Translating HOL4 Developments into ACL2 (Progress Since Last Year)

Rump Session, ACL2 Workshop 2026

Matt Kaufmann (speaker) and
Konrad Slind

November 16, 2026

Outline

Overview

Examples

Concluding Remarks

Outline

Overview

Examples

Concluding Remarks

Introduction

This talk is about a translator from higher order logic, specifically, HOL4, to ACL2.

Introduction

This talk is about a translator from higher order logic, specifically, HOL4, to ACL2.

- **Challenge**: Higher order logic has functions as first class objects, e.g., they can go into lists.

Introduction

This talk is about a translator from higher order logic, specifically, HOL4, to ACL2.

- **Challenge**: Higher order logic has functions as first class objects, e.g., they can go into lists.
- **Solution**: Translate to ACL2(zfc), an extension of ACL2 that comprehends set theory.

Introduction

This talk is about a translator from higher order logic, specifically, HOL4, to ACL2.

- **Challenge**: Higher order logic has functions as first class objects, e.g., they can go into lists.
- **Solution**: Translate to ACL2(zfc), an extension of ACL2 that comprehends set theory.

Timeline:

Introduction

This talk is about a translator from higher order logic, specifically, HOL4, to ACL2.

- **Challenge**: Higher order logic has functions as first class objects, e.g., they can go into lists.
- **Solution**: Translate to ACL2(zfc), an extension of ACL2 that comprehends set theory.

Timeline:

- 2006: Translate ACL2 into HOL4 — so, the other direction (with Gordon, Hunt, and Reynolds); sadly, bit rot

Introduction

This talk is about a translator from higher order logic, specifically, HOL4, to ACL2.

- **Challenge**: Higher order logic has functions as first class objects, e.g., they can go into lists.
- **Solution**: Translate to ACL2(zfc), an extension of ACL2 that comprehends set theory.

Timeline:

- 2006: Translate ACL2 into HOL4 — so, the other direction (with Gordon, Hunt, and Reynolds); sadly, bit rot
- 2025 Rump Session: Early version of this work

Introduction

This talk is about a translator from higher order logic, specifically, HOL4, to ACL2.

- **Challenge**: Higher order logic has functions as first class objects, e.g., they can go into lists.
- **Solution**: Translate to ACL2(zfc), an extension of ACL2 that comprehends set theory.

Timeline:

- 2006: Translate ACL2 into HOL4 — so, the other direction (with Gordon, Hunt, and Reynolds); sadly, bit rot
- 2025 Rump Session: Early version of this work
- 2026 Rump Session: Highlights progress since then (but the talk is still self-contained)

Introduction

This talk is about a translator from higher order logic, specifically, HOL4, to ACL2.

- **Challenge**: Higher order logic has functions as first class objects, e.g., they can go into lists.
- **Solution**: Translate to ACL2(zfc), an extension of ACL2 that comprehends set theory.

Timeline:

- 2006: Translate ACL2 into HOL4 — so, the other direction (with Gordon, Hunt, and Reynolds); sadly, bit rot
- 2025 Rump Session: Early version of this work
- 2026 Rump Session: Highlights progress since then (but the talk is still self-contained)
- 2029: Someone's PhD dissertation?

Introduction

This talk is about a translator from higher order logic, specifically, HOL4, to ACL2.

- **Challenge**: Higher order logic has functions as first class objects, e.g., they can go into lists.
- **Solution**: Translate to ACL2(zfc), an extension of ACL2 that comprehends set theory.

Timeline:

- 2006: Translate ACL2 into HOL4 — so, the other direction (with Gordon, Hunt, and Reynolds); sadly, bit rot
- 2025 Rump Session: Early version of this work
- 2026 Rump Session: Highlights progress since then (but the talk is still self-contained)
- 2029: Someone's PhD dissertation?

No trust tags or `skip-proofs` are used in this work.

Two HOL-in-ACL2 Versions

See community books directory `projects/hol-in-acl2/`.

Two HOL-in-ACL2 Versions

See community books directory `projects/hol-in-acl2/`.

- Subdirectory `v1/` corresponds roughly to last year's rump session talk and the following paper.

