Relative completeness of a Hoare-calculus for while-programs

Kurt Sieber

A 80/01
February 1980

Fachbereich 10 -
Angewandte Mathematik
und Informatik
Universität des Saarlandes
6600 Saarbrücken
West Germany

Abstract:

In several papers,e.g. [COOK] or [APT] the problems of correctness and completeness of Hoare calculi have been studied. The purpose of this paper is to present a simple approach to this subject by restricting the attention to a very small class of programs, the so-called while-programs.

1. Preliminaries

As Hoare logic is an extension of first order predicate logic, a few definitions and notations of mathematical logic are shortly recalled:

A first order predicate language $\mathcal{L}$ is built up from a *basis* $B = (\underline{F}, \underline{P}, \underline{V})$, where

- $\underline{F}$ is a set of *function symbols*; with each element of $\underline{F}$ is associated an integer $k \geq 0$, called its *arity*
- $\underline{P}$ is a set of *predicate symbols*; with each element of $\underline{P}$ is associated an integer $h \geq 0$, called its *arity*
- $\underline{V}$ is an infinite set of *variables.*
- $\underline{F}$, $\underline{P}$ and $\underline{V}$ are disjoint

The *terms* and *formulas* (*well-formed formulas, wff's*) of the language $\mathcal{L}$ are constructed in the well-known manner.

An *interpretation* J for $\mathcal{L}$ consists of

- a nonempty set $\underline{D}$ (the *domain* of J)
- a function J_0, which assigns
  to each k-ary function symbol $f \in \underline{F}$ a function
  $J_0(f)\colon \underline{D}^k \to \underline{D}$,
  to each h-ary predicate symbol $p \in \underline{P}$ a predicate
  $J_0(p)\colon \underline{D}^h \to \{\underline{\text{true}}, \underline{\text{false}}\}$

A *state* σ (for $\mathcal{L}$ and J) is a function $\sigma : \underline{V} \to \underline{D}$.
$\underline{\Sigma}$ will denote the set of all states ($\mathcal{L}$ and J will always be known from the context). For $\sigma \in \underline{\Sigma}$, $x \in \underline{V}$ and $d \in \underline{D}$ we denote by $\sigma[x/d]$ the state σ' with

$$\begin{aligned} \sigma'(x) &= d \\ \sigma'(y) &= \sigma(y) \quad \text{for} \quad y \neq x \end{aligned}$$

Let J be an interpretation and σ a state.
Then it is well-known, how to define

$J(t,\sigma) \in \underline{D}$ for a term t of $\mathcal{L}$,
$J(p,\sigma) \in \{\underline{true}, \underline{false}\}$ for a wff $p \in \mathcal{L}$.

Instead of $J(p,\sigma) = \underline{true}$ we also write $\models_{J,\sigma} p$.
p is called *valid in* J (or: J-*valid*, notation: $\models_J p$), if $\models_{J,\sigma} p$ for all $\sigma \in \underline{\Sigma}$.
p is called *valid* (notation: $\models P$), if $\models_J p$ for all interpretations J of $\mathcal{L}$.

Let T be a subset of $\mathcal{L}$.
q is called a *valid consequence of* T (notation: $T \models q$), if $\models_J q$ holds, whenever $\models_J p$ holds for all $p \in T$.
T is called an *axiom system for* J, if

- every $p \in T$ is J-valid
- every J-valid formula $q \in \mathcal{L}$ is a valid consequence of T.

p_x^t denotes the formula obtained by substituting the term t for every free occurrence of x (being understood that variables of p must possibly be renamed before substitution).
The substitution lemma then states that

$$J(p_x^t,\sigma) = J(p,\sigma[x/J(t,\sigma)])$$

The definition and the lemma may be generalized in the usual way for simultaneous substitution (notation: $p_{x_1,\ldots,x_k}^{t_1,\ldots,t_k}$)

2. A Hoare logic for while programs

2.1 Syntax

Let $\mathcal{L}_E$ and $\mathcal{L}_A$ be two first order predicate languages with $\mathcal{L}_E \subseteq \mathcal{L}_A$; $\mathcal{L}_E$ is called the *expression language*, $\mathcal{L}_A$ the *assertion language*.

The set $\mathcal{S}$ of *while programs* (for $\mathcal{L}_E$) is defined as the least set for which:

i) For each variable x and term t of $\mathcal{L}_E$: $x := t \in \mathcal{S}$

ii) If $S_1, S_2 \in \mathcal{S}$, then: $S_1; S_2 \in \mathcal{S}$

iii) If $S_1, S_2 \in \mathcal{S}$ and e is a quantifier free formula of $\mathcal{L}_E$, then : $\underline{\text{if}}$ e $\underline{\text{then}}$ S_1 $\underline{\text{else}}$ S_2 $\underline{\text{fi}}$ $\in \mathcal{S}$

iv) If $S \in \mathcal{S}$ and e is a quantifier free formula of $\mathcal{L}_E$, then : $\underline{\text{while}}$ e $\underline{\text{do}}$ S $\underline{\text{od}}$ $\in \mathcal{S}$

The language $\mathcal{H}$ of *Hoare logic* (for $\mathcal{L}_A$ and $\mathcal{S}$) is now defined as the set

$$\mathcal{H} = \mathcal{L}_A \cup \{\{p\}S\{q\} \mid p,q \in \mathcal{L}_A, S \in \mathcal{S}\}$$

The elements of $\mathcal{H}$ are called *Hoare-formulas*.

