MIT/LCS/TM-211

A NOTE ON EQUIVALENCES AMONG LOGICS OF PROGRAMS

Albert R. Meyer

Jerzey Tiuryn

December 1981

A Note On Equivalences Among Logics Of Programs

1 December 1981

Albert R. Meyer
Massachusetts Institute of Technology[1]

Jerzy Tiuryn
Massachusetts Institute of Technology and Warsaw University

This work was supported in part by The National Science Foundation, Grant Nos. MCS 7719754 and MCS 8010707, and by a grant to the M.I.T. Laboratory for Computer Science by the IBM Corporation.

[1] Laboratory for Computer Science, Cambridge, Massachusetts 02139, USA

Abstract. Several different first order formal logics of programs-- Algorithmic Logic, Dynamic Logic, and Logic of Effective Definitions -- are compared and shown to be equivalent to a fragment of constructive $L_{\omega_1 \omega}$. When programs are modelled as effective flowcharts, the logics of deterministic and nondeterministic programs are equivalent.

1 Introduction

A number of systems of formal logics which extend predicate calculus have been proposed for reasoning about sequential and nondeterministic programs. These include in rough chronological order

1. The infinitary logic $L_{\omega_1 \omega}$ -- suggested by ENGELER 67 as a logic for programming,

2. Algorithmic Logic (AL) -- defined and developed by SALWICKI, *et.al.* 70,

3. μ-calculus -- defined by HITCHCOCK and PARK 73; extended by DE BAKKER 80,

4. Dynamic Logic (DL) -- PRATT 76,

5. Programming Logic (PL) -- CONSTABLE and O'DONNELL 78,

6. Logic of Effective Definitions (LED) -- TIURYN 80.

Each of these logical systems actually represents a family of formal logics, instances of the family being determined by the choice of a few parameters. The principal parameter is the class of programs allowed in formulas. For example, in the case of DL some variants which have been considered are

- *regular* DL, in which programs are taken essentially to be finite, possibly nondeterministic, flowchart schemes with atomic formulas as tests and with *simple assignment statements* of the form $x := \tau$ where τ is a term,

- *regular-array* DL in which *array assignments* of the form $\tau_1 = \tau_2$ may also occur (cf. MEYER and WINKLMANN 80),

- *regular* DL$^+$ in which for every finite flowchart α, the predicate $LOOPS_\alpha$, which asserts that α has an infinite computation, is included as an extra atomic formula (cf. MEYER and WINKLMANN 80).[2]

- *recursive-call* DL in which programs are taken to be flowchart schemes containing recursive calls with arguments (cf. GREIBACH 75, DE BAKKER 80).

In general, such different choices of the parameters lead to logics which differ in expressive power. For example, TIURYN 81 has recently shown that there is a formula of recursive-call DL, as well as one of regular-array DL, which is not equivalent to any formula of regular DL. On the other hand, MEYER and WINKLMANN 80 have shown that regular DL and regular DL$^+$ are equivalent in expressive power. MEYER and PARIKH 81 have also demonstrated distinctions among the expressive powers of several other versions of DL and $L_{\omega_1\omega}$.

Thus there are genuine distinctions in the expressive, and also model theoretic and undecidability properties among the various instances of DL. These distinctions complicate the problem of comparing the six systems of programming logics listed above. For example, the bulk of the literature on AL defined that system in the particular version where programs are deterministic **while** schemes.[3] Since the original DL allowed nondeterministic schemes, it appeared that DL and AL represented genuinely distinct conceptions of programming logic.

Nevertheless, we claim that with appropriately matched parameters, DL, AL, and LED, are actually equivalent systems. We believe that PL can be incorporated into this common framework as well, although its numerous "practical" features make it harder to grasp theoretically.

[2]However, $LOOPS_\alpha$ may not occur as a test in a program.

[3]Only recently has an AL with nondeterministic schemes been considered by MIRKOWSKA 80.

