HARDWARE/SOFTWARE COASSURANCE USING ALGORITHMIC C AND ACL2

David Hardin
Collins Aerospace

<first>.<last>@collins.com

© 2019 Collins Aerospace, a United Technologies company. All rights reserved.

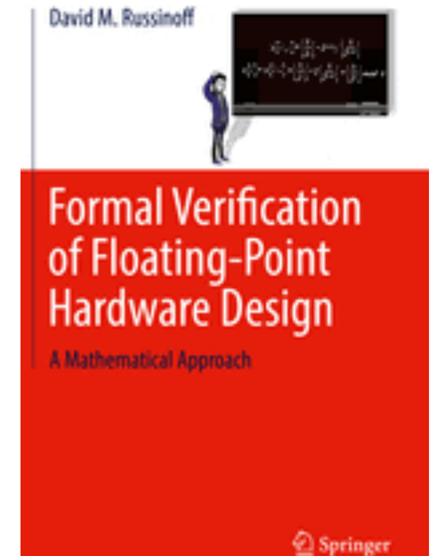
INTRODUCTION

- Floating-point hardware verification is a signature success of formal methods
- Automated theorem proving has been used in the verification of many floating-point hardware designs, including those from:
  - AMD
  - ARM
  - Centaur (x86-compatible)
  - Intel
  - Oracle (SPARC)
- Yet, outside the reuse of basic bit-vector libraries, little use of the specific tools and techniques of floating-point verification has been made outside of that domain
- In this talk, we will describe an experiment in the use of a particular approach to floating-point verification to produce verified algebraic data types, suitable for implementation in either hardware or software

© 2019 Collins Aerospace, a United Technologies company. All rights reserved.

THE RUSSINOFF-O'LEARY APPROACH TO FLOATING POINT HARDWARE VERIFICATION

- The floating-point hardware verification approach we employ was developed by David Russinoff and John O'Leary, while both were at Intel (ACL2 Workshop 2014)
  - The approach was initially based on SystemC, and was called MASC
  - Russinoff changed the source language from SystemC to Algorithmic C after he moved to ARM, made several enhancements, and renamed the system RAC (Restricted Algoritmic C)
- RAC is extensively documented in Russinoff's 2018 book, *Formal Verification of Floating-Point Hardware Design: A Mathematical Approach*, wherein RAC is applied to the verification of realistic ARM floating-point designs
  - RAC, and the verifications described in the book, are all available as part of the standard ACL2 distribution



Collins Aerospace

© 2019 Collins Aerospace, a United Technologies company. All rights reserved.

ALGORITHMIC C

- The Algorithmic C datatypes “provide a basis for writing bit-accurate algorithms to be synthesized into hardware”
- The Algorithmic C datatypes are defined via an open source C++ header file that users can #include in their designs
  - No runtime library required
- Example use:
  - `typedef ac_int<112,false> ui112;`

  declares an unsigned 112-bit type used in floating-point hardware datapaths
- Supported by Mentor hardware synthesis tools
- Further information is available at **`https://hlslibs.org`**

Collins Aerospace

© 2019 Collins Aerospace, a United Technologies company. All rights reserved.

