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
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).
Thus, an ADA-CS program consists of a fixed set of tasks. These tasks are all activated simultaneously and executed 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 execution 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 'first'-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:

\[(1) \{p\} P_i!a \{q\} \quad \text{and} \quad (2) \{p\} P_j?x \{q\}.\]

The actual test whether these pre- and post-assertions are compatible with the commu-
nication 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 distin-
guish 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 introduc-
tion 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:

\[
\begin{align*}
\text{ADA-CS:} & & \text{CSP:} \\
T_1: & \text{call } T_2, \text{entry}(e\#x) & T_2!(e,x); T_2?x \\
T_2: & \text{accept entry}(u\#v) \text{ do } S \text{ end accept} & T_1?(u,v); S; T_1!v
\end{align*}
\]

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

\[
\begin{align*}
A1. \text{call} & \quad \{p\} \text{call } T, \text{entry}(e\#x) \{q\} \\
R1. \text{accept} & \quad \{p_1\} S \{q_1\} \\
        & \quad \{p\} \text{accept entry}(u\#v) \text{ do } S \text{ end accept} \{q\}
\end{align*}
\]

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

\[
\begin{align*}
R2. \text{select} & \quad \{p\wedge b_1\} S_1 \{q\}, \ldots , \{p\wedge b_n\} S_n \{q\} \\
        & \quad \{p\} \text{select } b_1:S_1 \text{ or } \ldots \text{ or } b_n:S_n \text{ end select} \{q\}
\end{align*}
\]

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 pro-
grams. 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 $(p) S (q)$

\[
(p[t/z]) S (q)
\]

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. $S_1; \text{call entry}$($\#_1$); $S_2$, or
2. $\text{accept entry}$(\#_2$)$ do $T_1$, or
3. $T_2 \text{end accept}$

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

$$\text{(2') accept entry}$(\#_2$)$ do $T_1; T; T_2 \text{end accept.}$

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 $x \in \text{AV}$ implies $x$ appears in $S'$ only in assignments of the form $y:=x$, where $y \in \text{AV}$. Then

\[
(p) S' (q),
\]

\[
(p) S (q)
\]

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

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:

$S_1 = S'_1; \text{call T.entry}_1(e=x); S'_2$

$S_2 = \text{accept entry}_2(\#_2) \text{ do } S'_2; S <;S'_2 \text{ end accept}$

($S_1$ 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 \ldots \end{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) \}_{i=1..n} 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. \{\text{pre}(S_1) \land \text{pre}(S_2) \land GI \} S_1 \parallel S_2 \{\text{post}(S_1) \land \text{post}(S_2) \land GI \}

holds for all matching communication pairs \langle S_1 \rangle and \langle S_2 \rangle 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, \(P_i \downarrow a\) and \(P_j \uparrow x\), 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 \begin{call} call \text{\textbf{entry}}(e_1,e_2,\ldots,e_m \# x_1,x_2,\ldots,x_n) \end{call} to an entry declared as \begin{entry} entry(u_1,u_2,\ldots,u_m \# v_1,v_2,\ldots,v_n) \end{entry} the following assumptions are made about the parameter lists \(\bar{e}, \bar{x}, \bar{u}\) and \(\bar{v}\):

1. \(\text{FV(}\bar{e})\cap \bar{x} = \emptyset\) 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. \((s \in \text{FV(}\bar{e}) \cup \bar{u}) \land s \in \text{FV(}\bar{v})\) \implies s \notin \text{FV("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]):

\begin{verbatim}
begin new \bar{u}, \bar{v}; \bar{u} := \bar{e}; \bar{v} := \bar{x}; S ; \bar{x} := \bar{v} end = S[\bar{e}/\bar{u}, \bar{x}/\bar{v}].
\end{verbatim}  (S1)
Moreover, if \( \{p\} S \{q\} \) then \( \{p[a]\} \{q[b]\} \) (where \( \cdot[a/b] = \{a\} \cdot \{b\} \)). Hence, the input-output behaviour of a call is obtained by substituting in the canonical (1) 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:

\[
R10. \text{formation} \quad \begin{aligned}
\{p_1 & \wedge p_2 \wedge GI\} S_1^1; S_2^1[\cdot] \{\overline{p}_1 \wedge \overline{p}_2[\cdot] \wedge GI\} \\
\{\overline{p}_2\} S \{\overline{q}_2\} \\
\{\overline{p}_1 \wedge \overline{q}_2[\cdot] \wedge GI\} S_2^2[\cdot]; S_1^2 \{q_1 \wedge q_2 \wedge GI\}
\end{aligned}
\]

where (1) the call is contained in task \( T_i \) and the accept in task \( T_j \) (i\#j),
(2) \( \cdot[a/b] = \{a\} \cdot \{b\} \),
(3) \( p_1 \), \( \overline{p}_1 \) and \( q_1 \) belong to the proof of \( T_i \) and \( FV(\overline{p}_1) \equiv \emptyset \), 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:

\[
R11. \text{parallel composition} \quad \begin{aligned}
\text{proofs of } \{p_i\} T_i \{q_i\}, i=1..n \text{ cooperate w.r.t. GI} \\
\{p_1 \wedge \ldots \wedge p_n \wedge GI\} \begin{task} T_1 \ldots \end{task} T_n \text{ end } \{q_1 \wedge \ldots \wedge q_n \wedge GI\}
\end{aligned}
\]

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) \( P_i \cdot a?x \) and \( P_j \cdot b!e \), 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 \( P_i \cdot !e \) (or \( !e \)) in any other process. Execution is as usual.

(3) \( P_i \cdot ?(x_1, \ldots, x_m) \) and \( P_j \cdot !e \).
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 \( !(e,t) \), \( P_i \cdot a?(x,y) \) and \( .c?(u,v,w) \), with obvious semantics.

The translation \( T: \text{ADA-CS} \rightarrow \text{CSP}^+ \) is now defined as follows\(^1\) (induction on complexity):

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

\[
T(\text{call } T_i \cdot a(\overrightarrow{x})) = P_i \cdot a(\overrightarrow{x}); P_j \cdot a?\overrightarrow{x}
\]

\[
T(\text{select } b_1:S_1 \text{ or } \cdots \text{ or } b_n:S_n \text{ end select}) , \text{ if } S_i = \text{ accept } a_{i}(\overrightarrow{u},\overrightarrow{v}) \text{ do } S_{i1} \text{ end accept}; S_{i2}
\]

In all remaining occurrences of accept-statements

\[
T(\text{accept } a(\overrightarrow{u},\overrightarrow{v}) \text{ do } S \text{ end accept}) = [a(\overrightarrow{u},\overrightarrow{v}) \rightarrow T(S); a!\overrightarrow{v}]
\]

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

Correctness of this translation is stated by

**Theorem 1.** For each program \( T: \text{ADA-CS} \) and assertions \( p \) and \( q \)

\[
\vdash [p]T[q] \rightarrow \vdash [p]T(T)[q],
\]

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

A proof system for \( \text{CSP}^+ \) 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 \( \text{CSP}^+ \) 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 \( L \). \( T_j \) is a model of \( L \) with \( \varphi \) and we write \( \vdash_{\varphi} \varphi \) if the formula \( \varphi \) can be proven in the proof system \( D \), using the assertions in \( B \) in the consequence-rule. When this is clear from the context, references to \( B \) and/or \( D \) 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 $S$ and $p,q \in L$

$$\text{Tr}_f \vdash A (p) S (q) \Rightarrow \text{J} \vdash (p) S (q)$$

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, $\text{pre}(R)$, and a post-assertion, $\text{post}(R)$. 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 $p,q \in L$. Then

$$\text{Tr}_f \vdash A (p) S (q) \text{ iff there is a proof outline of } S \text{ such that for each sub-statement } R \text{ of } S, \text{ the following verification conditions are true in } J:$$