These systems can be described in more classical terminology as fragments of the constructive portion of $L_{\omega_1\omega}$, with the different instances of the systems characterized by various simple syntactic conditions on infinitary formulas. Thus, we argue that there is a common intuition which leads to the DL-AL-LED-PL framework for programming logic. In what follows we focus on this framework.[4]

In order to compare the DL-AL-LED-PL frameworks, we restrict ourselves to instances of these systems using what we regard as the mathematically most natural and robust notion of *computability* over arbitrary structures, namely computability by *effective flowcharts.* Effective flowcharts may be described informally as generally infinite, nondeterministic, uninterpreted flowchart schemes whose basic instructions are assignment statements and whose basic tests consist of atomic formulas (including equations). Moreover, given a box of the flowchart, one can effectively find the instruction in that box, the number of edges leaving the box, and the endpoints of those edges. For technical convenience we require that the *signature* (i.e., set of symbols occurring, including variables) of any flowchart is *finite.*

A *state* provides an interpretation for all function, predicate, and variable symbols. Given a state, a nondeterministic flowchart defines a set of executable instruction sequences. The set of states in which execution of these instruction sequences can finally terminate is the set of *output states* for the given input state. Thus, any flowchart α defines a binary *input-output* relation R_α on states where

$R_\alpha = \{(s,t) |$ starting in state s, there is an executable sequence of instructions in α which finishes in output state t$\}$.

[4]Technical results of PARK 76 for μ-calculus, and MEYER and PARIKH 81 for the constructive fragment of $L_{\omega_1\omega}$, show that these latter logics are incomparable in expressive power, and both are strictly greater in expressive power than logics in the DL-AL-LED-PL framework unless the notion of program scheme is stretched unreasonably.

If there is an infinite executable sequence starting in state s, then α is said to *loop from state* s. Formal definitions are available in MEYER and WINKLMANN 80, MEYER and HALPERN 80, MEYER and PARIKH 80, TIURYN 80.

Friedman, cf. SHEPHERDSON 73, proposed a notion of *effective definitional scheme* as the most general model of effective computability in arbitrary structures. These may be described as the special case of effective flowcharts which are of the form

```
if $P_1$ then $ASSIGN_1$ else
if $P_2$ then $ASSIGN_2$ else
if $P_3$ ...
```

where P_i is a finite conjunction of atomic formulas or their negations, and $ASSIGN_i$ is a sequence of assignment statements of the form $x := \tau$ with distinct variables x on the lefthand side of each statement in the sequence.

We can generalize effective definitional schemes to be nondeterministic. These nondeterministic effective definitional schemes can be informally described as the infinite parallel **OR** of statements of the form

if P_i **then** $ASSIGN_i$ **else** ABORT **fi**,

where ABORT is a program with empty input-output relation, e.g., **while** *true* **do** *anything* **od**. Equivalent notions of universal classes of effective procedures on arbitrary structures have been proposed by many other researchers. In particular, it is easy to show

Lemma 1: The following classes of program schemes define the same class of input-output relations:

1. (Non)Deterministic effective flowcharts without array assignments (i.e., simple assignments only),

2. (Nondeterministic) Effective definitional schemes,

3. (Non)Deterministic *finite* flowcharts without array assignments but with stacks.

Similar definitions and lemma can be given for the case that array assignments are allowed. These results indicate the invariance of the class of *computable input-output relations* between states defined by effective flowcharts.

Our main observation is that when effective flowcharts are taken as the notion of program in the programming logics listed above, then all can be reduced to a simple fragment of constructive $L_{\omega_1\omega}$ which we define next.

Definition 2: Let L_{re} be the class of infinitary first order formulas defined inductively as follows:

(a).if P_1, P_2,... is a recursively enumerable sequence of quantifier-free formulas of predicate calculus among which there are only finitely many free variables, then $\vee\{P_i | i \leq 1\}$ is a basic formula of L_{re},

(b).if p,q are formulas of L_{re}, then so are $\neg p$, $p \wedge q$, $p \vee q$, $\exists x[p]$, $\forall x[p]$.

