An Integration of Axiomatic Set Theory with ACL2 (Extended Abstract)

ACL2 Workshop 2026

Matt Kaufmann

UT Austin (retired)

November, 2026

Introduction

It can be awkward to use ACL2 to reason about infinite objects, e.g.:

Introduction

It can be awkward to use ACL2 to reason about infinite objects, e.g.:

- paths in temporal logic;
- infinite structures;
- sets of real numbers.

Introduction

It can be awkward to use ACL2 to reason about infinite objects, e.g.:

- paths in temporal logic;
- infinite structures;
- sets of real numbers.

One solution:
Use a theorem prover based on higher-order logic.

Introduction

It can be awkward to use ACL2 to reason about infinite objects, e.g.:

- paths in temporal logic;
- infinite structures;
- sets of real numbers.

One solution:
Use a theorem prover based on higher-order logic.

My solution:
Integrate set theory with ACL2.

Introduction

It can be awkward to use ACL2 to reason about infinite objects, e.g.:

- paths in temporal logic;
- infinite structures;
- sets of real numbers.

One solution:
Use a theorem prover based on higher-order logic.

My solution:
Integrate set theory with ACL2.

Note: Several other provers have integrated set theory, including Isabelle more than 30 years ago!

Introduction

It can be awkward to use ACL2 to reason about infinite objects, e.g.:

- paths in temporal logic;
- infinite structures;
- sets of real numbers.

One solution:
Use a theorem prover based on higher-order logic.

My solution:
Integrate set theory with ACL2.

Note: Several other provers have integrated set theory, including Isabelle more than 30 years ago!
This talk is about extending the reach of ACL2.

Two Goals of This Talk (and the paper)

Two Goals of This Talk (and the paper)

GOAL #1: Introduce **ACL2(zfc)** and point to the following:

Two Goals of This Talk (and the paper)

GOAL #1: Introduce **ACL2(zfc)** and point to the following:

- the *set-theory directory* in the community books, `projects/set-theory/`;

Two Goals of This Talk (and the paper)

GOAL #1: Introduce **ACL2(zfc)** and point to the following:

- the *set-theory directory* in the community books, `projects/set-theory/`;
- :DOC zfc for documentation and :DOC zfc-model for foundational issues (which points to file `logic.txt` for details); and

Two Goals of This Talk (and the paper)

GOAL #1: Introduce **ACL2(zfc)** and point to the following:

- the *set-theory directory* in the community books, `projects/set-theory/`;
- :DOC zfc for documentation and :DOC zfc-model for foundational issues (which points to file `logic.txt` for details); and
- slides and video recordings of three 1.5-hour talks, on the ACL2 Seminar page.

Two Goals of This Talk (and the paper)

GOAL #1: Introduce **ACL2(zfc)** and point to the following:

- the *set-theory directory* in the community books, `projects/set-theory/`;
- :DOC zfc for documentation and :DOC zfc-model for foundational issues (which points to file `logic.txt` for details); and
- slides and video recordings of three 1.5-hour talks, on the ACL2 Seminar page.

GOAL #2: Generate interest in pursuing some of the many opportunities for proof developments in ACL2(zfc).

Two Goals of This Talk (and the paper)

GOAL #1: Introduce **ACL2(zfc)** and point to the following:

- the *set-theory directory* in the community books, `projects/set-theory/`;
- :DOC zfc for documentation and :DOC zfc-model for foundational issues (which points to file `logic.txt` for details); and
- slides and video recordings of three 1.5-hour talks, on the ACL2 Seminar page.

GOAL #2: Generate interest in pursuing some of the many opportunities for proof developments in ACL2(zfc).
Ph.D. topic(s)?

Two Goals of This Talk (and the paper)

GOAL #1: Introduce **ACL2(zfc)** and point to the following:

- the *set-theory directory* in the community books, `projects/set-theory/`;
- :DOC zfc for documentation and :DOC zfc-model for foundational issues (which points to file `logic.txt` for details); and
- slides and video recordings of three 1.5-hour talks, on the ACL2 Seminar page.

GOAL #2: Generate interest in pursuing some of the many opportunities for proof developments in ACL2(zfc).
Ph.D. topic(s)?

See :DOC zfc-future-work.

ACL2(zfc)

ACL2(zfc) combines ACL2 with ZFC:

ACL2(zfc)

ACL2(zfc) combines ACL2 with ZFC:

ZFC = **Z**ermelo-**F**raenkel set theory (ZF) extended by the Axiom of Choice (AC).

