A SOUND AND COMPLETE HOARE AXIOMATIZATION OF THE ADA-RENDEZVOUS

(extended abstract)

Rob Gerth

RUU-CS-82-5

April 1982

Rijksuniversiteit Utrecht

Vakgroep informatica

Princetonplein 5
Postbus 80.002
3508 TA Utrecht
Telefoon 030–53 1454
The Netherlands

This paper will be presented at the
9th International Colloquium on
Automata, Languages and Programming,
Aarhus, July 12-16, 1982.

A SOUND AND COMPLETE HOARE AXIOMATIZATION OF THE ADA-RENDEZVOUS

(extended abstract)

Rob Gerth[1]
Department of Computer Science, University of Utrecht
P.O. Box 80.002, 3508 TA Utrecht, the Netherlands.

0. ABSTRACT.

A HOARE-axiomatization is constructed for the ADA rendezvous, embedded in a subset of the ADA concurrency section. The well-known CSP proof system of Apt, Francez and de Roever is taken as a starting point. We prove the axiomatization to be sound and relatively complete with respect to partial correctness properties.

1. INTRODUCTION.

We study the basic synchronization primitive - the *rendezvous* - of the programming language ADA [ARM81]. A proof system is constructed for partial correctness properties for a subset, ADA-CS, of its concurrency section. The proof system is then proven to be sound and relatively complete.

As CSP [Hoa78] is one of the languages which influenced the design of ADA, it should come as no surprise that the resulting proof system is similar to proof systems for CSP; especially to the one in [AFR80].

Within the latter system, CSP-synchronization is captured by a *general invariant* and a *cooperation test*. A rendezvous can be characterized as something akin to a procedure-call. We will show how the notion of cooperation can be combined with a procedure-call-rule, as expressed in our formation-rule. Notably, we will retain the idea of one proof per procedure-body which suffices for each call. This rule is inspired by [RGR81] and [GRR82].

In this abstract, we first describe the subset ADA-CS. Next, in section 3, a proof system for ADA-CS is constructed that closely follows the example of the CSP proof system in [AFR80]. In section 4, the soundness and completeness proof are sketched. Space limitations prevents us from including full proofs. For these and an example-proof of an ADA-CS program, the reader is referred to [Ger82]. We conclude, in section 5, with a discussion of our work.

The reader is assumed to be familiar with the CSP-system in [AFR80].

2. THE SUBSET, ADA-CS.

The syntax of the subset of ADA is described by the following augmented BNF-grammar. The conventions used are similar to those in [ARM81]: *Italicized* prefixes in the non-terminals are irrelevant; "[...]" denotes an optional part, "{...}" denotes repetition, zero or more times. We have taken some liberties with the ADA-syntax, which is somewhat verbose.

---

[1] The author is supported by the Netherlands Organization for the Advancement of Pure Research (ZWO).

```
program ::= begin task {task} end
task ::= task task id decl begin stats end
decl ::= {entry_decl}{var_decl}
entry_decl ::= entry entry_id (formal_part)
var_decl ::= var_id_list:int | var_id_list:bool
var_id_list ::= var_id{,var_id}
formal_part ::= [var_id_list][# var_id_list]
stats ::= stat{;stat}
stat ::= null | ass_st | if_st | while_st | call_st | acc_st | sel_st
ass_st ::= var_id := expr
if_st ::= if bool_expr then stats else stats end if
while_st ::= while bool_expr do stats end while
call_st ::= call task id.entry id (actual_part)
actual_part ::= {expr}[# var_id_list]
acc_st ::= accept entry id (formal_part) do stats end accept
sel_st ::= select sel_br {or sel_br} end select
sel_br ::= bool_expr : acc_st [;stats]
expr ::= "expression"
bool_expr ::= "boolean expression"
id ::= "identifier"
```

Thus, an ADA-CS program consists of a fixed set of tasks. These tasks are all activated simultaneously and excuted in parallel. When execution reaches the end of the task-body, the task terminates. Each task can have declarations for entries, which may be called by other tasks. The actions to be performed, when such an entry is called, are specified by corresponding accept-statements. Execution of an accept is synchronized with the excution of a corresponding entry-call. Consequently, a task executing an accept or entry-call, will be suspended until another process reaches a corresponding entry-call or accept,after which the statements of the accept-body are executed by the called task, while the calling task remains suspended. This action is called a *rendezvous* and is the primary means of communication between and synchronization of tasks; in particular, there are no global variables. After a rendezvous, the two tasks continue their execution in parallel.

Apart from the synchronization, an entry-call is executed as an ordinary procedure-call. An entry-declaration may specify a formal_part. Only parameters of type int are allowed. The first set of parameters, closed off by the '#'-sign, is of mode in; the second set is of mode in out. Hence, in the actual_part of a corresponding call, the first set of actual parameters may be expressions, the second set must be variables.

The formal_part of an accept-statement must match the one of the corresponding entry-declaration. A task may only contain accept-statements for one of its own entries, but it may contain more than one accept-statement for the same entry.

The select-statement allows a task to wait for synchronization with one of a set of alternatives. First, all boolean expressions are evaluated to determine which branches of the select-statement are open. If all are closed, the statement aborts. Otherwise, the task, if necessary, waits until a rendezvous corresponding with one of the open branches is possible. (Notice that each branch starts with an accept-statement.) If more than one rendezvous is possible, one is selected arbitrarily.

Two more or less implicit changes w.r.t. the ADA semantics are the following:

(1) the removal of entry-queues[1], and

(2) deadlock instead of abortion in case an entry is called of an already terminated task.

3. THE PROOF SYSTEM.

A characteristic tendency in proof systems for concurrent languages is the reduction to sequential reasoning. Of course there is a price to pay: In [OwG76] an *interference freedom test* has to be introduced and in [AFR80] a *cooperation test*.

