From Formal Correctness to Engineering Validity

Giuseppe Primiero

UNIVERSITÀ DEGLI STUDI DI MILANO

DIPARTIMENTO DI FILOSOFIA
"PIERO MARTINETTI"

PROGRAMme Workshop
Bertinoro

Curry-Howard-de Bruijn Correspondence

First observed by [Curry (1934)], then later independently reformulated by [Howard (1969)] and [de Bruijn (1970)] in the 60s and 70s respectively:

Principle (Curry-Howard-de Bruijn Correpondence)

Propositions are Types.

Early Steps: Gentzen I

$$\frac{}{A \vdash A}\, Id$$

$$\frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \wedge B}\, \wedge\text{-}intro$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\, \wedge\text{-}elim(left) \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\, \wedge\text{-}elim(right)$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B}\, \supset\text{-}intro \qquad \frac{\Gamma \vdash A \qquad \Delta \vdash A \supset B}{\Gamma, \Delta \vdash B}\, \supset\text{-}elim$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\, \vee\text{-}intro(left) \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\, \vee\text{-}intro(right)$$

$$\frac{\Gamma \vdash A \vee B \quad \Delta \vdash A \supset C \qquad \Delta \vdash B \supset C}{\Gamma, \Delta \vdash C}\, \vee\text{-}elim$$

Curry

- [Curry (1934)]: a function $f : A \to B$ with an input of type A and an output B can be intuitively read as a corresponding implication $A \supset B$ and hence for any computable function there is a provable proposition of that form, and viceversa.

- [Howard (1969)], making use of results by [Prawitz (1965)]: now every natural deduction pair of rules can be reformulated in the typed λ-calculus.

Early Steps: Typed Church I

$$\frac{}{x : A \vdash x : A}\, Id$$

$$\frac{\Gamma \vdash t : A \qquad \Delta \vdash u : B}{\Gamma, \Delta \vdash \langle t, u \rangle : A \wedge B}\, \wedge\text{-}intro$$

$$\frac{\Gamma \vdash t : A \wedge B}{\Gamma \vdash \pi_1(t) : A}\, \wedge\text{-}elim(left) \qquad \frac{\Gamma \vdash t : A \wedge B}{\pi_2(t) : B}\, \wedge\text{-}elim(right)$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \supset B}\, \supset\text{-}intro \qquad \frac{\Gamma \vdash t : A \qquad \Delta \vdash u : A \supset B}{\Gamma, \Delta \vdash t(u) : B}\, \supset\text{-}elim$$

$$\frac{\Gamma \vdash t : A}{inl(t) : A \vee B}\, \vee\text{-}intro(left) \qquad \frac{\Gamma \vdash u : B}{inr(u) : A \vee B}\, \vee\text{-}intro(right)$$

$$\frac{\Gamma \vdash t : A \vee B \quad \Gamma, x : A \vdash u : C \quad \Gamma, y : B \vdash v : C}{\Gamma \vdash \lambda x.t(u) \mid \lambda y.t(v) : C}\, \vee\text{-}elim$$

Equivalence

FUNCTION TYPE	PROPOSITION
$A \to B$	$A \supset B$
$A \times B$	$A \wedge B$
$A + B$	$A \vee B$
$\lambda(b) : \Pi(A, B)$	$(\forall x : A)B(x)$
$\langle a, b \rangle : \Sigma(A, B)$	$(\exists x : A)B(x)$

Figure: Correspondence between function types and propositions

Correspondence revisited

Principle (Curry-Howard-de Bruijn Correpondence – Revisited)

Proofs are Programs.

Equivalence, [Martin-Löf (1982)]

Programming	Mathematics
program, procedure, algorithm	function
input	argument
output, result	value
$x := e$ (assignment)	$x = e$
$S_1; S_2$	composition of functions
if B then S_1 else S_2	definition by cases
while B do S	definition by recursion
data structure	element, object
data type	set, type
value of data type	element of a set, object of a type
$a : A$	$a \in A$
integer	Z
real	R
boolean	$\{0, 1\}$
$(c_1, \ldots, c_n)$	$\{c_1, \ldots, c_n\}$
array $[I]$ of T	T^I, $I \to T$
record $s_1 : T_1; s_2 : T_2$ end	$T_1 \times T_2$
record case $s : (c_1, c_2)$ of $c_1 : (s_1 : T_1); c_2 : (s_2 : T_2)$ end	$T_1 + T_2$
set of T	$\{0, 1\}^T$, $T \to \{0, 1\}$

Figure: Key notions of Programming with mathematical counterparts

Correctness

Definition (Program Correctness Problem)

Given any program p, is it possible to prove whether p executed on the appropriate input i returns the intended output o, or in other words whether it satisfies its intended function?

