# NOTES ON TOPICS THAT HAVE PHILOSOPHICAL AND HISTORICAL ASPECTS

PARADIGMS AND PROGRAMS

The different programming language paradigms impact upon the nature of programs. Roughly, different paradigms would seem to generate different notions of program. Functional, procedural, object oriented and logical paradigms present different notions of what a program is and different notions of computation. This provides a philosophical and historical perspective on the question what is a program. The parallel, non-deterministic dimension also contributes to the notion of program. On the other hand, are there aspects of programs that transcend the paradigms?

COMPUTATIONS AND PROGRAMS

What is a physical computation? Putnam paper. The book *physical computation* and the SEP article good starting points. The arguments of the paper are central to any philosophical analysis of the notion of program. Physical Computation: A Mechanistic Account

Gualtiero Piccinini

CORRECTNESS AND PROGRAMS

Central to any analysis of programs- roughly change the notion of correctness and you change the notion of program. Mathematical and verification notions generate mathematical and empirical notions of program. Giuseppe tackles the problem by introducing different definitions that reflect different degrees of abstraction. At the bottom, the physical notion of correctness links back to the physical computation topic.

CHURCH TURING THESIS – THE CONTEMPORARY SCENE

Two dimensions:

1. Gandy and physical constraints- argues that physics sets constraints on computation and that these constraints lead to Turing machines.
2. Mathematical arguments for the CT thesis start with SEP article,

LOGIC AND COMPUTATION

Logic programming versus constructive programming. They have different notions of computation and program. Philosophical differences in underlying semantics of the specification rather than the programs. In constructive approach the type is the specification that has constructive content that is unpacked in terms of the programs that provide witness to the truth of the proposition. In the logic paradigm the logical assertion is both the specification and the program but is given different interpretations – with all that is entailed by calling it a specification. Namely as assertion it provides the correctness criteria for its guise as program. This is reflected in the difference between the truth conditional and the operational semantics.

Cluster: Logic

Title of the the chapter: Programming with proofs: the Curry-Howard correspondance

Content: The idea is to have an introduction to the Curry-Howard isomorphism showing the connection between proofs and programs. In particular the idea would be to let emerge the fact that use of different logic induces a use of a different style in programming. The chapter could be organized in the following way:

-) Introduction to pure lambda-calculus as an abstract functional programming language;

-) Possibility of defining “pathological” programs in pure lambda-calculus (like looping programs or never ending ones);

-) Use of a type system based on the logical implicational fragment in order to get always terminating programs (i.e. total functions). Correspondence between simple typed lambda-calculus and the implicational fragment in natural deduction;

-) Extension of the Curry-Howard correspondence to first-order logic and to arithmetic. Use of new functional programming instruction in order to decorate the rules of natural deduction.

-) Introduction of dependent type systems and extension of Curry-Howard correspondence to second order logic. This could be interesting to show that some of the “pathological” programs of pure lambda-calculus can now be typed (in the Girard-Reynolds system F), e.g. the program (x x).

-) Extension of the proofs-as-programs correspondence to the case of classical logic (presentation of Parigot’s lambda-mu and Krivine’s classical realizability). Use of imperative programming instruction in order to do this, namely control operators.

-) Translation of classical logic into intuitionistic logic and use of the continuation passing style (CPS) translations.

Possible authors: Jean-Baptiste Joinet, Simone Martini, Felice Cardone, Gilles Dowek, Alberto Naibo

(Otherwise, outside the people of the PROGRAMme group, maybe David Turner (Univ. of Kent) could be a good name. Well, also Philip Wadler would be a great name! ;-) )

——————

2nd proposal

Cluster: Logic/Programming languages and notations

Title of the chapter: Identifying algorithms and programs

(Comment: It seems to me that in the philosophical debate the notion of “algorithm” and that of “program” are often confused. This motivates such a chapter. Well, clearly this is also linked with my interests: the ANR project that I submitted last year with my colleague Thomas Seiller touched this topic.)

Content: The idea would be to compare the notion of algorithm and that of program, in order to understand common points and differences. The chapter could be organized in the following way:

-) Does the Church-Turing thesis capture only the notion of computable function or also that of algorithm? If it does not capture the notion of algorithm, as it is argued by people like Gurevich and Blass, what is an algorithm? (They argue, for instance, that interactive algorithms cannot be directly covered by the Church-Turing analysis.)

-) But what is an algorithm? Algorithms seems to be general procedures that can be used to solve a problem which do not depend from a specific language. Only later they can be implemented in some specific language and generate then a program. [Ok, I am very sketchy on this idea!!]

-) Would it then be possible to define algorithms by abstracting on programs? More precisely, do algorithm correspond to equivalence classes defined over the class of programs?

In this sense algorithms would be defined by an abstraction principle (similar to the one used by Frege for defining natural numbers) like:

the algorithm expressed by program P1 = the algorithm expressed by program P2 if and only if P1 and P2 are equivalent (with respect to some equivalence relation)

But which is this equivalence relation?

-) Is there a way to identify programs on an extensional way (i.e. by an equivalence relation)\_ Or is the notion of program a purely intensional one?

Possible authors:

-) Walter Dean (Univ. Warwick): even he is not involved in the PROGRAMme project, he is one of the expert of this topic (he is a philosopher, trained in maths and computer science).

-) Jean-Baptiste Joinet (Jean-Baptiste est en train de travailler en ce moment sur des questions liées aux définitions par abstraction.)

-) Giuseppe Primiero

-) Raymond Turner (?