Note that the language $\mathcal{L}_E$ is used in the definition of $\mathcal{S}$ and the language $\mathcal{L}_A$ in the definition of $\mathcal{H}$. The reason why $\mathcal{L}_E \subset \mathcal{L}_A$ is allowed, will become clear in the sequel.

2.2 Semantics

Intuitively the meaning of a Hoare formula $\{p\}\ S\ \{q\}$ is:

"If p holds before the execution of S and
S terminates,
then q holds after the execution of S".

(partial correctness of S with respect to p and q).

In order to formulate this more precisely, it is necessary to define "what the program S does", i.e. the semantics for the programming language $\mathcal{S}$ has to be defined clearly.

2.2.1 An operational semantics for $\mathcal{S}$

Let J be an interpretation for $\mathcal{L}_E$.

The relation $\Rightarrow$ on $(\mathcal{S} \cup \{\varepsilon\}) \times \underline{\Sigma}$ (which of course depends on J) is defined as follows:

i) $(x := t;\ R,\sigma) \Rightarrow (R,\sigma[x/J(t,\sigma)])$

ii) $(\underline{\text{if}}\ e\ \underline{\text{then}}\ S_1\ \underline{\text{else}}\ S_2\ \underline{\text{fi}};\ R,\sigma)$

$$\Rightarrow \begin{cases} (S_1;R,\sigma) & \text{if } J(e,\sigma) = \underline{\text{true}} \\ (S_2;R,\sigma) & \text{if } J(e,\sigma) = \underline{\text{false}} \end{cases}$$

iii) $(\underline{\text{while}}\ e\ \underline{\text{do}}\ S\ \underline{\text{od}};\ R,\sigma)$

$$\Rightarrow \begin{cases} (R,\sigma) & \text{if } J(e,\sigma) = \underline{\text{false}} \\ S;\ \underline{\text{while}}\ e\ \underline{\text{do}}\ S\ \underline{\text{od}};\ R,\sigma) & \text{if } J(e,\sigma) = \underline{\text{true}} \end{cases}$$

(where $R \in \mathcal{S} \cup \{\varepsilon\}$, $\sigma \in \underline{\Sigma}$ and e,x,t,S,S_1,S_2 as in section 2.1).

$\overset{*}{\Rightarrow}$ is defined to be the reflexive, transitive closure of $\Rightarrow$.
The while program $S \in \mathcal{S}$ is said to *terminate* for σ, if there is a state $\sigma' \in \underline{\Sigma}$, such that:

$$(S,\sigma) \overset{*}{\Rightarrow} (\varepsilon,\sigma')$$

(i.e. there is a finite "computation sequence"
$(S,\sigma) = (S_1,\sigma_1) \Rightarrow (S_2,\sigma_2) \Rightarrow \ldots \Rightarrow (S_k,\sigma_k) = (\varepsilon,\sigma')$)

Note that the relation $\Rightarrow$ is even a partial function, which is defined for any (R,σ) provided $R \neq \varepsilon$. Therefore, if S terminates for ε, the state σ', such that $(S,\sigma) \overset{*}{\Rightarrow} (\varepsilon,\sigma')$, is uniquely determined (in other words: the introduced semantics is "deterministic").

2.2.2 Interpretation of Hoare formulas

Let $\mathcal{H}$ be the language of Hoare logic for $\mathcal{L}_A$ and $\mathcal{S}$. Let J be an interpretation for $\mathcal{L}_A$ and $\sigma \in \underline{\Sigma}$ a state for $\mathcal{L}_A$ and J. As $\mathcal{L}_E \subseteq \mathcal{L}_A$, J may be considered as an interpretation for $\mathcal{L}_E$ and σ as a state for $\mathcal{L}_E$ and J, and the definitions of section 2.2.1 may be applied.

Now, for every $h \in \mathcal{H}$ - quite similar to mathematical logic - an element $J(h,\sigma) \in \{\underline{true},\underline{false}\}$ is defined by :

i) $h \in \mathcal{L}_A$:

Then $J(h,\sigma)$ is the usual value from mathematical logic.

ii) h is of the form {p} S {q} :

$$J(h,\sigma) = \text{true} \underset{\text{def.}}{\Longleftrightarrow} \begin{cases} \text{If } J(p,\sigma) = \underline{\text{true}} \text{ and } (S,\sigma) \overset{*}{\Rightarrow} (\varepsilon,\sigma'), \\ \text{then } J(q,\sigma') = \underline{\text{true}} \end{cases}$$

$$\Longleftrightarrow \begin{cases} J(p,\sigma) = \underline{\text{false}} \\ \text{or: } S \text{ does not terminate for } \sigma \\ \text{or: } (S,\sigma) \overset{*}{\Rightarrow} (\varepsilon,\sigma') \text{ and } J(q,\sigma') = \underline{\text{true}} \end{cases}$$

Instead of $J(h,\sigma) = \underline{\text{true}}$, one again writes: $\models_{J,\sigma} h$.

h is called *valid in* J ($\models_J h$), if $\models_{J,\sigma} h$ for every $\sigma \in \underline{\Sigma}$.

