequence of elments of } E, \text{may be infinite}\} \cup \{T\}$,

ε: the empty sequence,
T: an un-fair sequence,

$$\mathbf{app}_{seq}(a, s) = \begin{cases} T, & \text{if } a = \mathbf{null}_E, \\ T, & \text{if } s = T, \\ a \hat{} s, & \text{otherwise}, \end{cases}$$

$s_1 \leqslant s_2$ iff $s_1 = \varepsilon$ or $s_1 = \mathbf{app}_{seq}(a, s_1')$ and $s_2 = \mathbf{app}_{seq}(a, s_2')$ for some s_1' and s_2' such that $s_1' \leqslant s_2'$.

We can see that Seq(E) is ω-complete with respect to the sort seq, and it is a free extension of E with respect to CSpec.Sequence-of-events.

Note that if we take E to be Event, then we get the model, Seq(Event), of Cspec.Sequence-of-events. We take this model as the semantics of the specification CSpec.Sequence-of-events. Q.E.D.

As what we have done for CSpec.Sequence-of-events, we can obtain a model of CSpec.Set-of-sequences as a free extension of Seq(Event) that we take as the semantics of CSpec.Set-of-sequences.

Theorem 4.2. *Given E as model of NSpec.Events, and let Seq(E) be the model of CSpec.Sequence-of-events constructed in Theorem 4.1, then there is a model of CSpec.Set-of-sequences which is a free extension of Seq(E).*

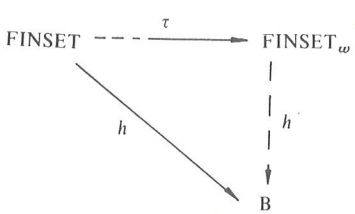
Proof. Let $\mathrm{SET} = (E, \mathrm{Seq}(E), \mathrm{SET}_{set}, \mathrm{Empty}, \mathrm{All}, \mathrm{Insert}, \text{Set-app}, \bigsqcup_{set}, \bigsqcap_{set})$, where

$\mathrm{SET}_{set} = \{S: S \text{ is a set of elements of Seq}(E) \text{ without T in it}\}$,
Empty $= \phi$, the empty set,

$$\text{Insert}(s, S) = \{s\} \cup S,$$
$$\text{Set-app}(a, S) = \{a \hat{} s : s \in S\},$$
$$S_1 \sqcup_{set} S_2 = S_1 \cup S_2,$$
$$S_1 \sqcap_{set} S_2 = S_1 \cap S_2,$$
$$\text{All} = \text{Seq}_{seq},$$
$$S_1 \leqslant S_2 \text{ iff } S_1 \cup S_2 = S_2 \text{ iff } S_1 \cap S_2 = S_1.$$

Note that NSpec . Events defines a class of lattices . So every model of this specification can be regarded as a lattice of sets and we can assume that each element in the carrier set of evt in E is a set in this proof.

It is important to note that all finite sets together with All form an initial ordered algebra of CSpec . Set- of - sequences , and we write it as FINSET . From a theorem in [3] , we have that there is an ω-complete ordered algebra FINSET_ω and an order preserving embedding $\tau : \text{FINSET} \to \text{FINSET}_\omega$ with the following property . For any ω- complete ordered algebra B of CSpec . Set - of - sequences and any order preserving homomorphism $h : \text{FINSET} \to B$, there is a unique ω - continuous homomorphism $h' : \text{FINSET}_\omega \to B$ such that the following diagram commutes .



From the initiality of FINSET , we get the initiality of FINSET_ω in the class of all ω-complete algebras of CSpec . Set-of-sequences .

Let i : FINSET → SET be the inclusion , and i' be the uniquely defined continuous homomorphism in the above diagram , and $i'[\text{FINSET}_\omega] \subseteq$ SET be the image of i', that is a subalgebra of SET , it is clear that i' is a one - one homomorphism . Obviously , $i'[\text{FINSET}_\omega]$ is a free extension of Seq (E). Q . E . D .