In the CSP-system, this reduction is brought about by introducing two axioms for local reasoning about processes in isolation:

(1) $\{p\}\ P_i!a\ \{q\}$ and (2) $\{p\}\ P_j?x\ \{q\}$.

The actual test whether these pre- and post-assertions are compatible with the communication action, is deferred to a second global stage: that of the cooperation test. To express this test, a *general invariant*, GI, is introduced to tie the local reasonings, within each of the processes, globally together. In particular, GI is used to distinguish among all communication possibilities, i.e., the *syntactically* matching ones, the ones which may actually occur, i.e., the *semantically* matching ones. Also, auxiliary variables have to be introduced to express the necessary assertions and invariants. As the variables appearing free in GI have to be updated when communication occurs to model synchronization, GI cannot hold throughout the whole program. Hence the introduction of *bracketed sections* (each associated with a unique communication-action) to which the updating of GI-variables is confined and inside of which GI need not hold.

The reader will find these concepts reappearing (for the same reasons) in the ADA-CS proof system to be developed now.

A rendezvous may be viewed as a double CSP-communication, as the translation shows:

	ADA-CS:	CSP:
T_1:	call T_2.entry(e#x)	$T_2!(e,x);\ T_2?x$
T_2:	accept entry(u#v) do S end accept	$T_1?(u,v);\ S;\ T_1!v$

Correspondingly, in the proof system the following axiom and proof rules are adopted:

A1. *call* $\{p\}\ \underline{\text{call}}\ T.\text{entry}(\vec{e}\#\vec{x})\ \{q\}$

R1. *accept*
$$\frac{\{p_1\}\ S\ \{q_1\}}{\{p\}\ \underline{\text{accept}}\ \text{entry}(\vec{u}\#\vec{v})\ \underline{\text{do}}\ S\ \underline{\text{end}}\ \underline{\text{accept}}\ \{q\}}$$

The reason to use an accept-*rule* rather than an *axiom*, is to force proofs of the accept-bodies to be given.

R2. *select*
$$\frac{\{p\wedge b_1\}\ S_1\ \{q\},\ \ldots\ ,\ \{p\wedge b_n\}\ S_n\ \{q\}}{\{p\}\ \underline{\text{select}}\ b_1:S_1\ \underline{\text{or}}\ \ldots\ \underline{\text{or}}\ b_n:S_n\ \underline{\text{end}}\ \underline{\text{select}}\ \{q\}}$$

Remember that waiting, in case no rendezvous is immediately possible, is not a partial correctness notion.

---

[1] This is not serious, as in ADA-CS entry-queues cannot be used explicitely within programs. Hence, entry-queues only define an implementation of a fair scheduler, see [PR82]. As such, they are not relevant for partial correctness properties.

These are augmented by the usual *assignment*-axiom (A2), the *null*-axiom (A3 {p}null{p}) and the *if*-, *while*-, *composition*-, *consequence*- and *conjunction*-rules (R3...R7). We also need a substitution rule to remove auxiliary variables from assertions (see below)

R8. *substitution* $\dfrac{\{p\}\ S\ \{q\}}{\{p[t/z]\}\ S\ \{q\}}$, provided $z \notin FV(S,q)$

Next, a *general invariant* GI is introduced and the concept of *bracketed sections*. Here we must be careful:

Definition. A task is *bracketed* if the brackets "<" and ">" are interspersed in its text, so that for each program-section <S> (to be called a *bracketed section*), S is of the form (1) S_1; call entry($\vec{e}\#\vec{x}$); S_2 , or

(2) accept entry($\vec{u}\#\vec{v}$) do T_1 , or

(3) T_2 end accept

where S_1, S_2, T_1 and T_2 do not contain calls or accepts and may be null-statements.

It would have been wrong to replace (2) and (3) together by the single clause

(2') accept entry($\vec{u}\#\vec{v}$) do T_1; T; T_2 end accept.

The idea is that GI must hold when a communication-action is executed; that is, upon entrance of and exit from the associated bracketed section. But as the "body", T, of an accept may contain other calls or accepts, the validity of GI would not have been assured at these points, since GI would not have been required to hold within S.

Auxiliary variables are introduced to express assertions and invariants. These variables do not affect program-control during execution. Correspondingly, a proof rule is needed to remove assignments to auxiliary variables.

R9. *AV* Let AV be a set of variables such that $x \in AV \Rightarrow$ x appears in S' only in assignments of the form y:=x, where $y \in AV$. Then

$$\frac{\{p\}\ S'\ \{q\}}{\{p\}\ S\ \{q\}}\ ,$$

provided S is obtained from S' by deleting assignments to variables in AV and $FV(q) \cap AV = \emptyset$.

Although a rendezvous is modelled by two CSP-communications, it makes sense not to separate them entirely, because, disregarding the synchronization, a rendezvous is no more than an ordinary procedure-call, which we want to treat as a meaningful whole. This will influence the formulation of our cooperation test.

Definition. Let <S_1> and <S_2> be a *communication pair*:

$S_1 \equiv S_1'$; call T.entry$_1$($e\#x$); S_1''

$S_2 \equiv$ accept entry$_2$($\vec{u}\#\vec{v}$) do S_2';> S <;S_2'' end accept

(S_1 and S_2 contained in different tasks)

We say that <S_1> and <S_2> (syntactically) *match*, if 'entry$_1$' and 'entry$_2$' are the same name, and S_2 is part of the body of task T.