Floyd, [Floyd (1967)]

- an association of steps in the control flow of a program with logically holding propositions.
- A semantic definition of the program syntactically specifies linguistic constructs expressing valid commands and their conditions
- an interpretation corresponds to a mapping of logical sentences to each transition from step to step (called the entry and exit of the command).
- A verification of the flowchart of the program is a proof that for every command expressed in it, if there is an entrance such that a given sentence is true, then there must be an exit such that another sentence is true.

Floyd, [Floyd (1967)]

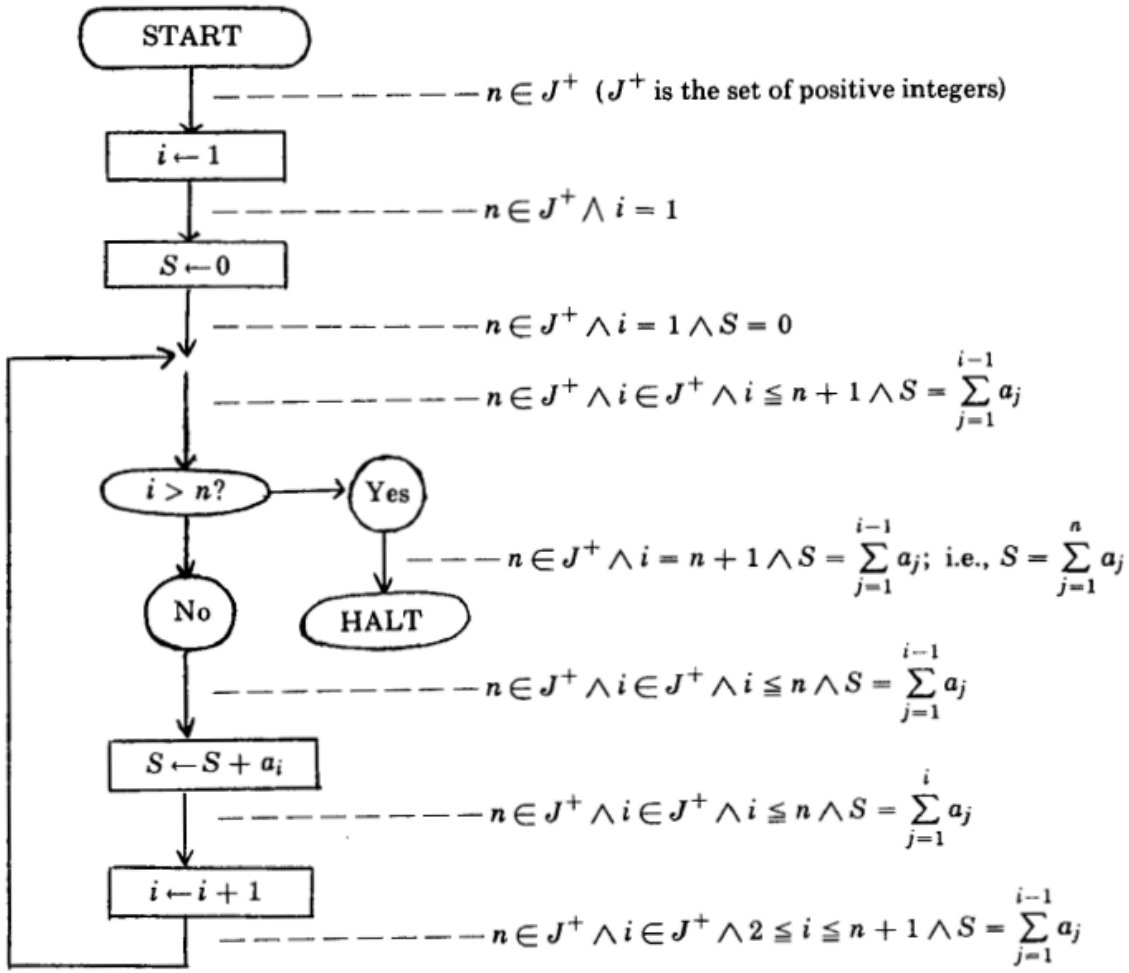

FIGURE 1. Flowchart of program to compute $S = \sum_{j=1}^{n} a_j$ $(n \geqq 0)$

Figure: Flowchart from [Floyd (1967)].