1. $p \Rightarrow \text{pre}(S) \Rightarrow \text{post}(S) \Rightarrow q$
2. $\text{pre}(\text{null}) \Rightarrow \text{post}(\text{null})$
3. $\text{pre}(R) \Rightarrow \text{post}(R), \text{if } R=x:=e$
4. $\text{pre}(R) \Rightarrow \text{pre}(R) \land B \Rightarrow \text{pre}(R) \land \text{pre}(R) \Rightarrow \text{post}(R) \quad (i=1..2)$
   - if $R=\text{if } b \text{ then } R_1 \text{ else } R_2 \text{ end if}$
5. $\text{pre}(R) \Rightarrow \text{pre}(R) \Rightarrow \text{post}(R) \Rightarrow \text{pre}(R) \land \neg B \Rightarrow \text{post}(R)$
   - if $R=\text{while } b \text{ do } R_1 \text{ end while}$
6. $\text{pre}(R) \land B \Rightarrow \text{pre}(R) \Rightarrow \text{post}(R) \Rightarrow \text{pre}(R) \land \neg B \Rightarrow \text{post}(R)$
   - if $R=\text{select } b;R_1 \text{ or } \ldots \text{ or } b_i;R_i \text{ end select}$
7. $\text{pre}(R) \Rightarrow \text{pre}(R) \Rightarrow \text{pre}(R) \Rightarrow \text{post}(R)$
   - if $R=R_1 \ldots R_n$
   - and such that $\text{TV}(\text{pre}(R))_{\text{not}}=\emptyset$ if $R=\text{call } T.a(\text{#x})$

Such a proof outline is called valid (w.r.t. $J$) for $(p)S(q)$, and $\text{VC}((p)S(q))$ 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,

$$S_1; \text{call } T.a(\text{#x});S_2 \text{ and accept } a(\text{#y}) \text{ do } S_2;S_3 \text{ end accept},$$

the following two formulae should be valid in $J$

$$\{\text{pre}(S_1) \land \text{pre}(S_2) \land \text{GI}\} \Rightarrow \text{post}(S_1[S_2]^\bot) \Rightarrow \text{pre}(S_2[S_1]^\bot) \land \text{GI}, \ldots$$

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 $\text{CC}((p_i)T_i(q_i))_{i=1..n}, \text{GI})$.

Basing our argument on lemma 1, we replace the axiom $A_1$ and the proof rules $R_1, R_2$,

---

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 \( VC((p_i T_i(q_i)), CC((p_i T_i(q_i)) i=1..n, GI) \)

\( \{p_1\ldots Ap_n\wedge GI\} \begin{array}{c} \text{begin} \\ \text{task} T_1 \ldots \text{task} T_n \\ \text{end} \{q_1\ldots \wedge q_n\wedge GI\} \end{array} \)

provided (1) no assertion in the proof outline of \( T_i \) contains variables subject to change in \( T_j \) (i\#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 [Apt81a], \( C' \) 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 \( C' \)

\[ J \vdash \{p\} S \{q\} \iff J \vdash \{p\} T(S) \{q\} \iff Tr_j \vdash_{C'} \{p\} T(S) \{q\}. \]

This concludes the preliminaries and we continue with proving

Soundness of the combined rule.

By the above observation, it suffices to prove:

\[ Tr_j \vdash_{C'} \{p_1\ldots Ap_n\wedge GI\} \{P_1\ldots \|P_n\} \{q_1\ldots \wedge q_n\wedge GI\} \quad (P_i \equiv T(T_i)), \quad (S2) \]

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 premises 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 premise 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:

\[ \{p_1\}<S_1(p) call T_j.a(\alpha/\pi)(q) S_1'\{q_1\} \]

\[ \{p\}<\text{accept } a(\pi/\nu) do(p_2)S_2\{q_2\} S_2'\{q_2\} end accept\{q_2\} \]

