Cetera: Certified Termination with Agda

Dieter Hofbauer[1], Johannes Waldmann[2]

[1]ASW Saarland (Germany), [2]HTWK Leipzig (Germany)

20th Workshop on Termination
Leipzig, Germany, September 3–4, 2025

-
```
$ pure-matchbox --cetera -S cetera.strat z049.sys > z049.cert
Certificate
 { system = [Rule   { lhs = [0,1,0,0,1,1]
                    , rhs = [0,0,1,0,1,1,0]}]
 , reason = MatrixInterpretation
   { minterpretation = MatrixInterpretation
     { dim = 4
     , int = [(1,[[1,0,0,0],[0,0,1,0],[0,1,1,1],[0,0,0,1]])
             ,(0,[[1,1,0,0],[0,1,0,0],[0,0,0,0],[0,0,0,1]]) ]}
   , remove =
       [Rule { lhs = [0,1,0,0,1,1],rhs = [0,0,1,0,1,1,0]}]
   , sub = Certificate   { system = [], reason = Empty}}}

$ cetera-main z049.sys z049.cert
OK
```
- Using only weights and matrix interpretations, with a time-out of 30 seconds, Matchbox proves termination for 488 (of 1658) benchmarks in SRS_Standard. Total verification time over all certificates is < 5 s.

Cetera: Goals and Status

- general goal: a formally verified program that checks validity of certificates for termination of string rewriting
  . . . so, same approach as CeTA (Thiemann) –
- specific goals (for now)
  - matrix interpretations ($E_{\{1,n\}}$ (HW RTA06) (DONE)
  - sparse tiling (GHW FSCD19), approximate RFC match bounds (GHW WST22)
    needs
    - RCF theorem (nearly DONE)
    - partial models for local termination (not done)
- specific method: verification in Agda
  (then extract Haskell code, compile with GHC - like CeTA)
- implied by using Agda: constructive proofs

Agda

- Agda(2) (Norell 2007), based on Martin-Löf Type Theory (1972)
  proposition = type, proof = program
  each Agda program is (provably) total, each proof is constructive
- very few built-in assumptions/mechanisms
  - dependently typed functions
    example: the concept of equality
    ```
    data Eq {a : Set} : a -> a -> Set where
      refl : {x : a} -> Eq x x
    ```
    type checking involves normalisation and unification of type arguments
  - recursive functions where the Agda compiler can prove termination
    (Abel 1998, Abel and Altenkirch 2002)
- everything can be defined from these,
  there is no separate tactics language
- we want (for Cetera) to stay constructive,
  don't introduce classical logic via postulates (like Color/Coq does?)

Constructive (Non)Termination

- (this is not new, cf. *accessibility* in Paulson 1986)
- R is terminating for x: each R-successor of x is terminating
  ```
  data SN {a : Set} (R : Rel a) (x : a) : Set  where
    sn : (forall (y : a) -> R x y -> SN R y) -> SN R x
  ```
  a proof of `SN R` is a (Agda-definable!) function that constructs the levels of successors
- R is non-terminating for x if there is an infinite R-derivation
  $x = f(0) \to f(1) \to \ldots$ for Agda-definable f
  ```
  data INF {a : Set} (R : Rel a) (x : a) : Set  where
    inf : (f : Nat -> a) -> (f zero == x)
        -> (forall (y : Nat) -> R (f y) (f (succ y))) -> INF R
  ```
- this will miss some forms of termination, and of non-termination

RFC Theorem (proof plan)

- Dershowitz 1981: SN(R) $\iff$ SN(R on RFC(R)).
- constructive proof: block decomposition $w \in (\Sigma \cup \mathrm{RFC}(R))^*$
  embed $\to_R$ (arbitrary derivation) into length-lex. (from the right)
  extension of $(\to_R \cup \sqsupset_s)^+$ on blocks.
-
```
17 : 0 0 1 0 0 1 0 0 1 1 0 0 0 1 1 1 1
14 : 0 0 1 0 0 1 0 0 1 1 0 111000 1 1
12 : 0 0 1 0 0 1 0 0 1 1 0 1110111000
 9 : 0 0 1 0 0 1 111000 0 1110111000
 8 : 0 0 1 0 0 1 11100111000 10111000
 8 : 0 0 1 0 0 1 1111110001000 10111000
 6 : 0 0 1 111000 111110001000 10111000
 6 : 0 0 1 1110111000 1110001000 10111000
 6 : 0 0 1 11101110111000 10001000 10111000
 4 : 111000 1101110111000 10001000 10111000
 4 : 1110111000 01110111000 10001000 10111000
 4 : 11101110011000 10111000 10001000 10111000
 4 : 1110111110001000 10111000 10001000 10111000
```
- SN$(\to_R \cup \sqsupset_s)$ via commutation $(\sqsupset_s \circ \to_R) \subseteq (\to_R \circ \sqsupset_s)$

Plans, Discussion

- even with proof of RFC theorem, need effective representation of RFC(R) (as finite automaton = partial algebra)
  then get sparse tiling via semantic labeling w.r.t. partial model
- constructive proof for dependency pairs method (for SRS)
  use multi-set of self-labelled strings
- implications for derivational complexity?
- compressed loop certificates (transport systems)?
- unified (with CeTA) format for certificates?
- competition of certificate checkers? (CeTA vs. Cetera?)
  not useful (e.g., it would compare efficiency of implementation of matrix multiplication, correctness proofs (e.g., matrix multiplication is associative) are *irrelevant* for that computation)

Random ideas for future competitions

- find proofs for restricted set of certificates (e.g., matrix only, or DP+weights only, finite models + weights only) so a new prover stands a chance against established ones that have a full range of methods
- make a minimal change to a fixed (open-sourced) prover ("minisat hack track")
- . . . to the strategy expression used by a fixed prover (matchbox, aprove, ttt2 have strategy language)
  can take part in competition without writing a prover
- god's book of proofs: for each problem in TPDB: bring any certificate (no matter how it was computed), bring a smaller certificate.
- busy (elusive) beaver hunt: bring a small problem that cannot be solved by current provers (of most recent competition) ("small" = not larger than a known unsolved problem of the same category)