The wary reader will have noticed that the formal parameters of the entries are *local* w.r.t. the corresponding accepts. However, to facilitate the proofs in section 4, we do not introduce a *block rule*. Instead we will restrict, without loss of generality, the set of ADA-CS programs: *ADA-CS tasks may not have name-clashes between the formal parameters of its entries and its 'normal' variables*. This allows us to restrict the variables appearing free in assertions, thus obliterating the need of a block rule. The following definition reflects this.

Definition. Consider (a proof of) an ADA-CS program begin task T_1 ... task T_n end. The proofs of $\{p_i\}\ T_i\ \{q_i\}$ (i=1..n) *cooperate w.r.t. GI* if

(1) the assertions used in the proof of $\{p_i\}\ T_i\ \{q_i\}$ have no free variables subject to change in T_j ($i \neq j$) and contain formal parameters of an entry only if they appear in the proof of the body for an accept for this entry.

(2) $\{pre(S_1) \wedge pre(S_2) \wedge GI\}\ S_1 \parallel S_2\ \{post(S_1) \wedge post(S_2) \wedge GI\}$
holds for all matching communication pairs $<S_1>$ and $<S_2>$ within the program.

As in the CSP proof system, we need an additional rule to establish cooperation. This rule basically expresses the value transfer during a communication. In CSP this is simple, as the execution of two matching io-commands, $P_i!a$ and $P_j?x$, is equivalent with executing the assignment x:=a. In ADA however, a rendezvous results in a procedure-call. In order to render the semantics of such a call simple, a few restrictions are placed on the actual parameters of a call. (After all, we are not interested in the purely sequential aspects of ADA.) Hence:

For any entry-call $\underline{call}\ entry(e_1,e_2,\ldots,e_m\#x_1,x_2,\ldots,x_n)$
to an entry declared as $\underline{entry}\ entry(u_1,u_2,\ldots,u_m\#v_1,v_2,\ldots,v_n)$
the following assumptions are made about the parameter lists $\vec{e}$, $\vec{x}$, $\vec{u}$ and $\vec{v}$:

(1) $FV(\vec{e}) \cap \vec{x} = \emptyset$ and no u_i appears on the left-hand side of any assignment or as value-result actual parameter of any call within the "bodies of entry",

(2) the x_i are all distinct, and

(3) $(s \in (FV(\vec{e}) \cup \vec{x}) \wedge s \notin \vec{u} \cup \vec{v})) \Rightarrow s \notin FV(\text{"bodies of entry"})$
where "bodies of entry" denote the bodies of the accepts for entry.

A remark should be made now. According to the ADA-standard [ARM81, 6.2], programs should not depend on specific mechanisms for parameter-passing and should not use aliasing. This justifies our using call-by-value-result (call-by-copy for the purist) in the ADA-CS subset and the introduction of the above restrictions; only (1) appears to be a real restriction. In fact, instead of (3) the following even stronger assumption will be made: *An ADA-CS program may not have tasks which contain variables with the same name as in other tasks of the program.* Proofs of such tasks simply cannot be combined because of the resulting name-clashes.

Under the above restrictions the parameter-transfer during the execution of an entry-call may be simulated by a substitution ([Apt81b]):

$$\underline{begin}\ \underline{new}\ \vec{u},\vec{v};\ \vec{u}:=\vec{e};\ \vec{v}:=\vec{x};\ S\ ;\vec{x}:=\vec{v}\ \underline{end} \equiv S[\vec{e}/\vec{u}\,,\ \vec{x}/\vec{v}]. \qquad (S1)$$

Moreover, if $\{p\}\ S\ \{q\}$ then $\{p[\cdot]\}\ S[\cdot]\ \{q[\cdot]\}$ (where $[\cdot] \equiv [\vec{e}/\vec{u}, \vec{x}/\vec{v}]$). Hence, the input-output behaviour of a call is obtained by substituting in the canonical (!) proof, $\{p\}\ S\ \{q\}$, of the procedure-body S.

The well-known call-rules, associated with the above result on substitution (cf. [Apt81b]), are too simple because we have bracketed sections to consider too. Hence the following rule:

R10. *formation*

$$\frac{\begin{array}{l}\{p_1 \wedge p_2 \wedge GI\}\ S_1';\ S_2'[\cdot]\ \{\overline{p}_1 \wedge \overline{p}_2[\cdot] \wedge GI\} \\ \{\overline{p}_2\}\ S\ \{\overline{q}_2\} \\ \{\overline{p}_1 \wedge \overline{q}_2[\cdot] \wedge GI\}\ S_2''[\cdot];\ S_1''\ \{q_1 \wedge q_2 \wedge GI\}\end{array}}{\{p_1 \wedge p_2 \wedge GI\}\ S_1';\ \underline{\text{call}}\ T_j.a(\vec{e}\#\vec{x});S_1''\ \|\ \underline{\text{accept}}\ a(\vec{u}\#\vec{v})\ \underline{\text{do}}\ S_2';>S<;S_2''\ \underline{\text{end}}\ \underline{\text{accept}}\ \{q_1 \wedge q_2 \wedge GI\}}$$

where (1) the call is contained in task T_i and the accept in task T_j $(i \neq j)$,
(2) $[\cdot] = [\vec{e}/\vec{u}, \vec{x}/\vec{v}]$,
(3) p_1, $\overline{p}_1$ and q_1 belong to the proof of T_i and $FV(\overline{p}_1) \cap \vec{x} = \emptyset$, and
(4) p_2, $\overline{p}_2$, q_2 and $\overline{q}_2$ belong to the proof of T_j.

