Quantification in Tail-recursive Function Definitions

Sandip Ray

Department of Computer Science
University of Texas at Austin

Email: sandip@cs.utexas.edu
web: http://www.cs.utexas.edu/users/sandip

Prologue

"ACL2 is a quantifier-free first order logic of recursive functions."

Prologue

"ACL2 is a quantifier-free first order logic of recursive functions."

The Truth: The syntax of ACL2 is quantifier-free, but ACL2 allows us to write quantified predicates via Skolemization.

Prologue

"ACL2 is a quantifier-free first order logic of recursive functions."

The Truth: The syntax of ACL2 is quantifier-free, but ACL2 allows us to write quantified predicates via Skolemization.

```
(defun-sk exists-foo (x) (exists y (foo x y)))
```

Prologue

"ACL2 is a quantifier-free first order logic of recursive functions."

The Truth: The syntax of ACL2 is quantifier-free, but ACL2 allows us to write quantified predicates via Skolemization.

```
(defun-sk exists-foo (x) (exists y (foo x y)))
```

```
(= (exists-foo x) (foo x (foo-witness x)))
(implies (foo x y) (exists-foo x))
```

A Preliminary Illustration

Consider defining a predicate `true` with the following axiom:

```
(= (true x)
   (if (done x) t
      (forall x (true (st x)))))
```

The equation is recursive, but in addition has quantification in the body.

A Preliminary Illustration

Consider defining a predicate `true` with the following axiom:

```
(= (true x)
   (if (done x) t
      (forall x (true (st x)))))
```

The equation is recursive, but in addition has quantification in the body.

ACL2 does not allow us to introduce definitional equations with both recursion and quantification.

A Preliminary Illustration

But if the axiom is introduced would the resulting theory be inconsistent?
No.

A Preliminary Illustration

But if the axiom is introduced would the resulting theory be inconsistent?

No.

```
(encapsulate
  (((true *) => *))
  (local (defun true (x) t))
  (defthm true-satisfies-its-equation
    (= (true x)
       (if (done x) t
         (forall x (true (st x)))))))
```

A Preliminary Illustration

But if the axiom is introduced would the resulting theory be inconsistent?
No.

```
(encapsulate
  (((true *) => *))
  (local (defun true (x) t))
  (defthm true-satisfies-its-equation
    (= (true x)
       (if (done x) t
         (forall x (true (st x)))))))
```

ACL2 users have from time to time wanted some form of recursion and quantification together.

This Talk

We show how to introduce in ACL2 a class of definitional axioms, called **extended tail-recursive axioms**, that contain both recursion and quantification.

This Talk

We show how to introduce in ACL2 a class of definitional axioms, called **extended tail-recursive axioms**, that contain both recursion and quantification.

The defining equation of a predicate `Q-iv` is extended tail-recursive if

- There is exactly one recursive branch.
- The outermost function call in the recursive branch is `Q-iv`, possibly enclosed by a sequence of quantifiers.

Admissibility of Extended Tail-recursive Definitions

Why are extended tail-recursive definitions admissible?

Admissibility of Extended Tail-recursive Definitions

Why are extended tail-recursive definitions admissible?

```
(= (F-iv1 x)
   (if (done x) (base x)
     (forall i (F-iv1 (st1 x i)))))
```

Admissibility of Extended Tail-recursive Definitions

Why are extended tail-recursive definitions admissible?

```
(= (F-iv1 x)
   (if (done x) (base x)
     (forall i (F-iv1 (st1 x i)))))
```

We view `st1` as a transformation function that transforms an object `x` given a choice `i`.

`F-iv1` postulates an invariant over this transformation.

Admissibility of Extended Tail-recursive Definitions

Why are extended tail-recursive definitions admissible?

```
(= (F-iv1 x)
   (if (done x) (base x)
     (forall i (F-iv1 (st1 x i)))))
```

We view `st1` as a transformation function that transforms an object `x` given a choice `i`.

`F-iv1` postulates an invariant over this transformation.

If `(done x)` holds the invariant is equal to `(base x)`.

Otherwise the invariant holds for `x` if and only if it holds for each successor.

Admissibility of Extended Tail-recursive Definitions

We can introduce the equation by defining a witnessing invariant that posits the same thing a little differently.

```
(defun sn1 (x ch) (if (endp ch) x (sn1 (st1 x (car ch)) (cdr ch))))

(defun n-done (x ch)
  (if (endp ch) (not (done ch))
    (and (not (done x)) (n-done (st1 x (car ch)) (cdr ch)))))

(defun done-ch1 (x ch)
  (and (done (sn1 x ch))
       (implies (consp ch) (n-done x (dellast ch)))))

(defun-sk F-iv1 (x)
  (forall ch (implies (done-ch1 x ch) (base (sn1 x ch)))))
```

Admissibility of Extended Tail-recursive Definitions

Consider a variant of the above equation.

```
(= (E-iv1 x)
   (if (done x) (base x)
     (exists i (E-iv1 (st1 x i)))))
```

Admissibility of Extended Tail-recursive Definitions

Consider a variant of the above equation.

```
(= (E-iv1 x)
   (if (done x) (base x)
     (exists i (E-iv1 (st1 x i)))))
```

We can introduce the equation the same way as above.

```
...

(defun-sk E-iv1 (x)
  (exists ch (and (done-ch1 x ch) (sn1 x ch))))
```

Summing Up the Witnesses

```
(= (F-iv1 x)
   (if (done x) (base x)
     (forall i (F-iv1 (st1 x i)))))
```

The witnessing predicate: "For each sequence ch of choices, such the first descendant of x that satisfies done also satisfies base."

