Constructive proof of Kannan's theorem based upon a much simpler construction

Sunny Daniels
e-mail: sdaniels@lycos.com

Abstract:

I appreciate the acknowledgement by Fortnow[1] of my, historically significant, I believe, first ever constructive proof of Kannan's theorem[2], which I think was acknowledged as "folklore" (I thank Watanabe for this) in Watanabe[3] (I don't think I currently have access to any university library system so I don't think I currently have access to the full text of this). If I am not mistaken, I now have a much simpler construction (I thought of this years ago if I remember rightly but was a bit too busy with an IT job unrelated to complexity theory until recently to have the time to publish it) giving a constructive proof of Kannan's theorem, based upon a particular type of universal Turing machine. The fact that this construction (if I am not mistaken) works is an easy consequence of either my earlier constructive proof[2] or Watanabe's later proof of the same result as discussed by Fortnow[1]. (I know that my presentations here of this construction and proof are quite sketchy at present; I would appreciate some feedback from others before attempting to write up more detailed versions of them). Also, I think it can be used as the basis of a much simpler constructive proof of Kannan's theorem: I also discuss this here.

Introduction:

I appreciate the acknowledgement by Fortnow[1] of my, historically significant I believe, first ever constructive proof of Kannan's theorem[2], which I think was acknowledged as "folklore" (I thank Watanabe for this) in Watanabe[3] (I don't think I currently have access to any university library system so I don't think I currently have access to the full text of this).

If I am not mistaken, I now have a much simpler construction (I thought of this years ago if I remember rightly but was a bit too busy with an IT job unrelated to complexity theory until recently to have the time to publish it) giving a constructive proof of Kannan's theorem, based upon a particular type of universal Turing machine.

The Construction Itself:

1) The Σ_2^P machine in question is a universal alternating Turing machine that is both time-bounded and alternation-bounded, in a particular way.

2) An example of an input that it could possibly accept is:

111111aaabbbaabbaba111000111

depending upon the outcome of the simulation of the Turing

machine, time bound polynomial input and Turing machine input represented by this input string.

2) In general, the machine uses the set of tape symbols {blank, 0, 1} and will certainly reject unless the input is of the form: encoding_of(xyz)

Where:

2a) x is a string of 1s for the input to the time bound polynomial (length of y gets added on before input to time bound polynomial) which we will discuss later.

2b) y is representation of control unit of tm being simulated, as as and bs rather than 1s and 0s (to avoid need for separators from x and z).

z is input to machine being stimulated.

The following encoding of the input xyz as 0s and 1s would work for the function that we call encoding_of() above:

0 by 00
1 by 11
a by 01
b by 10

3) The alternation count of the machine being simulated is capped at 1: existential first then universal. This alternation count is kept in a variable somewhere on the tape of the simulating machine. If the simulated machine tries to enter a second existential state then the simulating machine will halt and reject. This behaviour of the simulating machine, together with the polynomial time bound condition below, if I am not mistaken, ensures that the simulating machine is in Σ_2^P .

3) A polynomial time bound P of the simulation (as function of x length plus y length) is hard coded into simulator machine, which we will call M(2,P). Recall that x is a string of 1s in the input to the simulating machine; y is an encoding of the simulated machine's control unit as as and bs. The number of steps (where each step is a control unit state transition and / or a tape read or write and / or a tape motion; I think that there is a standard definition of this somewhere!) taken in the simulated machine by the simulator machine is kept in a variable somewhere on the tape of the simulator machine. If this step count exceeds P(x length + y length) then the simulator machine halts and rejects. **Note that this is a generalization of the apparently "standard" (I don't have a reference to it in the literature; I would be happy to be made aware of one) construction of a universal Σ_2^P Turing machine described to me years ago by Allender (Professor Eric Allender of Rutgers) in which the polynomial P is just the trivial polynomial (commonly written as "x", although we are using "x" to mean**

something else here) that, when treated as a function, outputs the number input to it. I think that the fact that the "standard" polynomial-time universal Turing machine construction described by Allender can be generalized in this way to give a construction for a constructive version of Kannan's theorem has never been previously clearly recognised!