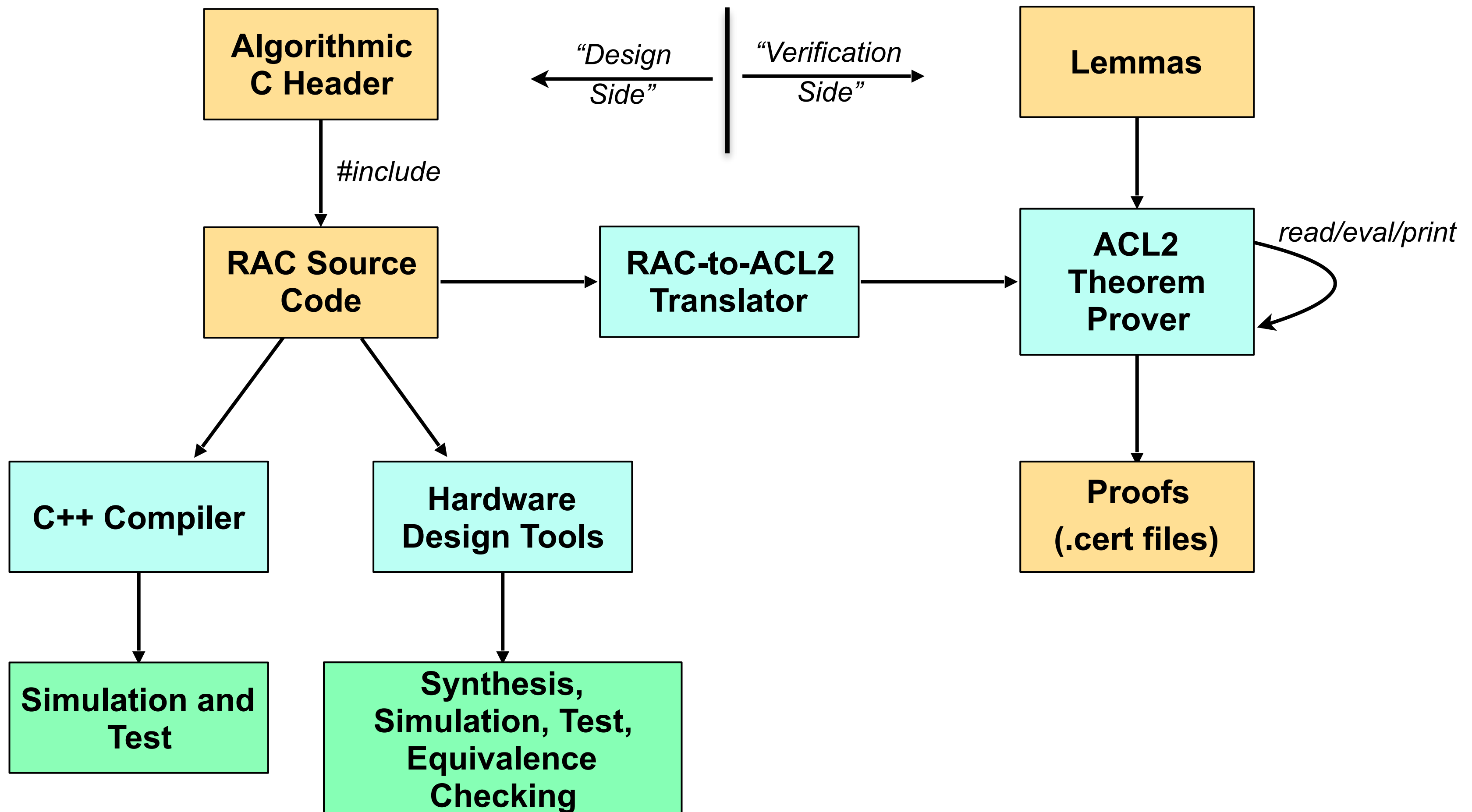
RESTRICTED ALGORITHMIC C (RAC)

- Restricted Algortihmic C defines a C subset that promotes proof, hardware synthesis, and simulation
- Use case: A hardware developer expresses hardware functionality in RAC, which is then translated into a theorem prover language used by the verification expert
- RAC encompasses many of the restrictions common in “high-assurance” C, such as no function pointers, disallowing recursion, etc.
  - RAC also disallows all pointers, as well as function side-effects
  - Certain control constructs (such as breaking out of a **`for`** loop) are disallowed
- Provides support for bit slices and multiple-value return
- For more information on RAC, please consult Chapter 15 of Russinoff’s book

Collins Aerospace

© 2019 Collins Aerospace, a United Technologies company. All rights reserved.

RAC TOOLCHAIN (SIMPLIFIED VIEW)



© 2019 Collins Aerospace, a United Technologies company. All rights reserved.