ACL2(zfc)

ACL2(zfc) combines ACL2 with ZFC:

ZFC = **Z**ermelo-**F**raenkel set theory (ZF) extended by the Axiom of Choice (AC).

- ZFC is an established, intuitive foundation for mathematics.

ACL2(zfc)

ACL2(zfc) combines ACL2 with ZFC:

ZFC = **Z**ermelo-**F**raenkel set theory (ZF) extended by the Axiom of Choice (AC).

- ZFC is an established, intuitive foundation for mathematics.
- Technically: ZFG, where G is Global Choice:

ACL2(zfc)

ACL2(zfc) combines ACL2 with ZFC:

ZFC = Zermelo-Fraenkel set theory (ZF) extended by the Axiom of Choice (AC).

- ZFC is an established, intuitive foundation for mathematics.
- Technically: ZFG, where G is Global Choice:

```
(defthmz min-in-1
  (equal (in (min-in z) z)    ; min-in(z) ∈ z
         (not (equal z 0))))  ; z ≠ ∅
```

ACL2(zfc)

ACL2(zfc) combines ACL2 with ZFC:

ZFC = Zermelo-Fraenkel set theory (ZF) extended by the Axiom of Choice (AC).

- ZFC is an established, intuitive foundation for mathematics.
- Technically: ZFG, where G is Global Choice:

```
(defthmz min-in-1
  (equal (in (min-in z) z)    ; min-in(z) ∈ z
         (not (equal z 0))))  ; z ≠ ∅
```

Here's the expansion (we'll come back to this):

```
(defthm min-in-1
  (implies (force (zfc)) ; ``true'' hypothesis
           (equal (in (min-in z) z)
                  (not (equal z 0)))))
```

Basics of Using ACL2(zfc)

```
(include-book "projects/set-theory/top" :dir :system)
(in-package "ZF")
```

BASICS OF USING ACL2(zfc)

```
(include-book "projects/set-theory/top" :dir :system)
(in-package "ZF")
```

The "top" book includes "base" (and others), which has the following *initial encapsulate* for basic ZFG axioms.

BASICS OF USING ACL2(zfc)

```
(include-book "projects/set-theory/top" :dir :system)
(in-package "ZF")
```

The "top" book includes "base" (and others), which has the following *initial encapsulate* for basic ZFG axioms.

```
(encapsulate
  (((zfc) => *)          ; hypothesis function
   ((in * *) => *)       ; (in a x) ⇔ a ∈ x
   ((pair * *) => *)     ; (pair x y) = {x,y}
   ((min-in *) => *)     ; (min-in x) ∈ x (for x ≠ ∅) ∧
                         ; (min-in x) ∩ x = ∅
   ((union *) => *)      ; (union x) = ⋃{a: a ∈ x}
   ((omega) => *)        ; the set of natural numbers
   ((powerset *) => *))
                         ; (powerset x) = {y: y ⊆ x}
```

BASICS OF USING ACL2(zfc)

```
(include-book "projects/set-theory/top" :dir :system)
(in-package "ZF")
```

The "top" book includes "base" (and others), which has the following *initial encapsulate* for basic ZFG axioms.

```
(encapsulate
  (((zfc) => *)          ; hypothesis function
   ((in * *) => *)       ; (in a x) ⇔ a ∈ x
   ((pair * *) => *)     ; (pair x y) = {x,y}
   ((min-in *) => *)     ; (min-in x) ∈ x (for x ≠ ∅) ∧
                         ; (min-in x) ∩ x = ∅
   ((union *) => *)      ; (union x) = ⋃{a: a ∈ x}
   ((omega) => *)        ; the set of natural numbers
   ((powerset *) => *))
                         ; (powerset x) = {y: y ⊆ x}
  (local (defun zfc () nil))
  ...)
```

Some Axioms Exported from that Encapsulate

```
(defthmz in-pair
  (equal (in a (pair x y)) ; a ∈ {x,y}
         (or (equal a x)
             (equal a y))))
(defthmz in-powerset
  (equal (in a (powerset x))
         (subset a x)))
(defthmdz cons-as-ordered-pair ; <x,y> = {{x},{x,y}}
  (equal (cons x y)
         (pair (singleton x)
               (pair x y))))
(defthmz infinity
  (equal (in n (omega))
         (natp n)))
```

Two Views of ACL2(zfc)

Let's discuss the following two views, which are crucial.

Two Views of ACL2(zfc)

Let's discuss the following two views, which are crucial.

