10
LCF, A Logic for Computable Functions

LCF consists of a logic for cpo's and continuous functions, and of a corresponding calculus allowing in particular the formal verification of programs. It differs from the Hoare logic and calculus in the following points. First, the Hoare calculus was developed for a particular programming language, namely the while-programming language. LCF was developed for cpo's and is applicable to any programming language whose semantics is denotationally described. Second, it is possible to formulate and prove total correctness in LCF as well as partial correctness. Properties like those in Examples 9.4 and 9.5 are also provable. Third, LCF does not build on the predicate logic, but instead starts from scratch. The reason is that it is necessary to consider the ω-extensions of predicates, that is, those predicates with range $Bool_\omega$ and not the 'usual' predicates with range *Bool*. For a similar reason it was not possible in Section 3.3 to build the (operational) semantics of recursive programs on the predicate logic.

10.1 The Logic

Essentially terms in LCF are defined as λ-terms of a λ-notation with two distinguished function symbols interpreted as the if-then-else function and the least fixpoint operator respectively.

10.1.1 Syntax

A *basis* for LCF is a basis for the λ-notation (Section 5.1) which satisfies the following five conditions:

(1) There is a distinguished basis type *bool*.
(2) There are two function symbols TT and FF of type *bool*. (TT and FF stand for 'true' and 'false'.)
(3) There is a function symbol UU_τ (or UU for short) of type τ, for every type τ. (UU_τ stands for the 'undefined value' of type τ.)

J. Loeckx et al., *The Foundations of Program Verification*

(4) There is a function symbol *if-then-else*$_\tau$ (or *if-then-else* for short) of type $(bool, \tau, \tau \to \tau)$, for every type τ.
(5) There is a function symbol MM_τ (or MM for short) of type $((\tau \to \tau) \to \tau)$, for every type τ. (MM_τ stands for the least fixpoint operator of Theorem 4.32.)

An *LCF-term* (and its *type*) for a given basis is a λ-term (and its type) for this basis.

Formulas are constructed using the (logical) symbol '$\sqsubseteq$' which is intended to be interpreted as the partial order 'is less or equally defined than'. An *atomic LCF-formula* has the form

$$t_1 \sqsubseteq t_2$$

where t_1 and t_2 are LCF-terms of the same type. An *LCF-formula* has the form $\mathrm{P}_1, \ldots, \mathrm{P}_n$, where the $\mathrm{P}_i, i = 1, \ldots, n$, are all atomic LCF-formulas. The case that $n = 0$ is also allowed, so that the 'empty formula' is also a formula. Finally, an *LCF-sentence* has the form $\mathrm{P} \Rightarrow \mathrm{Q}$, where P and Q are LCF-formulas and '$\Rightarrow$' is one more logical symbol.

10.1.2 Semantics

An *interpretation* $\mathscr{I}$ of a basis for LCF is an interpretation according to Definition 5.2 which satisfies the following five conditions:

(1) The set D_{bool} is taken to be the set $Bool_\omega$ ($= \{\text{true}, \text{false}, \omega\}$).
(2) $\mathscr{I}_0(\mathrm{TT}) = \text{true}$ and $\mathscr{I}_0(\mathrm{FF}) = \text{false}$.
(3) For every type τ, $\mathscr{I}_0(\mathrm{UU}_\tau) = \bot_\tau$ (= the least element of D_τ). So in particular, $\mathscr{I}_0(\mathrm{UU}_{bool}) = \omega$.
(4) For every type τ, $\mathscr{I}_0(\textit{if-then-else}_\tau)$ is taken to be the usual ω-extension if-then-else: $Bool_\omega \times D_\tau \times D_\tau \to D_\tau$ defined by

$$\text{if-then-else}(b, d_1, d_2) = \begin{cases} d_1 & \text{if } b = \text{true} \\ d_2 & \text{if } b = \text{false} \\ \bot_\tau & \text{if } b = \omega. \end{cases}$$

(5) For every type τ, $\mathscr{I}_0(\mathrm{MM}_\tau)$ is taken to be the fixpoint operator $\mu\colon [D_\tau \to D_\tau] \to D_\tau$ as described in Theorem 4.32.

An interpretation fulfilling these requirements has the property that $\mathscr{I}_0(f) \in D_\tau$ for every function symbol f of type τ. In particular, that the if-then-else function is continuous for every type τ, i.e. that

$$\mathscr{I}_0(\textit{if-then-else}_\tau) \in [Bool_\omega \times D_\tau \times D_\tau \to D_\tau] = D_{(bool, \tau, \tau \to \tau)}$$

can be easily proved. (Note that the proof of the continuity of *if-then-else* in Section 4.2 applies to flat cpo's only). As for every function $f \in [D_\tau \to D_\tau]$, $\mu(f) \in D_\tau$ (by definition of μ) and as moreover the fixpoint operator μ is continuous (see Theorem 4.32),

$$\mathscr{I}_0(\mathrm{MM}_\tau) \in [[D_\tau \to D_\tau] \to D_\tau] = D_{((\tau \to \tau) \to \tau)}.$$

Armed now with an interpretation, the semantics of LCF-terms, LCF-formulas, and LCF-sentences can be given. As usual $\mathscr{I}$ will also denote the semantic functional.

The semantics of LCF-terms is defined as in Definition 5.3. Hence $\mathscr{I}$ maps an LCF-term t to a function $\mathscr{I}(t)\colon \Gamma \to D_\tau$, where Γ is the set of all assignments (for the basis and the interpretation under consideration).

The semantics of an atomic LCF-formula $t_1 \sqsubseteq t_2$ are defined—as usual—by an extension of the semantic functional $\mathscr{I}$:

$$\mathscr{I}(t_1 \sqsubseteq t_2)(\gamma) = \begin{cases} \text{true} & \text{if } \mathscr{I}(t_1)(\gamma) \sqsubseteq \mathscr{I}(t_2)(\gamma) \\ \text{false} & \text{otherwise,} \end{cases}$$

for all assignments $\gamma \in \Gamma$. The symbol '$\sqsubseteq$' on the right-hand side of this equation stands for the partial order of the appropriate cpo.

