Generalization Correctness

David Greve
Rockwell Collins
March 15, 2017

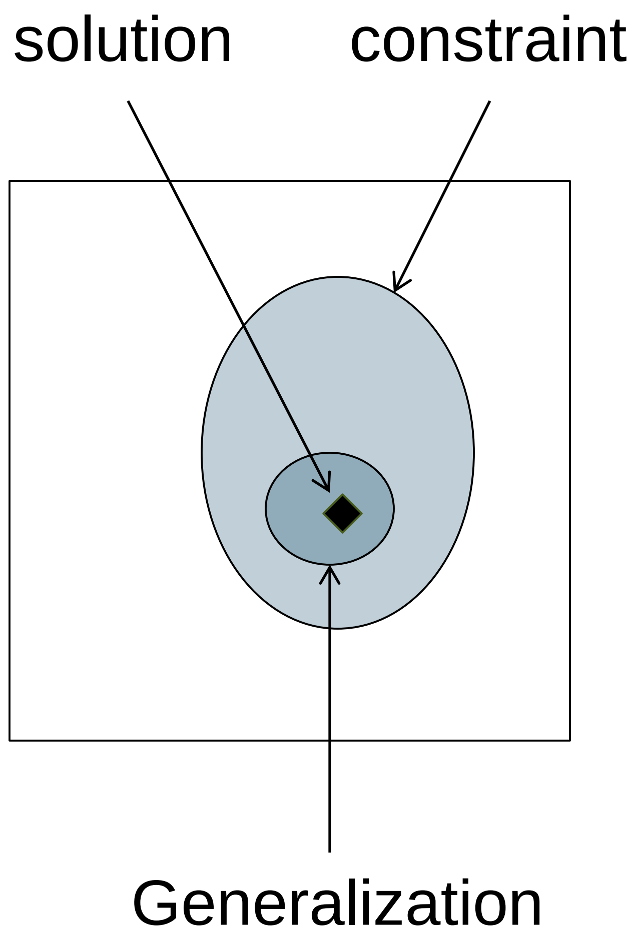
This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA) under Contract FA8750-16-C-0218.