ACL2

- ACL2 is "A Computational Logic for Applicative Common Lisp", developed by Matt Kaufmann and J Moore
  - ACL2 is a winner of the ACM Software Systems Award

- ACL2 developers model their system as Common Lisp functions, then state and prove theorems about their model using ACL2's highly automated proof heuristics
  - These functions and theorems are gathered into libraries, called books, which are proved once, then utilized many times

- ACL2 has been used in many large academic and industrial verification efforts:
  - Floating-point unit verification (AMD, ARM, Centaur, Intel, Oracle)
  - AAMP7 separation kernel microcode and Green Hills INTEGRITY-178B kernel information flow verification (Collins Aerospace)
  - Used to certify the correctness of the "world's largest math proofs" (Heule)
    - Proofs are discovered by massively parallel SAT solving

© 2019 Collins Aerospace, a United Technologies company. All rights reserved.

BRIDGING THE DESIGN/VERIFICATION GULF

- A key issue in the formal verification of engineering artifacts is the gulf between the sorts of programs that can be readily specified and verified, and the sorts of programs that “real-world” developers actually write:

Formal Verification “Comfort Zone”	***Real World***
Functional Programming	**Imperative Programming**
Non-tail-recursive functions	**Loops**
Okasaki-style pure functional algebraic data types	**Structs and Arrays**
Infinite-precision Integers	**Modular Integers**
Linear arithmetic	**Linear and non-linear arithmetic**

- The Russinoff-O’Leary toolchain, in combination with the ACL2 theorem prover, does an admirable job of bridging these two worlds

Collins Aerospace

© 2019 Collins Aerospace, a United Technologies company. All rights reserved.

RAC-TO-ACL2 TRANSLATOR

- Translates loops into tail-recursive functions

- Generates ACL2 “measures” to aid in function termination proofs
  - All functions to be admitted into ACL2 must be proved to terminate
  - Termination proofs are conducted mostly automatically by ACL2, with hints provided by the measure annotations (if needed)

- Translates fixed-width integer operations into functions defined in Russinoff’s “RTL” (Register Transfer Language) ACL2 books
  - Ensures that translated operations are “wrapped” with an appropriate RTL bit-width coercion operator so as to accurately translate modular integer aritmetic
  - RTL is described in detail in Part I of Russinoff’s book

Collins Aerospace

© 2019 Collins Aerospace, a United Technologies company. All rights reserved.

RAC-TO-ACL2 TRANSLATOR (CONT'D.)

- Converts assignments to Lisp let-bindings
- Converts struct/array reads/writes to ACL2 record gets/sets, for which get-over-set, set-over-get, etc. theorems are available
- In addition, ACL2's powerful arithmetic capability allows it to reason about non-linear arithmetic expressions
- ACL2 also features a very capable induction scheme generator
  - ACL2 automatically finds suitable induction schemes for the vast majority of inductive proof attempts, including hybrid schemes
- …allowing us to reason about real-world designs expressed in RAC

Collins Aerospace

© 2019 Collins Aerospace, a United Technologies company. All rights reserved.

THE EXPERIMENT

- Utilize the RAC toolchain to
  - specify,
  - verify,
  - and generate executable code

  for a number of classic algebraic datatypes, suitable for implementation either hardware or software components

- Document the experiment and its results in a public forum (this talk)
  - Discuss advantages, shortcomings, and next steps

Collins Aerospace

© 2019 Collins Aerospace, a United Technologies company. All rights reserved.

A USEFUL SIMPLIFICATION

- We don’t want to provide heap management in hardware
- We have experience in utilizing an array-based representation for graph data structures
- Thus, we adopt our array-based approach in our RAC code for algebraic datatypes

© 2019 Collins Aerospace, a United Technologies company. All rights reserved.

ARRAY-BASED GRAPH REPRESENTATION

- Based on a data structure layout approach created for efficient GPU execution (Harish and Narayanan, HiPC 2007); used to code Dijkstra’s All-Pairs Shortest Path algorithm (APSP)