Hence $\models_J \{p\}\ S\ \{q\}$ means:

"For every $\sigma \in \Sigma$: If $J(p,\sigma) = \underline{\text{true}}$ and $(S,\sigma) \overset{*}{\Rightarrow} (\varepsilon,\sigma')$ then $J(q,\sigma') = \underline{\text{true}}$".

This is now the formal definition of "partial correctness" needed in the sequel.

Finally, h is called *valid* ($\models$ h), if $\models_J$ h for every interpretation J for $\mathcal{L}_A$.

2.3 Examples

a) Let $\mathcal{L}_A = \mathcal{L}_E = \mathcal{L}_N$ be the language of Peano arithmetic and J the standard interpretation.

Consider the formula h : $\{x \geq 0\}$ x := 2*x $\{x = 2\}$

Then

$$J(h,\sigma) = \underline{\text{true}} \iff \begin{cases} \sigma(x) < 0 \\ \text{or:} \quad \sigma(x) = 1 \end{cases}$$

b) With the same notations as in a) we have :

$$\models_J \{x \geq 0\}\ x := 2*x\ \{x \geq 0\} ,$$

but of course this may be wrong, if J is <u>not</u> the standard interpretation.

c) The formula {<u>true</u>} <u>while</u> <u>true</u> <u>do</u> x := x+1 <u>od</u> {<u>false</u>} is <u>valid</u>, because the program does not terminate for any state σ, independently of the interpretation J.

3. A Hoare calculus for while-programs

3.1 The axioms and rules

Let $\mathcal{L}_E$, $\mathcal{L}_A$, $\mathcal{S}$ and $\mathcal{H}$ be as above. The Hoare calculus for $\mathcal{H}$ consists of one axiom scheme and four rules of inference : (see e.g. [APT])

(A1) $\{p_x^t\}\ x := t\ \{p\}$ for every variable x and term t of $\mathcal{L}_E$ and every $p \in \mathcal{L}_A$

(R1) $$\frac{\{p\}S_1\{q\},\{q\}S_2\{r\}}{\{p\}S_1;\ S_2\{r\}}$$ for $S_1, S_2 \in \mathcal{S}$, $p,q,r \in \mathcal{L}_A$

(R2) $$\frac{\{p\wedge e\}S_1\{q\},\{p\wedge\neg e\}S_2\{q\}}{\{p\}\ \underline{\text{if}}\ e\ \underline{\text{then}}\ S_1\ \underline{\text{else}}\ S_2\ \underline{\text{fi}}\ \{q\}}$$ for every quantifier free formula $e \in \mathcal{L}_E$, $p,q \in \mathcal{L}_A$, $S_1,S_2 \in \mathcal{S}$

(R3) $$\frac{\{p\wedge\neg e\}S\{p\}}{\{p\}\ \underline{\text{while}}\ e\ \underline{\text{do}}\ S\ \underline{\text{od}}\ \{p\wedge\neg e\}}$$ for every quantifier free formula $e \in \mathcal{L}_E$, $p \in \mathcal{L}_A$, $S \in \mathcal{S}$

(R4) $$\frac{p\supset q,\{q\}S\{r\},\ r\supset s}{\{q\}S\{s\}}$$ for $p,q,r,s \in \mathcal{L}_A$, $S \in \mathcal{S}$

As this calculus can be applied to any arbitrarily given languages $\mathcal{L}_E$ and $\mathcal{L}_A$, one may really speak of a "calculus for while programs". In order to deduce reasonable formulas for a given $\mathcal{L}_E$, $\mathcal{L}_A$ and interpreation J for $\mathcal{L}_A$, it must of course be augmented by :

a) The axioms and inference rules of the predicate calculus.
b) An axiom system T for J.

This leads to the following

Definition:

Let $\mathcal{H}$ be the language of Hoare logic for $\mathcal{L}_A$ and $\mathcal{S}$, and let T be a subset of $\mathcal{L}_A$.

Then we write

$$\vdash_T h$$

for $h \in \mathcal{H}$,

if h can be derived from :

- instances of (A1)
- formulas of T
- the axioms of the predicate calculus

by finitely many applications of :

- the rules (R1) - (R4)
- the inference rules of the predicate calculus

3.2 Correctness of the calculus

What do we mean by saying that the calculus for $\mathcal{H}$ is correct ?

As we are only interested in the axioms and inference rules of the Hoare calculus, we have to abstract from those of the predicate calculus and the axiom system T.

Theorem 1: (Correctness of the Hoare calculus)

Let $\mathcal{H}$ be the language of Hoare logic for $\mathcal{L}_A$ and $\mathcal{S}$ and let J be an arbitrary interpretation for $\mathcal{L}_A$. If T is a set of J-valid formulas of $\mathcal{L}_A$, then for every formula $h \in \mathcal{H}$:

$$\vdash_T h \quad \text{implies} \quad \models_J h$$

Proof:

In order to prove this theorem, one shows :

- every instance of (A1) is a valid Hoare formula
- for each of the rules (R1) - (R4) :

  if all the premises of the rule are valid in some interpretation J of $\mathcal{L}_A$, then the conclusion is valid in J

A detailed proof is left to the reader.