Similarly for an LCF-formula consisting of the atomic LCF-formulas $\mathrm{P}_1, \mathrm{P}_2, \ldots, \mathrm{P}_n$:

$$\mathscr{I}(\mathrm{P}_1, \mathrm{P}_2, \ldots, \mathrm{P}_n)(\gamma) = \begin{cases} \text{true} & \text{if } \mathscr{I}(\mathrm{P}_i)(\gamma) = \text{true, for every } i,\ i = 1, \ldots, n; \\ \text{false} & \text{otherwise,} \end{cases}$$

for all assignments $\gamma \in \Gamma$. In particular for $n = 0$, $\mathscr{I}(\ )(\gamma) = \text{true}$.

Finally, for an LCF-sentence $\mathrm{P} \Rightarrow \mathrm{Q}$ one defines

$$\mathscr{I}(\mathrm{P} \Rightarrow \mathrm{Q})(\gamma) = \begin{cases} \text{true} & \text{if } \mathscr{I}(\mathrm{P})(\gamma) = \text{true implies } \mathscr{I}(\mathrm{Q})(\gamma) = \text{true} \\ \text{false} & \text{otherwise,} \end{cases}$$

for all assignments $\gamma \in \Gamma$.

If $\mathscr{I}(\mathrm{P} \Rightarrow \mathrm{Q})(\gamma) = \text{true}$ for every interpretation $\mathscr{I}$ and every assignment γ, then $\mathrm{P} \Rightarrow \mathrm{Q}$ is said to be a *logically valid* LCF-sentence; if P is empty, then Q is said to be a *logically valid* LCF-formula.

Notice that for an LCF-term t of type τ (and an assignment γ) $\mathscr{I}(t)(\gamma) \in D_\tau$, so in the case that $\tau = bool$, $\mathscr{I}(t)(\gamma) \in \{\text{true}, \text{false}, \omega\}$. On the other hand, for an LCF-formula P, $\mathscr{I}(\mathrm{P})(\gamma) \in \{\text{true}, \text{false}\}$. So on the level of LCF-terms (of type *bool*) one works with a three-valued logic (namely true, false, and ω), while on the level of LCF-formulas one works with the usual two-valued logic. This reflects the fact that LCF-terms represent objects under discussion, in particular non-terminating programs, while LCF-formulas represent statements about these objects.

Now follow some examples of logically valid LCF-formulas and LCF-sentences.

EXAMPLE 10.1 Let s, t, and u be LCF-terms of type τ, and x a variable of type σ. The formulas

$$s \sqsubseteq s \quad \text{and} \quad \mathrm{UU}_\tau \sqsubseteq s$$

are logically valid due to the obvious properties of the partial order '$\sqsubseteq$'. The following are logically valid LCF-sentences:

$$s \sqsubseteq t, t \sqsubseteq u \Rightarrow s \sqsubseteq u$$
$$[\lambda x.s] \sqsubseteq \mathrm{UU}_{(\sigma \to \tau)} \Rightarrow s \sqsubseteq \mathrm{UU}_{\tau}$$
$$\mathrm{TT} \sqsubseteq \mathrm{UU}_{bool} \Rightarrow s \sqsubseteq \mathrm{UU}_{\tau}.$$

Notice that this last LCF-sentence trivially holds, since $\mathscr{I}(\mathrm{TT} \sqsubseteq \mathrm{UU}_{bool})(\gamma) =$ false for every interpretation $\mathscr{I}$ and every assignment γ. □

In comparison with predicate logic, LCF contains '$\sqsubseteq$' and '$\Rightarrow$' as the only logical symbols. Hence, in LCF the atomic formulas cannot be joined by any connectives such as '$\neg$' or '$\vee$', even if these are available as function symbols in the basis. One may very well want to write

$$\Rightarrow \neg(\mathrm{TT} \sqsubseteq \mathrm{UU}_{bool})$$

but this is not a legal LCF-sentence, because the function symbol '$\neg$' may be 'applied' on LCF-terms only, not on atomic formulas. A further difference with predicate logic is the lack of quantifiers. Additionally, LCF can make statements about continuous (that is, 'computable') functions only. On the other hand LCF is provided with function variables—something that first-order predicate logic is not. All these differences make it difficult to compare the expressive power of predicate logic and LCF.

In the treatment of the semantics of LCF the fact that the fixpoint operator is continuous was used for the first time in this book. The reason this is necessary is because LCF allows more 'complicated' expressions than those used above in the denotational semantics of Chapter 5. LCF allows in particular terms with 'nested' fixpoint operators. An example of such a term is

$$\ldots[\lambda \mathrm{F}.\ldots.\mathrm{MM}([\lambda \mathrm{G}.\ldots.\mathrm{F}\ldots])\ldots]\ldots$$

10.1.3 Syntactic Simplifications

In order to simplify the calculus to be described four simplifications are introduced.

Instead of

$$\textit{if-then-else}_{\tau}(e, t_1, t_2)$$

where e, t_1, t_2 are LCF-terms of type *bool*, τ, and τ respectively, one writes simply

$$(e \to t_1, t_2)$$

The second simplification is a more fundamental one. It consists in requiring $n = 1$ for each type

$$(\tau_1, \ldots, \tau_n \to \tau);$$

informally, this means that one considers only functions with one argument. Syntactically, this simplification has as a consequence that application leads to LCF-terms

$$u(t)$$

rather than

$$u(t_1, \ldots, t_n)$$

and λ-abstraction to

$$[\lambda x \,.\, t]$$

rather than

$$[\lambda x_1, \ldots, x_n \,.\, t]$$

(see Definition 5.1). Semantically this simplification does not constitute a restriction since by currying it is possible to replace any n-place continuous function by a 1-ary continuous function (see Section 4.2 and, in particular, Theorem 4.23).

