What is CS?

Discuss nature of the subject. Does it exist?
Analogy with telescope or microscope: subject matter is determined by what the instrument reveals
Danger of seeing subject solely through the major applications
Might be viewed as having stimulated fields which have been dormant for years in which profound questions concerning psychological and philosophical basis of knowledge and intelligence are involved.
Possible parallel between general perception of computing today, and view in AI circles in 1960s: dangers of misconceptions about how much revolutionary progress can be expected
Problems of balanced perception at present: transitions
    Hardware developments
    Software system developments of unparallelled complexity
    Involvement of the advertisors

Background

Algorithms: Euclid, Gauss, Galois, Hilbert
Theories: Newton, Boole, Frege
Theorems: Turing, Godel

1950-60: -> Machines + Languages + Algorithms + Applications

1. FORTRAN compiler development:

Objectives - Develop a HHL for programming, and write a compiler

Consequences - innovations / development resulting
    (a) non-numeric applications
    (b) structural design of "large" programs (phases of a compiler)
    (c) identification/solution of syntactic problems
    Initiated proliferation of HHLs,
        search for a better general purpose programming language
        diversification of applications

1.1 Programming Language Development

Objective - Develop a general puprose language for programming
        Eliminate the difficulties of reasoning about algorithms
        at a low-level of abstraction

Consequences - Two aspects of the problem identified: design / implementation
    Principles of design lead to syntactic problems solution (ALGOL60)
        Data structures recognised as important
    Implementation leads to a caucus of important programming techniques
        hashing, parsing, tree-building, translating, sorting/searching
        Prompts deep questions about the semantics of PLs:
            how to identify and interpret meaningful programs
    Applications indicate that a general purpose paradigm may be difficult
        Trend towards monolithic languages: PL1, ALGOL68

1.2 Applications

Objectives: Develop hardware/software tools for specific uses

Consequences - Many different programming paradigms initiated
    AI -> LISP, computer engineering control applications -> APT
    graphics -> SKETCHPAD, linear algebra -> APL, business -> COBOL
    operating systems -> ??
        Became clear that no single paradigm for programming is
    appropriate for all applications, that there are many choices of

"virtual machine", that the theory of algorithms is very complex

Several really clever ideas in algorithms discovered; eg fact that there are many different algorithms for sorting/searching; new ideas in theorem-proving - resolution

Also became clear in AI particularly that issues of feasibility were very significant: had been anticipated that computers would outperform us much more dramatically in some tasks than is the case

1.1.1 Programming Language Semantics

Objective: Investigate how to design languages so that we can understand the meaning of any program within the language, and what formal models to use for reasoning about specific programs in a language

Consequences: Development of a variety of different mathematical models for modelling programs. Algebra and Logic become significant part of the mathematical foundation of programming. Recursive function theory, lambda calculus, algebraic theories for data types.

Primary mathematical achievement: Scott-Strachey semantics

Paramount importance of having a good semantic model and difficulty of giving semantics to complex procedural language recognised

Idea of developing a language from the intended semantic model first conceived: the motivating idea behind functional languages

1.3 Programming Methodology

Objective: How do we write correct programs? How do we communicate about program design? How do we document programs / program designs?

Consequences: Recognition that proving correctness of algorithms formally is hard. Clear that a simple programming language helps cf Dijkstra's guarded commands vs Algol 68. Particular difficulties presented by concurrent programs first encountered. Importance of program structure first recognised. No "goto"s, appropriate abstractions essential ideas behind "structured programming".

Concept of invariants as device for proving correctness of a procedural program. Development of programming logics, specifically for reasoning about programs.

Management aspects of software design task first considered

2.1 Theory of Algorithms

Objective: How do we recognise a good algorithm? How do we write efficient algorithms? What is the impact of choosing different primitive actions as a computational basis? When does a computational problem have a feasible solution?

Consequences: Revival of interest in mathematical theory of algorithms

Investigation of upper / lower bounds on computational complexity of problems. Surprises: eg matrix multiplication by Strassen, many traditional algorithms unexpectedly subtle (coefficient growth), lower bounds very hard to prove even when non-constructive proofs indicate that many or most instances are hard.

Cook's Theorem indicates that several important algorithmic problems are essentially equivalent in complexity. Identification of computational hierarchies.

Ground rules under which computations are performed seen to be very significant. Search for satisfactory machine models for evaluating the "real" complexity of a practical algorithm initiated.

Practical importance of parallel, approximate, probabilistic algorithms recognised.

Contemporary view

Increasingly has become clear that the role which computers will play in the future depends upon broader issues than have been addressed hitherto.

The complexity of computer hardware now such that conventional methods of designing software fail to exploit its apparent potential. This is the case even if we approach the design task in a relatively informal way, not seeking to justify our systems rigorously in every detail. Ironically, it is more than ever evident that very large systems can only be designed with appropriate formal methods. The nature, and degree of formality, of these methods is a particularly topical issue. Might see this as analogous to the position of an engineer who recognises that the theory of materials is hazy and incomplete, but still wishes to investigate structural theories.

At the high-level of abstraction characteristic of contemporary research into applications, have strong connections with other disciplines.

AI: expert systems, theories of knowledge, psychology/philosophy
human-computer interaction, object-oriented paradigm,
special purpose applications using semantic elements

SE: enormous software systems, design tools, management problems
data bases as unifying resources
distributed systems, communicating sequential processes

New Architectures: Problems with VN machines
Alice, Flagship
Transputers, highly parallel systems
VLSI and special purpose machines

Robotics: Real-time programming
Links with AI, sensory devices
Control engineering, simulation, vision

CSE Systems — increasingly blurring between hardware/software design. Shared VLSI designs, enormous software component, abstract description of circuits
Image processing.
Glass annealing solution to travelling salesman