Rockwell Collins
Building trust every day

Rockwell Collins

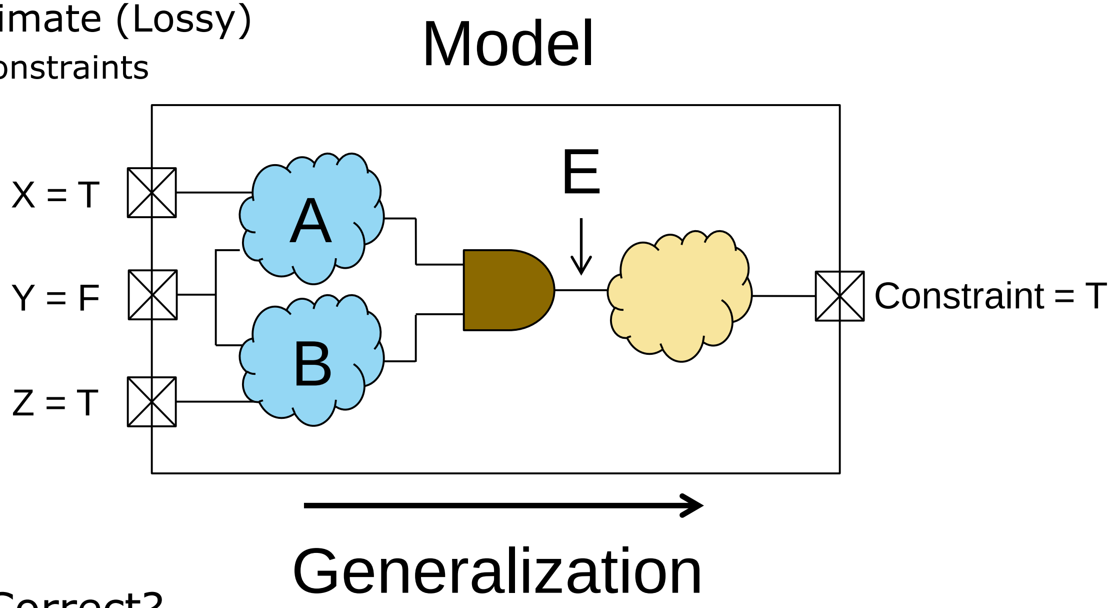
Problem Statement

- Given
  - System Model
  - Constraint
  - Solution provided by Constraint Solver
- Generate a Generalization
  - Convert a single solution into a set of solutions
  - Express Result Concisely
    - Usually Generalization != Constraint
    - Result is Inexact



Generalization Illustration

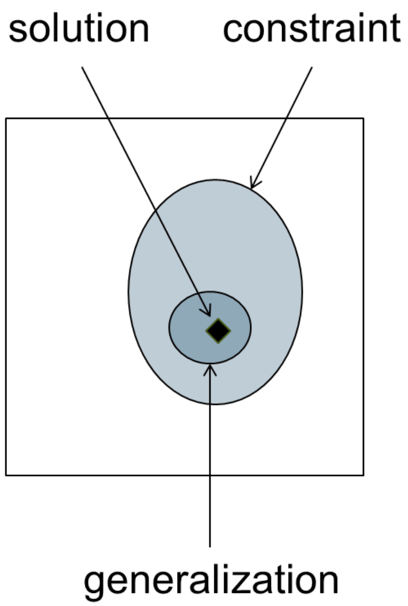
- Computed via Symbolic Simulation
  - System Model + Constraint
  - Original Solution
  - Simulation is Approximate (Lossy)
    - Representational constraints



- Is the Generalization Correct?
  - Formalize Correctness
  - Articulate Generalization Rules
  - Prove Rules Satisfy Correctness

Generalization Correctness Statements

- Top Level Correctness Statement
  - Generalization Contains Original Solution
  - Generalization is a Subset of Original Constraint

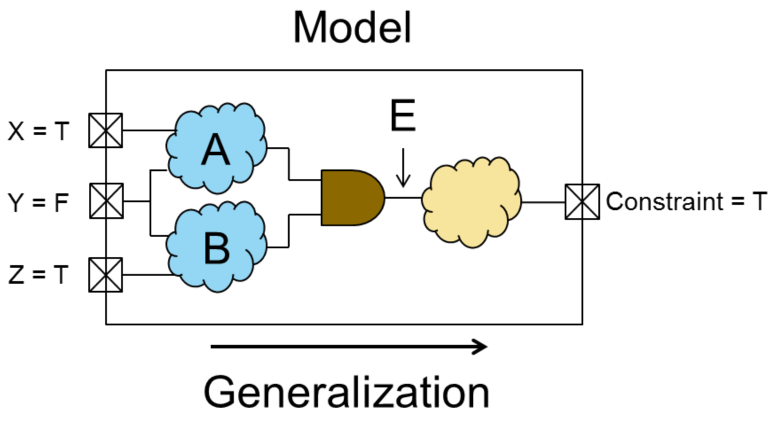


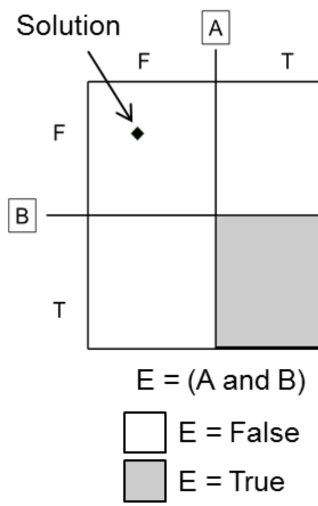
- Invariants
  - Can be enforced incrementally
    - During Symbolic Simulation
  - Reduce to Correctness when applied to top level constraint

- Correctness Invariants
  - 1. Evaluating Solution on Generalization must be the same as Evaluating Solution on original expression
  - 2. An input whose evaluation differs from that of the solution on the original expression must also differ on the Generalization