In this rule, the first and third premiss (above the line) express the invariance checks of GI over the bracketed sections. The second premiss does not refer to actual parameters. This means that a proof of the body, S, of an accept *need only be given once* and that this *canonical* proof suffices for the cooperation test for matching communication pairs containing this accept. In the first premiss, we must show that the input is legal by deriving $\overline{p}_2[\cdot]$. If the input is legal, $\overline{q}_2[\cdot]$ specifies the output of the body S. The intermediate assertion $\overline{p}_1$ is used to retain information about the variables of T_i which do not appear in the actual result-parameter list of the call. This information cannot be placed in $\overline{p}_2$, because $\overline{p}_2$ is an assertion belonging to the proof of T_j and hence may not contain variables subject to change in T_i.

Now, the (usual) parallel composition (meta-) rule can be stated :

R11. *parallel composition*

$$\frac{\text{proofs of } \{p_i\}\ T_i\ \{q_i\},\ i=1..n \text{ cooperate w.r.t. GI}}{\{p_1 \wedge \ldots \wedge p_n \wedge GI\}\ \underline{\text{begin}}\ \underline{\text{task}}\ T_1\ \ldots\ \underline{\text{task}}\ T_n\ \underline{\text{end}}\ \{q_1 \wedge \ldots \wedge q_n \wedge GI\}},$$

provided no variable free in GI is subject to change outside a bracketed section.

4. SOUNDNESS AND COMPLETENESS.

In this section, soundness and completeness proofs for the ADA-CS proof system are sketched. For full proofs, the reader is referred to the full version of the paper [Ger 82]. The proof is based on the translation of ADA-CS into CSP. However, to circumvent some purely technical details, a slight generalization of CSP, CSP^+, will be used as target language.

CSP^+ is CSP augmented with some additional communication primitives, but without the distributed termination convention. The new primitives are characterized by generalization of syntactical matching:

(1) $P_i.a?x$ and $P_j.b!e$, where 'a' and 'b' are arbitrary identifiers.
These only match if, besides the usual constraints, a≡b holds. Execution of a matching pair results, as usual, in assignment of expression e to variable x.

(2) ?x and !e.
A command ?x (within P_i) matches with P_i!e (or !e) in *any other* process. Execution is as usual.

(3) $P_i?(x_1,\ldots,x_m)$ and $P_j!(e_1,\ldots,e_m)$.
These match if, besides the usual constraints, the number of variables and values are equal. Execution results in the simultaneous assignment of e_i to x_i. All variables x_i should be different.

Of course, these new primitives may be combined so as to allow constructs like !(e,t), $P_i.a?(x,y)$ and .c?(u,v,w), with obvious semantics.

The translation T: ADA-CS→CSP$^+$ is now defined as follows[1] (induction on complexity):

$T(\underline{\text{begin}}\ \underline{\text{task}}\ T_1\ \ldots\ \underline{\text{task}}\ T_n\ \underline{\text{end}}) = [P_1 \| \ldots \| P_n] \quad (P_i \equiv T(T_i))$

$T(\underline{\text{call}}\ T_i.a(\vec{e}\#\vec{x})) = P_i.a!(\vec{e},\vec{x});\ P_j.a?\vec{x}$

$T(\underline{\text{select}}\ b_1:S_1\ \underline{\text{or}}\ \ldots\ \underline{\text{or}}\ b_n:S_n\ \underline{\text{end select}})$, where $S_i \equiv \underline{\text{accept}}\ a_i(\vec{u}_i\#\vec{v}_i)\ \underline{\text{do}}\ S_{i1}\ \underline{\text{end accept}};\ S_{i2}$

$= [\square_i b_i;a_i?(\vec{u}_i,\vec{v}_i)\rightarrow T(S_{i1});a_i!\vec{v}_i;T(S_{i2})]$

In all remaining occurrences of accept-statements

$T(\underline{\text{accept}}\ a(\vec{u}\#\vec{v})\ \underline{\text{do}}\ S\ \underline{\text{end accept}}) = [a?(\vec{u},\vec{v})\rightarrow T(S);a!\vec{v}]$

In the other cases, the translation is the identity mapping.

Correctness of this translation is stated by

Theorem 1. For each program T∈ADA-CS and assertions p and q

$$\models \{p\}\ T\ \{q\} \Leftrightarrow \models \{p\}\ T(T)\ \{q\},$$

where validity is defined relative to some semantics, not further specified in this abstract.

A proof system for CSP$^+$ is obtained by an obvious redefinition of syntactical matching in the CSP system, besides some trivial changes which are connected with the different communication primitives. Soundness and completeness of the CSP system (see [Apt81a]) carry over to the CSP$^+$ system. This is intuitively clear, as the differences between CSP$^+$ and CSP are (1) a refinement of *syntactical* matching and (2) the ability to transfer a *syntactically* determined sequence of values between processes. For proofs we refer to [Ger82] or [FCN82]. As we shall see, having a sound and complete proof system allows us to answer semantic questions by the construction of formal proofs.

We need some notation: A denotes the ADA-CS proof system and C the CSP$^+$ proof system. L denotes the language of Peano arithmetic (or some first order countable extension) and is the assertion language; J is some model of L. $Tr_J \stackrel{D}{=} \{\varphi\in L \mid J\models\varphi\}$ and we write $B \vdash_{\mathcal{D}} \Phi$ if the formula Φ can be proven in the proof system $\mathcal{D}$, using the assertions in B in the consequence-rule. When this is clear from the context, references to B and/or $\mathcal{D}$ will be omitted.

[1] We assume that no ADA-CS task has entries with the same name as entries in other tasks.

