

# Automated Reasoning with Quantified Formulae

May 11, 2009







## Quantification in ACL2

- 2<sup>nd</sup> Class citizen in a 1<sup>st</sup> order world
  - ACL2 is "Quantifier Free"
    - No Syntactic Construct for quantification ie: (forall (x) ..)
  - "Quantification" is a top-level event .. via a choice axiom
    - Cannot be nested in function definitions or theorems

(defun-sk prop () (forall (a) (pred a)))

- Quantification is effectively hidden from user during proof
  - Quantified variables are modeled as constrained function symbols

Goal' (implies (pred (prop-witness)) (pred x))

- Insubstantial native reasoning support
  - One point for :rewrite :direct

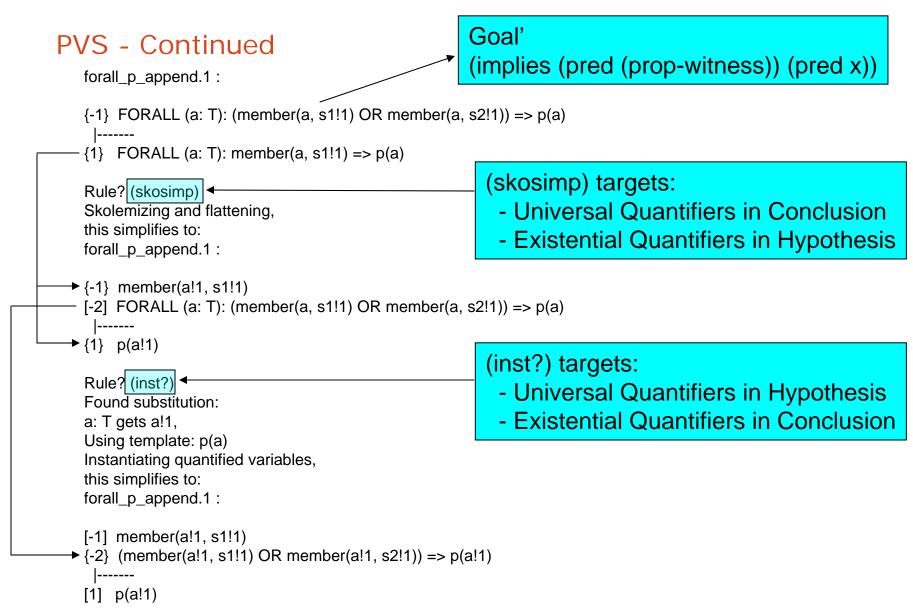




```
PVS
```

```
member_of_append: LEMMA
                                          / 11 66
 FORALL (a:T, s1,s2: set):
                                            (skosimp)
  member(a,append(s1,s2)) =
                                            (auto-rewrite "forall_p")
    (member(a,s1) or member(a,s2))
                                            (auto-rewrite "member_of_append")
                                            (assert)
p(a:T): bool
                                            (iff)
                                            (apply
forall_p(x: set) : bool =
                                             (then (ground)
 FORALL (a: T): member(a,x) \Rightarrow p(a)
                                            (then (skosimp)
                                                   (repeat* (then (inst?) (ground))))))
forall_p_append: LEMMA
 FORALL (s1,s2: set):
  forall_p(append(s1,s2)) =
    (forall_p(s1) and forall_p(s2))
```





Rockwell

llins





### What was our objective?

- Add support for reasoning about quantified formulae in ACL2
  - In particular, automated instantiation
  - Power should approach that of the PVS (inst?) Command
  - For fun, also support something like (skosimp)
- At least Identify quantified formulae in subgoals
  - Give the user an idea of what they have to work with



### How did we do it?

Rockwell

- Constructed a wrapper for defun-sk (def::un-sk)
  - Same interface as defun-sk
  - Saves information about quantified formula in a table
  - Makes information about quantified formulae available at proof time
- Defined computed hints for (quant::inst?) and (quant::skosimp)
  - Used (bash-to-dnf) to simplify formulae before/during matching
  - Pattern match table entries against current goal
    - Detect formulae and their polarity
  - Search for suitable instances of existing formulae from goal
  - Generate hints to advance proof
    - Skosimp: generalize quantified variables
      - Rewrite them into (generalize (quantified-variable ..))
      - Apply generalization clause processor
    - Inst: instantiate the appropriate quantification lemma (-necc or -suff)



### What were the challenges?

- Propositional simplification of quantified formulae
  - Makes it hard to even identify formulae
- Simplification (rewriting) **during** pattern matching
  - (member a x) where  $(x \cdot (append y z)) =>$  (member a x) or (member a z)
  - Required for forall-p-append solution
- Theory management during simplification
  - Not easy ... I still don't understand it
- Lack of standard form for quantified formula
  - Subterm matching => Support for equality
    - (forall (x) (equal (goo x) (foo x)))
    - Cannot look for (equal (goo x) (foo x))
      - pattern match on (goo a) .. and then on (foo a)
- Avoiding duplicate/specious instantiations
- Limiting introduction of instances