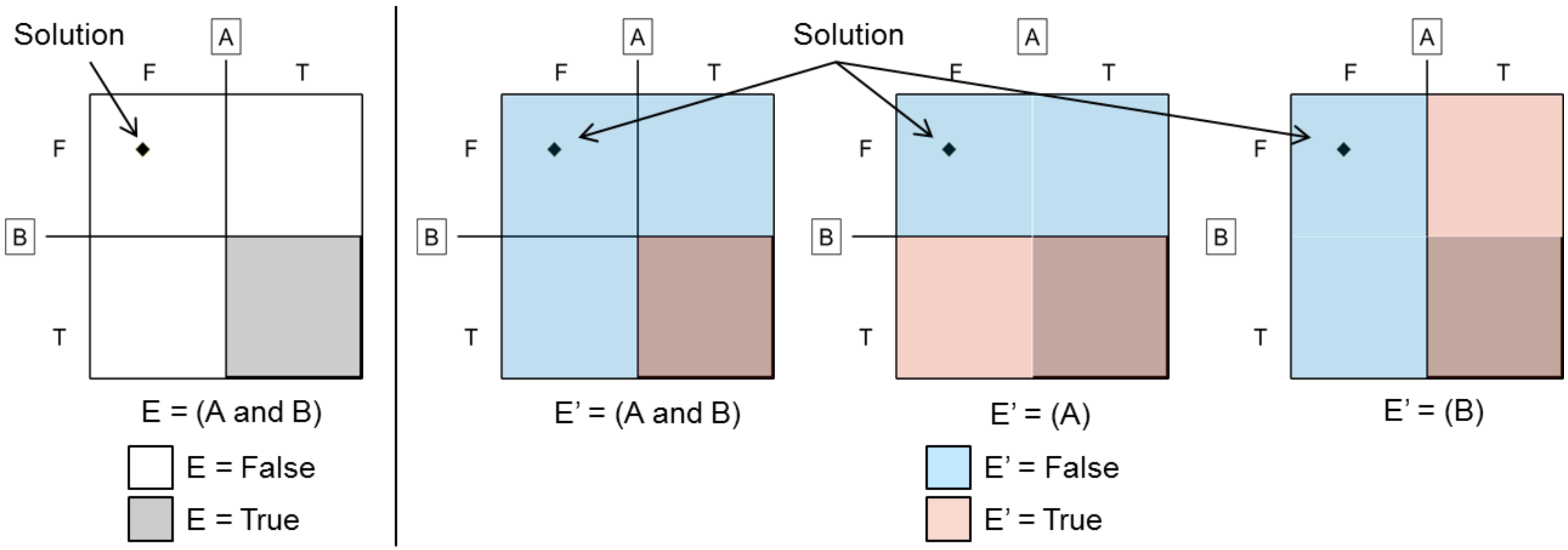
Generalization Rules

- Generalizing Boolean Expressions
  - AND, OR, NOT, ID
- **One** Choice:
  - Drop Terms or Not?
- Visualization
  - State Space
    - Original Solution is one Point
  - Organized as Truth Table w/to A,B
- Consider rules for Generalizing AND
  - OR follows from De Morgan's



