COMPUTER IMPLICATION AND CURRY'S PARADOX

WAYNE AITKEN, JEFFREY A. BARRETT

Abstract. There are theoretical limitations to what can be implemented by a computer program. In this paper we are concerned with a limitation on the strength of computer implemented deduction. We use a version of Curry's paradox to arrive at this limitation.

1. Introduction

Computers can be used to check properties. Indeed, properties of computer programs might themselves be checked by appropriately designed programs. As is well-known there are limits on what can be done. Testing for the halting property is one of these. In this paper we are concerned with another limitation on what properties computers can check. Here we are interested in a limitation on the strength of computer-implemented deduction.

An *implication program* is a program that takes as input two statements concerning the behavior of programs, then tries to deduce, by way of a specified library of rules, the second statement from the first. It keeps looking until it finds a deduction, thus it may never halt. If it does find a deduction, it halts and outputs 1 to signal that a proof has been found. An implication program can be used to prove statements involving the implication program itself. Since recursive and partially recursive functions can be implemented as computer programs, an implication program might also be used to prove relationships between such functions. Throughout we assume that programs are written in a fixed language for a computer with unlimited memory.

In this paper we point out a limitation on implication programs, namely that *no sufficiently powerful implication program can incorporate an unrestricted form of modus ponens.* More generally, we show that there are valid rules of inference that can be defined algorithmically, but cannot be used by a given implication program. Here we present an informal version of the argument. A formal, more rigorous, and consequently much longer version will be given in a future paper. The future paper will be axiomatic and prove the result not just for computer programs or Turing machines, but for more general partial combinatory structures. The future paper will also discuss the relationship between our results and John Myhill's levels of implication [3].

2. Statements

A *statement* is a list $[prog, in, out]$ where *prog* is a program considered as data, *in* is an input for *prog*, and *out* is an anticipated output. A statement $[prog, in, out]$

Date: August 17, 2003.

is called *true* if the program *prog* halts with input *in* and output *out*. A statement which is not true is called *false*.

There is a program to check but not test whether [*prog*, *in*, *out*] is a true statement. Given [*prog*, *in*, *out*] as an input, it first runs *prog* as a subprocess with input *in*. If and when *prog* halts, it compares the actual output with *out*. If they match then the program outputs 1; if they do not match, our program does something else (say, outputs 0). This program will output 1 if and only if [*prog*, *in*, *out*] is true, but it might not halt if [*prog*, *in*, *out*] is false. Due to the halting problem, no program can check for falsity.

In what follows, we continue to use 1 to signal a positive result. We use 0 to signal a negative result, but failure to halt also indicates (but does not signal to a real user) a negative result.

3. Valid Rules

A *rule* is a program that takes as input a list of statements and outputs a list of statements. For convenience we require that the output list include the input list as a sublist, and that the rule halt for all input lists. A *valid rule* is a rule with the property that whenever the input list consists of all true statements, the output list also consists of all true statements.

Consider the program AND which expects as an input a list of two statements $[A, B]$. The program first checks the truth of A in the manner indicated above. If it determines that A is true, then it checks the truth of B. If B is also true, it outputs 1. If either A or B is false, AND fails to halt.

An example of a valid rule concerning AND is the program AND-ELIM. This program expects as input a list of statements (if the input is not of this form then AND-ELIM just outputs 0). It looks for statements of the form $\big[\textsc{and}, [A, B], 1\big]$ on the input list. If no such statement is found, AND-ELIM outputs the list which was given as an input. Otherwise, AND-ELIM creates an output list consisting of every statement on the input list plus, for each statement of the form $\big[\textsc{and}, [A, B], 1\big]$ appearing on the input list, the two statements A and B. Given the definition of AND, if all the statements on the input list are true and if $\big[\textsc{and}, [A, B], 1\big]$ is on the input list, then A and B are also true. So the program AND-ELIM only adds true statements to the input list and hence is a valid rule.

4. An Implication Program

A *library* is the list of rules used by an implication program in its proofs. We assume here that the library is finite at any given time. A *valid library* is one that contains only valid rules.