Theorem 3: There is an effective procedure to translate a formula of any one of the following formal logics into an equivalent formula of any of the others:

1. L_{re},

2. DL of deterministic effective flowcharts without array assignments (i.e., only simple assignments occur), henceforth called *DDL-w/o-array*

3. DL of deterministic effective flowcharts (i.e., array assignments may occur) henceforth called *DDL*,

4. DL^+ of nondeterministic effective flowcharts without array assignments, henceforth called *DL^+-w/o-array*,

5. LED,

6. Logic of nondeterministic effective definitional schemes (without array assignments),

7. AL of deterministic effective flowcharts without array assignments,

8. AL of nondeterministic effective flowcharts without array assignments and without the iteration quantifier $\cap$.

We would like to emphasize that according to Theorem 3, DDL-w/o-array and DL$^+$-w/o-array are equivalent, viz., *adding nondeterminism to effective flowcharts does not increase the expressive power of the dynamic logic.*

Although in many programming situations nondeterminism is a significant addition, we can explain informally why it adds nothing to the logic of deterministic effective schemes: the rich control structure provided by arbitrary effective flowcharts enables a deterministic scheme α_d to "check the results" of any nondeterministic scheme α by carrying out a backtracking search. In particular, suppose α is a nondeterministic effective flowchart without array assignments whose registers, i.e., free variables, are $\mathbf{x} = x_0,\ldots,x_{n-1}$. Then there is a deterministic effective flowchart α_d such that $\alpha_d(\mathbf{x},\mathbf{y})$ halts iff $\alpha(\mathbf{x})$ can halt with the final contents of registers $\mathbf{x}$ set to $\mathbf{y}$. Thus the assertion that after $\alpha(\mathbf{x})$ halts, it is possible that some property $p(\mathbf{x})$ holds, is equivalent to the assertion that *there exist* $\mathbf{y}$ such that $\alpha_d(\mathbf{x},\mathbf{y})$ halts and $p(\mathbf{y})$ holds. In this way, an existentially quantified assertion about a deterministic flowchart has the same expressive power as an assertion about a nondeterministic flowchart.

For more restricted control structures which cannot carry out the backtrack search, nondeterminism indeed makes a difference: P. Berman, J. Halpern, and J. Tiuryn have recently shown that for *regular* programs, DDL is strictly less expressive than DL.

In the case that array assignments do occur in nondeterministic programs, our proof of Theorem 3 breaks down. The nondeterministic flowchart α may have registers $\mathbf{x}$ and also assignable arrays, i,e., function symbols $\mathbf{f}$. Again, there is a deterministic "checking" flowchart α_d such that $\alpha_d(\mathbf{x},\mathbf{f},\mathbf{y},\mathbf{g})$ halts iff $\alpha(\mathbf{x},\mathbf{f})$ can halt with the final values of registers $\mathbf{x}$ and arrays $\mathbf{f}$ equal to $\mathbf{y},\mathbf{g}$. Now, however, in order to reduce an assertion about α to one about α_d as above, it is necessary to existentially bind not only the $\mathbf{y}$ variables by also the function symbols **g**. This second order quantification exceeds the power of DL. But because the values of the arrays g differ only finitely from the values of the f, the full power of second order quantification is not necessary. If there are elements in the domain of interpretation which can serve to represent finite sets, it is possible to simulate this weak second order quantification by first order quantifiers. Any infinite set of finitely generated elements will serve to represent finite sets, so, aside from the pathological case of (essentially) finite domains, we can extend the theorem to nondeterministic effective flowcharts even with array assignments.

Namely, let Σ be some finite set of function symbols. A state is n,Σ-*infinite* iff there are n elements of the domain of the state such that the set of elements generated by applying the functions (which are the interpretations in the state of the symbols) in Σ to these n elements is infinite.

