What's a Computational Constraint?

One possible answer to the question in the title is simple: anything that narrows down a computation is a computational constraint. It might however trivialize the issue because everything begins to look like a computational constraint, for we know since the work of Bennett (1973) that all computations including the classical Turing computations are reversible; since Feynman (1982) that all reversible physical processes may be computable, which is to say quantum mechanics—hence the whole natural world—is computable; and since Deutsch (1985) that quantum computations satisfy his Church-Turing Principle: all finitely realizable physical systems can be perfectly made computational by finite means.

The simple answer gives us no understanding of the differences we make, either implicitly or explicitly at some level of abstraction, among functional, constitutive, cognitive, logical and automatic constraints (as opposed to interactive), since by this way of thinking they will have to boil down to a computational constraint sooner or later. Down this path lies physicalism to the point of eliminative materialism of Churchlands (e.g. 1998), but an alternative is that we stop midway, like Dennett's (1989) intentional stance, or Marr's (1977) levels, and try to be explicit and predictive about the internal mechanisms that might be causing the perceived (or perhaps intuitive) differences among the nature of the constraints mentioned above. For computational constraints, we suggest that the right level to stop is automata, more precisely, classes of automata.

From this perspective, not all constraints are computational, and it may serve well not to consider computational ones as something else. Take for example the so-called functional constraints on language, the kind Hawkins (1994) identifies as some word orders causing "processing difficulties" because they purportedly make it difficult to connect earlier constituents in parsing, which is supposedly a function, to mother nodes in a tree of constituents. Judging from the terminology, it appears to be a computational constraint. But no automata class is singled out by this constraint. The theory itself is not constructed in a grammar formalism that spells a certain class of automata either (finite-state, push-down, embedded push-down etc.). We have no grounds to see what kind of computations is left out by the constraint, and we are left with an impression that parsing is a performance function. However, children parse to learn, rather than learn to parse (Fodor, 1998), and there are computational constraints on it. For example, Joshi's (1990) constraints are computational in this sense. A truly functional constraint would not be computational. (There are functional constraints on language, such as its use in communication, topic versus new information in sentences, but parsing is probably not one of them, and the automaton perspective makes it clear.)

Some constraints presented as cognitive may turn out to be computational as well. For example, Tomasello and Call (1997) first argued that chimpanzees don't have a mind, but their later work suggested they might have a mind, but not to the extent of human psychological states and understanding of beliefs (Tomasello et al., 2003). Chimpanzees can do instrumental plans in which their actions are well organized and depend on knowledge states, accomplishments, or failures of other agents, but they don't seem to be able to do recursive plans which humans are capable of. Assuming that instrumental plans can be embedding, which is to say they might correspond to a push-down (stack) automaton with some limits on embedding and self-embedding, this class of

automata would be distinct from stack of stacks, which seems to be what is required for collaborative multi-agent plans of humans. (We know that this is minimally required for human languages and for their recursion in the limit; see Shieber 1985; Joshi 1985.) At the formal descriptive level, a simple stack gives us context-freeness, whereas a stack of stacks gives us linear-indexed grammars (Gazdar, 1988), with limited extension of possible computations. Notice that here the term *grammar* does not necessarily mean a linguistic grammar, but a tool to model organized behavior in limited ways including plans, vision, music, etc. It is the descriptive counterpart of a class of automata. If it turns out that this way of thinking might suggest experiments which can elucidate stack of stacks versus simple stack constraint in various manifestations of complex behavior in chimpanzees, then conceiving the chimpanzee mind with a computational constraint serves its scientific purpose.

Automatic computations and interactive ones can be compared from this perspective as well. Turing (1936) called the first kind a-machines and the second c-machines (c for 'choice'). Oracles (o-machines; Turing 1939) differ from both. They need not specify an internal i.e. computational mechanism.