Consider the implication program $\Rightarrow$ defined as follows. The program $\Rightarrow$ expects as input a list of two statements $[A, B]$. Then it sets up and manipulates a list of sentences called the *consequence list*. The consequence list begins as a singleton list consisting only of A. The program $\Rightarrow$ then goes to the library and chooses a rule. It applies the rule to the consequence list, and the result becomes the new consequence list. Since rules are required to include the input list as a sublist of the output list, once a statement appears on any consequence list it will appear on all subsequent consequence lists. After applying a rule, the program $\Rightarrow$ checks whether the consequent B is on the new consequence list. If so, it outputs 1; otherwise it chooses another rule, applies it to update the consequence list, and checks for B

on the new consequence list. It continues to apply the rules in an exhaustive way until B is found, in which case $\Rightarrow$ outputs 1. If the consequent B is never found, the implication program $\Rightarrow$ does not halt.

It does not matter how the program $\Rightarrow$ selects rules to apply to the consequence list as long as each rule is repeatedly applied. The program $\Rightarrow$ might, for example, use the sequence $1, 1, 2, 1, 2, 3, 1, 2, 3, 4, \ldots$ to determine which order to apply the rules. This sequence directs $\Rightarrow$ to first use the first rule in the library, then the first rule again, then the second rule in the library, etc. If it is time to use the nth rule, but there is no nth rule in the library, then it just ignores that step and goes to the next.

If one adds new rules to the library, a currently running implication program might be designed to dynamically incorporate the new rules in its search for the consequent. An implication program need not have this feature for the considerations below to hold. But adding new rules and using the sequence described above allows one to consider a countable infinite number of rules if needed.

5. Modus Ponens

Consider *Modus Ponens Program* MP. The program MP expects as an input a list of statements. It begins by forming an empty *result list.* It then searches the input list for all statements of the form $\big[\Rightarrow, [A, B], 1\big]$ where A and B are statements. For each such statement, it searches to see if A also appears as a statement on the input list. If A is found, then MP adds B to the result list. After constructing the result list, the program MP outputs a list consisting of all the statements in the input list followed by all the statements of the result list (if any).

The program MP is a rule since its output list contains every statement on its input list. A rule is valid if, for an input list of true statements, it only adds true statements. From the definition of $\Rightarrow$, if $\big[\Rightarrow, [A, B], 1\big]$ and A are on the input list and if they are both true and if the library is valid, then B will be true. So MP *is a valid rule if the library used by* $\Rightarrow$ *is valid.*

6. The Curry Program

Some statements are clearly false. Consider the program EQ. It expects as input a list $[m, n]$ of two natural numbers. If $m = n$ then EQ outputs 1, otherwise it outputs 0. Let FALSE be the false statement $\big[\text{EQ}, [0, 1], 1\big]$.

Consider the program CURRY defined as follows. It expects a program X as input. Then it runs $\Rightarrow$ as a subprocess with input $\big[[X, X, 1], \text{FALSE}\big]$. The output of the subprocess (if any) is then used as the output of CURRY.

If X checks for a particular property of programs, then the statement $[X, X, 1]$ asserts that the program X has the very property for which it checks. The program CURRY when applied to program X can be thought of as trying to find a proof by contradiction that the statement $[X, X, 1]$ does not hold.

7. The Ad Hoc Rule

From the definition of CURRY, the only way that CURRY could output 1 with input X is if $\Rightarrow$ outputs 1 with input $\big[[X, X, 1], \text{FALSE}\big]$. In other words, if $[\text{CURRY}, X, 1]$ is a true statement, then so is $\Big[\Rightarrow, \big[[X, X, 1], \text{FALSE}\big], 1\Big]$. This is the idea behind the Ad Hoc Rule.

The *Ad Hoc Rule* AH expects as input a list of statements. It begins by producing an empty result list. Then it checks its input for statements of the form $[\textsc{curry}, X, 1]$ where X is a program. For each such statement on the input list, AH adds the statement $\big[\Rightarrow, \big[[X, X, 1], \textsc{false}\big], 1\big]$ to the result list. When AH is finished constructing the result list, it outputs a list consisting of the statements in the input list followed by the statements of the result list (if any).