The next two simplifications are again purely notational ones. An LCF-term

$$\mathrm{MM}_\tau([\lambda x \,.\, t])$$

with x a variable of type τ and t an LCF-term of type τ is written as

$$[\alpha x \,.\, t].$$

Note that this notational convention allows one to eliminate any occurrence of the function symbol MM_τ as any LCF-term u of type $(\tau \to \tau)$ may be written as

$$[\lambda x \,.\, u(x)]$$

where x is a variable of type τ not occurring free in u; in other words, $\mathrm{MM}_\tau(u)$ may be written as

$$[\alpha x \,.\, u(x)].$$

Finally,

$$t_1 \equiv t_2$$

is allowed as an abbreviation for the LCF-formula

$$t_1 \sqsubseteq t_2, t_2 \sqsubseteq t_1$$

where t_1, t_2 are LCF-terms of the same type. Note that, semantically, '$\equiv$' expresses the equality in D_τ, τ being the type of t_1 and t_2.

Notice the difference between '$\equiv$' and the function symbol '$=$' with type $(\tau, \tau \to \mathit{bool})$ or, more precisely, $(\tau \to (\tau \to \mathit{bool}))$. The interpretation of the LCF-term $t_1 = t_2$ leads to a value from $\{\text{true}, \text{false}, \omega\}$. On the other hand, $t_1 \equiv t_2$ is an LCF-formula and hence when interpreted leads to a value from $\{\text{true}, \text{false}\}$. By the way, when considered as a function, '$\equiv$' is not only not continuous, it is not monotonic, since '$\bot \equiv \bot$' has the value true, and '$\bot \equiv$ true' has the value false (see also the remark preceding Example 10.1).

10.2 The Calculus

The LCF-calculus is a calculus over (the set of all) LCF- sentences. Its goal is to derive the logically valid LCF-sentences and, in particular, the logically valid LCF-formulas. Notice the interest here is only in deriving sentences from the

empty set although the abstract notion of a calculus permits the derivation of LCF-sentences from an arbitrarily given set of LCF-sentences (see Section 2.3).

Before the calculus can be presented, the definition of substitution in λ-terms (hence also in LCF-terms) given in Section 5.1 (see Theorem 5.9) must first be generalized to LCF-formulas. The first step is to define the substitution for atomic LCF-formulas. Let x be a variable and t an LCF-term of the same type. Then t substituted for x in the atomic formula $t_1 \sqsubseteq t_2$ is written $(t_1 \sqsubseteq t_2)^t_x$ and is defined to stand for $(t_1)^t_x \sqsubseteq (t_2)^t_x$. Since t_1 and t_2 are LCF-terms, the substitution of t for x is already understood. For an LCF-formula which consists of the atomic formulas $P_1, \ldots, P_n$, $(P_1, \ldots, P_n)^t_x$ stands for the LCF-formula $(P_1)^t_x, \ldots, (P_n)^t_x$.

Furthermore the following convention will be used. If P and Q are LCF-formulas which consist of the atomic LCF-formulas $P_1, \ldots, P_n$ and $Q_1, \ldots, Q_m$ respectively, then P, Q stands for the LCF-formula

$$P_1, \ldots, P_n, Q_1, \ldots, Q_m.$$

This convention holds also for $n = 0$ or $m = 0$.

The eleven axiom schemes and seven inference rules of the LCF-calculus now follow. The axioms and rules are labelled with an identifier like INCL or FIXP. They are grouped in seven parts corresponding to their 'meaning'. In the axioms and rules P, Q, R and S stand for LCF-formulas, s, t and u stand for terms, f stands for a 1-ary function symbol, x stands for a variable and τ and σ are types. All LCF-terms and LCF-formulas are assumed to be correctly typed.

(i) '$\Rightarrow$'-Definition

$$P, Q \Rightarrow P \qquad \text{INCL1}$$

$$P, Q \Rightarrow Q \qquad \text{INCL2}$$

$$\frac{P \Rightarrow Q \quad Q \Rightarrow R}{P \Rightarrow R} \qquad \text{CUT}$$

$$\frac{P \Rightarrow Q \quad P \Rightarrow R}{P \Rightarrow Q, R} \qquad \text{CONJ}$$

(ii) '$\sqsubseteq$'-Definition

$$\frac{P \Rightarrow s \sqsubseteq t}{P \Rightarrow u(s) \sqsubseteq u(t)} \qquad \text{APPL}$$

$$\Rightarrow s \sqsubseteq s \qquad \text{REFL}$$

$$\frac{P \Rightarrow s \sqsubseteq t \quad P \Rightarrow t \sqsubseteq u}{P \Rightarrow s \sqsubseteq u} \qquad \text{TRANS}$$

(iii) 'UU'-Definition

$$\Rightarrow UU_\tau \sqsubseteq s \qquad \text{MIN1}$$

$$\Rightarrow UU_{(\tau \to \sigma)}(s) \sqsubseteq UU_\sigma \qquad \text{MIN2}$$

(iv) 'if-then-else'-Definition

$$\Rightarrow (\mathrm{TT} \rightarrow s, t) \equiv s \qquad \text{CONDT}$$

$$\Rightarrow (\mathrm{FF} \rightarrow s, t) \equiv t \qquad \text{CONDF}$$

$$\Rightarrow (\mathrm{UU}_{bool} \rightarrow s, t) \equiv \mathrm{UU}_{\tau} \qquad \text{CONDU}$$