$\square$

3.3 The problem of completeness

In section 3.2 we have seen, that our axiom system yields only J-valid Hoare formulas, if the formulas of the underlying set T are J-valid.

The question is now, whether one can deduce all J-valid Hoare formulas by using an appropiate set T of J-valid formulas. The following theorem states that, in general, there is no recursively enumerable set T with this property.

Theorem 2: (Incompleteness of the Hoare calculus)

Let $\mathcal{L}_A = \mathcal{L}_E = \mathcal{L}_N$ be the language of Peano arithmetic and J_N its standard interpretation.
Then there is no recursively enumberable set T of J_N-valid formulas, such that for every $h \in \mathcal{H}$:

$$\models_{J_N} h \qquad \text{implies} \qquad \vdash_T h$$

(i.e. by theorem 1 : $\models_{J_N} h \quad \text{iff} \quad \vdash_T h$)

Proof: Consider the formulas of the form {true} S {false}

a) Let T be a recursively enumerable set of J_N-valid formulas. Then the set of axioms and inference rules of our proof system (i.e. the Hoare calculus together with the predicate calculus and the set T) is recursively enumerable. This implies that the set of formulas, which can be deduced from this system, i.e. $\{h \in \mathcal{H} \mid \vdash_T h\}$ is recursively enumerable, and finally $\{S \in \mathcal{S} \mid \vdash_T \{\underline{true}\}\ S\ \{\underline{false}\}\}$ is recursively enumerable (as it is possible to decide whether h has the form {true} S {false}).

b) $\{S \mid \models_{J_N} \{\underline{true}\}\ S\ \{\underline{false}\}\}$ is the set of all $S \in \mathcal{S}$, which do not terminate for any $\sigma \in \underline{\Sigma}$. This set is not recursively enumerable, because one can simulate Turing machines by programs of $\mathcal{S}$, and the set of Turing machines which do not halt for any input, is not recursively enumerable. (Note that we need the interpretation J_N for simulating Turing machines.)

From a) and b) (and theorem 1) we get :

There exists a program $S \in \mathcal{S}$ for which

$$\models_{J_N} \{\underline{\text{true}}\}\ S\ \{\underline{\text{false}}\} \quad \text{but not} \quad \vdash_T \{\underline{\text{true}}\}\ S\ \{\underline{\text{false}}\}$$

□

<u>Another proof:</u>

It is a well-known (but non-trivial) result of first order predicate logic, that there exists no recursively enumerable axiom system T for J_N. As $\mathcal{L}_N$ is a subset of $\mathcal{H}$ (and formulas of $\mathcal{L}_N$ can not be deduced with the aid of (A1), (R1) - (R4)), the theorem is obvious.

□

The result of theorem 2 (or even more the second proof) suggests us to start from a - not necessarily recursively enumerable - axiom system T for every interpretation J of $\mathcal{L}_A$, e.g. let T be the set of all J-valid formulas of $\mathcal{L}_A$. Of course the existence of a non recursively enumerable axiom system T for J is of no help for practical purposes, but it enables us to study the completeness of the Hoare calculus while making abstraction from incompleteness features caused by the set T.

Unfortunately, there is still another reason which can lead to incompleteness of the Hoare calculus: It is possible, that the assertion language $\mathcal{L}_A$ is not "powerful enough". An example may be found in [WAND]. As the study of such examples is rather difficult, we only try to explain by a small example, what "powerful enough" means, in order to give a motivation for the definitions of section 3.4 :

Let $\mathcal{L}_A = \mathcal{L}_E = \mathcal{L}_N$.

Then the following Hoare formula is clearly valid in J_N :

$\{z > 0\}$ y := 1; x := z;

<u>while</u> x > 0 <u>do</u> y := y*x; x := x-1 <u>od</u> ;

<u>while</u> x < z <u>do</u> x := x+1; y := y$\dot{-}$x <u>od</u> $\{y = 1\}$

Both assertions, "z > 0" and "y = 1" are wff's of $\mathcal{L}_A$.
In order to deduce the above formula one needs an assertion p, for which

$\models_{J_N}$ $\{z > 0\}$ y := 1; x := z; <u>while</u> x > 0 <u>do</u> y := y*x; x := x - 1 <u>od</u>$\{p\}$

and

$\models_{J_N}$ $\{p\}$ <u>while</u> x < z <u>do</u> x := x+1; y := y$\dot{-}$x <u>od</u> $\{y = 1\}$,

e.g. let p be "$z > 0 \wedge x = 0 \wedge y = z!$".

But there is no factorial symbol in $\mathcal{L}_N$, hence one must find a formula in $\mathcal{L}_N$, which is equivalent to p. We shall prove in section 4, that such a formula exists, i.e. that $\mathcal{L}_A$ is "powerful enough" in our example. But, as mentioned above, there are other cases (for other languages $\mathcal{L}_A$ and $\mathcal{L}_E$) where such formulas do not exist.