If the statements on the input list are true, then AH only adds true statements to form the output list. So AH *is a valid rule.*

The valid rule AH is *ad hoc* since it is designed specifically for CURRY. With a little more work, the same effect can be achieved by using a collection of more natural, logically motivated rules, each independent of CURRY. This more natural approach is pursued in the promised future paper.

8. Curry's Paradox

We now describe a version of Curry's paradox[1] for the implication program $\Rightarrow$. We assume that the library is valid and contains the Modus Ponens Rule MP and the Ad Hoc Rule AH.

Consider what happens when we run $\Rightarrow$ with input $\big[[\textsc{curry}, \textsc{curry}, 1], \textsc{false}\big]$. First a consequence list containing the statement $[\textsc{curry}, \textsc{curry}, 1]$ is set-up. Next rules from the library are applied to the consequence list. At some point the Ad Hoc Rule AH is applied and, since $[\textsc{curry}, \textsc{curry}, 1]$ is on the consequence list, $\big[\Rightarrow, \big[[\textsc{curry}, \textsc{curry}, 1], \textsc{false}\big], 1\big]$ is added to the consequence list. Because of this, when MP is next applied to the consequence list, FALSE will be added to the list. Since the initial input had the statement FALSE as the second item on the input list, $\Rightarrow$ will halt with output 1 when FALSE appears on the consequence list.

So $\Rightarrow$ outputs 1 with input $\big[[\textsc{curry}, \textsc{curry}, 1], \textsc{false}\big]$. From the definition of CURRY, this implies that CURRY will output 1 when CURRY is given as an input. That is, the statement $[\textsc{curry}, \textsc{curry}, 1]$ is true.

Consider again what happens when $\Rightarrow$ is applied to $\big[[\textsc{curry}, \textsc{curry}, 1], \textsc{false}\big]$. Since the antecedent $[\textsc{curry}, \textsc{curry}, 1]$ is true, every statement added to the consequence list is also true. Above we noted that FALSE will eventually appear on the consequence list. Therefore, FALSE is true, a contradiction.

Curry's paradox typically arises in logical systems designed to capture natural inference, and in such cases there are several assumptions one might question. But here the paradox occurs in the concrete setting of perfectly well-defined programs and seemingly careful reasoning about their expected behavior. So what went wrong?

9. The Resolution

We assume above that the inference library is valid *and* that it contains the Modus Ponens Rule MP. We showed that if the library is valid, then the rule MP is in fact valid, *but this does not mean that* MP *is in the valid library.* Indeed, the argument above can be regarded as a proof (by contradiction) that MP is *not* in any valid library containing the rule AH (or containing rules sufficiently strong to mimic AH).

[1] Named for its use in [2]. See [1] for more information.

We conclude from this that, given an implication program, there are valid inference rules (including the associated Modus Ponens Rule MP) that are valid only so long as they are not included in the library of rules used by the implication. Informally, we can say that *there are valid rules that one is not allowed to use (in an unrestricted manner) in one's proofs*.

In the discussion above we allowed for the possibility of an open library; that is, a library where valid rules can be added as developed. It is now clear that a rule's validity is not itself a sufficient condition for it being safe to add it to a valid library. Rather, in order to maintain a valid open library, one must check that the rule is valid and that it remains valid when added to the library. Call a rule *independently valid* if it is valid regardless of which library is used by the implication program. Examples of independently valid rules include AH and AND-ELIM. Clearly, *any library consisting only of independently valid rules is valid.*

The rule MP is not independently valid; its validity is contingent on the nature of the library. In fact, Curry's paradox gives examples of libraries for which MP is not valid. Our proof that MP was valid only works for some libraries, namely the valid libraries.

References

[1] Beall, JC, *Curry's Paradox*, The Stanford Encylopedia of Philosophy (Spring 2001 Edition), Edward N. Zalta (ed.).

`URL = http://plato.stanford.edu/archives/spr2001/entries/curry-paradox/`

[2] H. Curry, *The inconsistency of certain formal logics*, Journal of Symbolic Logic **7** (1942), 115–117.

[3] J. Myhill, *Paradoxes*, Synthese **60** (1984), 129–143.

Cal. State, San Marcos, CA 92096, USA
E-mail address: `waitken@csusm.edu`

UC Irvine, Irvine, CA 92697, USA
E-mail address: `jabarret@uci.edu`