- **System View**: Include the `"top"` book and view your work as a standard ACL2 development.

Two Views of ACL2(zfc)

Let's discuss the following two views, which are crucial.

- **System View**: Include the `"top"` book and view your work as a standard ACL2 development.
  - No `defaxiom` events, no trust tags, no source code changes, etc. — just books

Two Views of ACL2(zfc)

Let's discuss the following two views, which are crucial.

- **System View**: Include the `"top"` book and view your work as a standard ACL2 development.
  - No `defaxiom` events, no trust tags, no source code changes, etc. — just books

- **Logic View**: ACL2 primitives are **defined in** ZFG, and `(zfc)` is defined to be `t`.

Two Views of ACL2(zfc)

Let's discuss the following two views, which are crucial.

- **System View**: Include the `"top"` book and view your work as a standard ACL2 development.
  - No `defaxiom` events, no trust tags, no source code changes, etc. — just books

- **Logic View**: ACL2 primitives are **defined in** ZFG, and `(zfc)` is defined to be `t`.
  - Ignores `(zfc)` hypothesis, which is `t`; similarly for other *hypothesis functions* (discussed below)

More on the Logic View

A metatheoretic argument (see :DOC zfc-model) provides a set-theoretic interpretation for which `(zfc)` is true...

More on the Logic View

A metatheoretic argument (see :DOC zfc-model) provides a set-theoretic interpretation for which `(zfc)` is true...
so `defthmz` is like `defthm` but for set theory.

More on the Logic View

A metatheoretic argument (see :DOC zfc-model) provides a set-theoretic interpretation for which `(zfc)` is true...
so `defthmz` is like `defthm` but for set theory.

From an earlier slide:

More on the Logic View

A metatheoretic argument (see :DOC zfc-model) provides a set-theoretic interpretation for which `(zfc)` is true...
so `defthmz` is like `defthm` but for set theory.

From an earlier slide:

```
(defthmz min-in-1
  (equal (in (min-in z) z)    ; min-in(z) ∈ z
         (not (equal z 0))))  ; z ≠ ∅

(defthm min-in-1
  (implies (force (zfc)) ; ``true'' hypothesis
           (equal (in (min-in z) z)
                  (not (equal z 0)))))
```

Axiom schemes

ZF formulations incorporate two infinite sets of axioms (schemes):

Axiom schemes

ZF formulations incorporate two infinite sets of axioms (schemes):

- **Comprehension (Subset)**:
  $\{x \in s : P(x)\}$ is a set.

Axiom schemes

ZF formulations incorporate two infinite sets of axioms (schemes):

- **Comprehension (Subset)**:
  $\{x \in s : P(x)\}$ is a set.

- **Replacement**:
  The image of a function $P(x, y)$ on a set s is contained in a set.

Axiom schemes

ZF formulations incorporate two infinite sets of axioms (schemes):

- **Comprehension (Subset)**:
  $\{x \in s : P(x)\}$ is a set.

- **Replacement**:
  The image of a function $P(x, y)$ on a set s is contained in a set.

I'll show you now how Comprehension is implemented in ACL2(zfc).

Comprehension example

Example from the paper:
the *restriction* of a function or relation to a sub-domain.

Comprehension example

Example from the paper:
the *restriction* of a function or relation to a sub-domain.

Specification of restrict:

```
(restrict f s) = {p ∈ f: (car p) ∈ s}
```

Comprehension example

Example from the paper:
the *restriction* of a function or relation to a sub-domain.

Specification of restrict:

```
(restrict f s) = {p ∈ f: (car p) ∈ s}
```

Implementation of restrict:

```
(zsub restrict (f s)
      p
      f
      (in (car p) s))
```

```
(zsub restrict (f s) p f (in (car p) s))

(PROGN
 (ENCAPSULATE (((RESTRICT$PROP) => *)
               ((RESTRICT * *) => *))
   (CHECK-ZSUB RESTRICT (F S)
               P F (IN (CAR P) S))
   (LOCAL (DEFUN RESTRICT$PROP NIL NIL))
   (LOCAL (DEFUN-NX RESTRICT (F S) ...))
   (DEFTHM RESTRICT$COMPREHENSION
     (IMPLIES (FORCE (RESTRICT$PROP))
              (EQUAL (IN P (RESTRICT F S))
                     (AND (IN P F)
                          (IN (CAR P) S))))))
 (TABLE ZFC-TABLE
        'RESTRICT$PROP
        '(ZSUB RESTRICT (F S) P F (IN (CAR P) S)))
 (VALUE-TRIPLE 'RESTRICT))
```