3.4 Expressiveness

As indicated in 3.3, the language $\mathcal{L}_A$ must be powerful enough to express certain formulas in order to have a chance to deduce every J-valid Hoare formula. This notation of "powerful enough" will now be defined precisely.

<u>Definitions:</u>

Let $\mathcal{L}_A$, $\mathcal{L}_E$, $\mathcal{S}$ and J be as usual.

a) For every $q \in \mathcal{L}_A$ and $S \in \mathcal{S}$ the Hoare formula

$\{$<u>true</u>$\}$ S $\{q\}$

is called the *weakest precondition* of S and q.

b) $\mathcal{L}_A$ is called J-*expressive* with respect to $\mathcal{L}_E$ (or $\mathcal{S}$) if for every $q \in \mathcal{L}_A$ and $S \in \mathcal{S}$ there exists a formula $p \in \mathcal{L}_A$, which is equivalent to $\{\underline{true}\}\ S\ \{q\}$ (i.e. $J(p,\sigma) = J(\{\underline{true}\}\ S\ \{q\},\sigma)$ for every $\sigma \in \underline{\Sigma}$).

We shall use the name "weakest precondition" also for the formula p. Of course p is only determined up to equivalence, but this will suffice for our purpose. (A way out is to choose the first such p in an enumeration of the language $\mathcal{L}_A$).

Note that

$$J(\{\underline{true}\}\ S\ \{q\},\sigma) = \underline{true} \Longleftrightarrow$$

$$(\text{if } (S,\sigma) \overset{*}{\Rightarrow} (\varepsilon,\sigma') \text{ then } J(q,\sigma') = \underline{true}\ )$$

As a straightforward consequence we get the following lemma, which explains the name "weakest precondition".

<u>Lemma 1:</u>

a) For every $r \in \mathcal{L}_E$: $\models_J \{r\}\ S\ \{q\}$ iff for every $\sigma \in \underline{\Sigma}$

$$J(r,\sigma) = \underline{true} \quad \text{implies} \quad J(\{\underline{true}\}\ S\ \{q\},\sigma) = \underline{true}$$

b) If $p \in \mathcal{L}_A$ is the weakest precondition of S and q, then for every $r \in \mathcal{L}_E$:

$$\models_J \{r\}\ S\ \{q\} \quad \text{iff} \quad \models_J r \supset q\ ,$$

i.e. p is the "weakest" formula for which $\models_J \{p\}\ S\ \{q\}$

<u>Proof:</u>

a) "$\Rightarrow$": If $J(r,\sigma) = \underline{true}$

then $\models_J \{r\}\ S\ \{q\}$ implies: If $(S,\sigma) \overset{*}{\Rightarrow} (\varepsilon,\sigma')$ then

$J(q,\sigma') = \underline{true}$

i.e.: $J(\{\underline{true}\}\ S\ \{q\},\sigma) = \underline{true}$

"$\Leftarrow$": If $J(r,\sigma) = \underline{true}$, then $J(\{\underline{true}\}\ S\ \{q\},\sigma) = \underline{true}$

i.e.:

if $(S,\sigma) \overset{*}{\Rightarrow} (\varepsilon,\sigma')$ then $J(q,\sigma) = \underline{true}$

But this is exactly the definition for $\models_J \{r\}\ S\ \{q\}$.

b) is an immediate consequence of a). □

Examples:

Let $\mathcal{L}_A = \mathcal{L}_E = \mathcal{L}_N$ and $J = J_N$

a) The weakest precondition of $y := 4$ and $y = 5$
is: $\underline{false}$,

because $J(\{\underline{true}\}\ y := 4\ \{y = 5\},\sigma) = \underline{true}$

iff $J(y = 5, \sigma[y/4]) = \underline{true}$

and this doesn't hold for any σ.

Hence $\models_J \{r\}\ y := 4\ \{y = 5\}$,

iff r is equivalent to $\underline{false}$.

b) The weakest precondition of

$\underline{while}\ y \neq 5\ \underline{do}\ y := y+1\ \underline{od}$ and $\underline{false}$

is: $y > 5$,
because $J(\{\underline{true}\}\ \underline{while}\ \ldots\ \underline{od}\ \{\underline{false}\},\sigma) = \underline{true}$

iff the while-statement does not terminate for σ,
i.e. iff $\sigma(y) > 5$.

The weakest precondition of the same program and
$y = 5$ is : $\underline{true}$,

because $J(\{\underline{true}\}\ \underline{while}\ \ldots\ \underline{od}\ \{y = 5\},\sigma) = \underline{true}$ for every σ.

Note that the definition of weakest precondition slightly differs from Dijkstra's definition (see e.g. [DIJ], section 3). In his sense, the weakest precondition of S and q is true for a state σ iff S terminates for σ with $(S,\sigma) \overset{*}{\Rightarrow} (\varepsilon,\sigma')$ and $J(q,\sigma') = \underline{true}$.

3.5 Relative completeness of the calculus

The notion of relative completeness was first introduced in [COOK].

