McCarthy 91 function

The **McCarthy 91 function** is a recursive function, defined by the computer scientist John McCarthy as a test case for formal verification within computer science.

The McCarthy 91 function is defined as

$$M(n) = \begin{cases} n - 10, & \text{if } n > 100 \\ M(M(n + 11)), & \text{if } n \leq 100 \end{cases}$$

The results of evaluating the function are given by $M(n) = 91$ for all integer arguments $n \leq 100$, and $M(n) = n - 10$ for $n > 100$. Indeed, the result of M(101) is also 91 (101 - 10 = 91). All results of M(n) after n = 101 are continually increasing by 1, e.g. M(102) = 92, M(103) = 93.

Contents

History

The 91 function was introduced in papers published by Zohar Manna, Amir Pnueli and John McCarthy in 1970. These papers represented early developments towards the application of formal methods to program verification. The 91 function was chosen for being nested-recursive (contrasted with single recursion, such as defining $f(n)$ by means of $f(n-1)$). The example was popularized by Manna's book, *Mathematical Theory of Computation* (1974). As the field of Formal Methods advanced, this example appeared repeatedly in the research literature. In particular, it is viewed as a "challenge problem" for automated program verification.

It is easier to reason about tail-recursive control flow, this is an equivalent (extensionally equal) definition:

$$M_t(n) = M'_t(n, 1)$$

$$M'_t(n, c) = \begin{cases} n, & \text{if } c = 0 \\ M'_t(n - 10, c - 1), & \text{if } n > 100 \text{ and } c \neq 0 \\ M'_t(n + 11, c + 1), & \text{if } n \leq 100 \text{ and } c \neq 0 \end{cases}$$

As one of the examples used to demonstrate such reasoning, Manna's book includes a tail-recursive algorithm equivalent to the nested-recursive 91 function. Many of the papers that report an "automated verification" (or termination proof) of the 91 function only handle the tail-recursive version.

This is an equivalent mutually tail-recursive definition:

$$
\begin{aligned}
M_{mt}(n) &= M'_{mt}(n, 0) \\
M'_{mt}(n, c) &= \begin{cases} M''_{mt}(n - 10, c), & \text{if } n > 100 \\ M'_{mt}(n + 11, c + 1), & \text{if } n \leq 100 \end{cases} \\
M''_{mt}(n, c) &= \begin{cases} n, & \text{if } c = 0 \\ M'_{mt}(n, c - 1), & \text{if } c \neq 0 \end{cases}
\end{aligned}
$$

A formal derivation of the mutually tail-recursive version from the nested-recursive one was given in a 1980 article by Mitchell Wand, based on the use of continuations.

Examples

Example A:

```
M(99) = M(M(110)) since 99 ≤ 100
      = M(100)    since 110 > 100
      = M(M(111)) since 100 ≤ 100
      = M(101)    since 111 > 100
      = 91        since 101 > 100
```

Example B:

```
M(87) = M(M(98))
      = M(M(M(109)))
      = M(M(99))
      = M(M(M(110)))
      = M(M(100))
      = M(M(M(111)))
      = M(M(101))
      = M(91)
      = M(M(102))
      = M(92)
      = M(M(103))
      = M(93)
   .... Pattern continues increasing till M(99), M(100) and M(101), exactly as we saw on the
example A)
      = M(101)    since 111 > 100
      = 91        since 101 > 100
```

Code

Here is an implementation of the nested-recursive algorithm in Lisp:

```lisp
(defun mc91 (n)
  (cond ((<= n 100) (mc91 (mc91 (+ n 11))))
        (t (- n 10))))
```

Here is an implementation of the nested-recursive algorithm in Haskell:

```haskell
mc91 n
  | n > 100   = n - 10
  | otherwise = mc91 $ mc91 $ n + 11
```

Here is an implementation of the nested-recursive algorithm in OCaml:

```
let rec mc91 n =
  if n > 100 then n - 10
  else mc91 (mc91 (n + 11))
```