(v) 'λ'-Definition

$$\frac{\mathrm{P} \Rightarrow s \sqsubseteq t}{\mathrm{P} \Rightarrow [\lambda x . s] \sqsubseteq [\lambda x . t]} \quad (x \text{ not free in P}) \qquad \text{ABSTR}$$

$$\Rightarrow [\lambda x . s](t) \equiv s_x^t \qquad \text{CONV}$$

$$\Rightarrow [\lambda x . f(x)] \equiv f \qquad \text{FUNC}$$

(vi) Case Distinction

$$\frac{\mathrm{P}, s \equiv \mathrm{TT} \Rightarrow \mathrm{Q} \quad \mathrm{P}, s \equiv \mathrm{FF} \Rightarrow \mathrm{Q} \quad \mathrm{P}, s \equiv \mathrm{UU}_{bool} \Rightarrow \mathrm{Q}}{\mathrm{P} \Rightarrow \mathrm{Q}} \qquad \text{CASES}$$

(vii) 'α'-Definition

$$\Rightarrow [\alpha x . s] \equiv s_x^{[\alpha x . s]} \qquad \text{FIXP}$$

$$\frac{\mathrm{P} \Rightarrow \mathrm{Q}_x^{\mathrm{UU}} \quad \mathrm{P}, \mathrm{Q} \Rightarrow \mathrm{Q}_x^t}{\mathrm{P} \Rightarrow \mathrm{Q}_x^{[\alpha x . t]}} \quad (x \text{ not free in P}) \qquad \text{INDUCT}$$

These axioms and rules capture formally what one would expect to be true. INCL1, INCL2, CUT, and CONJ express the semantics of LCF-sentences and LCF-formulas. In particular they imply that the order of occurrence of the atomic LCF-formulas in an LCF-formula is immaterial. APPL captures the monotonicity of an LCF-term u. REFL and TRANS express the reflexivity and transitivity of the partial order '$\sqsubseteq$'. MIN1 and MIN2 characterize the least element of each type. CONDT, CONDF, CONDU axiomatically define the function if-then-else. ABSTR is the familiar way of extending (by 'λ-abstraction') a cpo of a domain to a cpo of functions to that domain. CONV and FUNC make sure that functions obtained by λ-abstraction behave correctly. CASES corresponds to the proof method based on case distinction. FIXP expresses that the least fixpoint $[\alpha x . s]$ of the function $[\lambda x . s]$ is a fixpoint of this function, that is,

$$[\alpha x . s] \equiv [\lambda x . s]([\alpha x . s]).$$

INDUCT is the fixpoint induction principle; the proof of Theorem 10.7 sheds some light on this rule.

Before the soundness of the calculus is proved some simple deductions in the LCF-calculus are given. It appears that, as in the predicate calculus, the formal deduction of 'obviously' correct theorems is not necessarily trivial. In these example deductions P, Q and R stand for LCF-formulas, s, t and u stand for LCF-terms, and x and y for variables. All LCF-terms and LCF-formulas are

assumed to be correctly typed. The first three examples establish derived rules (in the sense of Section 8.1).

EXAMPLE 10.2 It is to prove that the LCF-sentence $P \Rightarrow R$ is derivable from the LCF-sentence $P \Rightarrow Q, R, S$ or, more formally, that

$$\frac{P \Rightarrow Q, R, S}{P \Rightarrow R} \quad \text{is a derived rule.}$$

Here is the deduction proving this statement:

$P \Rightarrow Q, R, S$	assumption	(1)
$Q, R, S \Rightarrow Q, R$	INCL1	(2)
$P \Rightarrow Q, R$	CUT on (1), (2)	(3)
$Q, R \Rightarrow R$	INCL2	(4)
$P \Rightarrow R$	CUT on (3), (4)	□

EXAMPLE 10.3 That

$$\frac{R \Rightarrow P}{Q, R, S \Rightarrow P}$$

is a derived rule is proved by the following deduction:

$R \Rightarrow P$	assumption	(1)
$R, S \Rightarrow R$	INCL1	(2)
$R, S \Rightarrow P$	CUT on (2), (1)	(3)
$Q, R, S \Rightarrow R, S$	INCL2	(4)
$Q, R, S \Rightarrow P$	CUT on (4), (3)	□

EXAMPLE 10.4

$$\frac{P \Rightarrow s \sqsubseteq t \quad Q \Rightarrow t \sqsubseteq u}{P, Q \Rightarrow s \sqsubseteq u} \quad \text{is a derived rule:}$$

$P \Rightarrow s \sqsubseteq t$	assumption	(1)
$Q \Rightarrow t \sqsubseteq u$	assumption	(2)
$P, Q \Rightarrow s \sqsubseteq t$	Example 10.3 on (1)	(3)
$P, Q \Rightarrow t \sqsubseteq u$	Example 10.3 on (2)	(4)
$P, Q \Rightarrow s \sqsubseteq u$	TRANS on (3), (4)	□

The previous three examples established a derived rule; the next two each establish a 'theorem' or, equivalently, a 'derived axiom'.

EXAMPLE 10.5 If the variable x does not occur free in the LCF-terms s and t, the LCF-sentence

$$s(t) \sqsubseteq t \Rightarrow [\alpha x \,.\, s(x)] \sqsubseteq t$$

is derivable (from the empty set). This theorem expresses Park's Theorem (Corollary 4.31).