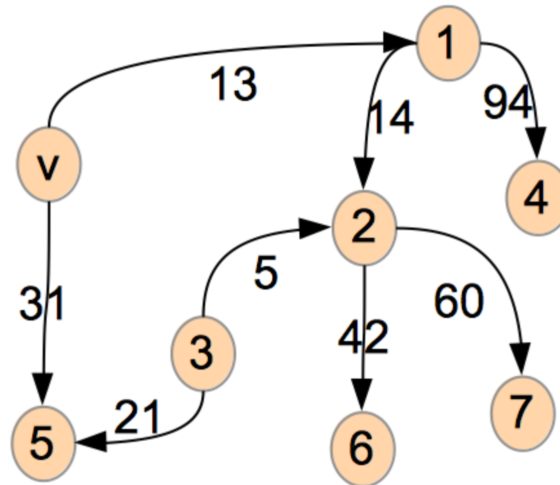
- Amenable to efficient CUDA, OpenCL implementation, as well as hardware implmentation (VHDL)

- Implementated as an ACL2 single-threaded object (ACL2 Workshop 2013)
  - Execution of Dijkstra’s shortest path algorithm on compiled graph using stobjs was linear in number of vertices up to at least 1 million vertices at 10 edges per vertex

- Applied to DASL toolchain (ACL2 Workshop 2018)
  - DASL compiler analyzes datatype, graphtype, and sized declarations, creates appropriate array-based layout, and instantiates runtime functions

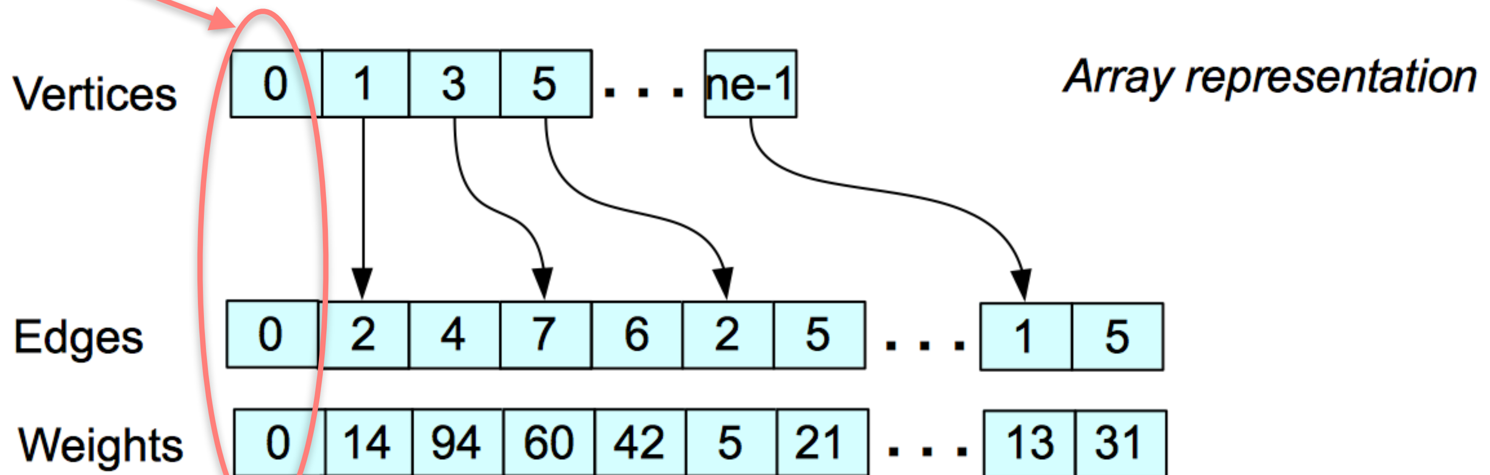
© 2019 Collins Aerospace, a United Technologies company. All rights reserved.

GRAPH COMPILATION EXAMPLE, TWO EDGES PER VERTEX



Graph (incomplete)

"Null" indices



Array representation

© 2019 Collins Aerospace, a United Technologies company. All rights reserved.