Proof that the Construction Works:

So if our machine M(2,P) can be simulated by a CCT family C of size O(n^99) (say), then we can connect inputs of C as follows, for any given Σ_2 machine N (that is any alternating Turing machine N that starts off in the existential state and then does at most one alternation, that alternation if it happens taking N into the universal state):

(We will refer back to this point in the paper later. We will call it ***HARDWIRING CONSTRUCTION***).

x inputs can be hardwired to string of 1s of same length as z minus length of y (or zero if this subtraction result is negative).

y inputs are hardwired to representation of N.

z inputs are hardwired to input to N.

So if input size is sufficiently large then the resulting cct family should be of size big oh of, if m is the length of z:

(4m)^99 = 4^99m^99

Which is big oh of m^99. 4 is because we are replacing 1s and 0s with pairs of 1s and 0s and then in worst case doubling this again by concatenating x y and z.

And length of x is hardwired such that input to P time bound of simulator machine is always at least length of input z to machine being simulated. We assume P is monotonic (all positive coefficients should be sufficient to ensure this I think) so polynomial time bound of simulation should be at least P(length of z). So simulation should run to completion (not be terminated early by time bound) provided that P is big enough to accommodate running time of machine being stimulated for all possible input sizes, irrespective of whatever value we put in place of 99. (But the degree and coefficients of P *will*, in general, depend upon the value that we put in place of 99).

So setting P big enough to accommodate the running time of my machine in [2] (for n^99 circuits) or Cai and Watanabe's sort-of equivalent machine (for n^99 circuits) for ALL input sizes (not just all sufficiently large input sizes) should give circuit family for my [2] machine or Cai and Watanabe's sort-of equivalent

machine contradicting the fact that these machines are constructed in such a way as to provably not have such a cct family: contradiction!

I think that any other exponent >= 2 should be able to be used for the cct size: cool IMHO!

My idea of using 99 for the exponent comes from the cmu complexity lecturer whose name I have temporarily forgotten (I recently saw the lectures online in YouTube).

First Conclusion:

So this new construction (for a constructive proof of Kannan's theorem) is much simpler than either my old construction in [2] or Cai and Watanabe's sort-of equivalent construction. So I think that this new, much simpler, construction is an important result by Occam's razor!

Conjectured Possible Next Step: Replace [2] Constructions Completely

Also: I think that I can probably replace either my construction in [2] or Cai and Watanabe's sort-of equivalent construction by a simpler layer by layer proof as follows:

Existence of n^99 cct family for any given polynomial simulation machine (as we define) for any time bound polynomial would imply existence of n^99 cct family for lower time bound polynomial Q (some simple relationship between the two time bound polynomials would exist I think) for analogous universal tm construction with alternation count capped at 3 rather than 2, which we call M(3,Q).

The reason is that simulation machine could be used to simulate SAT or tautology machine implying existence of SAT and tautology cct families of big oh of n^99 size (by hardwiring inputs to simulation machine in analogous way to ***HARDWIRING CONSTRUCTION** above): this could be used to get rid of last alternation in simulated universal three alternation (Karp Lipton type construction) machine: cool!

By "SAT or tautology" machine I mean a single machine that computes either sat or tautology (the former is in NP I believe, latter co-NP I believe) depending upon the first bit of its input: remainder of input is input to SAT or tautology problem.

Clearly this "SAT or tautology" problem is in Σ_2^P .

The analogous result holds between this three alternation construction and analogous four alternation construction M(4,R) for the appropriate time bound R.

Then the four alternation construction can be used to simulate

Kannan's Σ_4^P machine directly if I am not mistaken.

This is a sketchy proof at present: details need to be worked out.

Also, doing it for n^99 gives exponents in running time bound increase here that sound extreme, although I think that the whole thing is still polynomial time: trying lesser exponent as example would give a more plausible looking proof IMHO.

Second Conclusion:

If this proposed simplified proof of Kannan's theorem works, I believe that it will also be good by Occam's razor!

References:

[1] https://blog.computationalcomplexity.org/2014/08/sixteen-years-in-making.html

[2] A constructive proof presenting languages in Σ_2^P that cannot be decided by circuit families of size n^k : *arXiv: 1408.6334*

[3] Watanabe, O. (2014). On the Limit of Some Algorithmic Approach to Circuit Lower Bounds. In: Calude, C., Freivalds, R., Kazuo, I. (eds) Computing with New Resources. Lecture Notes in Computer Science(), vol 8808. Springer, Cham. https://doi.org/10.1007/978-3-319-13350-8_29