Here is the deduction:

$\Rightarrow \mathrm{UU} \sqsubseteq t$	MIN1	(1)
$s(t) \sqsubseteq t \Rightarrow \mathrm{UU} \sqsubseteq t$	Example 10.3 on (1)	(2)
$x \sqsubseteq t \Rightarrow x \sqsubseteq t$	INCL1	(3)
$x \sqsubseteq t \Rightarrow s(x) \sqsubseteq s(t)$	APPL on (3)	(4)
$s(t) \sqsubseteq t, x \sqsubseteq t \Rightarrow s(x) \sqsubseteq s(t)$	Example 10.3 on (4)	(5)
$s(t) \sqsubseteq t, x \sqsubseteq t \Rightarrow s(t) \sqsubseteq t$	INCL1	(6)
$s(t) \sqsubseteq t, x \sqsubseteq t \Rightarrow s(x) \sqsubseteq t$	TRANS on (5), (6)	(7)
$s(t) \sqsubseteq t \Rightarrow [\alpha x . s(x)] \sqsubseteq t$	INDUCT on (2), (7) (remember that x does not occur free in s and t)	□

EXAMPLE 10.6 If the variable y does not occur free in the LCF-term s, then the LCF-sentence

$$\Rightarrow [\alpha x . s] \sqsubseteq [\alpha y . s_x^y]$$

is derivable. One can also derive the LCF-sentence with '$\equiv$' instead of '$\sqsubseteq$' in which case the theorem expresses the obvious semantical property, that the name of the variable bound by α is irrelevant.

$\Rightarrow \mathrm{UU} \sqsubseteq [\alpha y . s_x^y]$	MIN1	(1)
$x \sqsubseteq [\alpha y . s_x^y] \Rightarrow x \sqsubseteq [\alpha y . s_x^y]$	INCL1	(2)
$x \sqsubseteq [\alpha y . s_x^y] \Rightarrow [\lambda x . s](x) \sqsubseteq [\lambda x . s]([\alpha y . s_x^y])$	APPL on (2)	(3)
$\Rightarrow [\lambda x . s](x) \equiv s_x^x \equiv s$	CONV	(4)
$\Rightarrow s \sqsubseteq [\lambda x . s](x)$	Example 10.2 on (4)	(5)
$x \sqsubseteq [\alpha y . s_x^y] \Rightarrow s \sqsubseteq [\lambda x . s]([\alpha y . s_x^y])$	Example 10.4 on (5), (3)	(6)
$\Rightarrow [\lambda x . s]([\alpha y . s_x^y]) \equiv s_x^{[\alpha y . s_x^y]} \equiv (s_x^y)_y^{[\alpha y . s_x^y]}$	CONV since y is not free in s	(7)
$\Rightarrow [\lambda x . s]([\alpha y . s_x^y]) \sqsubseteq (s_x^y)_y^{[\alpha y . s_x^y]}$	Example 10.2 on (7)	(8)
$\Rightarrow [\alpha y . s_x^y] \equiv (s_x^y)_y^{[\alpha y . s_x^y]}$	FIXP	(9)
$\Rightarrow (s_x^y)_y^{[\alpha y . s_x^y]} \sqsubseteq [\alpha y . s_x^y]$	Example 10.2 on (9)	(10)
$\Rightarrow [\lambda x . s]([\alpha y . s_x^y]) \sqsubseteq [\alpha y . s_x^y]$	TRANS on (8), (10)	(11)
$x \sqsubseteq [\alpha y . s_x^y] \Rightarrow s \sqsubseteq [\alpha y . s_x^y]$	Example 10.4 on (6), (11)	(12)
$\Rightarrow [\alpha x . s] \sqsubseteq [\alpha y . s_x^y]$	INDUCT on (1), (12)	□

10.3 Soundness and Completeness Problems

THEOREM 10.7 (Soundness of the LCF-calculus) An LCF-sentence $P \Rightarrow Q$ which is derivable in the calculus from the empty set, is logically valid.

Proof. The proof proceeds by showing that every axiom is a logically valid LCF-

sentence, and that for every inference rule if the premises are logically valid, then the conclusion is too.

By way of example the axiom INCL1 and the inference rule INDUCT will be treated. The remaining cases are left to the reader.

(1) To show that the axiom P, Q ⇒ P is logically valid, one must show for every interpretation $\mathscr{I}$ and every assignment γ that

$$\mathscr{I}(\mathrm{P}, \mathrm{Q})(\gamma) = \text{true} \quad \text{implies} \quad \mathscr{I}(\mathrm{P})(\gamma) = \text{true}.$$

Recalling the notational convention about P, Q it is clear that $\mathscr{I}(\mathrm{P}, \mathrm{Q})(\gamma)$ is true iff $\mathscr{I}(\mathrm{P})(\gamma) = \text{true}$ and $\mathscr{I}(\mathrm{Q})(\gamma) = \text{true}$, so the proof is immediate.

(2) Now the proof for INDUCT. By assumption the premises are logically valid. So for all interpretations $\mathscr{I}$ and all assignments γ

$$\mathscr{I}(\mathrm{P})(\gamma) = \text{true} \quad \text{implies} \quad \mathscr{I}(\mathrm{Q}_x^{UU})(\gamma) = \text{true} \tag{1}$$

and

$$\mathscr{I}(\mathrm{P})(\gamma) = \text{true and } \mathscr{I}(\mathrm{Q})(\gamma) = \text{true} \quad \text{imply} \quad \mathscr{I}(\mathrm{Q}_x^t)(\gamma) = \text{true}. \tag{2}$$

It must be proved that for all interpretations $\mathscr{I}$ and assignments γ

$$\mathscr{I}(\mathrm{P})(\gamma) = \text{true} \quad \text{implies} \quad \mathscr{I}(\mathrm{Q}_x^{[\alpha x.t]})(\gamma) = \text{true}.$$

Let $\mathscr{I}$ be an interpretation and γ an assignment such that $\mathscr{I}(\mathrm{P})(\gamma) = \text{true}$. Define the predicate Φ: $D_\tau \to \mathit{Bool}$ by

$$\Phi(d) = \text{true} \qquad \text{iff } \mathscr{I}(\mathrm{Q})(\gamma[x/d]) = \text{true},$$

where τ is the type of the variable x. It is now proved that

$$\Phi(\mu f) = \text{true} \tag{3}$$

where f is defined to be

$$f = \mathscr{I}([\lambda x\,.\,t])(\gamma).$$

Notice that Q has the form

$$t_1 \sqsubseteq t_1', \ldots, t_n \sqsubseteq t_n'$$

where t_i and t_i' are LCF-terms, $i = 1, \ldots, n$ and $n \geq 0$. Therefore for all $d \in D_\tau$

$$\begin{aligned}\Phi(d) = \text{true} \qquad \text{iff } & \mathscr{I}(t_1)(\gamma[x/d]) \sqsubseteq \mathscr{I}(t_1')(\gamma[x/d]) \text{ and } \ldots \\ & \text{and } \mathscr{I}(t_n)(\gamma[x/d]) \sqsubseteq \mathscr{I}(t_n')(\gamma[x/d]);\end{aligned}$$

so Φ is an admissible predicate and the property (3) may be proved by fixpoint induction.