Soundness theorem. For any program S∈ADA-CS and p,q∈L

$Tr_J \vdash_A \{p\}\ S\ \{q\} \Rightarrow J \models \{p\}\ S\ \{q\}$

As in the case of the soundness proof of the CSP proof system, we too, run into some difficulties. Firstly, the parallel composition rule is not a Hoare-style proof rule because of its reference to cooperating proofs. Secondly, it is not clear how to assign meaning to an isolated call or accept, which makes it difficult to prove soundness of the corresponding rules and axioms. Therefore, the proof system A is changed into an equivalent proof system A' which does use the ordinary notion of proof, by explicitly listing all conditions required in establishing cooperation.

To this end, *proof outlines* are introduced. If S is some ADA-CS task-body, a proof outline for S associates with each sub-statement R of S, a pre-assertion, pre(R), and a post-assertion, post(R). The following lemma connects the existence of a formal proof with 'validity' of a proof outline (see also [Apt81a])

Lemma 1. Let S be an ADA-CS task (or part of one) and p,q∈L. Then

$Tr_J \vdash_A \{p\}\ S\ \{q\}$ iff there is a proof outline of S such that for each sub-statement R of S, the following *verification conditions* are true in J:

1. $p \supset pre(S)$, $post(S) \supset q$
2. $pre(\underline{null}) \supset post(\underline{null})$
3. $pre(R) \supset post(R)[e/x]$, if R≡x:=e
4. $pre(R) \wedge b \supset pre(R_1)$, $pre(R) \wedge \neg b \supset pre(R_2)$, $post(R_i) \supset post(R)$ (i=1..2)
   if R≡<u>if</u> b <u>then</u> R_1 <u>else</u> R_2 <u>end if</u>
5. $pre(R) \wedge b \supset pre(R_1)$, $post(R_1) \supset pre(R)$, $pre(R) \wedge \neg b \supset post(R)$
   if R≡<u>while</u> b <u>do</u> R_1 <u>end while</u>
6. $pre(R) \wedge b_i \supset pre(R_i)$, $post(R_i) \supset post(R)$ (i=1..n)
   if R≡<u>select</u> $b_1:R_1$ <u>or</u> ... <u>or</u> $b_n:R_n$ <u>end select</u>
7. $pre(R) \supset pre(R_1)$, $post(R_1) \supset pre(R_2)$, $post(R_2) \supset post(R)$
   if $R \equiv R_1;R_2$

and such that $FV(pre(R)) \cap \vec{x} = \emptyset$ if R≡<u>call</u> $T.a(\vec{e}\#\vec{x})$

Such a proof outline is called *valid* (w.r.t. J) for {p}S{q}, and VC({p}S{q}) denotes the set of verification conditions.

In order to combine the isolated, sequential proofs (or proof outlines), A imposes a second global set of constraints on the assertions of the proof outlines, the *cooperation conditions*, as embodied in the cooperation test:

For any matching communication pair,

S_1';<u>call</u> $T.a(\vec{e}\#\vec{x})$;S_1'' and <u>accept</u> $a(\vec{u}\#\vec{v})$<u>do</u> S_2';>S<;S_2'' <u>end accept</u>,

the following two formulae should be valid in J

$\{pre(S_1') \wedge pre(S_2') \wedge GI\}\ S_1';S_2'[\cdot]\ \{pre(\text{'call'}) \wedge post(S_2'[\cdot]) \wedge GI\}$

$\{pre(\text{'call'}) \wedge pre(S_2'')[\cdot] \wedge GI\}\ S_2''[\cdot];S_1''\ \{post(S_1'') \wedge post(S_2'') \wedge GI\}$.[1]

The set of these conditions for valid proof outlines for $\{p_i\}T_i\{q_i\}$ (i=1..n) w.r.t. GI, is denoted by $CC(\{p_i\}T_i\{q_i\}\ i=1..n, GI)$.

Basing our argument on lemma 1, we replace the axiom A1 and the proof rules R1, R2,

---

[1] Defining cooperation directly in terms of the formulae to be proven, instead of relying on the formation-rule, was suggested by W.P. de Roever. Thus, some problems are circumvented in defining validity of the formation-rule.

R10 and R11 of A by the single rule

R11'. *combined rule*
$$\frac{VC(\{p_i\}T_i\{q_i\}) \ , \ CC(\{p_i\}T_i\{q_i\}\ i=1..n,\ GI)}{\{p_1\wedge\ldots\wedge p_n\wedge GI\}\ \underline{\text{begin}}\ \underline{\text{task}}\ T_1\ \ldots\ \underline{\text{task}}\ T_n\ \underline{\text{end}}\ \{q_1\wedge\ldots\wedge q_n\wedge GI\}}$$

provided (1) no assertion in the proof outline of T_i contains variables subject to change in T_j $(i\neq j)$ and formal parameters only appear in assertions belonging to the outlines of corresponding accept-bodies,
(2) GI does not have free variables subject to change outside bracketed sections.

The resulting proof system A' consists of the axioms A2..3 and the rules R3..9 and R11'. Because of lemma 1, equivalence of A and A' is not difficult to show, hence it suffices to prove A' sound. Similar changes can be made to the CSP$^+$ proof system C. In particular, a lemma analogous to lemma 1 holds (see [Apt81a]). The resulting proof system is called C' and is equivalent with C. (In order to stay in the spirit of [Apt 81a], C' retains the C formation rule.)

As A' uses the ordinary notion of proof, it suffices to prove soundness of the axioms and proof rules. For these proofs, an important observation is the following one:
Because of theorem 1 and soundness and completeness of C'

$$J \models \{p\}\ S\ \{q\} \ \Leftrightarrow\ J \models \{p\}\ T(S)\ \{q\} \ \Leftrightarrow\ Tr_J \vdash_{C'} \{p\}\ T(S)\ \{q\}.$$

This concludes the preliminaries and we continue with proving

Soundness of the combined rule.