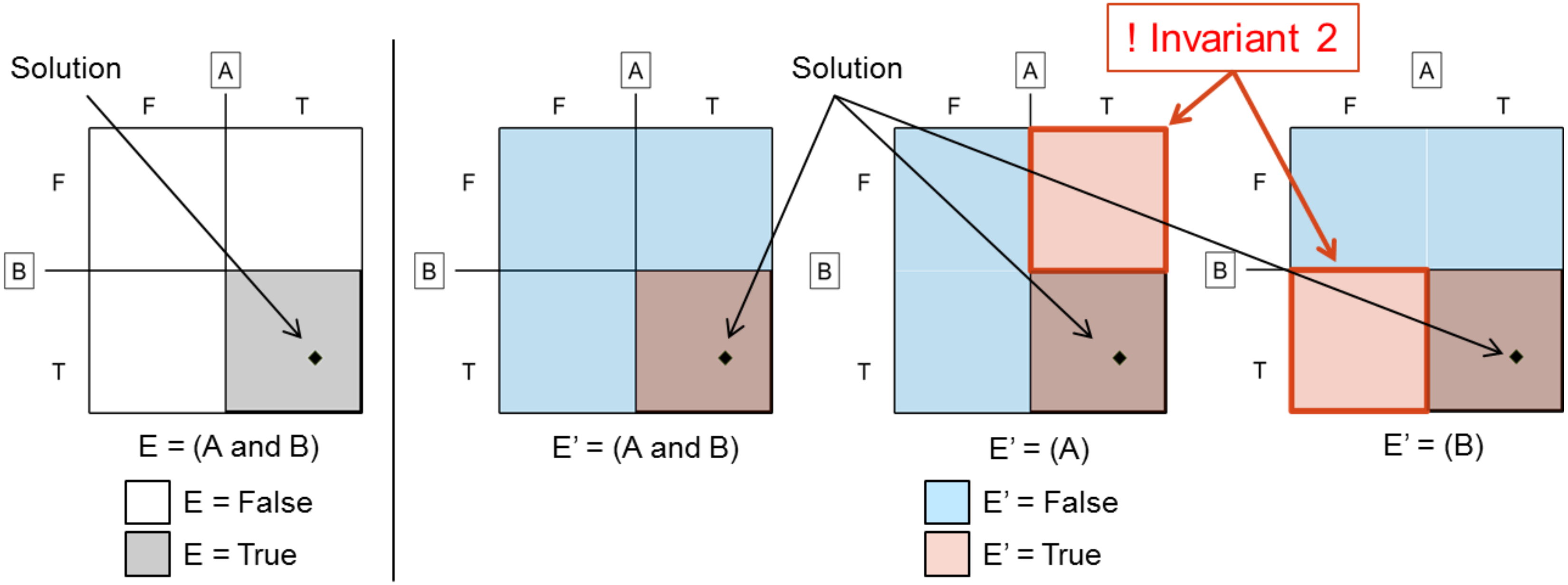


Rule #1: (AND F F)



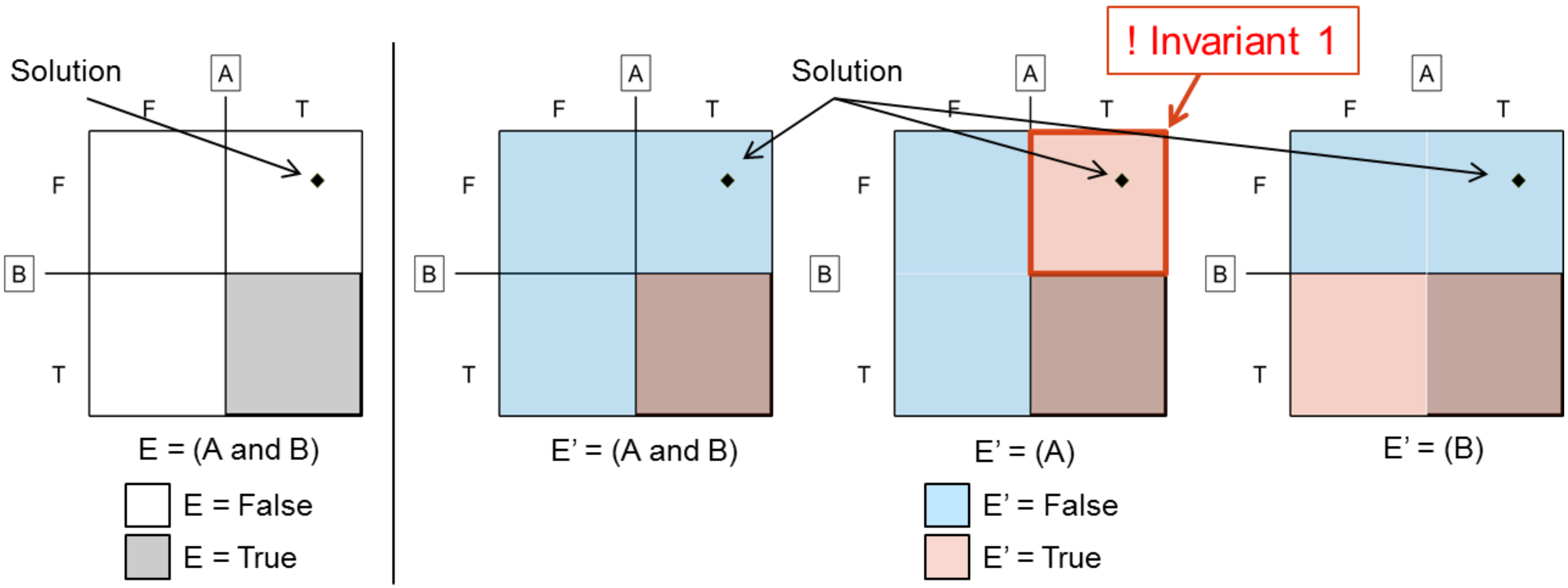
- Correctness Invariants
  - 1. Evaluating Solution on Generalization must be the same as Evaluating Solution on original expression
  - 2. An input whose evaluation differs from that of the solution on the original expression must also differ on the Generalization
- Generalization Rule #1
  - If both expressions evaluate to False, we can either keep both or keep just one