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

\[ \{p_1\}<R_1(p) P_j.a(\alpha/\pi)(q) R_1'\{q_1\} \]

\[ \{p\}<\alpha(\pi/\nu)-\{p_2\} R_2\{q_2\} R_2'\{q_2\} a(\nu/\pi) \{q_2\} \]

By way of example, we prove CSP\(^+\) cooperation of the first matching pair of bracketed sections:

\[ \{p_1\wedge \alpha/\pi GI\} R_1 R_1' P_j.a(\alpha/\pi) \equiv a(\nu/\pi) \equiv R_2 \{p_1\wedge \alpha/\pi GI\}. \]

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 premises:

\[ H(p_1\wedge \alpha/\pi GI; R_1' \{r\}, H(r) P_j.a(\alpha/\pi) \equiv a(\nu/\pi) \equiv s, H(s) R_2 \{p_1\wedge \alpha/\pi GI\}, \quad (S3) \]

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 \( \vdash (p_1^1 \land \neg A^1 G I) R_1^1 ; R_2^2 ; [\cdot] (p \land \neg q_2^2 [\cdot] \land A^2 G I) \) to hold. Hence, defining \( r \) to be the intermediate assertion between \( R_1^1 \) and \( R_2^2 ; [\cdot] \), takes care of the first premiss in (S3) (this assertion exists because we have a formal proof). As \( FV(r) \cap \{U \cup V \} = \emptyset \), we can take \( \sigma = U = \emptyset, V = \emptyset \) 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. (S1)). This implies that

\[ \vdash (r) \ R_2^2 ; [\cdot] (p \land q_2^2 [\cdot] \land A^2 G I) \iff \vdash (r) \ U = \emptyset; \ V = \emptyset; \ R_1^1 ; \ U = \emptyset; \ V = \emptyset; \ R_2^2 ; [\cdot] (p \land q_2^2 [\cdot] \land A^2 G I), \]

since \( FV(r,p,q_2^2 [\cdot],G I) \cap \{U \cup V \} = \emptyset \). And hence \( \vdash (r) \ U = \emptyset; \ V = \emptyset; \ R_2^2 ; [\cdot] (p \land q_2^2 [\cdot] \land A^2 G I). \)

Finally, using completeness of \( C' \), applying the preservation axiom to \( \{U = \emptyset\} \) and noticing that \( \vdash q_2^2 [\emptyset \cup U] U = \emptyset \Rightarrow q_2^2 \), we get \( \vdash (s) \ R_2^2 ; [\cdot] (p \land q_2^2 [\cdot] \land A^2 G I), \) 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

\( (p \land q_2^2 [\cdot] \land A^2 G I) P_1 \ \& \ R_1^1 ; [\cdot] (q_1^1 \land q_2^1 [\cdot] \land A^2 G I), \)

we run into a difficulty, because we need invariance of the value parameters, \( U \), 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 strengthened, 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, \( CC(p_1^1 P_1 \{q_1^1\} \ i = 1 \ldots n, G I) \). Hence the rule may be applied, thus proving (S3) 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 \( S \in \text{ADA-CS} \) and \( p, q \in E \)

\[ J \vdash (p) S (q) \iff \text{Tr}_J \vdash (p) S (q). \]

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

\[ \text{Tr}_J \vdash C (p) T(S) (q) \iff \text{Tr}_J \vdash (p) S (q). \]

Let \( S \) begin task \( T_1 \) \ldots task \( T_n \) end 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

\[ \vdash (p_1 \land \ldots \land p_n \land G I) [T(T_1') \| \ldots \| T(T_n') \}] (q_1 \land \ldots \land q_n \land G I), \]

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 \( (p_1 \land T(T_1')(q_1) \), simply by defining for each component statement \( S \) of \( T_i \), \( \text{pre}(S) \subseteq \text{pre}(T(S)) \) and \( \text{post}(S) \subseteq \text{post}(T(S)) \). Validity of these proofs is obvious; ADA-CS cooperation remains to be shown. I.e., for each communication
pair $S_1;\text{call } T_j.a(\overline{e},\overline{x});S_1^u$, accept $a(\overline{u}\#\overline{v})$ do $S_2';\triangleright S_<S_2^u$ and accept

the premisses of the A formation rule must be shown to hold. This is in fact straight-
forward, using the following lemma which states some general conditions under which
variable-substitution preserves provability.