Termination and Machine-independence

- *"we have not proved that an exit will ever be reached; the methods so far described offer no security against nonterminating loops. "*
- Properties of programs in a given language are independent of their physical instantiation and processors for that language, and can be reduced to *"establishing standards of rigor for proofs about programs in the language"*.

Hoare's Axiomatic Semantics, [Hoare (1969)]

- an *assertion* corresponds to a predicate describing the state of a program at any point of its execution.
- determine the valid assignments of these variables *before* program execution, its *preconditions*.
- check that the program execution leads to the expected set of variable assignments or program states *after* the program operations have been performed, its *postconditions*.

$$P\{Q\}R$$

Hoare's Axiomatic Semantics, [Hoare (1969)]

$$\frac{}{P\{x := v\}P'}\ \textit{Assignment}$$

$$\frac{P\{Q_1\}R \qquad R\{Q_2\}S}{P\{Q_1; Q_2\}S}\ \textit{Sequencing}$$

$$\frac{P \vdash P' \qquad P'\{Q\}R' \qquad R' \vdash R}{P\{Q\}R}\ \textit{Consequence}$$

$$\frac{P \wedge R\{Q\}P}{P\{\textit{while } R \textit{ do } Q\}\neg R \wedge P}\ \textit{Iteration}$$

Figure: Rules of Floyd-Hoare Logic

Hoare's Axiomatic Semantics, [Hoare (1969)]

Definition (Hoare Correctness)

A program is correct with respect to a pre- and post-condition specification if the execution of the program on a machine whose initial state satisfies the pre-condition will result in the machine terminating in a state which satisfies the post-condition.

No side-effects and Universality

- *"The axioms and rules of inference quoted in this paper have implicitly assumed the absence of side effects of the evaluation of expressions and conditions."*

- The axiomatic method is *general*, implementations can be designed so that they should be required to satisfy the axioms, and it is *universally applicable*, so that aspects of the program – like data-types – can be left underspecified.

Dijsktra, [Dijkstra (1975)]

Definition (Weakest Pre-Condition)

The weakest pre-condition, denoted $wp(P, R)$ is the condition that characterizes the set of *all* initial states such that execution of program P will result with certainty in a final state satisfying a given post-condition R.

The semantics of P is given by determining for any post-condition R the corresponding weakest pre-condition $wp(P, R)$. If the initial state does not satisfy a weakest pre-condition, P may not establish the truth of R in its final state, or P may fail to reach a final state at all, because it enters an infinite loor or it gets stuck,[Dijkstra (1975), p.17].

Dijkstra: Total Correctness

Definition (Total Program Correctness)

For a given specification $(pre(P), R)$, a program P is totally correct with respect to its specification iff $pre(P) \rightarrow wp(P, R)$

Dijkstra: Partial Correctness

Definition (Partial Program Correctness)

A program P is partially correct if it can be proved that $pre(P) \wedge wp(P, true) \rightarrow wp(P, R)$, where $wp(P, true)$ is the weakest pre-condition ensuring termination in any state.

Correctness vs. Efficiency

"most techniques for proving the correctness of a program treat the program text as a static, rather formal, mathematical object that can be dealt with independently of the fact that there may exist machines that could execute such a program. As such, a clear separation of concerns emerges: we might call them the mathematical concerns about correctness and the engineering concerns about efficiency. In contrast to the correctness concerns, the efficiency concerns are only meaningful in relation to implementations and it is only during the efficiency concerns that we need remember that the program text is intended to evoke computational processes. Both the mathematical and the engineering concerns have, of course, always been with us, but once they used to be dealt with inextricably intertwined."

Principles

Principle (Algorithmic dependence)

Computation is a mathematical process in which an algorithmic description is lingustically encoded.