Theorem 3:

(Relative completeness of the Hoare Calculus or : Completeness in the sense of Cook)

Let $\mathcal{L}_A, \mathcal{L}_E, \mathcal{S}$ and J be as usual.
If

- T is an axiom system for J and
- $\mathcal{L}_A$ is J-expressive with respect to $\mathcal{L}_E$

then

for every Hoare formula h :

$$\models_J h \quad \text{implies} \quad \vdash_T h$$

$$(\text{i.e.: } \models_J h \ \text{ iff } \ \vdash_T h)$$

Proof:

The theorem is trivial for $h \in \mathcal{L}_A$.

For h of the form {p} S {q} the property is proved by induction on the structure of S.

a) Assume $\models_J \{p\}\ x := t\ \{q\}$

In order to prove

$$\vdash_T \{p\}\ x := t\ \{q\}$$

it is sufficient to prove

$$\vdash_T \ p \supset q_x^t \qquad \text{(see (A1) and(R4))}$$

or, because T is complete

$$\models_J \ p \supset q_x^t \qquad (1)$$

As for every $\sigma \in \underline{\Sigma}$: $(x := t,\sigma) \Rightarrow (\varepsilon,\sigma[x/J(t,\sigma)])$,
$J(p,\sigma) = \underline{true}$ implies $J(q,\sigma[x/J(t,\sigma)]) = \underline{true}$
and by the substitution lemma of the predicate calculus one gets

$$J(q_x^t,\sigma) = J(q,\sigma[x/J(t,\sigma)]) = \underline{true}$$

Hence (1) is proved.

b) Assume $\models_J \{p\}\ S_1;\ S_2\ \{q\}$ (1)

Let $r \in \mathcal{L}_A$ be the weakest precondition of S_2 and q; note that r exists because of expressiveness.
Then, by lemma 1b), $\models_J \{r\}\ S_2\ \{q\}$ holds, and the induction hypothesis yields: $\vdash_T \{r\}\ S_2\ \{q\}$

Now it is sufficient to prove : $\models_J \{p\}\ S_1\ \{r\}$ (2) ;
in fact from (2) one gets $\vdash_T \{p\}\ S_1\ \{r\}$ by the induction hypothesis and an application of (R1) leads to $\vdash_T \{p\}\ S_1;\ S_2\ \{q\}$.

In order to prove (2) assume

$$J(p,\sigma) = \underline{true} \quad \text{and} \quad (S_1,\sigma) \overset{*}{\Rightarrow} (\varepsilon,\sigma').$$

Then one has to show $J(r,\sigma') = \underline{true}$,
i.e., because r is the weakest precondition of S_2 and q:

$$J(\{\underline{true}\}\ S_2\ \{q\},\sigma') = \underline{true} \qquad (3)$$

To prove (3) assume that $(S_2,\sigma') \overset{*}{\Rightarrow} (\varepsilon,\sigma'')$.
This leads to $(S_1;S_2,\sigma) \overset{*}{\Rightarrow} (S_2,\sigma') \overset{*}{\Rightarrow} (\varepsilon,\sigma'')$,
hence $J(q,\sigma'') = \underline{true}$ by (1).
Thus, (3) is proved and the proof of (2) is completed.

c) Assume that S if of the form <u>if</u> e <u>then</u> S_1 <u>else</u> S_2 <u>fi</u> and $\models_J \{p\}\ S\ \{q\}$ holds. (1)

With the usual arguments (see b)) it is sufficient to prove

$$\models_J \{p \wedge e\}\ S_1\ \{q\} \qquad (\ 2\ )$$

and

$$\models_J \{p \wedge \neg e\}\ S_2\ \{q\} \qquad (\ 3\ )$$

Proof of (2):

Assume that $J(p \wedge e,\sigma) = \underline{true}$ and $(S_1,\sigma) \overset{*}{\Rightarrow} (\varepsilon,\sigma')$.

Then, as $J(e,\sigma) = \underline{true}$,

$$(S,\sigma) \overset{*}{\Rightarrow} (S_1,\sigma) \overset{*}{\Rightarrow} (\varepsilon,\sigma')$$

and, as $J(p,\sigma) = \underline{true}$, (1) yields :

$$J(q,\sigma') = \underline{true}$$

(3) is proved in a similar way.

d) Assume that S is of the form <u>while</u> e <u>do</u> S' <u>od</u> and $\models_J \{p\}\ S\ \{q\}$ holds. (1)

Let $r \in \mathcal{L}_A$ be the weakest precondition of S and q.

Then, by lemma 1b), $\models_J p \supset r$ holds and it is sufficient to prove

$$\models_J \{r \wedge e\}\ S'\ \{r\} \qquad (\ 2\ )$$

and

$$\models_J (r \wedge \neg e) \supset q \qquad (\ 3\ )$$

because the induction hypothesis and the rules (R3) and (R4) then lead to $\vdash_T \{p\}\ S\ \{q\}$.

To prove (2) and (3), first note that

$$\models_J \{r\}\ S\ \{q\} \qquad (\ 4\ )$$

because of lemma 1b).

Proof of (2) :

Assume that $J(r \wedge e,\sigma)$ = true and $(S',\sigma) \overset{*}{\Rightarrow} (\varepsilon,\sigma')$.

One has to prove $J(r,\sigma')$ = true,

i.e. $J(\{\underline{\text{true}}\}\ S\ \{q\},\sigma')$ = true. (5)

Assume $(S,\sigma') \overset{*}{\Rightarrow} (\varepsilon,\sigma'')$.