By the above observation, it suffices to prove

$$Tr_J \vdash_{C'} \{p_1\wedge\ldots\wedge p_n\wedge GI\}\ [P_1\|\ldots\|P_n]\ \{q_1\wedge\ldots\wedge q_n\wedge GI\} \quad (P_i\equiv T(T_i)), \qquad (\$2)$$

assuming the conditions in $VC(\{p_i\}T_i\{q_i\})$, i=1..n, to hold and proofs of the translations of the formulae in $CC(\{p_i\}T_i\{q_i\}\ i=1..n,\ GI)$ to be given.

We will establish the premisses of the C' combined rule.
Using the ADA-CS proof outlines, it is straightforward to construct (valid) CSP$^+$ proof outlines for the translations and hence to obtain validity of $VC(\{p_i\}P_i\{q_i\})$ i=1..n, the first premiss of the C' combined rule.

Next, we must show that ADA-CS cooperation implies CSP$^+$ cooperation. Consider the following proof outlines of a matching communication pair in the ADA-CS program:

$\{p_1'\}\langle S_1'\{p\}\underline{\text{call}}\ T_j.a(\vec{e}\#\vec{x})\{q\}S_1''\rangle\{q_1''\}$
$\{\overline{p}\}\langle\underline{\text{accept}}\ a(\vec{u}\#\vec{v})\ \underline{\text{do}}\{p_2'\}S_2'\rangle\{q_2'\}S\{p_2''\}\langle S_2''\{q_2''\}\underline{\text{end}}\ \underline{\text{accept}}\rangle\{\overline{q}\}$

The proof outlines of the translation, are as follows $(R_i\equiv T(S_i))$:

$\{p_1'\}\langle R_1'\{p\}P_j.a!(\vec{e},\vec{x})\rangle\{p\}\langle P_j.a?\vec{x}\{q\}R_1''\rangle\{q_1''\}$
$\{\overline{p}\}\langle a?(\vec{u},\vec{v})\rightarrow\{p_2'\}R_2'\rangle\{q_2'\}R\{p_2''\}\langle R_2''\{q_2''\}a!\vec{v}\rangle\{\overline{q}\}$

By way of example, we prove CSP$^+$ cooperation of the first matching pair of bracketed sections: $\{p_1'\wedge\overline{p}\wedge GI\}\ R_1';P_j.a!(\vec{e},\vec{x})\ \|\ a?(\vec{u},\vec{v})\rightarrow R_2'\ \{p\wedge q_2'\wedge GI\}$.
Using the arrow rule, we may replace '→' by ';' in the statement on the right. We intend to use the C' formation rule, hence we must prove its premisses:

$$\vdash\{p_1'\wedge\overline{p}\wedge GI\}R_1'\{r\}\ ,\ \vdash\{r\}P_j.a!(\vec{e},\vec{x})\|a?(\vec{u},\vec{v})\{s\}\ ,\ \vdash\{s\}R_2'\{p\wedge q_2'\wedge GI\}, \qquad (\$3)$$

where r and s are yet to be determined.

We may assume to have proofs of the translations of the A' cooperation conditions; in particular, we may assume $\vdash\{p_1'\wedge\overline{p}\wedge GI\}R_1';R_2'[\cdot]\{p\wedge q_2'[\cdot]\wedge GI\}$ to hold. Hence, defining r to be the intermediate assertion between R_1' and $R_2'[\cdot]$, takes care of the first premiss in (§3) (this assertion exists because we have a formal proof). As $FV(r)\cap(\vec{u}\cup\vec{v})=\emptyset$, we can take $s\equiv r\wedge\vec{u}=\vec{e}\wedge\vec{v}=\vec{x}$ in the second premiss (apply the C' preservation and communication axioms).

When introducing the formation rule, we argued the equivalence between assignment and substitution in parameter-passing (cf. (§1)). This implies that

$$\models \{r\}\ R_2'[\cdot]\ \{p\wedge q_2'[\cdot]\wedge GI\} \Leftrightarrow \models \{r\}\ \vec{u}:=\vec{e};\ \vec{v}:=\vec{x};\ R_2';\ \vec{x}:=\vec{v}\ \{p\wedge q_2'[\cdot]\wedge GI\},$$

since $FV(r,p,q_2'[\cdot],GI)\cap(\vec{u}\cup\vec{v})=\emptyset$. And hence $\models \{r\wedge\vec{u}=\vec{e}\wedge\vec{v}=\vec{x}\}\ R_2'\ \{p\wedge q_2'[\vec{e}/\vec{u}]\wedge GI\}$. Finally, using completeness of C', applying the preservation axiom to $\{\vec{u}=\vec{e}\}$ and noticing that $\models q_2'[\vec{e}/\vec{u}]\wedge\vec{u}=\vec{e}\supset q_2'$, we get $\vdash \{s\}\ R_2'\ \{p\wedge q_2'\wedge GI\}$, the third premiss of the C' formation rule. Hence, this rule may be applied, thus establishing cooperation of the above matching pair.

When trying to prove cooperation of the second matching pair

$$\{p\wedge p_2''\wedge GI\}\ P_j.a?\vec{x};R_1''\ \|\ R_2'';a!\vec{v}\ \{q_1''\wedge\overline{q}\wedge GI\},$$

we run into a difficulty, because we need invariance of the value parameters, $\vec{u}$, over the translation of the accept-body. For this, the translations of the ADA-CS outlines are too weak (in ADA-CS, invariance is a direct consequence of the semantics). Hence, the CSP^+ proof outlines and GI have to be slightly strenghtened, implying that the above cooperation proof changes too. None of this, however, introduces essential difficulties and the method of proof stays the same; cf. [Ger82].

