On Vickrey's Theorem and the Use of ACL2 for Formal Reasoning in Economics

Ruben Gamboa and John Cowles

Department of Computer Science
University of Wyoming
Laramie, Wyoming 82071

{ruben,cowles}@uwyo.edu

ACL2 Workshop 2014
Vienna, Austria

Motivation

- ForMaRE project: **for**mal **ma**thematical **r**easoning in **e**conomics
- Interesting application for formal reasoning—is it even possible?
- This is a problem looking for a solution
- One of the early projects is to choose a formal reasoning platform:
  - Isabelle
  - Theorema
  - Mizar
  - Hets/CASL/TPTP
- Challenge problem: Vickrey's Theorem for second price auctions

Vickrey's Theorem

- An **auction** consists of a good to be sold and a list of participants indexed by $N = \{1, \ldots, n\}$, each with a **bid** b_i and a secret and personal **value** v_i assigned to the good
- An **outcome** is a winner i and a price
- Outcomes are expressed as a tuple (x, p), where x is an n-dimensional vector with $x_i = 1$ and $x_j = 0$ for all other j, and p is a price vector, which is zero except for p_i
- For a second price auction, the price p is the highest of the b_j (other than the winner's bid, b_i)
- An auction is **efficient** if it maximizes $\sum_{i \in N} x_i v_i$
- An auction strategy supports an **equilibrium in weakly dominant strategies** if no bidder i can increase his or her utility $(x_i v_i - p_i)$

Vickrey's Theorem

Theorem

The strategy where each bidder bids his or her value (i.e., $b_i = v_i$) supports an equilibrium in weakly dominant strategies and is efficient.

Proof (Outline) of Vickrey's Theorem

- Fix i, and start with a strategy b where $b_i = v_i$
- Consider another strategy b' identical to b, except $b'_i \neq b_i = v_i$

- Break into four cases, depending on whether
  - i is the winner with the first strategy b_i
  - i is the winner with the new strategy b'_i
- Each case is resolved by chasing simple inequalities

Formalizing Vickrey's Theorem in ACL2

```
(defun valid-bid-p (l n)
  (and (equal (len l) n)
       (all-positive-p l)))
```

Formalizing Vickrey's Theorem in ACL2

```
(defun almost-equal-bids (b1 b2 i)
  (if (and (consp b1)
           (consp b2))
      (if (zp i)
          (equal (cdr b1) (cdr b2))
        (and (equal (car b1) (car b2))
             (almost-equal-bids (cdr b1) (cdr b2) (1- i))))
    nil))
```

Formalizing Vickrey's Theorem in ACL2

```
(defthmd nth-almost-equal-bids
  (implies (and (almost-equal-bids b1 b2 i)
                (natp i)
                (natp j)
                (not (equal i j)))
           (equal (nth j b1)
                  (nth j b2))))
```

Formalizing Vickrey's Theorem in ACL2

```
(defun max-bid (b)
  (if (consp b)
      (max (car b)
           (max-bid (cdr b)))
    0))

(defun 2nd-max-bid (b w)
  (if (consp b)
      (if (zp w)
          (max-bid (cdr b))
        (max (car b)
             (2nd-max-bid (cdr b) (1- w))))
    0))
```

Formalizing Vickrey's Theorem in ACL2

```
(defun utility (b v i w)
  (if (equal i w)
      (- (nth w v)
         (2nd-max-bid b w))
    0))
```

Formalizing Vickrey's Theorem in ACL2

```
(encapsulate
 ( ((pick-winner *) => *) )

 ;; two local definitions and three lemmas

 (defthmd pick-winner-correctness
   (implies (and (valid-bid-p b n)
                 (< 0 n))
            (and (natp (pick-winner b))
                 (< (pick-winner b) (len b))
                 (equal (nth (pick-winner b) b)
                        (max-bid b))))
   :hints (("Goal"
            :use ((:instance pick-winning-bid-if-exists
                             (bmax (max-bid b))
                             (i 0))
                  (:instance member-max-bid))))
   ))
```

Formalizing Vickrey's Theorem in ACL2

```
(encapsulate
 nil

 (local
  (defun induction-hint (b1 b2 i n)
    (if (and (consp b1)
             (consp b2))
        (1+ (induction-hint (cdr b1) (cdr b2) (1- i) (1- n)))
      (+ i n))))
```