The first step is to show that $\Phi(\perp) = \text{true}$:

$$\begin{aligned}\mathscr{I}(\mathrm{Q}_x^{UU})(\gamma) &= \mathscr{I}(\mathrm{Q})(\gamma[x/\perp]) \qquad \text{(by a straightforward generalization to LCF-formulas of the Substitution Theorem 5.9)} \\ &= \Phi(\perp).\end{aligned}$$

By assumption (1), $\Phi(\perp) = \text{true}$.

Now comes the harder part. Let d be an arbitrary element of D_τ. It must be

shown that

$$\Phi(d) = \text{true} \quad \text{implies} \quad \Phi(f(d)) = \text{true}.$$

Suppose $\Phi(d) = \text{true}$, then $\mathscr{J}(\mathrm{Q})(\gamma[x/d]) = \text{true}$. Moreover

$$\begin{aligned}\mathscr{J}(\mathrm{P})(\gamma[x/d]) &= \mathscr{J}(\mathrm{P})(\gamma) \qquad \text{(by the Coincidence Theorem 5.7 since } x \text{ is not free in P)}\\ &= \text{true} \qquad \text{(by choice of } \mathscr{J} \text{ and } \gamma).\end{aligned}$$

This means the hypothesis of assumption (2) is true (with $\gamma[x/d]$ for γ), so one may conclude $\mathscr{J}(\mathrm{Q}^{t}_{x})(\gamma[x/d]) = \text{true}$. Now

$$\begin{aligned}\mathscr{J}(\mathrm{Q}^{t}_{x})(\gamma[x/d]) &= \mathscr{J}(\mathrm{Q})((\gamma[x/d])[x/\mathscr{J}(t)(\gamma[x/d])]) \qquad \text{(by the Substitution Theorem)}\\ &= \mathscr{J}(\mathrm{Q})(\gamma[x/\mathscr{J}(t)(\gamma[x/d])])\\ &= \Phi(\mathscr{J}(t)(\gamma[x/d])) \qquad \text{(by the definition of } \Phi)\\ &= \Phi(\mathscr{J}([\lambda x\,.\,t])(\gamma)(d)) \qquad \text{(by the semantics of the } \lambda\text{-notation)}\\ &= \Phi(f(d)).\end{aligned}$$

So $\Phi(f(d)) = \text{true}$, too. This completes the proof of (3). Now

$$\begin{aligned}\mu f &= \mu.\mathscr{J}([\lambda x\,.\,t])(\gamma) \qquad \text{(by definition of } f)\\ &= \mathscr{J}(\mathrm{MM}([\lambda x\,.\,t]))(\gamma) \qquad \text{(by definition of } \mathscr{J}_0(\mathrm{MM}))\\ &= \mathscr{J}([\alpha x\,.\,t])(\gamma) \qquad \text{(by definition of the notation } \alpha)\end{aligned}$$

Hence, by (3)

$$\Phi(\mathscr{J}([\alpha x\,.\,t])(\gamma)) = \text{true}$$

which means that

$$\mathscr{J}(\mathrm{Q})(\gamma[x/\mathscr{J}([\alpha x\,.\,t])(\gamma)]) = \text{true}$$

and so, by the Substitution Theorem,

$$\mathscr{J}(\mathrm{Q}^{[\alpha x.t]}_{x})(\gamma) = \text{true}. \qquad \square$$

The LCF-calculus is not complete; moreover, there can be no complete calculus for LCF. A proof of this assertion will not be given here, instead a sketch of it is given which is based on the following facts (which will not be proved here either):

(i) The Peano axioms can be formulated as LCF-formulas. (See also Section 10.4.)

(ii) The principle of induction over the natural numbers can be formulated as an LCF-formula. (This fact is directly related to the availability of function variables in LCF; see Newey (1973).)

(iii) The LCF-terms which may be constructed from the function symbols of Peano arithmetic represent exactly the computable functions in the standard model of Peano arithmetic. In other words, starting from the functions (contained in the basis) of Peano arithmetic, the operations of composition, λ-abstraction, and least fixpoint operation lead to exactly the computable functions.

From (i) and (ii) it is possible to deduce that Peano arithmetic or, more

precisely, the Peano axioms *and* the principle of induction can be expressed as an LCF-formula, say PA. Now for this LCF-formula there can only be one model—up to isomorphism—namely, the natural numbers model (see Section 2.4). So consider the LCF-sentences of the form

$$\mathrm{PA} \Rightarrow t_1 \equiv t_2,$$

where t_1 and t_2 are LCF-terms which are interpreted (in the standard interpretation of Peano arithmetic) as functions over the natural numbers. From computation theory it is known that this set of LCF-sentences cannot be recursively enumerable. Therefore, there can be no complete calculus for LCF.

10.4 Application to Program Verification

To prove something about a program it is necessary to be able to use properties about the underlying data or, more precisely, about the domain of the interpretation. As in Hoare calculus these properties must be introduced as formulas. These formulas may be the formulas of the theory $Th(\mathscr{I})$ where $\mathscr{I}$ is the interpretation under consideration. Alternatively, these formulas may be axioms for this theory, for instance the Peano axioms in the case of programs dealing with natural numbers. Clearly, in the LCF calculus these axioms must first be rewritten as LCF-formulas. In this rewriting process at least two problems arise.