If we take E to be Event and Seq (E) to be Seq (Event), the free extension of Seq (Event) constructed in Theorem 4 . 2 , denoted as ISET , is to be taken as the semantics of CSpec . Set- of - sequences .

Finally , we have the important result :

Theorem 4 . 3 . *The model of CSpec . Set- of -sequences , ISET , is consistent with the model of the temporal logic semantics of CSP given in* [13] , *i . e . , ISET is a model of the temporal logic in* [13] .

Proof. Straight-forward and omitted . Q . E . D .

5 . Discussion

The non-continuous algebraic specifications of abstract data types have been well investigated in literature , but there are few researches in general studies of continuous algebraic specifications of abstract data types. At present we still do not know whether the requirements are sufficient or necessary for the axioms to allow initial models of the specifications , and guarantee the completeness of the partial ordering of the

quotient of terms . In further research we feel the imperativeness of more general studies on continuous specifications .

On the basis of the algebraic semantics of CSP given in this paper , we can define the traces of the processes of [6] in a formal way . Some properties of communicating systems , such as fairness , liveness , termination , etc ., can also be specified . But to make the algebraic specification language more expressive to define the auxiliary variables as in [13] , harder work is still waiting for us . We are now working on a more powerful specification language which combines the algebraic specifications , temporal logic , and CSP notations .

Acknowledgements

I am grateful to my adviser , Professor Zhou Chaochen for his constant guidance and encouragement. My special thanks go to Doctor David Rydeheard for his careful reading the draft of this paper and his good suggestions . I want to express my thanks to my fellow students Gong Yu , Liu Junbo , and Liu Xinxin for their discussions and help while I was working on this subject .

References

[1] J . A . Goguen , J . W . Thatcher and E . G . Wagener , An Initial Algebra Approach to the Specification , Correctness , and Implementation of Abstract Data Types , *in* Current Trends in Programming Methodology, **4** (1978), 80 — 194 .

[2] J . A . Goguen , J.W. Thatcher , E . G .Wagener and J.B.Wright, Initial algebra semantics and continuous algebras , *J . ACM*, **24** (1977).

[3] S . L . Bloom , Varieties of ordered algebras , *Journal of Computer and System Sciences* , **13** (1979), 200 — 212 .

[4] R . M . Burstall and J . A . Goguen , Putting theories together to make specifications , Proc . Fifth Int 'l Joint Conf . on Artificial Intelligence , 2 (1977) , 1045 —1058 .

[5] M . Broy and M . Wirsing , A systematic study of models of abstract data types , *Theoretical Computer Science* , **33** (1984) , 139 — 174 .

[6] C . A . R . Hoare , Communicating Sequential Processes , Prentice-Hall , International Series in Compter Science .

[7] International Standardization Organization , LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour , Draft Proposal ISO / OP 8807 , 1985 .

[8] R . Milner , A Calculus of Communicating Systems , Leture Notes in Computer Science 92 (1979).

[9] M . J . O 'Donnell , Computing in Systems Described by Equations , Lecture Notes in Computer Science 58 .

[10] Udo Pletat , Algebraic Specifications of Abstract Data Types and CCS : An Operational Junction , Proc . 6th IFIP Int 'l Workshop on Protocol Specification , Testing and Verification , Montreal , June 1986 .

[11] M . Wirsing , P . Peper , H . Partsh , W . Dosech and M . Broy , On hierarchies of abstract data types , Acta Informatica , **20** (1983) , 1 —33 .

[12] Zhou Chaochen , Specifying Communicating Systems with Temporal Logic , Lecture Notes at the Technical University of Denmark , Jan . 1986 .

[13] Zhou Chaochen , A Temporal Semantics of Communicating Processes , Proc . of PPCC — I , 1985 , Melbourne , Australia , 617 —630 .

[14] Zhou Chaochen , A study of communicating sequential processes , *Chinese Journal of Computers* , **6** :1 (1983), 1 —8 .