EXPERIMENT 1: DOUBLY-LINKED LIST

- In this experiment, we implement a classic doubly-linked list, with “cons” and “rest” mutators on both the head and tail of the list
    - No logic for concurrent update of head and tail in this experiment
- The RAC code maintains the vertices (elements) of the list, as well as the “next” and “previous” edges for the vertices of the list, using our array-based form
- Experiment Goals:
    - Specify a doubly-linked list with millions of vertices in RAC
        - Generate an executable using the clang C++ compiler
    - Translate into ACL2 using the Russinoff-O’Leary toolchain
        - Validate in ACL2 by way of basic unit tests
        - Prove basic functional correctness properties for the translated implementation of the doubly-linked list using ACL2

Collins Aerospace

© 2019 Collins Aerospace, a United Technologies company. All rights reserved.

DOUBLY-LINKED LIST SPECIFICATION IN RAC

```
// Note: MAX_VTX1 == 2**23 (~8 million vertices)

struct DLSTObj {
  ui23 vtxHd;
  ui23 vtxTl;
  ui23 vtxCount;
  // (V) stores the first edge array index for each vertex
  // Note: Must use the new C++ syntax for array declarations
  array<ui24, MAX_VTX1> vtxArr;
  // (D) Data Value array
  array<ui64, MAX_VTX1> valArr;
  // (E) stores the destination vertex for each edge
  array<ui23, MAX_EDGE1> edgeArr;
};
```


© 2019 Collins Aerospace, a United Technologies company. All rights reserved.

DOUBLY-LINKED LIST OPERATORS IMPLEMENTED TO DATE

Operator	*Description*
hd	**Value at head of list**
tl	**Value at tail of list**
ln	**Length of list**
cns	**Add to front of list**
snc	**Add to tail of list**
rst	**Remove head element of list**
tsr	**Remove tail element of list**
ins	**Insert element in front of nth element of list**
del	**Remove nth element of list**

Collins Aerospace

© 2019 Collins Aerospace, a United Technologies company. All rights reserved.

DOUBLY-LINKED LIST "CONS" FUNCTION IN RAC

```
DLSTObj cns (ui64 n, DLSTObj Obj) {
  if (Obj.vtxCount == MAX_VTX) {
    return Obj;
  } else {
    ui23 prevHd = Obj.vtxHd;
    ui23 index = find_free_vtx(Obj);
    if (index == 0) {
      return Obj;
    } else {
      Obj.vtxHd = index;
      if (Obj.vtxCount == 0) {
        Obj.vtxTl = index;
      }
      Obj = add_vtx_at_index(index, n, Obj);
      Obj.edgeArr[Obj.vtxArr[index]] = prevHd;
      if ((prevHd > 0) && (Obj.vtxArr[prevHd] > 0)) {
        Obj.edgeArr[Obj.vtxArr[prevHd] + 1] = index;   // backptr
      }
      return Obj;  }}}
```


© 2019 Collins Aerospace, a United Technologies company. All rights reserved.

DOUBLY-LINKED LIST "CONS" FUNCTION TRANSLATED TO ACL2

```
(DEFUN CNS (N OBJ)
  (IF1 (LOG= (AG 'VTXCOUNT OBJ) 8388607) OBJ
    (LET ((PREVHD (AG 'VTXHD OBJ)) (INDEX (FIND_FREE_VTX OBJ)))
         (IF1 (LOG= INDEX 0) OBJ
           (LET* ((OBJ (AS 'VTXHD INDEX OBJ))
                  (OBJ (IF1 (LOG= (AG 'VTXCOUNT OBJ) 0)
                            (AS 'VTXTL INDEX OBJ) OBJ))
                  (OBJ (ADD_VTX_AT_INDEX INDEX (BITS N 63 0) OBJ))
                  (OBJ (AS 'EDGEARR
                           (AS (AG INDEX (AG 'VTXARR OBJ))
                               PREVHD (AG 'EDGEARR OBJ)) OBJ)))
                 (IF1 (LOGAND1 (LOG> PREVHD 0)
                               (LOG> (AG PREVHD (AG 'VTXARR OBJ)) 0))
                  (AS 'EDGEARR
                      (AS (+ (AG PREVHD (AG 'VTXARR OBJ)) 1)
                          INDEX (AG 'EDGEARR OBJ)) OBJ)
                   OBJ))))))
```

Note: 'AG' and 'AS' are ACL2 untyped record get and set, respectively

© 2019 Collins Aerospace, a United Technologies company. All rights reserved.