Theorem 4: For any $n>0$ and finite set Σ of function symbols, there is an effective procedure to translate any formula p of the logics 9.-11. below, into a formula p' of L_{re} such that for every n,Σ-infinite state s,

$$s\models p \text{ iff } s\models p'.$$

9. DL^+ of nondeterministic effective flowcharts,

10. Logic of nondeterministic effective definitional schemes (with array assignments),

11. AL of nondeterministic effective flowcharts without the iteration quantifier $\cap$.

It remains an interesting open question whether the hypothesis of n, Σ-infinity can be eliminated from Theorem 4. Whether the iteration quantifier $\cap$ makes a difference in the presence of nondeterministic programs is also open, but appears to be of technical interest only.

In the next section we present the main definitions among the logics 1.-11., and prove Theorems 3 and 4.

2 Definitions and Proofs

All of the logics 1.-11. are subsets of the following class L_{univ} of formulas which is obtained by combining the features of all the languages.

Definition 5: L_{univ} is defined inductively as follows:

(a).Any atomic formula of predicate calculus with equality is a formula of L_{univ},

(b).if α is an effective flowchart, then $LOOPS_\alpha$ is a formula of L_{univ},

(c).if p,q are formulas of L_{univ}, then so are $\neg p$, $p \wedge q$, $p \vee q$, $\exists x[p]$, $\forall x[p]$,

(d).if $P_1, P_2, \ldots,$ is an r.e. sequence of formulas of L_{univ}, then so are $\vee\{P_i | i \geq 1\}$ and $\wedge\{P_i | i \geq 1\}$,

(e).if α is an effective flowchart and p is a formula of L_{univ}, then so are $\langle\alpha\rangle p$ and $[\alpha]p$,

(f).if α is an effective flowchart and p is a formula of L_{univ}, then so are $(\cap\alpha)p$ and $(\cup\alpha)p$.

Whether a state s *satisfies* a formula p of L_{univ}, denoted $s \models p$, is defined in the usual way for p of the form (a), (c), or (d) above.

For case (b), $s \models LOOPS_\alpha$ iff α loops from state s.

For case (e), $s \models \langle\alpha\rangle p$ iff $t \models p$ for *some* state t such that $(s,t) \in R_\alpha$; $s \models [\alpha]p$ iff $t \models p$ for *all* states t such that $(s,t) \in R_\alpha$.

Case (f) covers the *iteration quantifiers* of AL. $s \models (\cup\alpha)p$ iff $s \models \langle\alpha^*\rangle p$, where α^* is an effective flowchart such that R_{α^*} is the reflexive transitive closure of R_α. $s \models (\cap\alpha)p$ iff $s \models \langle\alpha^n\rangle p$ for *all* $n \geq 0$, where α^n is an effective flowchart such that R_{α^n} = the relational composition of R_α with itself n times.

This defines the *semantics* of L_{univ}.

L_{re} is easily embeddable in all of the logics of Theorem 3, and all are obviously embeddable into one of DDL or DL^+-w/o-array, so we give precise definitions and proofs only for these latter two logics.

Definition 6: *DDL* is the class of formulas defined by rules (a,c,e) of Definition 5 such that the flowcharts α of rule (e) are deterministic. *DL^+-w/o-array* is the class of formulas defined by rules (a,b,c,e) such that the flowcharts α of rule (e) do not contain array assignments.

To prove Theorem 3, we describe translations between L_{re} and DDL, and between L_{re} and DL^+-w/o-array.

The translation from L_{re} actually takes formulas of L_{re} into the intersection of DDL and DL^+-w/o-array. It is obtained trivially from the observation that the atomic formula $\vee\{P_i | i \geq 1\}$ of L_{re} is equivalent to $\langle\alpha\rangle true$ where α is the effective flowchart

```
if P_1 then x:=x else
if P_2 then x:=x else
if P_3 then x:=x else....
```

The translation from DDL to L_{re} is based on

Lemma 7: The following formulas are valid for any flowchart α and formula p of L_{univ}:

1. $\langle\alpha\rangle(p \vee q) \equiv (\langle\alpha\rangle p \vee \langle\alpha\rangle q)$,

2. $\langle\alpha\rangle\exists x[p] \equiv \exists z[\langle\alpha\rangle(p[z/x])]$, where z does not occur in α or p, and $p[z/x]$ is the result of substituting z for x in p.