  > *Matt Kaufmann and Konrad Slind, Translating HOL to ACL2 with Set Theory. In:* ***Logic, Computation and Proof: A Festschrift in Honour of Lawrence C. Paulson****, Springer LNCS Vol. 16870 (to appear), edited by Tobias Nipkow, Grant Passmore, and Nik Sultana.*

TWO HOL-IN-ACL2 VERSIONS

See community books directory `projects/hol-in-acl2/`.

- Subdirectory `v1/` corresponds roughly to last year's rump session talk and the following paper.

  > *Matt Kaufmann and Konrad Slind, Translating HOL to ACL2 with Set Theory. In:* ***Logic, Computation and Proof: A Festschrift in Honour of Lawrence C. Paulson****, Springer LNCS Vol. 16870 (to appear), edited by Tobias Nipkow, Grant Passmore, and Nik Sultana.*

- Subdirectory `v2/` (today's focus):
  Adds user-defined datatypes.

Modeling HOL Values and Types

We model HOL values as *value*-*type pairs* with respect to a *HOL type alist*.

Modeling HOL Values and Types

We model HOL values as *value*-*type pairs* with respect to a *HOL type alist*.

```
(include-book "projects/hol-in-acl2/v2/acl2/hol"
              :dir :system)
(in-package "ZF")
(thmz (hpp '(3 . :num) (hta0)))
```

Modeling HOL Values and Types

We model HOL values as *value-type pairs* with respect to a *HOL type alist*.

```
(include-book "projects/hol-in-acl2/v2/acl2/hol"
              :dir :system)
(in-package "ZF")
(thmz (hpp '(3 . :num) (hta0)))

(defun hta0 ()
; added arities to support datatype constructors
  (declare (xargs :guard t))
  `((:num    . (,(omega)       . 0))
    (:bool   . (,(pair t nil)  . 0))
    (:arrow  . (,(zfun-space)  . 2))
    ...))
```

Modeling HOL Values and Types

We model HOL values as *value-type pairs* with respect to a *HOL type alist*.

```
(include-book "projects/hol-in-acl2/v2/acl2/hol"
              :dir :system)
(in-package "ZF")
(thmz (hpp '(3 . :num) (hta0)))

(defun hta0 ()
; added arities to support datatype constructors
  (declare (xargs :guard t))
  `((:num    . (,(omega)       . 0))
    (:bool   . (,(pair t nil)  . 0))
    (:arrow  . (,(zfun-space)  . 2))
    ...))
```

(apply (zfun-space) (list A B)) = (A → B)

Outline

Overview

Examples

Concluding Remarks

We will consider a **binary tree** datatype.

We will consider a **binary tree** datatype.

- I'll go quickly, to give a sense of how translation works.

We will consider a **binary tree** datatype.

- I'll go quickly, to give a sense of how translation works.
- THE TAKEAWAY: Translation is generally quite direct, making a clean semantics possible.

"Mirror" Project Translation Steps

"MIRROR" PROJECT TRANSLATION STEPS

Files in `projects/hol-in-acl2/v2/examples/`:

"MIRROR" PROJECT TRANSLATION STEPS

Files in `projects/hol-in-acl2/v2/examples/`:

- `mirror`Script.sml
  The HOL4 source (actually a Standard ML file)

"MIRROR" PROJECT TRANSLATION STEPS

Files in `projects/hol-in-acl2/v2/examples/`:

- `mirror`Script.sml
  The HOL4 source (actually a Standard ML file)
- `mirror`.defhol
  The output of Konrad's translator from HOL4: `defhol` calls

"MIRROR" PROJECT TRANSLATION STEPS

Files in `projects/hol-in-acl2/v2/examples/`:

- `mirror`Script.sml
  The HOL4 source (actually a Standard ML file)
- `mirror`.defhol
  The output of Konrad's translator from HOL4: `defhol` calls
- `mirror`-thy.lisp

```
(in-package "HOL")
(include-book "../acl2/theories")
; Creates events from defhol forms:
(import-theory mirror)
```

"MIRROR" PROJECT TRANSLATION STEPS

Files in `projects/hol-in-acl2/v2/examples/`:

- `mirror`Script.sml
  The HOL4 source (actually a Standard ML file)
- `mirror`.defhol
  The output of Konrad's translator from HOL4: `defhol` calls
- `mirror`-thy.lisp

```
(in-package "HOL")
(include-book "../acl2/theories")
; Creates events from defhol forms:
(import-theory mirror)
```