Then, as $J(e,\sigma)$ = true,

$$(S,\sigma) \Rightarrow (S';S,\sigma) \overset{*}{\Rightarrow} (S,\sigma') \overset{*}{\Rightarrow} (\varepsilon,\sigma'')$$

and, as $J(r,\sigma)$ = true, (4) yields

$$J(q,\sigma'') = \underline{\text{true}}.$$

Hence (5) is proved, which completes the proof of (2).

Proof of (3) :

Assume that $J(r \wedge \neg e,\sigma)$ = true

Then, as $J(e,\sigma)$ = false, $(S,\sigma) \Rightarrow (\varepsilon,\sigma)$

and by (4) : $J(q,\sigma)$ = true

$\square$

5. Expressiveness of certain languages

In section 4 we have defined "expressiveness" in order to discuss completeness problems of the Hoare calculus. But up to now we have not indicated whether there exist languages $\mathcal{L}_A, \mathcal{L}_E$ and an interpretation J, such that $\mathcal{L}_A$ is J-expressive with respect to $\mathcal{L}_E$. We now want to examine the problem of expressiveness for a few important cases :

Definitions:

a) For a first order predicate formula p and a while-program S, we denote by var(p) and var(S) the set of variables which occur in p and S respectively.

b) For a Hoare formula h of the form {p} S {q} we define

$\text{var}(h) = \text{var}(p) \cup \text{var}(S) \cup \text{var}(q)$.

Putting $\text{var}(h) = \{x_1,\ldots,x_k\}$, one easily proves by induction on the structure of S, that $J(h,\sigma)$ depends on $\sigma(x_1),\ldots,\sigma(x_k)$ only, or, more precisely, that $J(h,\sigma) = J(h,\sigma')$ whenever $\sigma(x_i) = \sigma'(x_i)$ for all i, $1 \leq i \leq k$.
This leads to the following

Definition:

Let $\mathcal{L}_E, \mathcal{L}_A, \mathcal{S}, \mathcal{H}$, J be as above and let $\underline{D}$ be the domain of J. Then for every $h \in \mathcal{H}$ the relation $\underline{R} \subseteq \underline{D}^k$:

$$\underline{R} = \{(d_1,\ldots,d_k) \mid J(h,\sigma) = \underline{\text{true}} \text{ whenever } \sigma(x_1) = d_1,\ldots,\sigma(x_k) = d_k$$

is called the *relation induced by* h.

By the last definition the problem of expressiveness has been reduced to the question, whether certain relations can be "expressed" by first order formulas.

Definition:

Let $\mathcal{L}$ be a first order predicate language and J an interpretation for $\mathcal{L}$ with domain $\underline{D}$. We say that $p \in \mathcal{L}$ *expresses* a relation $\underline{R} \subseteq \underline{D}^k$ (in the variables $x_1,\ldots,x_k$) iff
$J(p,\sigma) = \underline{\text{true}} \iff (\sigma(x_1),\ldots,\sigma(x_k)) \in \underline{R}$.

Lemma 2:

Let $\mathcal{L}_N$ and J_N be the language and the standard interpretation of Peano arithmetic respectively.

If $\underline{R} \subseteq N^k$ is recursively enumerable, then there exists a formula $p \in \mathcal{L}_N$, which expresses $\underline{R}$.

(For a proof see e.g. [MAL], p. 119).

Theorem 4:

$\mathcal{L}_N$ is J_N-expressive with respect to $\mathcal{L}_N$.

Proof:

Let $S \in \mathcal{S}$ be an arbitrary program and $q \in \mathcal{L}_N$ an arbitrary formula. By definition $\mathcal{L}_N$ is J_N-expressive if there exists a formula $r \in \mathcal{L}_N$ such that

$$J_N(\{\underline{\text{true}}\}\ S\ \{q\},\sigma) = J_N(r,\sigma) \qquad \text{for all } \sigma \qquad (1)$$

Let $var(\{\underline{\text{true}}\}\ S\ \{q\}) = \{x_1,\ldots,x_k\}$ and $\underline{R} \subseteq N^k$ the relation induced by $\{\text{true}\}\ S\ \{q\}$. If $r \in \mathcal{L}_N$ is a formula which expresses $\underline{R}$, then r satisfies (1).

Hence the theorem is proved if we succeed in finding such a formula r.

Let f be the partial function

$$f : \mathbb{N}^k \rightsquigarrow \mathbb{N}^k$$
$$(\sigma(x_1),\ldots,\sigma(x_k)) \mapsto (\sigma'(x_1),\ldots,\sigma'(x_k)) \text{ if } (S,\sigma) \overset{*}{\Rightarrow} (\varepsilon,\sigma')$$

(f is well-defined, as $var(S) \subseteq \{x_1,\ldots,x_k\}$)

Of course f is a recursive function, because it is computed by the program S. Hence, the domain of f :

$$dom(f) = \{(n_1,\ldots,n_k) \mid f(n_1,\ldots,n_k) \text{ is defined}\} \subseteq \mathbb{N}^k$$

and the graph of f :

$$graph(f) = \{(n_1,\ldots,n_k,m_1,\ldots,m_k) \mid (m_1,\ldots,m_k) = f(n_1,\ldots,n_k)\} \subseteq \mathbb{N}^2$$

are recursively enumerable sets.