In addition, the following formula is valid for any *deterministic* flowchart α and formula p of L_{univ}:

3. $\langle\alpha\rangle\neg p \equiv (\langle\alpha\rangle true \wedge \neg\langle\alpha\rangle p)$.

The equivalences of Lemma 7 allow one to "move the $\langle\rangle$'s in" thereby converting an arbitrary formula of DDL into an equivalent formula built solely by first order constructs, i.e., the rules of Definition 5.(c), starting from formulas of the form $\langle\beta_1\rangle...\langle\beta_n\rangle P$ where P is an atomic formula of predicate calculus. But a formula of the form $\langle\beta_1\rangle...\langle\beta_n\rangle P$ is equivalent to an r.e. disjunction of formulas $\langle\alpha_i\rangle P$ where α_i ranges over the terminating instruction sequences of the program $\beta_1;...;\beta_n$. Each formula $\langle\alpha_i\rangle P$, where α_i is a finite sequence of assignments and atomic tests and P is quantifier free, is equivalent to a quantifier free formula of predicate calculus, cf. PRATT 76, MEYER and PARIKH 80. In this way DDL translates into L_{re}.

The translation from DL^+-w/o-array into L_{re} proceeds by induction on the definition of DL^+. The only interesting case in the basis of the induction is for formulas of the form $LOOPS_\alpha$. These are obviously equivalent to the r.e. conjunction of the quantifier-free first order formulas which assert that a terminating instruction sequence in α is not executable.

The essential step in the inductive definition of the translation is $\langle\,\rangle$- elimination. Let α be a nondeterministic effective flowchart without array assignments and let p be a formula of DL^+-w/o-array. By induction, we may assume there is a formula q of L_{re} equivalent to p. Let $x_0,...,x_{n-1}$ be all the variables occurring in flowchart α. It is easy to define an r.e. set of quantifier-free first order formulas $\{P_i | i \geq 0\}$ and an r.e. set of terms $\{\tau_{i,j} | i \geq 0, j < n\}$ such that for all $j<n$ and states s, $s \models P_i$ iff it is possible for α, started in state s, to terminate with the terminal value of x_j equal to the value of $\tau_{i,j}$ in state s.

Let $y_0,...,y_{n-1}$ be new variables which occur neither in α nor in q. The reader can easily check that

$$\exists y_0...\exists y_{n-1}[\ \vee_i\{P_i \wedge (\wedge_{j<n}\ y_j = \tau_{i,j})\ \} \wedge q[y_0,...,y_{n-1}/x_0,...,x_{n-1}]\ ] \qquad (1)$$

is equivalent in all states to $\langle\alpha\rangle p$.

We remark that introducing quantifiers in formula (1), or indeed any such formula which accomplishes $\langle\rangle$-elimination, is unavoidable. This follows from the fact that the quantifier-free fragment of Deterministic DL^+-w/o-array, which is equivalent to quantifier-free L_{re}, is strictly weaker than the quantifier-free fragment of DL^+-w/o-array, (cf. MEYER and WINKLMANN 80).

This completes the proof of Theorem 3.

In proving Theorem 4, we note that all of the logics 9.-11. are no more expressive that DL^+. We therefore only describe the translation of DL^+ into L_{re}.

As in the proof of Theorem 3, the translation is given inductively. The only interesting case is $\langle\rangle$-elimination.