- `mirror`-thy-exports.lisp
  Explicit events created by `import-theory` (above)

DATATYPE DEFINITION TO DEFHOL FORM

From `mirror.defhol` (comment via `mirrorScript.sml`):

DATATYPE DEFINITION TO DEFHOL FORM

From `mirror.defhol` (comment via `mirrorScript.sml`):

```
#|
Datatype α btree
Constructors:
   Leaf : α -> α btree
   Node : α btree -> α -> α btree -> α btree
|#
```

DATATYPE DEFINITION TO DEFHOL FORM

From `mirror.defhol` (comment via `mirrorScript.sml`):

```
#|
Datatype α btree
Constructors:
   Leaf : α -> α btree
   Node : α btree -> α -> α btree -> α btree
|#

(defhol
   :datatype (:btree a)
   :constructors ((leaf (:arrow* a (:btree a)))
                  (node (:arrow* (:btree a)
                                 a
                                 (:btree a)
                                 (:btree a)))))
```

Function Definition to `defhol` Form

From `mirror.defhol`, continued:

Function Definition to defhol Form

From `mirror.defhol`, continued:

```
#|
Definition:
   ⊢ ∀ a0. btree_Leaf_proj (Leaf a0) = a0
|#
```

Function Definition to defhol Form

From `mirror.defhol`, continued:

```
#|
Definition:
   ⊢ ∀ a0. btree_Leaf_proj (Leaf a0) = a0
|#


(defhol
   :fns ((btree_Leaf_proj (:arrow* (:btree a) a)))
   :defs ((:forall ((a0 a))
      (equal
       (hap* (btree_Leaf_proj (typ (:arrow* (:btree a)
                                            a)))
             (hap* (Leaf (typ (:arrow* a (:btree a))))
                   a0))
       a0))))
```

Function Definition to ACL2(zfc)

Function Definition to ACL2(zfc)

```
⊢ ∀ a0. btree_Leaf_proj (Leaf a0) = a0
```

Function Definition to ACL2(zfc)

```
⊢ ∀ a0. btree_Leaf_proj (Leaf a0) = a0
(defhol :fns ... :defs ...)
```

Function Definition to ACL2(zfc)

```
⊢ ∀ a0. btree_Leaf_proj (Leaf a0) = a0
 (defhol :fns ... :defs ...)
```

From `mirror-thy-exports.lisp`:

```
(DEFTHM HOL::HOL{BTREE_LEAF_PROJ}
  (IMPLIES
   (AND (ALIST-SUBSETP (HOL::MIRROR$HTA) HTA)
        (HPP HOL::A0 HTA)
        (EQUAL (HP-TYPE HOL::A0) (TYP HOL::A))
        (FORCE (HOL::MIRROR$PROP)))
   (EQUAL
    (HAP*
     (HOL::BTREE_LEAF_PROJ
      (TYP (:ARROW* (:BTREE HOL::A) HOL::A)))
     (HAP* (HOL::LEAF (TYP (:ARROW* HOL::A
                                    (:BTREE HOL::A))))
           HOL::A0))
    HOL::A0)))
```

Outline

Overview

Examples

Concluding Remarks

New Since Last Year's Talk

New Since Last Year's Talk

- We translate HOL4 **theorems** and **datatype definitions**, not just function definitions.
  — Includes mutual recursion

New Since Last Year's Talk

- We translate HOL4 **theorems** and **datatype definitions**, not just function definitions.
  — Includes mutual recursion
- Translation handles **quantifiers** and **lambdas** (by lifting).

New Since Last Year's Talk

- We translate HOL4 **theorems** and **datatype definitions**, not just function definitions.
  — Includes mutual recursion
- Translation handles **quantifiers** and **lambdas** (by lifting).
- We added a small **proof example**: correspondence of simple expression evaluators written in HOL and in ACL2.
  - See `examples/eval-poly-proof.lisp` under either `v1/` or `v2/`.
  - This drove further development of the ACL2(zfc) library.