By lemma 2 there is a formula $d \in \mathcal{L}_A$, such that

$$J(d,\sigma) = \underline{true} \iff (\sigma(x_1),\ldots,\sigma(x_k)) \in dom(f)$$

and a formula $g \in \mathcal{L}_A$, such that

$$J(g,\sigma) = \underline{true} \iff (\sigma(x_1),\ldots,\sigma(x_k),\sigma(y_1),\ldots,\sigma(y_k)) \in graph(f)$$

Now it is easily proved that the formula

$$\neg\, d \vee \exists y_1,\ldots,y_k.(g \wedge q^{y_1,\ldots,y_k}_{x_1,\ldots,x_k})$$

expresses $\underline{R}$. □

<u>Lemma 3:</u>

Let $\mathcal{L}_+$ be the language of Presburger arithmetic (i.e. $\mathcal{L}_N$ without a multiplication symbol) and J_+ its standard interpretation.

If $\underline{R} \subseteq \mathbb{N}^k$ can be expressed by a formula $p \in \mathcal{L}_+$, then $\underline{R}$ is recursive.

(For a proof, see e.g. [END], p. 188).

<u>Theorem 5:</u>

$\mathcal{L}_+$ is <u>not</u> J_+-expressive with respect to $\mathcal{L}_+$.

<u>Proof:</u>

Let $\underline{R} \subseteq \mathbb{N}$ be a set, which is recursively enumerable, but not recursive. Then the acceptor function f of $\underline{R}$, i.e.

$$\begin{aligned} f:\ & \mathbb{N} \rightsquigarrow \{1\} \\ & x \mapsto 1 \qquad \text{iff} \quad x \in \underline{R} \end{aligned}$$

is recursive.

Hence there exists a program S, which computes f, i.e. S terminates for σ, iff f is defined for $\sigma(x)$, with

$$(S,\sigma) \overset{*}{\Rightarrow} (\varepsilon,\sigma[x/f(\sigma(x))]) = (\varepsilon,\sigma[x/1]).$$

Note that it is possible to compute recursive functions by while programs for $\mathcal{L}_+$; the multiplication symbol is not necessary for this purpose.

For the weakest precondition of S and <u>false</u> we get :

$J(\{\underline{\text{true}}\}\ S\ \{\underline{\text{false}}\},\sigma) = \underline{\text{true}}$

$\Longleftrightarrow$ S does not terminate for

$\Longleftrightarrow$ f is undefined for $\sigma(x)$

$\Longleftrightarrow \sigma(x) \in \mathbb{N} - \underline{R}$

But, as $\underline{R}$ is not recursive, $\mathbb{N} - \underline{R}$ is not recursive, i.e. by lemma 3, there exists no formula in $\mathcal{L}_+$ which expresses the relation $\mathbb{N} - \underline{R}$. Hence there exists no formula in $\mathcal{L}_+$, which is equivalent to the weakest precondition of S and <u>false</u>. $\square$

Theorem 6:

Let $\mathcal{L}_A$, $\mathcal{L}_E$ and J be as usual.

If the domain $\underline{D}$ of J is finite, and if for each $d \in \underline{D}$ there exists a term t of $\mathcal{L}_A$ with $J(t,\sigma) = d$ for every σ, then $\mathcal{L}_A$ is J-expressive with respect to $\mathcal{L}_E$.

Proof:

Let $\underline{R} \subseteq \underline{D}^k$ be a relation. As $\underline{D}$ is finite, $\underline{R}$ is finite, e.g. $\underline{R} = \{(d_{11},\ldots,d_{1k}),\ldots,(d_{m1},\ldots,d_{mk})\}$.

Let t_{ij}, $1 \leq i \leq m$, $1 \leq j \leq k$, be terms for which $J(t_{ij},\sigma) = d_{ij}$ for every σ. Then the formula

$$(x_1 = t_{11} \wedge \ldots \wedge x_k = t_{1k}) \vee \ldots \vee (x_1 = t_{m1} \wedge \ldots \wedge x_k = t_{mk})$$

expresses the relation $\underline{R}$.

In particular there exists an equivalent formula in $\mathcal{L}_A$ for each weakest precondition.

□

Acknowledgement:

I am grateful to J. Loeckx for useful discussions during the preparation of this paper.

References:

[APT] Apt, K.R., "Ten Years of Hoare's Logic, a Survey", Internal Report, Erasmus University, Rotterdam, Apr. 1979

[COOK] Cook, S.A., "Soundness and Completeness of an Axiom System for Program Verification", SIAM Journal on Computing 7, 1 (Febr. 1978), pp. 70 - 90

[DIJ] Dijkstra, E.W., "A Discipline of Programming", Prentice Hall, 1976

[END] Enderton, H.B., "A Mathematical Introduction to Logic", Academic Press, 1972

[MAL] Malitz, J., "Introduction to Mathematical Logic", Springer Verlag, 1979

[WAND] Wand, M., "A New Incompleteness Result for Hoare's System", Journal of the ACM 25, 1 (Jan. 1978), pp. 168 - 175