Let p be a formula of DL^+ and let α be a nondeterministic effective flowchart. By induction, let q be an L_{re} formula equivalent over all n,Σ-infinite states to p. According to Lemma 1 we can find a nondeterministic effective definitional scheme which defines the same input-output relation as α. This effective definitional scheme is an infinite parallel **OR** of finite deterministic programs α_i of the form

if P_i **then** $ASSIGN_i$ **else** ABORT **fi**

where $ASSIGN_i$ is a finite sequence of assignments.

Obviously $\langle\alpha\rangle p$ is equivalent to the L_{univ} formula

$$\vee_i\langle\alpha_i\rangle q. \tag{2}$$

It is not hard to show that any formula $\langle\alpha_i\rangle q$ is equivalent to a formula of L_{re}, but this still leaves the difficulty that (2) is an infinite disjunction of L_{re}, not first order, formulas, and L_{re} is not closed under infinite disjunctions. We could eliminate this difficulty if the integer variable i in $\langle\alpha_i\rangle q$ could somehow be taken as a variable of DL, for then the infinite disjunction over i in (2) could simply be replaced by an existential quantification of i. With the aid of the hypothesis of n,Σ-infinity, we will accomplish this as follows.

Each formula $\langle\alpha_i\rangle q$ can be transformed using the equivalences of Lemma 7 so that all the occurrences of $\langle\alpha_i\rangle$ appear in the context

$$\langle\alpha_i\rangle\vee_m G_m \tag{3}$$

where the G_m are quantifier-free formulas of predicate calculus. Let the formula obtained in this way be denoted q_i, so that formula (2) is equivalent to $\vee_i q_i$. (The transformation is uniform in i, so the same set of disjunctions $\vee_m G_m$ occur in each q_i.) In order to eliminate the outermost disjunction in (2) we use the assumption of n,Σ-infinity of states.

Let $\mathbf{y}=y_0,...,y_{n-1}$, and z be n+1 individual variables which occur neither in α nor in q_i, and choose some effective enumeration $\tau_1(\mathbf{y})$, $\tau_2(\mathbf{y})$,... of all the terms over $\mathbf{y} \cup$ (signature(α) - variables(α)), i.e., the terms with function symbols from α whose only variables are from $\mathbf{y}$.

For $k,i\geq 1$, let $D_{k,i}(\mathbf{y},z)$ be a quantifier-free formula of predicate calculus which expresses the following property: "z is the value of the k^{th} term (in the above enumeration), there are exactly i distinct values among those first k terms, and k is the least integer with the above two properties".

Let q' be a formula obtained from q_i by replacing every subformula of the form (3) by the r.e. disjunction

$$\vee\{D_{k,j}(\mathbf{y},z) \wedge G_{mj} | j,k,m\geq 1\}$$

where G_{mj} is a quantifier free first order formula equivalent to $\langle\alpha_j\rangle G_m$. Note that by the uniformity in i of the definition of q_i, it follows that the same q' is obtained for all i.

In q' we have apparently eliminated the index i, but it will be coded in the values of variables $\mathbf{y}$ and z. This coding is possible because, by definition of $D_{k,j}$, for every state s there is *at most* one pair of integers $k,j\geq 1$ such that $s\models D_{k,j}$. Moreover, for every n,Σ-infinite state s and for arbitrary $k,j\geq 1$, $s\models \exists\mathbf{y}\exists z[D_{k,j}(\mathbf{y},z)]$. Thus in n,Σ-infinite states we can code any pair of integers by using the formulas $D_{k,j}$.

We use the above observation to code the value of index i. Let q" be the formula

$$\exists\mathbf{y}\exists z[\vee\{P_i \wedge D_{k,i}(\mathbf{y},z)| k,i\geq 1\} \wedge q']$$

where P_i is the test portion of α_i.