Can be expressed in ACL2.

Summing Up the Witnesses

```
(= (E-iv1 x)
   (if (done x) (base x)
     (exists i (F-iv1 (st x i)))))
```

The witnessing predicate: "There exists a sequence ch of choices, such that the first descendant of x that satisfies done also satisfies base."

Can be expressed in ACL2.

Summing Up the Witnesses

```
(= (EF-iv2 x)
   (if (done x) (base x)
     (exists i (forall j (F-iv1 (st2 x i j))))))
```

The witnessing predicate: “ There exists a sequence `i-ch` of `i` choices, such that for each sequence `j-ch` of `j` choices, the first descendant of `x` that satisfies `done` also satisfies `base`.”

Can be expressed in ACL2.

Summing Up the Witnesses

```
(= (iv0 x)
   (if (done x) (base x)
     (iv0 (st0 x i))))))
```

The witnessing predicate: “The first descendant of x that satisfies done also satisfies base.”

This is essentially the witnessed designed by **Manolios and Moore (2000)**, to show that tail-recursive equations can always be introduced in ACL2.

Logical Impediments

We cannot allow arbitrary recursion and quantification. Doing so will violate conservativity.

Acknowledgement: This proof is due to an example provided by Matt Kaufmann. **(Thanks, Matt!)**

1. A truth predicate of Peano arithmetic is not conservative over Peano Arithmetic.
2. If we have both recursion and quantification then we can define a predicate `true-formula` in ACL2.
3. We can then prove by induction that `true-formula` holds for all formulas that are provable.
4. Details are in the paper.

Upshot of Logical Impediments

It is possible to define `true-formula` if we allow two recursive branches and quantification.

Therefore in general a recursive definition containing quantification and more than one recursive branch is not conservative.

A Potential Application

Moore (2003) showed how to use inductive assertions on operationally modeled sequential programs.

A Potential Application

Moore (2003) showed how to use inductive assertions on operationally modeled sequential programs.

```
(= (inv s) (if (cutpoint s) (assertion s) (inv (step s))))
```

Attempting to prove `(implies (inv s) (inv (step s)))` causes symbolic simulation of the operational semantics from each cutpoint.

A Potential Application

Moore (2003) showed how to use inductive assertions on operationally modeled sequential programs.

```
(= (inv s) (if (cutpoint s) (assertion s) (inv (step s))))
```

Attempting to prove `(implies (inv s) (inv (step s)))` causes symbolic simulation of the operational semantics from each cutpoint.

But suppose `step` is non-deterministic and also takes an input oracle.

A Potential Application

Moore (2003) showed how to use inductive assertions on operationally modeled sequential programs.

```
(= (inv s) (if (cutpoint s) (assertion s) (inv (step s))))
```

Attempting to prove `(implies (inv s) (inv (step s)))` causes symbolic simulation of the operational semantics from each cutpoint.

But suppose `step` is non-deterministic and also takes an input oracle.

To apply Moore's method, we now need to write `inv` as:

```
(= (inv s)
   (if (cutpoint s) (assertion s) (forall i (inv (step s i)))))
```

This equation can be introduced since it is extended tail-recursive.

Future Work

We are looking at more avenues for using extended tail-recursive equations.

Future Work

We are looking at more avenues for using extended tail-recursive equations.

One possible area might be in formalizing programming language metatheories.

Swords: Extended tail-recursive equations might be useful in that domain in some cases, but probably not sufficient for all the interesting properties.

Future Work

We are looking at more avenues for using extended tail-recursive equations.

One possible area might be in formalizing programming language metatheories.

Swords: Extended tail-recursive equations might be useful in that domain in some cases, but probably not sufficient for all the interesting properties.

We are also looking at extending the class of equations.

Might be possible to have more general equations if we restrict to only well-founded recursions?

Future Work

We are looking at more avenues for using extended tail-recursive equations.

One possible area might be in formalizing programming language metatheories.

Swords: Extended tail-recursive equations might be useful in that domain in some cases, but probably not sufficient for all the interesting properties.

We are also looking at extending the class of equations.

Might be possible to have more general equations if we restrict to only well-founded recursions?

An obvious and frustrating drawback: The semantics of LTL involves both recursion and quantification but is not extended tail-recursive (requires more than one recursive branch).

Acknowledgements

- **J Strother Moore** for challenging me to find a way to make his inductive assertions work applicable for non-deterministic systems.

- **Matt Kaufmann** for extensive discussions on conservativity in ACL2.