### What kinds of problems can it solve?

"Simple" instantiations where the required instance is deducible more or less immediately by pattern matching the quantified formula with the goal ©





### forall-p-append

(defstub p (x) t)

(def::un-sk forall-p (x) (forall a (implies (member a x) (p a))))

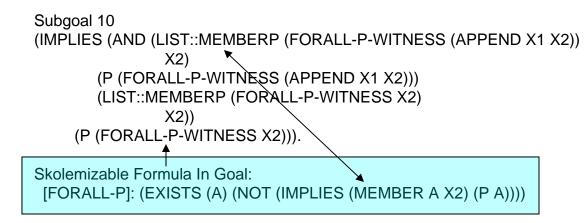
(defthm member-append (iff (member a (append x1 x2)) (or (member a x1) (member a x2))))

```
(defthm forall-p-append
(equal (forall-p (append x1 x2))
(and (forall-p x1) (forall-p x2)))
:hints ((quant::skosimp)
(quant::inst?)))
```

From the ACL2 documentation		
	This was my primary motivating example	



### forall-p-append proof



Computed Hint: (:DO-NOT '(PREPROCESS) :IN-THEORY (ENABLE FORALL-P-SKOLEMIZATION) :RESTRICT ((FORALL-P-SKOLEMIZATION ((X X2)))))

[Note: A hint was supplied for our processing of the goal above. Thanks!]

This simplifies, using the :meta rule \*META\*-BETA-REDUCE-HIDE and the :rewrite rule FORALL-P-SKOLEMIZATION, to



### forall-p-append proof

[Note: A hint was supplied for our processing of the goal below. Thanks!]

```
Subgoal 10'
(IMPLIES
(AND (LIST::MEMBERP (FORALL-P-WITNESS (APPEND X1 X2))
X2)
(P (FORALL-P-WITNESS (APPEND X1 X2)))
(LIST::MEMBERP (GENSYM::GENERALIZE (HIDE (FORALL-P-WITNESS X2)))
X2))
(P (GENSYM::GENERALIZE (HIDE (FORALL-P-WITNESS X2)))))).
We now apply the verified :CLAUSE-PROCESSOR function
GENERALIZE-CLAUSE-PROCESSOR-WRAPPER to produce one new subgoal.
Subgoal 10''
(IMPLIES (AND (LIST::MEMBERP (FORALL-P-WITNESS (APPEND X1 X2))
X2)
(P (FORALL-P-WITNESS (APPEND X1 X2)))
(LIST::MEMBERP HIDE10 X2))
(P (HDE10)).
```



### forall-p-append proof

Rockwell

Instantiable Formula In Goal: FORALL-P : (FORALL (A) (IMPLIES (MEMBER A (BINARY-APPEND X1 X2)) (P A)))

Computed Hint: (:USE (:INSTANCE FORALL-P-NECC (A HIDE10) (X (BINARY-APPEND X1 X2)))) [Note: A hint was supplied for our processing of the goal above. Thanks!]

We augment the goal with the hypothesis provided by the :USE hint. The hypothesis can be derived from FORALL-P-NECC via instantiation. We are left with the following subgoal.

```
Subgoal 10'''
(IMPLIES (AND (IMPLIES (NOT (IMPLIES (MEMBER HIDE10 (APPEND X1 X2)))
(P HIDE10)))
(NOT (FORALL-P (APPEND X1 X2))))
(LIST::MEMBERP (FORALL-P-WITNESS (APPEND X1 X2))
X2)
(P (FORALL-P-WITNESS (APPEND X1 X2)))
(LIST::MEMBERP HIDE10 X2))
(P HIDE10)).
```

But simplification reduces this to T, using the :definition FORALL-P, the :executable-counterpart of NOT, the :rewrite rules LIST::MEMBER-IS-MEMBERP-PROPOSITIONALLY and MEMBER-OF-APPEND and the :type-prescription rule LIST::MEMBERP.





One other (interesting?) example ...

```
(def::un-sk subetp (x y)
  (forall (a) (implies (member a x) (member a y))))
```

```
(defthm subset-transitivity
 (implies
    (and (subsetp x y)
        (subsetp y z))
    (subsetp x z))
    :hints ((quant::inst?)))
```





### How could it be better?

Rockwel

- Inspired by PVS (inst?) command
  - I have no idea how inst? works ...
    - I'm no longer that motivated, either.
  - There may be other/better ideas there .. or elsewhere.
    - If you are motivated, the ACL2 code is available.
- Improve integration with/leverage ACL2 simplification/unification
  - My solution is just a hack using (bash-to-dnf)
- Improve support for nested quantification
  - Although (inst?) didn't always solve that, either
- Access to type-alist
  - Would improve deductive capability
  - Computed hints do not have access to type-alist



### Conclusion

- ACL2 book providing (quant::inst?) and (quant::skosimp)
  - In the spirit of PVS (inst?) and (skosimp)
- Automate proofs of select theorems involving quantified formulae
  - A "reasonable" subset
    - Able to prove forall-p-append from ACL2 documentation
    - Appears limited by nested quantification
- Many enhancements possible
  - type-alist access would be nice