New Since Last Year's Talk

- We translate HOL4 **theorems** and **datatype definitions**, not just function definitions.
  — Includes mutual recursion
- Translation handles **quantifiers** and **lambdas** (by lifting).
- We added a small **proof example**: correspondence of simple expression evaluators written in HOL and in ACL2.
  - See `examples/eval-poly-proof.lisp` under either `v1/` or `v2/`.
  - This drove further development of the ACL2(zfc) library.
- We outlined a **soundness argument** (not mechanized) in the aforementioned paper and expanded it in `projects/hol-in-acl2/soundness/`.

Future Work

The basics are in place, but **much work remains**, including:

Future Work

The basics are in place, but **much work remains**, including:

- the use of induction for reasoning about functions on recursive user-defined datatypes;

Future Work

The basics are in place, but **much work remains**, including:

- the use of induction for reasoning about functions on recursive user-defined datatypes;
- better proof automation;

Future Work

The basics are in place, but **much work remains**, including:

- the use of induction for reasoning about functions on recursive user-defined datatypes;
- better proof automation;
- more applications, e.g., to equivalence of ACL2 and HOL4 formalizations;

Future Work

The basics are in place, but **much work remains**, including:

- the use of induction for reasoning about functions on recursive user-defined datatypes;
- better proof automation;
- more applications, e.g., to equivalence of ACL2 and HOL4 formalizations;
- enhancing soundness argument by

Future Work

The basics are in place, but **much work remains**, including:

- the use of induction for reasoning about functions on recursive user-defined datatypes;
- better proof automation;
- more applications, e.g., to equivalence of ACL2 and HOL4 formalizations;
- enhancing soundness argument by
  - extending to user-defined datatypes, and

Future Work

The basics are in place, but **much work remains**, including:

- the use of induction for reasoning about functions on recursive user-defined datatypes;
- better proof automation;
- more applications, e.g., to equivalence of ACL2 and HOL4 formalizations;
- enhancing soundness argument by
  - extending to user-defined datatypes, and
  - formalizing it in ACL2(zfc)

Future Work

The basics are in place, but **much work remains**, including:

- the use of induction for reasoning about functions on recursive user-defined datatypes;
- better proof automation;
- more applications, e.g., to equivalence of ACL2 and HOL4 formalizations;
- enhancing soundness argument by
  - extending to user-defined datatypes, and
  - formalizing it in ACL2(zfc) (using AI to help?);

Future Work

The basics are in place, but **much work remains**, including:

- the use of induction for reasoning about functions on recursive user-defined datatypes;
- better proof automation;
- more applications, e.g., to equivalence of ACL2 and HOL4 formalizations;
- enhancing soundness argument by
  - extending to user-defined datatypes, and
  - formalizing it in ACL2(zfc) (using AI to help?);

Konrad and I tend to be application-driven. So....

Future Work

The basics are in place, but **much work remains**, including:

- the use of induction for reasoning about functions on recursive user-defined datatypes;
- better proof automation;
- more applications, e.g., to equivalence of ACL2 and HOL4 formalizations;
- enhancing soundness argument by
  - extending to user-defined datatypes, and
  - formalizing it in ACL2(zfc) (using AI to help?);

Konrad and I tend to be application-driven. So....

It will inspire us to have users of this work!

Future Work

The basics are in place, but **much work remains**, including:

- the use of induction for reasoning about functions on recursive user-defined datatypes;
- better proof automation;
- more applications, e.g., to equivalence of ACL2 and HOL4 formalizations;
- enhancing soundness argument by
  - extending to user-defined datatypes, and
  - formalizing it in ACL2(zfc) (using AI to help?);

Konrad and I tend to be application-driven. So....

It will inspire us to have users of this work!

Perhaps better yet:
We would be happy to help someone take over this work.

Future Work

The basics are in place, but **much work remains**, including:

- the use of induction for reasoning about functions on recursive user-defined datatypes;
- better proof automation;
- more applications, e.g., to equivalence of ACL2 and HOL4 formalizations;
- enhancing soundness argument by
  - extending to user-defined datatypes, and
  - formalizing it in ACL2(zfc) (using AI to help?);

Konrad and I tend to be application-driven. So....

It will inspire us to have users of this work!

Perhaps better yet:
We would be happy to help someone take over this work.

- It might be a good start for a PhD topic.
  And maybe Konrad and I could help.