We claim that for every n,Σ-infinite state s,

$$s \models \vee_i \langle\alpha_i\rangle q \equiv q''. \qquad (4)$$

In order to prove the claim (4), let us assume that $s \models \langle\alpha_i\rangle q$ for a certain $i \geq 1$. Let $\mathbf{a} = a_0,\ldots,a_{n-1}$ be the generators of an infinite substructure in s, let b be the i-th distinct value in the sequence $\tau_1(a), \tau_2(a),\ldots$ and let $k \geq 1$ be the least integer such that $b = \tau_k(a)$. Let s_i be the state in which **y** has the value **a**, z has the value b, and all other symbols have the same interpretation as in s. We have $s_i \models P_i \wedge D_{k,i}(y,z)$ because $s \models P_i$ and **y**,z do not occur in P_i.

In order to see that $s_i \models q'$, it is enough to observe that for any r.e. set of formulas $\{G_m | m \geq 1\}$,

$$s_i \models \vee\{D_{k,j}(y,z) \wedge G_{mj} | j,k,m \geq 1\} \text{ iff } s \models \langle\alpha_i\rangle \vee_m G_m.$$

In this way we have proved $s \models q''$. The other half of the equivalence (4) is proved similarly.

3 Conclusion

Having reduced essentially all the various programming logics to the L_{re} fragment of infinitary logic, it is easy to deduce a body of model theoretic and undecidability results about programming logic from known results for infinitary logic. Moreover, the reduction to L_{re} is sufficiently straightforward that various infinitary proof theoretic results can also be carried over directly to programming logic.

We interpret these results as evidence that no very new model theoretic or recursion theoretic issues arise from logics of programs on first order structures.

Nevertheless, we believe that the problem of developing formal systems for reasoning about programs offers significant challenges in at least two directions. First, to be true to the purpose for which high level programming languages were originally developed and continue to be developed -- namely for economy and ease in the expression of algorithms --

it is important to develop proof methods for dealing with high level programs as textual objects. This has in fact been the focus of the bulk of the literature on program correctness, although many of the complex features of modern programming languages have yet to be adequately addressed. (In our treatment we assumed in effect that the high level programs had already been transformed into effective flowcharts, and thereby we avoided the challenge of developing a proof theory.) A second challenge involves programs operating on higher-type domains which are often assumed to satisfy "domain equations" which appear inconsistent with standard set theory. Development of appropriate logics for reasoning about such domains has just begun, cf. SCOTT 80, and seems an intriguing subject for further research.

Acknowledgement. We are grateful to Piotr Berman for his comments.

4 References

1. BANACHOWSKI, L. *et al.* An Introduction to Algorithmic Logic; Metamathematical Investigations in the Theory of Programs, *Mathematical Foundations of Computer Science*, Banach Center Publications, vol. 2, (ed A. Mazurkiewicz and Z. Pawlak), Polish Scientific Publishers, Warsaw, 1977, 7-100.

2. BERGSTRA, J., TIURYN, J. and TUCKER, J. Floyd's Principle, Correctness Theories and Program Equivalence, *Mathematisch Centrum*, IW145/80. To appear in *Theoretical Computer Science*, 1981.

3. CONSTABLE, R.L., and O'DONNELL, M.J. *A Programming Logic*, Winthrop Publishers, 1978.

4. DE BAKKER, J. *Mathematical Theory of Program Correctness*, Prentice-Hall, 1980.

5. ENGELER, E. Algorithmic Properties of Structures, *Mathematical Systems Theory*, 1, 1967, 183-195.

6. ENGELER, E. Algorithmic Logic. In de Bakker (ed.) *Mathematical Centre*

Tracts (63) Amsterdam 1975, 57-85.

7. GALLIER, J. H. Nondeterministic flowchart programs with recursive procedures: semantics and correctness, *Theoretical Computer Science*, 13, 2(1981), 193-224.

8. GREIBACH, S. *Theory of Program Structures: Schemes, Semantics, Verification*, Lecture Notes in Computer Science, 36, Springer Verlag, 1975.