Thus, CSP^+ cooperation is shown, and this establishes the second premiss of the C' combined rule, $CC(\{p_i\}P_i\{q_i\}\ i=1..n, GI)$. Hence the rule may be applied, thus proving (§3) and soundness of the A' combined rule.

Because of the close correspondence between A' and C', soundness of the other rules and axioms is trivial.

Completeness theorem. For any $S\in$ADA-CS and $p,q\in L$

$$J \models \{p\}\ S\ \{q\} \Rightarrow Tr_J \vdash_A \{p\}\ S\ \{q\}.$$

Because of theorem 1 and completeness of C, this takes the following simpler form

$$Tr_J \vdash_C \{p\}\ T(S)\ \{q\} \Rightarrow Tr_J \vdash_A \{p\}\ S\ \{q\}.$$

Let S≡<u>begin task</u> T_1 ... <u>task</u> T_n <u>end</u> and assume we have $\vdash \{p\}T(S)\{q\}$. In this proof, go back to the (only) application of the C parallel composition rule, yielding

$$\vdash \{p_1\wedge\ldots\wedge p_n\wedge GI\}\ [T(T_1)'\|\ldots\|T(T_n)']\ \{q_1\wedge\ldots\wedge q_n\wedge GI\},$$

where $T(T_i)'$ denotes $T(T_i)$ augmented with auxiliary variables; we may assume $T(T_i)'=T(T_i')$. This implies that we have proofs for $\{p_i\}T(T_i')\{q_i\}$ and CSP^+ cooperation w.r.t. GI. From these, ADA-CS proofs are constructed for $\{p_i\}T_i'\{q_i\}$, simply by defining for each component statement S of T_i', $pre(S)\equiv pre(T(S))$ and $post(S)\equiv post(T(S))$. Validity of these proofs is obvious; ADA-CS cooperation remains to be shown. I.e., for each communication

pair $S_1';\underline{call}\ T_j.a(\vec{e},\vec{x});S_1''$, $\underline{accept}\ a(\vec{u}\#\vec{v})\ \underline{do}\ S_2';>S<;S_2''\ \underline{end}\ \underline{accept}$
the premisses of the A formation rule must be shown to hold. This is in fact straightforward, using the following lemma which states some general conditions under which variable- substitution preserves provability.

Lemma 2. Let $S\in$ADA-CS, $p,q\in L$, $\vec{u}$, $\vec{v}$ and $\vec{x}$ sequences of distinct variables and $\vec{e}$ a sequence of expressions, such that

(i) $FV(\vec{e},q)\cap\vec{x}=\emptyset$, $\vec{u}\cap\vec{v}=\emptyset$ and $FV(S,\vec{u},\vec{v})\cap(FV(\vec{e})\cup\vec{x})=\emptyset$

(ii) the variables in $\vec{u}$ do not appear on the left-hand side of any assignment in S or as value-result parameter of any call in S.

Then (1) $Tr_J \vdash_A \{p\}\ S\ \{q\} \Rightarrow Tr_J \vdash_A \{p[\cdot]\}\ S[\cdot]\ \{q[\cdot]\}$ $([\cdot]\equiv[\vec{e}/\vec{u}\,,\,\vec{x}/\vec{v}])$

(2) $Tr_J \vdash_C \{p\}\ T(S)\ \{q\} \Rightarrow Tr_J \vdash_C \{p[\cdot]\}\ T(S[\cdot])\ \{q[\cdot]\}$

CSP^+ cooperation of the first pair of bracketed sections in the translations of the above communication pair, yields $(T(S_i)=R_i)$:

$\vdash \{p_1\wedge p_2\wedge GI\}\ R_1'\ \{p\}$, $\vdash \{p\}\ P_j.a\ (\vec{e},\vec{x})\|a?(\vec{u},\vec{v})\ \{q\}$, $\vdash \{q\}\ R_2'\ \{p_1'\wedge p_2'\wedge GI\}$,

where $q\equiv p\wedge\vec{u}=\vec{e}\wedge\vec{v}=\vec{x}$, and the other assertions are taken from the appropriate outlines. Furthermore, we may assume $FV(p_1')\cap\vec{x}=\emptyset$. This is a consequence of the fact that only formulae of the form $\{r\}a|\vec{v}\|P_j.a?\vec{x}\{r\wedge\vec{x}=\vec{v}\}$ in which $FV(r)\cap\vec{x}=\emptyset$ can be proven in C (see the communication axiom). As R_1' and R_2' are *sequential* statements, we obtain by translating the CSP^+ proof outlines and using lemma 1 and 2 that:

$\vdash_A \{p_1\wedge p_2\wedge GI\}\ S_1';S_2'[\cdot]\ \{p_1'\wedge p_2'[\cdot]\wedge GI\}$,

the first premiss of the A formation rule.

The second premiss, $\vdash_A \{p_2'\}S\{q_2'\}$ $(q_2'\equiv pre(R_2''))$, is clear, as S is a sequential statement. Finally, CSP^+ cooperation of the second pair of bracketed sections yields

$\vdash \{p_1'\wedge q_2'\wedge GI\}\ R_2''\ \{\overline{p}\}$, $\vdash \{\overline{p}\wedge\vec{x}=\vec{v}\}\ R_1''\ \{q_1\wedge q_2\wedge GI\}$,

whence, using lemma 2 again (and translating the proofs)

$\vdash_A \{p_1'\wedge q_2'[\cdot]\wedge GI\}\ R_2''[\cdot]\ \{\overline{p}[\cdot]\}$, $\vdash_A \{\overline{p}[\vec{e}/\vec{u}]\wedge\vec{x}=\vec{v}\}\ S_1''\ \{q_1\wedge q_2\wedge GI\}$.