Principles

Principle (Encoding independence)

The output of a given equivalence class of computational processes is unaffected by the properties of the chosen linguistic encoding when these processes are considered at the highest level of abstraction.

Formal Validity

Definition (Valid Formal Computation)

A computational process p is formally valid if and only if any expression e required by the linguistic encoding of the algorithmic description of p is logically admissible and the process described by p terminates in a state whose linguistic encoding expresses its intended output S.

Resource Aware Logic I

$$\frac{}{\phi \vdash \phi}\ \text{Identity}$$

$$\frac{}{\{\}_m \vdash I}\ I\text{-I} \qquad \frac{\Gamma(\{\}_m) \vdash \chi \qquad \Delta \vdash I}{\Gamma(\Delta) \vdash \chi}\ I\text{-E}$$

with $\{\}_m$ the multiplicative unit and I the empty resource (in the multiplicative fragment).

$$\frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \mathrel{-\!\!*} \psi}\ \mathrel{-\!\!*}\text{-I} \qquad \frac{\Gamma \vdash \phi \mathrel{-\!\!*} \psi \qquad \Delta \vdash \phi}{\Gamma, \Delta \vdash \psi}\ \mathrel{-\!\!*}\text{-E}$$

$$\frac{\Gamma \vdash \phi \qquad \Delta \vdash \psi}{\Gamma, \Delta \vdash \phi * \psi}\ *\text{-I} \qquad \frac{\Gamma(\phi, \psi) \vdash \chi \qquad \Delta \vdash \phi * \psi}{\Gamma(\Delta) \vdash \chi}\ *\text{-E}$$

$$\frac{}{\{\}_a \vdash 1}\ 1\text{-I} \qquad \frac{\Gamma(\{\}_a) \vdash \chi \qquad \Delta \vdash 1}{\Gamma(\Delta) \vdash \chi}\ I\text{-E}$$

with $\{\}_a$ the additive unit and 1 the empty resource (in the additive fragment).

Figure: Rules of BI Logic, [O'Hearn, Pym (1999)]

Formal Correctness

Definition (Partial Correctness)

$\{\phi\}C\{\psi\}$ is true if for all $R = \langle s, h \rangle$ it is the case that $R \models \phi$ implies that $\langle C, R \rangle$ is safe and if there is a transformation according to C to a memory configuration $R' = \langle s', h' \rangle$, then $R' \models \psi$.

Formal Correctness

Definition (Partial Correctness)

$\{\phi\}C\{\psi\}$ is true if for all $R = \langle s, h\rangle$ it is the case that $R \models \phi$ implies that $\langle C, R\rangle$ is safe and if there is a transformation according to C to a memory configuration $R' = \langle s', h'\rangle$, then $R' \models \psi$.

Definition (Total Correctness)

$\{\phi\}C\{\psi\}$ is true if for all $R = \langle s, h\rangle$ it is the case that $R \models \phi$ implies that $\langle C, R\rangle$ terminates normally and if there is a transformation according to C to a memory configuration $R' = \langle s', h'\rangle$, then $R' \models \psi$.

Resources

1. The resources from which a program depends need to be actually accessible/available at run time;

2. the program terminates with the intended specification under all possible configuration of resources.

CAP

Definition (Consistency)

Every read process receives the most recent write process, or an error.

Definition (Availability)

Every request process receives a (non-error) response – without guarantee that it contains the most recent write process.

Definition (Partition Tolerance)

The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes.

CAP

Theorem (CAP Theorem)

One can only practically build a distributed system exhibiting any two characteristics among Consistency, Availability and Partition-tolerance.

Partiality

1. either processes are consistent and live, but the system may fail in the presence of dropped messages
2. or is consistent and persistent, but may contain error responses
3. or is persistent in the presence of dropped messages and live even in the presence of older messages, but then it might not preserve consistency.

Principles of Physical Computational Validity

1. *Functionality*: the ability of a physically constructed computational system to satisfy the aims and objectives for which it has been designed, possibly nothwistanding the presence of non-essential errors.

2. *Efficiency*: the principle that functionalities need to be delivered at least as fast as required, parametrized by computational resources, and possibly preferred to fully correct computations which are not efficient.

3. *Usability*: the principle that practical interaction with the system guarantees both service delivery and resource usage within admissible limits.