9. HAREL, D. *First-Order Dynamic Logic*, Lecture Notes in Computer Science 68, Springer-Verlag, 1979.

10. HAREL, D., A.R. MEYER and V. PRATT, Computability and Completeness in Logics of Programs: Preliminary Report, *9th ACM Symp. on Theory of Computing*, Boulder, Colorado, (May, 1977), 261-268. Revised version, M.I.T. Lab. for Computer Science TM-97, (Feb. 1978), 16 pp.

11. HAREL, D. and V. PRATT, Nondeterminism in logics of programs, *5th Annual Symposium on Principles of Programming Languages*, January 1978, 203-213.

12. HITCHCOCK, P. and D. PARK. Induction Rules and Termination Proofs, *Automata, Languages and Programming*, (ed M. Nivat), American Elsevier, New York, 1973, 225-251.

13. KEISLER, H.J. *Model Theory for Infinitary Logic*. North-Holland Publ. Co., Amsterdam 1972.

14. KFOURY, D.J. Comparing Algebraic Structures up to Algorithmic Equivalence. In Nivat (ed.) *Automata, Languages and Programming*. North-Holland Publ. Co., Amsterdam 1972, 253-264.

15. KFOURY, D.J. Translatability of schemes over restricted interpretations. *Journal of Comp. and Syst. Sc. 8* (1974), 387-408.

16. MEYER, A.R. Ten thousand and one logics of programming. *EATCS Bulletin*, 11-29; M.I.T. LCS TM 150, MIT Laboratory for Computer Science, Cambridge, Ma., February 1980.

17. MEYER, A.R. and J. Y. HALPERN, Axiomatic Definitions of Programming Languages: A Theoretical Assessment, (Preliminary Report) *Proc. of Seventh Annual POPL Conf.,* January 1980, 203-212; M.I.T. LCS TM 163, April, 1980, 34 pp.; to appear *JACM* (1981).

18. MEYER, A.R. and R. PARIKH, Definability in Dynamic Logic, *Proc. of ACM Symp. on Theory of Computing,* Los Angeles, Cal., April, 1980, 1-7; to appear *Jour. Computer and System Science* (1981).

19. MEYER, A.R. and K. WINKLMANN, On the Expressive Power of Dynamic Logic, Preliminary Report, *Proc. of the 11th Annual ACM Conf. on Theory of Computing,* Atlanta, Ga., May 1979, 167-175; M.I.T. LCS TM 157, February,1980, 36pp; to appear *Theoretical Computer Science* (1981).

20. MIRKOWSKA, G. Complete Axiomatization of Algorithmic Properties of Program Schemes with Bounded Nondeterministic Interpretations, *12th Annual ACM Symp. on Theory of Computing* (1980), 14-21.

21. PARK, D. Finiteness is mu-ineffable, *Theoretical Computer Science* 3, 1976, 173-181.

22. PRATT, V., Semantical considerations on Floyd-Hoare logic, *Proceedings 17th Symposium on Foundations of Computer Science,* Houston, Texas, October 1976, 109-121.

23. SALWICKI, A. Formalized Algorithmic Languages, *Bull. Acad. Pol. Sci.,Ser. Math. Astr. Phys.* 18, 1970, 227-232.

24. SCOTT, D. S. Relating Theories of the λ-Calculus, in *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism,* eds. Seldin and Hindley, Academic Press, New York, 1980, 403-450.

25. SHEPHERDSON, J.C. Computing over abstract structures: serial and parallel procedures and Friedman's effective definitional schemes, In Shepherdson and Rose (eds.) *Logic Colloquium 73.* North-Holland, Amsterdam, 1973, pp.445-513.

26. TIURYN, J. A Survey of the Logic of Effective Definitions, MIT/LCS/TR-246, MIT, Laboratory For Computer Science, Cambridge, Mass., September

1980.

27. TIURYN, J. Unbounded program memory adds to expressive power of first-order Dynamic Logic, *Proceedings 22nd IEEE Symposium on Foundations of Computer Science*, Nashville, Tennessee, October 1981, to appear.