Example theorem:

```
(defthmz relation-p-restrict
  (implies (relation-p f)
           (relation-p (restrict f s)))
  :props (zfc restrict$prop) ; hypothesis functions
  :hints (("Goal"
           :expand ((relation-p (restrict f s))))))
```

Example theorem:

```
(defthmz relation-p-restrict
  (implies (relation-p f)
           (relation-p (restrict f s)))
  :props (zfc restrict$prop) ; hypothesis functions
  :hints (("Goal"
           :expand ((relation-p (restrict f s))))))
```

That generates:

```
(defthm relation-p-restrict
  (implies (and (relation-p f)
                (force (zfc))
                (force (restrict$prop)))
           (relation-p (restrict f s)))
  :hints (("Goal"
           :expand ((relation-p (restrict f s))))))
```

"Higher-order" Capability

"Higher-order" Capability

```
(defun map (f lst)
  (declare (xargs :guard (true-listp lst)))
  (cond ((endp lst) nil)
        (t (cons (apply f (car lst))
                 (map f (cdr lst))))))
```

"Higher-order" Capability

```
(defun map (f lst)
  (declare (xargs :guard (true-listp lst)))
  (cond ((endp lst) nil)
        (t (cons (apply f (car lst))
                 (map f (cdr lst))))))
(defun fact (n)
  (declare (type (integer 0 *) n))
  (if (zp n) 1 (* n (fact (1- n)))))
```

"Higher-order" Capability

```
(defun map (f lst)
  (declare (xargs :guard (true-listp lst)))
  (cond ((endp lst) nil)
        (t (cons (apply f (car lst))
                 (map f (cdr lst))))))
(defun fact (n)
  (declare (type (integer 0 *) n))
  (if (zp n) 1 (* n (fact (1- n)))))
(zify zfact fact :dom (omega) :ran (omega))
```

"Higher-order" Capability

```
(defun map (f lst)
  (declare (xargs :guard (true-listp lst)))
  (cond ((endp lst) nil)
        (t (cons (apply f (car lst))
                 (map f (cdr lst))))))
(defun fact (n)
  (declare (type (integer 0 *) n))
  (if (zp n) 1 (* n (fact (1- n)))))
(zify zfact fact :dom (omega) :ran (omega))
(defthm zfact-is-fact ; REDUNDANT
  (implies (and (in fact-arg (omega))
                (force (zfact$prop))
                (force (zify-prop)))
           (equal (apply (zfact) fact-arg)
                  (fact fact-arg))))
```

"Higher-order" Capability

```
(defun map (f lst)
  (declare (xargs :guard (true-listp lst)))
  (cond ((endp lst) nil)
        (t (cons (apply f (car lst))
                 (map f (cdr lst))))))
(defun fact (n)
  (declare (type (integer 0 *) n))
  (if (zp n) 1 (* n (fact (1- n)))))
(zify zfact fact :dom (omega) :ran (omega))
(defthm zfact-is-fact ; REDUNDANT
  (implies (and (in fact-arg (omega))
                (force (zfact$prop))
                (force (zify-prop)))
           (equal (apply (zfact) fact-arg)
                  (fact fact-arg))))
(thmz (equal (map (zfact) '(2 3 4 5))
             '(2 6 24 120))
      :props (zify-prop zfact$prop))
```

Some ACL2(zfc) Developments

- ordinals, including trichotomy
- de Morgan's laws and other theorems on set algebra
- transfinite induction
- transitive closure (closure under set membership)
- finite sets and sequences
- iterated composition
- function spaces
- Cantor's Theorem (no surjection onto powerset)
- the Schröder-Bernstein Theorem (using Grant Jurgensen's ACL2 formalization and proof)
- Beginnings of pure math developments:
  - real numbers (as Dedekind cuts with addition and negative)
  - groups
  - topology (compact Hausdorff space is normal)
- translation from HOL4 to ACL2(zfc) [rump session talk]

What now?

I don't have further plans for ACL2(zfc), but....

What now?

I don't have further plans for ACL2(zfc), but....

- There is plenty to do; for example, see :DOC zfc-future-work.

What now?

I don't have further plans for ACL2(zfc), but....

- There is plenty to do; for example, see :DOC zfc-future-work.
- I'd be happy to help someone take this over

What now?

I don't have further plans for ACL2(zfc), but....

- There is plenty to do; for example, see :DOC zfc-future-work.
- I'd be happy to help someone take this over (PhD dissertation topic(s)?)