Here is an implementation of the tail-recursive algorithm in OCaml:

```
let mc91 n =
  let rec aux n c =
    if c = 0 then n
    else if n > 100 then aux (n - 10) (c - 1)
    else aux (n + 11) (c + 1)
  in
  aux n 1
```

Here is an implementation of the nested-recursive algorithm in Python:

```
def mc91(n: int) -> int:
    """McCarthy 91 function."""
    if n > 100:
        return n - 10
    else:
        return mc91(mc91(n + 11))
```

Here is an implementation of the nested-recursive algorithm in C:

```
int mc91(int n)
{
    if (n > 100) {
        return n - 10;
    } else {
        return mc91(mc91(n + 11));
    }
}
```

Here is an implementation of the tail-recursive algorithm in C:

```
int mc91(int n)
{
    return mc91taux(n, 1);
}

int mc91taux(int n, int c)
{
    if (c != 0) {
        if (n > 100) {
            return mc91taux(n - 10, c - 1);
        } else {
            return mc91taux(n + 11, c + 1);
        }
    } else {
        return n;
    }
}
```

Proof

Here is a proof that

$$M(n) = \begin{cases} n - 10, & \text{if } n > 100 \\ 91, & \text{if } n \leq 100 \end{cases}$$

which provides an equivalent non-recursive algorithm to compute M.

For $n > 100$, equality follows from the definition of M. For $n \leq 100$, a strong induction downward from 100 can be used.

For $90 \leq n \leq 100$,

```
M(n) = M(M(n + 11)), by definition
     = M(n + 11 - 10), since n + 11 > 100
     = M(n + 1)
```

So $M(n) = M(101) = 91$ for $90 \leq n \leq 100$. This can be used as base case.

For the induction step, let $n \leq 89$ and assume $M(i) = 91$ for all $n < i \leq 100$, then

```
M(n) = M(M(n + 11)), by definition
     = M(91), by hypothesis, since n < n + 11 ≤ 100
     = 91, by the base case.
```

This proves $M(n) = 91$ for all $n \leq 100$, including negative values.

Knuth's generalization

Donald Knuth generalized the 91 function to include additional parameters.[1] John Cowles developed a formal proof that Knuth's generalized function was total, using the ACL2 theorem prover.[2]

References

1. Knuth, Donald E. (1991). "Textbook Examples of Recursion". *Artificial Intelligence and Mathematical Theory of Computation*: 207–229. arXiv:cs/9301113 (https://arxiv.org/abs/cs/9301113). Bibcode:1993cs.....1113K (https://ui.adsabs.harvard.edu/abs/1993cs.....1113K). doi:10.1016/B978-0-12-450010-5.50018-9 (https://doi.org/10.1016%2FB978-0-12-450010-5.50018-9). ISBN 9780124500105.
2. Cowles, John (2013) [2000]. "Knuth's generalization of McCarthy's 91 function" (http://www.cs.utexas.edu/users/moore/acl2/workshop-1999/Cowles-abstract.html). In Kaufmann, M.; Manolios, P.; Strother Moore, J (eds.). *Computer-Aided reasoning: ACL2 case studies*. Kluwer Academic. pp. 283–299. ISBN 9781475731880.

- Manna, Zohar; Pnueli, Amir (July 1970). "Formalization of Properties of Functional Programs". *Journal of the ACM*. **17** (3): 555–569. doi:10.1145/321592.321606 (https://doi.org/10.1145%2F321592.321606).
- Manna, Zohar; McCarthy, John (1970). "Properties of programs and partial function logic". *Machine Intelligence*. **5**. OCLC 35422131 (https://www.worldcat.org/oclc/35422131).
- Manna, Zohar (1974). *Mathematical Theory of Computation* (4th ed.). McGraw-Hill. ISBN 9780070399105.
- Wand, Mitchell (January 1980). "Continuation-Based Program Transformation Strategies". *Journal of the ACM*. **27** (1): 164–180. doi:10.1145/322169.322183 (https://doi.org/10.1145%2F322169.322183).