EXAMPLE DOUBLY-LINKED LIST FUNCTIONAL CORRECTNESS THEOREMS

```
(defthm hd-of-cns--thm    ;; list head after cons
  (implies
   (and
    (dlstp Obj)            ;; basic "type" predicate
    (good-Objp Obj)        ;; record field consistency predicate
    (spacep Obj)           ;; space available?
    (acl2::unsigned-byte-p 64 n))
   (= (hd (cns n Obj))) n)))

(defthm ln-of-snc--thm    ;; list length after 'cons to tail'
  (implies
   (and
    (dlstp Obj)            ;; basic "type" predicate
    (good-Objp Obj)        ;; record field consistency predicate
    (spacep Obj)           ;; space available?
    (acl2::unsigned-byte-p 64 n))
   (= (ln (snc n Obj))) (1+ (ln Obj)))
```

Collins Aerospace

© 2019 Collins Aerospace, a United Technologies company. All rights reserved.

MORE DOUBLY-LINKED LIST FUNCTIONAL CORRECTNESS THEOREMS

```
(defthm hd-of-rst-of-cns--thm
  (implies
   (and
    (dlstp Obj)              ;; basic "type" predicate
    (good-Objp Obj)          ;; record field consistency predicate
    (spacep Obj)             ;; space available?
    (acl2::unsigned-byte-p 64 n))
   (= (hd (rst (cns n Obj))) (hd Obj))))

(defthm tl-of-tsr-of-snc--thm
  (implies
   (and
    (dlstp Obj)              ;; basic "type" predicate
    (good-Objp Obj)          ;; record field consistency predicate
    (spacep Obj)             ;; space available?
    (acl2::unsigned-byte-p 64 n))
   (= (tl (tsr (snc n Obj))) (tl Obj))))
```

Collins Aerospace

© 2019 Collins Aerospace, a United Technologies company. All rights reserved.

EXERCISING THE DOUBLY-LINKED LIST IN ACL2

```
RTL !>(cns 3 (cns 4 (cns 5 (reset nil))))
((EDGEARR (16777209 . 8388606)
          (16777211 . 8388607)
          (16777212 . 8388605)
          (16777214 . 8388606))
 (VALARR (8388605 . 3)
         (8388606 . 4)
         (8388607 . 5))
 (VTXARR (8388605 . 16777209)
         (8388606 . 16777211)
         (8388607 . 16777213))
 (VTXCOUNT . 3)
 (VTXHD . 8388605)
 (VTXTL . 8388607))
RTL !>(hd (cns 3 (cns 4 (cns 5 (reset nil)))))
3
RTL !>(tl (cns 3 (cns 4 (cns 5 (reset nil)))))
5
```

Note: Arrays are represented as association lists, with array elements in (index . value) form. '0' values are not explicitly stored.

Collins Aerospace

–

© 2019 Collins Aerospace, a United Technologies company. All rights reserved.

RESULTS

- We have developed several data types using the RAC coassurance approach:
  - Stacks
  - Lists
  - Doubly-Linked Lists
  - Binary Trees
  - Binary Search Trees
  - Graphs
- We translated these datatype specifications into ACL2 using the Russinoff-O'Leary toolchain, and developed basic correctness proofs
- We employed the clang C++ compiler to conduct basic simulation and test
  - We also performed validation testing using the ACL2 read/eval/print loop
- No access yet to a Mentor license for hardware synthesis; this is future work

© 2019 Collins Aerospace, a United Technologies company. All rights reserved.

CONCLUDING REMARKS

- Floating-point hardware verification tools and techniques can be employed for more general hardware/software coassurance tasks
- It is feasible to prove correctness properties for algebraic datatype components generated using the RAC toolchain
  - Such efforts would be aided by improvements in the RAC code generator, such as the use of typed ACL2 records
- The RAC-to-ACL2 translator is untrusted code; it would be a worthwhile project to give it a formal foundation
- Generation of verified software components that interface to verified hardware components, both generated using this technique, would make for an instructive future Experiment

© 2019 Collins Aerospace, a United Technologies company. All rights reserved.