(1) There is no universal quantifier in LCF. This problem can be circumvented by using λ-abstraction. For example, the Peano axiom (A1) in Section 2.4

$$\forall x \,.\, x + 0 = x$$

becomes

$$[\lambda x \,.\, x + 0] \equiv [\lambda x \,.\, x]$$

in LCF.

(2) One must take care of the undefined value in LCF. For example, the Peano axiom (M1)

$$\forall x \,.\, x * 0 = 0$$

cannot be written

$$[\lambda x \,.\, x * 0] \equiv [\lambda x \,.\, 0]$$

if the intended interpretation of '$*$' is to be the strict ω-extension of multiplication. One solution consists in testing for ω with the help of the (strict ω-extension of the) equality:

$$[\lambda x \,.\, ((x = x) \rightarrow x * 0, 0)] \equiv [\lambda x \,.\, ((x = x) \rightarrow 0, 0)].$$

This requires the additional axiom

$$\mathrm{UU} * 0 \equiv \mathrm{UU}$$

for the undefined value. Actually this only shifts the problem, as now the properties of the equality '$=$' must be axiomatized. This can be done by the following four LCF-formulas:

$$[\lambda x.((x = x) \to x, \mathrm{UU})] \equiv [\lambda x.x]$$
$$[\lambda x.[\lambda y.((x = y) \to x, \mathrm{UU})]] \equiv [\lambda x.[\lambda y.((x = y) \to y, \mathrm{UU})]]$$
$$[\lambda x.[\lambda y.((x = x) \to ((y = y) \to \mathrm{TT}, \mathrm{UU}), \mathrm{UU})]] \equiv [\lambda x.[\lambda y.((x = y) \to \mathrm{TT}, \mathrm{TT})]]$$
$$(\mathrm{UU} = \mathrm{UU}) \equiv \mathrm{UU}.$$

That these four axioms are correct is easy to check, but that they are enough to fully define equality is not trivial. For a complete treatment of the axiomatization of Peano arithmetic in LCF the interested reader is referred to Newey (1973).

The problems involved with the properties of the underlying data will now be ignored, and the 'real' problems of proving properties of programs will be addressed. As should be apparent LCF is readily adapted to proofs about the recursive programs. The next example concerns such a program. Subsequently, the case of the while-programming language will be discussed.

EXAMPLE 10.8 Let S be again the recursive program

$$\mathrm{F}(x, y) \Leftarrow \textit{if } x = y \textit{ then } 1 \textit{ else } (y + 1) * \mathrm{F}(x, y + 1) \textit{ fi}.$$

In Example 9.1 it was proved that $\mu\Phi(\mathrm{S}) \sqsubseteq [\lambda x, y.x!/y!]$. In LCF this is formulated as

$$\mathrm{PA} \Rightarrow [\alpha f.[\lambda x.[\lambda y.((x = y) \to 1, (y + 1) * f(x)(y + 1))]]] \sqsubseteq [\lambda x.[\lambda y.x!/y!]], \tag{1}$$

where PA is the LCF-formula that expresses all the necessary properties of arithmetic.

A deduction in the LCF-calculus in the style of Example 10.2 through Example 10.6 would be very long, not to mention obscure. So a sketch of the proof is given.

LCF-sentence (1) is derived by INDUCT from

$$\mathrm{PA} \Rightarrow \mathrm{UU} \sqsubseteq [\lambda x.[\lambda y.x!/y!]] \tag{2}$$

and

$$\mathrm{PA}, f \sqsubseteq [\lambda x.[\lambda y.x!/y!]] \Rightarrow [\lambda x.[\lambda y.((x = y) \to 1, (y + 1) * f(x)(y + 1))]] \sqsubseteq [\lambda x.[\lambda y.x!/y!]] \tag{3}$$

LCF-sentence (2) is obtained by MIN1. LCF-sentence (3) is derived by ABSTR from:

$$\mathrm{PA}, f \sqsubseteq [\lambda x.[\lambda y.x!/y!]] \Rightarrow ((x = y) \to 1, (y + 1) * f(x)(y + 1)) \sqsubseteq (x!/y!) \tag{4}$$

LCF-sentence (4) is derived by CASES from:

$$\mathrm{PA}, f \sqsubseteq [\lambda x.[\lambda y.x!/y!]], (x = y) \equiv \mathrm{UU} \Rightarrow ((x = y) \to 1, (y + 1) * f(x)(y + 1)) \sqsubseteq (x!/y!) \tag{5a}$$

$$\mathrm{PA}, f \sqsubseteq [\ldots], (x = y) \equiv \mathrm{TT} \Rightarrow ((x = y) \to 1, \ldots) \sqsubseteq (x!/y!) \tag{5b}$$

$$\mathrm{PA}, f \sqsubseteq [\ldots], (x = y) \equiv \mathrm{FF} \Rightarrow ((x = y) \to 1, \ldots) \sqsubseteq (x!/y!) \tag{5c}$$

LCF-sentence (5a) is derived by (among others) TRANS and MIN1 from:

$$(x = y) \equiv \mathrm{UU} \Rightarrow ((x = y) \rightarrow 1, \ldots) \sqsubseteq \mathrm{UU} \tag{6a}$$

LCF-sentences (5b) and (5c) are derived from similar LCF-sentences (6b) and (6c). To the proof of (6a):

$(x = y) \equiv \mathrm{UU} \Rightarrow (x = y) \sqsubseteq \mathrm{UU}$	by INCL1	(7)
$(x = y) \equiv \mathrm{UU} \Rightarrow [\lambda b.(b \rightarrow 1, \ldots)](x = y)$ $\sqsubseteq [\lambda b.(b \rightarrow 1, \ldots)](\mathrm{UU})$	by APPL	(8)
$\Rightarrow ((x = y) \rightarrow 1, \ldots) \sqsubseteq [\lambda b.(b \rightarrow 1, \ldots)](x = y)$	by CONV	(9)
$\Rightarrow [\lambda b.(b \rightarrow 1, \ldots)](\mathrm{UU}) \sqsubseteq (\mathrm{UU} \rightarrow 1, \ldots)$	by CONV	(10)
$\Rightarrow (\mathrm{UU} \rightarrow 1, \ldots) \sqsubseteq \mathrm{UU}$	by CONDU	(11)

LCF-sentence (6a) is then derived by several applications of TRANS applied to (8) through (11) (among other things).

