Reasoning about Dynamic Auctions

Marco Caminati\textsuperscript{1,2}, Manfred Kerber\textsuperscript{1,2}, Colin Rowat\textsuperscript{1,2}

\textsuperscript{1}University of Birmingham, UK
\textsuperscript{2}ForMaRE project: http://cs.bham.ac.uk/research/projects/formare/

9th April 2015
Auctions

Auctions allocate trillions of dollars in goods and services every year.

Auctions are a mechanism to distribute resources (e.g., eBay, ICANN, possibly High-Frequency Trading [Peter Cramton])

Given: a set of individual bids for a good (not necessarily the same as the value an individual ascribes to the good!)

Goals:
- give the good to the bidder who values it most
- determine prices
- maximize revenue

- New auctions are designed and some properties are proved.
- Strict rules must be followed.
- New auctions may have problems.
Dynamic Auctions

- Allow bidders to form their valuations in the iterated rounds.
- Make the evolution of the auction more complex.
- Make the formalization of their properties more complex.
- In particular: how to include in the verification the iterated input and output?
Read input → Compute output

Check termination ↑ ↓ Pretty print output
Alternatives:

- Encapsulate verified code for the static aspects of the auction in procedural loops.
- Reason about dynamic auctions in ad-hoc frameworks, e.g., temporal logics.
- Co-recursion
Formal methods for dynamic auctions

primcorec conditionalIterates where
  "conditionalIterates f x =
    (if   (fst x) then
     (LCons x (conditionalIterates f (f x)))
   else (LCons x LNil))"

definition "dynamicAuction input output
  = conditionalIterates
    (output o g o input) (True,[])"

theorem auctionTermination: fixes l::"int list"
  assumes "step > (0::nat)" "∀ i<size l. l!i ≤ M"
  "l ≠ []"
  shows
  "int_of_nat (firstInvalidBidIndex step l) ≤
   (M - hd l) div (int_of_nat step) + (2::int)"
proof - have
1: "?R ≥ 0" using assms(1,2,3) diff_right_mono diff_self
Divides.transfer_nat_int_function_closures(1) hd_conv_nth
length_greater_0_conv max.semilattice_strict_iff_order
int_of_nat_def of_nat_0_less_iff by metis
let ?i="firstInvalidBidIndex step l - 1"
{
    assume "¬ ?thesis" then have
    0: "?L > ?R+(2::int)" by linarith then have "?L > 0" using 1
    by linarith then have
    2: "?i < firstInvalidBidIndex step l" using One_nat_def diff_less
    int_of_nat_def lessI1I neq0_conv of_nat_0 order_less_irrefl
    by (metis (no_types, lifting))
    moreover have "firstInvalidBidIndex step l ≤ size l" using lm23b
    by auto ultimately have
    3: "?i < size l" by linarith moreover have
    4: "int_of_nat ?i > ?R+(1::int)" using 0 1 2 int_of_nat_def by auto
    moreover have
    "int_of_nat ?i <= (l! ?i - hd l) div (int_of_nat step)+(1::int)"
    using 2 assms(1) by (rule lm30)
    moreover have "... <= ?R+(1::int)"
    using assms(1,2) lm32 int_of_nat_def 2 3 by simp
    ultimately have False using 4 by linarith
}
thus ?thesis by (rule HOL.ccontr)
qed
References and Related Work

We build on work on lazy list in Isabelle started by Paulson. Other approaches include:

- Simulation
- Model checking

References:

- UCAM-CL-TR-69
- ta-gu-va-09
- song2004nonparametric