Because $\vdash \overline{p}[\vec{e}/\vec{u}]\wedge\vec{x}=\vec{v}\leftrightarrow\overline{p}[\cdot]\wedge\vec{x}=\vec{v}$ and $FV(\overline{p}[\cdot],S_1'',q_1,q_2,GI)\cap\vec{v}=\emptyset$, this implies

$\vdash \{p_1'\wedge q_2'[\cdot]\wedge GI\}\ R_2''[\cdot];R_1''\ \{q_1\wedge q_2\wedge GI\}$,

which is the third premiss of the A formation rule.

Thus, ADA-CS cooperation of the proofs has been shown. Application of the A parallel composition rule yields

$\vdash_A \{p_1\wedge\ldots\wedge p_n\wedge GI\}\ \underline{begin}\ \underline{task}\ T_1\ \ldots\ \underline{task}\ T_n\ \underline{end}\ \{q_1\wedge\ldots\wedge q_n\wedge GI\}$.

Finally, application of the A consequence, substitution and AV rule allows the conclusion $\vdash_A\{p\}S\{q\}$, thus establishing completeness of the ADA-CS proof system.

5. DISCUSSION.

In this paper we have studied the ADA rendezvous in a subset of the ADA concurrency section, ADA-CS. Using the basic ideas of the CSP proof system in [AFR80], we have constructed a partial correctness proof system and have proven that this completely axiomatizes ADA-CS (disregarding failure, termination and deadlock).

ADA-CS communication is basically captured by our formation rule, which combines

a simple procedure-call rule with a cooperation test. The idea of canonical proofs for the accept-bodies is retained, by observing that an ADA-CS communication action splits into two CSP-like communications, so that the cooperation test can be restricted to a small prelude and postlude of each accept-body. Along the way, some restrictions on the actual parameters were introduced. Although these restrictions do not reduce the 'power' of the subset, the reader may wonder whether it is really necessary to introduce them. We believe not. The restrictions were introduced so as to allow a very simple call rule to be used and we expect the same techniques to apply to more general rules such as in [GL80].

Canonicity (or rather the absence of it) subtly affects the cooperation test: Accept-bodies may contain other calls or accepts. So, if two different proofs of an accept-body are used in a program-proof, two different cooperation tests must be checked for each matching communication pair which references a call or accept in the accept-body, because the pre- and post-assertions of the call or accept will in general be different in the different proofs of the accept-body. Hence, more than for sequential languages, canonicity is something to be desired in proof systems for concurrent languages (a similar remark applies to interference freedom tests).

This paper does not discuss total correctness, absence of deadlock and the proof of safety properties. Using the CSP proof system, it is straightforward to strengthen the current system to prove total correctness and absence of deadlock. Extension to a safety proof system is more difficult and we refer the reader to [GRR82] which shows how to use proof outlines to derive safety properties (in a more general context than ADA-CS).

After constructing the ADA-CS proof system, we learned of independent work on ADA by Barringer & Mearns, [BM81]. They attempt to construct a proof system for a larger subset of the ADA concurrency section than ADA-CS, containing, f.i., real-time constructs and the terminate-statement; they also address the problems of deadlock and termination. Their starting point, too, is [AFR80]; there are however some problems with their application of cooperation.

ACKNOWLEDGEMENTS.

I would like to thank Adrie van Bloois, Marly Roncken and Job Zwiers for some fruitful discussions and especially Willem P. de Roever for his stimulating guidance.

REFERENCES.

Reference [Ger81] is not cited in the paper.

[Apt81a] Apt, K.: Formal Justification of a Proof System for Communicating Sequential Processes. to appear

[Apt81b] Apt, K.: Ten Years of Hoare's Logic: A Survey - Part 1. TOPLAS 3-4 p.431-484, 1981.

[AFR80] Apt, K., N.Francez, W.P.de Roever: A Proof System for Communicating Sequential Processes. TOPLAS 2-3 p.359-385, 1980.

[ARM81] The Programming Language ADA. Reference manual. LNCS 106, Springer Verlag, New York, 1981.

[BM81] Barringer, H., I.Mearns: Axioms and Proof Rules for ADA Tasks. Technical report, Dept. of Computer Science, University of Manchester, 1981.

[FCN82] Francez, N., E.M.Clarke, C.Nikolaou: Extended Naming Conventions for Communicating Processes and their Proof Rules. Proc. of the 9th ACM Symposium on the Principles of Programming Languages, 1982.

[Ger81] Gerth, R.: A Proof System for a Subset of the Concurrency Section of ADA. Technical report RUU-CS-81-17, Dept. of Computer Science, University of Utrecht, 1981.

[Ger82] Gerth, R.: A Sound and Complete Hoare Axiomatization of the ADA Rendezvous. Technical report, Dept. of Computer Science, University of Utrecht, 1982.

[GRR82] Gerth, R., W.P. de Roever, M.Roncken: Procedures and Concurrency: A Study in Proof. Proc. of the Vth International Symposium on Programming, LNCS 137 Springer Verlag, New York, 1982.

[GL80] Gries, D., G.Levin: Assignment and Procedure Call Proof Rules. TOPLAS 2-4 p.564-580, 1980.

[OwG76] Owicki, S., D.Gries: An Axiomatic Proof Technique for Parallel Programs I. Acta Inf. 6 p.319-340, 1976.

[PR82] Pnueli, A., W.P. de Roever: Rendezvous with ADA - A Proof Theoretical View. Technical report, the Weizmann Institute of Science, Israel, 1982.

[RGR81] Roncken, M., R.Gerth, W.P. de Roever: A Proof System for Brinch Hansen's Distributed Procecesses. Proc. 11th Anual Conference of the Gesellschaft für Informatik, Informatik Fachberichte, Springer Verlag, New York, 1981.