Rule #2: (AND T T)



- Correctness Invariants
  - 1. Evaluating Solution on Generalization must be the same as Evaluating Solution on original expression
  - 2. An input whose evaluation differs from that of the solution on the original expression must also differ on the Generalization
- Generalization Rule #2
  - If both expressions evaluate to True, then we must keep both

Rule #3: (AND T F)



- Correctness Invariants
  - 1. Evaluating Solution on Generalization must be the same as Evaluating Solution on original expression
  - 2. An input whose evaluation differs from that of the solution on the original expression must also differ on the Generalization
- Generalization Rule #3
  - If the expressions evaluate to different values, we can either keep both or keep just the False expression

ACL2 Model

- Defined an expression evaluator
  - Expression and variable binding
  - AND, OR, NOT, IDs
- Used encapsulation to characterize 3 Generalization rules for AND
  - Choice is .. pragmatic
- Defined a depth-first generalizer
  - Returns a "generalized" expression
  - NOT,ID performs no simplification
  - Encapsulated function generalizes AND expressions
  - De Morgan's rule to simplify OR
- Formalized Correctness Invariants
- Proved that generalizer satisfied invariants

Expression Evaluator

```
(defun eval-expr (expr env)
  (case-match expr
    (('and x y)
     (let ((x (eval-expr x env))
           (y (eval-expr y env)))
       (and x y)))
    (('or x y)
     (let ((x (eval-expr x env))
           (y (eval-expr y env)))
       (or x y)))
    (('not x)
     (let ((x (eval-expr x env)))
       (not x)))
    (('id n)
     (nth n env))
    (& expr)))
```

Generalizer Formalization

```
(defun gen-expr (expr sln)
  (case-match expr
    (('and x y)
     (let ((genx (gen-expr x sln))
           (geny (gen-expr y sln)))
       (gen-and genx geny sln)))
    (('or x y)
     (let ((genx (gen-expr x sln))
           (geny (gen-expr y sln)))
       (gen-or genx geny sln)))
    (('not x)
     (let ((genx (gen-expr x sln)))
       (not-expr genx)))
    (& expr)))
```

Applies 'and' Rules

Invariant Proofs

original expression

original solution

```
(defthm invariant-1
  (iff (eval-expr (gen-expr expr sln) sln)
       (eval-expr expr sln))
  :hints (("Goal" :induct (gen-expr expr sln))))
```