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

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

(ii) the variables in $\overline{u}$ 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) $T_j \vdash A(p) S(q) \rightarrow T_j \vdash A(p[S[\cdot]]) S[q[\cdot]] (\cdot) = (\overline{e}/\overline{u}, \overline{x}/\overline{v})$

(2) $T_j \vdash C(p) T(S(q)) \rightarrow T_j \vdash C(p[S[\cdot]]) T[q[\cdot]]$

**CSP** cooperation of the first pair of bracketed sections in the translations of the above communication pair, yields $(T(S_1)=R_1)$:

$\vdash \{p_1\land p_2\land \cdot \land GI\} R_1^* (p), \vdash (p) \vdash p_j.a (\overline{e}, \overline{x}) \parallel a(?r) (\overline{u}, \overline{v}) (q), \vdash (q) R_2^* \{p_1\land p_2\land \cdot \land GI\}$,

where $p=\overline{u}=\overline{e}=\overline{x}$, and the other assertions are taken from the appropriate outlines.

Furthermore, we may assume $\text{FV}(p_j^*) \cap \overline{x} = \emptyset$. This is a consequence of the fact that only for-

mulae of the form $\{r\} a_1 \parallel a_2 p_j.a ?(\overline{u}, \overline{v}) (\overline{u}\overline{x}=\overline{v})$ in which $\text{FV}(r) \cap \overline{x} = \emptyset$ 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:

$\vdash A \vdash \{p_1\land p_2\land \cdot \land GI\} S_1;S_2^u \rightarrow S_1;S_2^u \rightarrow \{p_1\land p_2\land \cdot \land GI\},$

the first premiss of the A-formation rule.

The second premiss, $T_A \vdash \{p_2'\} S[q_2'] (q_2'=\text{pre}(R_2'))$, is clear, as $S$ is a sequential state-

ment. Finally, CSP cooperation of the second pair of bracketed sections yields

$\vdash \{p_1\land q_2\land \cdot \land GI\} R_2^* (\overline{p}), \vdash (\overline{p}; \overline{x}=\overline{v}) R_1^* (q_1\land q_2\land GI),$

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

$\vdash A \vdash \{p_1\land q_2\land \cdot \land GI\} R_2^*[\cdot] (\overline{p}[\cdot]), \vdash A \vdash (\overline{p}[\overline{e}/\overline{u}]\overline{x}=\overline{v}) S_1^u (q_1\land q_2\land GI).$

Because $\vdash (\overline{p}[\overline{e}/\overline{u}]\overline{x}=\overline{v}) \rightarrow \overline{p}[\overline{e}/\overline{u}]\overline{x}=\overline{v}$ and $\text{FV}(\overline{e}/\overline{u}, S_1^u, q_1, q_2, \cdot \land GI) \cap \overline{u} = \emptyset$, this implies

$\vdash \{p_1\land q_2\land \cdot \land GI\} R_2^*[\cdot];R_1^* (q_1\land q_2\land GI),$

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

$\vdash A \vdash \{p_1\land \ldots \land p_n\land GI\} \begin{task} T_1 \ldots \text{task } T_n \end{task} \text{end } \{q_1\land \ldots \land q_n\land GI\}.$

Finally, application of the A consequence, substitution and AV rule allows the conclu-

sion $\vdash A(p)S(q)$, 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 con-

structed a partial correctness proof system and have proven that this completely axiom-
aizes 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 intro-
duce them. We believe not. The restrictions were introduced so as to allow a very sim-
pie 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 sim-
ilar 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, e.g., real-time con-
structs 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.

Processes. to appear.

TOPLAS 3-4 p.431-484, 1981.


Technical report, Dept. of Computer Science, University of Manchester, 1981.