Consider the difference between calculating an integral within an interval and playing backgammon. There is no external element in the first one that would force the symbols of its "computations" to communicate with the computer and/or the outside world. For backgrammon there is an external element, the dice. The dice throw, however, can be incorporated into a-machine configurations, say by having another tape to enumerate sequence of dice throws long enough to serve any finite game. If this a-machine must decide whether the same configuration reached several times in a game warrants continuation, asking this question puts a computational constraint on the game, unlike asking for a dice throw, because the automata class changes from a-machines to c-machines, with concomitant results such as potential undecidability. As an open system in this sense, it is constrained descriptively by whatever is delivered by the dice, and computationally by an interaction. For integral calculation, if e.g. switching from integers to reals could change the complexity of the calculation, then we would consider the integer/real constraint a computational one.

There are more examples where rethinking purportedly computational constraints as something else, and vice versa, might lead to a new understanding of the phenomenon under study. In the end the question arises about the choice of automata being the right level to call a constraint computational. Why not other criteria? We think the answer is that only an explicit and formal definition of computation can give us a yardstick to check whether something is computational or not, and the best one we have got is Turing's (1936) definition, either in the classical sense, or in Deutsch 1985-style quantum model, both of which are precise enough to go back and validate. (The latter does not give a new class of computable functions but new complexity classes for the computable ones.) Either way, we are faced only with the underlying automata as the algorithm, and with the corresponding hierarchy of formal descriptive devices (grammars) as encoders of further computational constraints (features, categories, vocabulary etc.) on an identified class of automata, which is essentially a Herbrand universe (finite or infinite, with foreseeable predictions for either case).

A computational constraint then is manifested in two ways: (i) what automata class is identified to narrow down the space of possible computations, (ii) what further descriptive devices can be

imposed on the class of automata via grammars to further reduce the class of possible structures computable by the automata.

References

Bennett, C. H. (1973). Logical reversibility of computation. *IBM journal of Research and Development*, 17(6):525–532.

Churchland, P. M. and Churchland, P. S. (1998). *On the contrary: Critical essays, 1987-1997*. MIT Press.

Dennett, D. C. (1989). *The intentional stance*. MIT press.

Deutsch, D. (1985). Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117.

Feynman, R. P. (1982). Simulating physics with computers. *International journal of theoretical physics*, 21(6):467–488.

Fodor, J. D. (1998). Parsing to learn. *Journal of Psycholinguistic research*, 27(3):339–374.

Gazdar, G. (1988). Applicability of indexed grammars to natural languages. In Reyle, U. and Rohrer, C., editors, *Natural Language Parsing and Linguistic Theories*, pages 69–94. Reidel, Dordrecht.

Hawkins, J. A. (1994). *A Performance Theory of Order and Constituency*. Cambridge University Press, Cambridge.

Joshi, A. (1985). How much context-sensitivity is necessary for characterizing complex structural descriptions—tree adjoining grammars. In Dowty, D., Karttunen, L., and Zwicky, A., editors, *Natural Language Parsing*, pages 206–250. Cambridge University Press, Cambridge.

Joshi, A. (1990). Processing crossed and nested dependencies: An automaton perspective on the psycholinguistic results. *Language and Cognitive Processes*, 5:1–27.

Marr, D. (1977). Artificial intelligence: A personal view. *Artificial Intelligence*, 9:37–48.

Shieber, S. (1985). Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8:333–343.

Tomasello, M. and Call, J. (1997). *Primate Cognition*. Oxford University Press.

Tomasello, M., Call, J., and Hare, B. (2003). Chimpanzees understand psychological states—the question is which ones and to what extent. *Trends in Cognitive Sciences*, 7(4):153–156.

Turing, A. M. (1936). On computable numbers, with an application to the entscheidungsproblem. *Proc. of the London Mathematical Society*, 42 (series 2):230–265.

Turing, A. M. (1939). Systems of logic based on ordinals. *Proceedings of the London Mathematical Society*, 2(1):161–228.