Formalizing Vickrey's Theorem in ACL2

```
(local
 (defthmd lemma-1
   (implies (and (valid-bid-p bid n)
                 (natp i)
                 (< i n)
                 (valid-bid-p bid2 n)
                 (almost-equal-bids bid bid2 i))
            (equal (2nd-max-bid bid2 i)
                   (2nd-max-bid bid i)))
   :hints (("Goal"
            :induct (induction-hint bid bid2 i n)))
   ))
```

Formalizing Vickrey's Theorem in ACL2

```
(defthmd case-1-1
  (implies (and (valid-bid-p value n)
                (valid-bid-p bid n)
                (natp i)
                (< i n)
                (equal (nth i value)
                       (nth i bid))
                (valid-bid-p bid2 n)
                (almost-equal-bids bid bid2 i)
                (equal i (pick-winner bid))
                (equal i (pick-winner bid2)))
           (<= (utility bid2 value i (pick-winner bid2))
               (utility bid value i (pick-winner bid))))
  :hints (("Goal"
           :expand ((utility bid2 value i (pick-winner bid2))
                    (utility bid value i (pick-winner bid)))
           :use ((:instance lemma-1
                            (i (pick-winner bid))))))))
```

Formalizing Vickrey's Theorem in ACL2

```
;; 7 lemmas and 2 local definitions (induction hints)
(defthmd case-2-1
  (implies (and (valid-bid-p value n)
                (valid-bid-p bid n)
                (natp i) (< i n)
                (natp j) (< j n)
                (not (equal i j))
                (equal (nth i value) (nth i bid))
                (valid-bid-p bid2 n)
                (almost-equal-bids bid bid2 i)
                (equal j (pick-winner bid))
                (equal i (pick-winner bid2)))
           (<= (utility bid2 value i (pick-winner bid2))
               (utility bid value i (pick-winner bid))))
  :hints (("Goal" :do-not-induct t
           :use ((:instance lemma-7 (bid bid2) (w (pick-winner bid2)))
                 (:instance lemma-5)
                 (:instance lemma-4)))))
```

“Unbiased” Assessment of ACL2

- **Criterion #1:** Level of detail and explicitness required

“Unbiased” Assessment of ACL2

- **Criterion #1:** Level of detail and explicitness required

- Needed to formally define trivial functions like utility and 2nd-highest bid
- Had to prove some “obvious” lemmas about these functions, e.g., utility is non-negative

“Unbiased” Assessment of ACL2

- **Criterion #2:** Expressiveness of language vs. efficiency of system

“Unbiased” Assessment of ACL2

- **Criterion #2:** Expressiveness of language vs. efficiency of system

- ACL2 is biased in favor of efficiency
- For this project, this was not a problem at all
- Specifically, FOL was not a limitation at all

- Biggest limitation is the lack of true randomness
- We used encapsulate as a workaround (but we don’t like it)

“Unbiased” Assessment of ACL2

- **Criterion #3:** Proof development and management of proof tools

“Unbiased” Assessment of ACL2

- **Criterion #3:** Proof development and management of proof tools

- Users like the idea of doing proofs where they concentrate on different aspects at a time
- ACL2’s skip-proofs supports this style of development

- Users also liked mixing different tools
- ACL2 seems headed in the same direction

“Unbiased” Assessment of ACL2

- **Criterion #4:** Input syntax

“Unbiased” Assessment of ACL2

- **Criterion #4:** Input syntax

- This is a religion
- “I love the smell of parentheses in the morning”

- Apparently economists preferred the input style of Mathematica
- It’s not clear that there is (or can be) a clear winner

“Unbiased” Assessment of ACL2

- **Criterion #5:** Comprehensibility and trustability of the output

“Unbiased” Assessment of ACL2

- **Criterion #5:** Comprehensibility and trustability of the output

- It’s **not** over at the last Q.E.D.
- So we got a Q.E.D. So what?
- Is it the right theorem?

- We had a bug in valid-bid-p
- Since that was a hypothesis in all theorems, the proofs were too easy
- Best practice: use tests **and** proofs
- We use `(thm (equal expr expected))` in our code as a poor man’s unit testing framework

Future Work?

- Economics is a very interesting application area
- Some obvious theorems include the Black-Scholes equation and binomial approximation

Future Work?

- Economics is a very interesting application area
- Some obvious theorems include the Black-Scholes equation and binomial approximation
- But there are also some practical applications
- Economists on both sides of the political spectrum have recently released Excel spreadsheet with errors
- (Whether those errors are significant is a topic for economists)
- What would be nice is to offer support for some verification of spreadsheets
- This can be minimal (as in data coverage analysis)
- It can be more significant, as in serious theorems
- We have some ideas, but we would welcome collaborators!