- Correctness Invariants
  - 1. Evaluating Solution on Generalization must be the same as Evaluating Solution on original expression
  - 2. An input whose evaluation differs from that of the solution on the original expression must also differ on the Generalization

Invariant Proofs

arbitrary vector

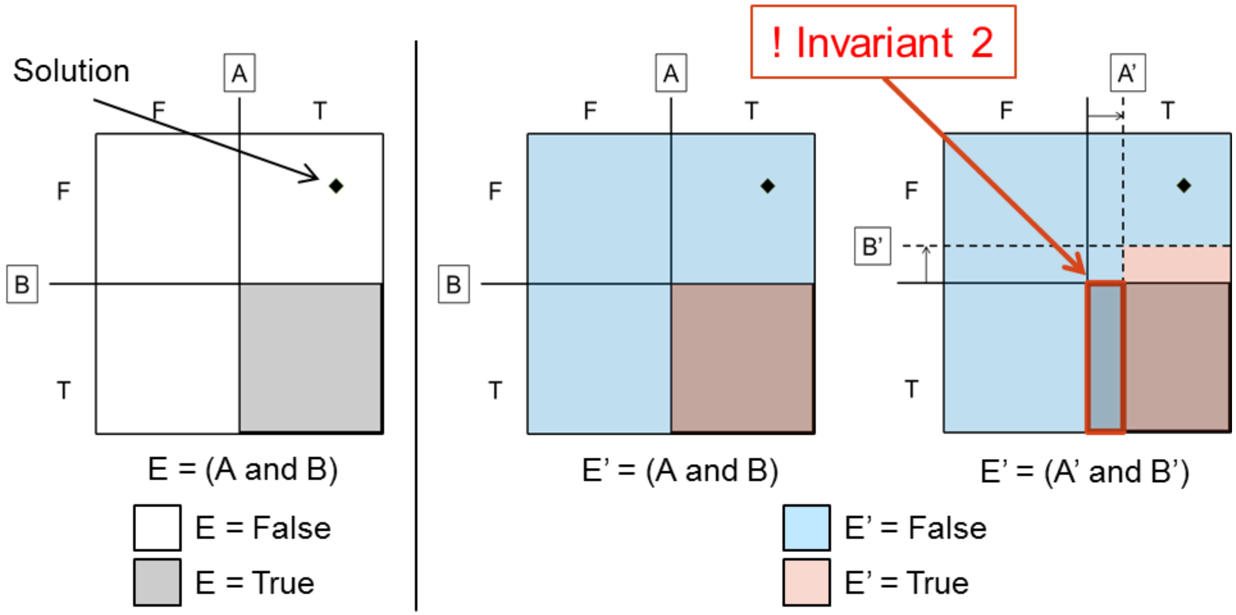
original solution

```
(defthm invariant-2
  (implies
   (iff (eval-expr expr any) (not (eval-expr expr sln)))
   (iff (eval-expr (gen-expr expr sln) any)
        (eval-expr expr any)))
  :hints (("Goal" :induct (gen-expr expr sln)
           :do-not-induct t)))
```

- Correctness Invariants
  - 1. Evaluating Solution on Generalization must be the same as Evaluating Solution on original expression
  - 2. An input whose evaluation differs from that of the solution on the original expression must also differ on the Generalization

PROOF FAILED!

Rule #3: (AND T F)



- Generalization Performed Depth-First
  - Solution space may get smaller (per correctness statement)
  - Predicate boundaries move **closer** to original solution
- Generalization Rule #3
  - If the expressions evaluate to different values, we may keep only the False expression

Conclusion

- We assumed that “Doing Nothing” was conservative
  - If you never change the expression, it trivially satisfies correctness
- We were wrong !
- It is easy to make these kinds of mistakes
  - ACL2 can help during algorithmic development
- Accomplishments
  - Formalized a notion of correctness for Generalization
  - Formalized rules for Generalization
  - Proved Generalization procedure
    - Corrected an error in our original Generalization rules