The LCF-sentences (6b) and (6c) are derived similarly. In the case of (6c) it is advantageous to use the derived rule of inference

$$\frac{\mathrm{P} \Rightarrow f \sqsubseteq g}{\mathrm{P} \Rightarrow f(s) \sqsubseteq g(s)}$$

proved in Exercise 10.2–2(i) together with Exercise 10.2–1(i)—when using the 'induction hypothesis' $f \sqsubseteq [\lambda x.[\lambda y.x!/y!]]$. □

Examples from the while-programming language can in principle be handled the same way. But as is clear from Example 9.21, the function $\mathscr{M}^{\omega}$ appears in the LCF-sentence to be proved. In order to be able to use the properties of $\mathscr{M}^{\omega}$ in the proof, they must be available as LCF-formulas. Now the domain of $\mathscr{M}^{\omega}$ (considered as a function) is L_2^B, the set of while-programs for the given basis B. These programs are introduced as an additional basis type. Furthermore, some basic predicates and functions on L_2^B must also be axiomatized (just like '$<$' and '$+$' for the natural numbers in Peano arithmetic). One such predicate distinguishes between an assignment statement and a conditional statement and one such function extracts the quantifier-free formula from a conditional statement (in view of its evaluation). For a complete treatment of the subject the interested reader is referred to Milner (1979).

EXERCISES

10.2–1 Let Q and R be LCF-formulas.

(i) Prove that the LCF-sentence

$$\mathrm{Q} \Rightarrow \mathrm{R}$$

can be derived in the LCF-calculus (from the empty set), iff

$$\frac{\mathrm{P} \Rightarrow \mathrm{Q}}{\mathrm{P} \Rightarrow \mathrm{R}} \quad \text{for all LCF-formulas P}$$

is a derived inference rule.

(ii) Give an example where

$$\frac{\Rightarrow Q}{\Rightarrow R}$$

is a derived inference rule, but

$$Q \Rightarrow R$$

is not a logically valid LCF-sentence.

10.2–2 Let s, t, u be LCF-terms (of the appropriate types).

(i) Prove that the LCF-sentence

$$s \sqsubseteq t \Rightarrow s(u) \sqsubseteq t(u)$$

is derivable in the LCF-calculus (from the empty set). (*Hint:* The LCF-sentence expresses the 'monotonicity of application to the argument u'. Try to express this 'application operator' as an LCF-term.)

(ii) Prove that the LCF-sentence

$$[\lambda x\,.\,s] \sqsubseteq [\lambda x\,.\,t] \Rightarrow [\alpha x\,.\,s] \sqsubseteq [\alpha x\,.\,t]$$

is derivable in the LCF-calculus (from the empty set).

10.3–1 Prove the soundness of the inference rules 'APPL' and 'ABSTR' of the LCF-calculus. Show that 'ABSTR' would not be sound, if x was allowed to occur free in P.

10.4–1 In Section 10.4 four 'axioms' for the strict equality were presented.

(i) Prove that these axioms hold for the strict equality.
(ii) Prove that these axioms are sufficient to characterize the strict equality. (*Hint:* First try to find out what the first two and the fourth formula say about the interpretation of '$=$' and use this information to 'interpret' the third one.)
(iii) Prove that the four axioms are independent of each other in the sense that none of them follows from the others. (*Hint:* Show that for any three of them there exists an interpretation of '$=$', which is different from the strict equality.)

***10.4–2** 'Axiomatize' the domain Nat_ω with LCF-formulas in the following way:

(i) As indicated in Section 10.4 use the strict equality to write the (first-order) Peano axioms of Chapter 2 as LCF-formulas.
(ii) Write the induction principle for the natural numbers as an LCF-formula by using a variable of type (*nat* $\longrightarrow$ *bool*). Note that, as in (i), the undefined element must be 'excluded'.
(iii) Add axioms which express that (the interpretations of) all function symbols are strict.

Note that an axiomatization of Nat_ω with LCF-*formulas* (rather than LCF-*sentences*) is presupposed in the incompleteness proof of the calculus in Section 10.3.

10.4–3 Let S be the recursive program

$$\begin{aligned} \mathrm{F}(x,y) \Leftarrow\ & \textit{if } x = y \textit{ then } x \\ & \textit{else if } x > y \textit{ then } \mathrm{F}(x - y, y) \\ & \qquad \textit{else } \mathrm{F}(x, y - x)\ \textit{fi} \\ & \textit{fi} \end{aligned}$$

Prove—in the style of Example 10.8—that

$$\mu\Phi(\mathrm{S}) \sqsubseteq [\lambda x, y\,.\,\gcd(x, y)]$$

HISTORICAL AND BIBLIOGRAPHICAL REMARKS

LCF is due to D. Scott and R. Milner; the basic paper on LCF is Milner (1972).

There are a relatively large number of publications on LCF, most of which are referenced in the bibliography of Gordon, Milner, and Wadsworth (1979). For a better and broader insight in LCF the reader may, for instance, consult Milner (1979), Aiello, Aiello, and Weyhrauch (1977), and Newey (1973).