Principles

Principle (Architectural Dependence)

A physical computation is the implementation in a physical infrastructure of the linguistic encoding of an algorithmic process.

Principles

Principle (Behavioural Dependence)

A physical computation may behave differently depending on the architecture's properties.

Principles

Definition (Functional Validity)

A physical computing artefact t is functionally valid if and only if the functionalities of its intended specification S are logically admissible.

Principles

Definition (Functional Correctness)

A physical computing artefact t is functionally correct if and only if the functionalities of its intended specification S are displayed with the intended efficiency even in the presence of negligible errors when t is used in the appropriate way.

Principles

Definition (Valid Physical Computation)

A physical artefact t performs a valid computation if and only if

1. t is functionally valid (i.e. its intended specification S is logically admissible according to an algorithmic description p), and
2. the information processing c implementing the linguistic encoding e of p is functionally correct and efficient (and possibly optimally so), and
3. c will terminate in a state satisfying S.

Final observations

- Formal correctness requires principles of algorithmic dependence and encoding independence
- Physical validity of computing requires principles of architectural and behavioural dependence
- Their co-existence is allowed by an analysis that includes the different levels of abstraction of a computational system.

Thanks.

References I

- Böhm, C., Jacopini, G., Flow diagrams, Turing Machines, and languages with only two formation rules, *Communications of the ACM*, 9(5), pp.366-371, 1966.
- Curry, H. B., Functionality in Combinatory Logic, *Proceedings of the National Academy of Sciences of the United States of America*, Vol. 20, No. 11, pp. 584-590, 1934.
- de Bruijn, N.G., A Survey of the Project AUTOMATH, *Symposium on Automatic Demonstration*, Lecture Notes in Mathematics, vol.125, pp.29–61, 1970.
- Dijkstra, E.W., Correctness concerns and, among other things, why they are resented. Proceedings of the *1975 International Conference on Reliable Software*, 21â23 April 1975, Los Angeles, California U.S.A, *ACM SIGPLAN Notices*, vol.10, issue 6, pp. 546–550, 1975.

References II


Floyd, R.W., Assigning meanings to Programs, *Proceedings of Symposia in Applied Mathematics*, 19:19-32, Provdience, RI, USA, AMS, 1967.


N. Fresco and G. Primiero.
Miscomputation.
Philosophy & Technology, 26(3):253–272, 2013.


Godfrey-Smith, P., Triviality Arguments Against Functionalism, *Philosophical Studies*, 145(2), pp.273–295, 2009.


Harel, D., On Folk Theorems, *Communications of the ACM*, 23(7), pp.379-388, 1980.


Hoare, C.A.R., An axiomatic basis for computer programming, *Communications of the ACM*, 12(10):576-580, 1969.

References III

- Hoare, C.A.R, Mathematics of Programming – Mathematical Laws help programmers control the complexity of tasks, in BYTE, August 1986, pp.115–149; reprinted in T.R. Colburn et al. (eds.), *Program Verification*, pp.135–154, Kluwer, 1989.
- Howard,W.A., The formulae-as-types notion of construction. Circulated privately in 1969. Printed in J.P.Seldin, J.R. Hendlin (eds.), *Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pp.479–491, Academic Press, NY, 1980.
- Martin-Löf, P. Constructive Mathematics and Computer Programming, in Cohen, Los, Pfeiffer, Podewski (eds.) *Proceedingsof the Sixth International Congress for Logic, methodology and Philosophy of Science*, pp.153–175, 1982.
- O'Hearn, P.W., Pym, D.J., The logic of bunched implications. Bulletin of Symbolic Logic, 5(02), 215–244, 1999.

References IV


Prawitz, D., *Natural Deduction*, Almqvist and Wiksell, Stockholm, 1965.


Primiero, G. (2015).
Realist Consequence, Epistemic Inference, Computational Correctness,
in A. Koslow, A. Buchsbaum (eds.), *The Road to Universal Logic, vol. II*,
Studies in Universal Logic, pp. 573-588, Springer International Publishing Switzerland.


Primiero, G., Information in the philosophy of computer science, in L. Floridi (ed.), *The Routledge Handbook of Philosophy of Information*, pp. 90–106, Routledge, 2016.