# Some Challenges for Future ITP

Andy Gordon
Microsoft Research

Workshop on Interactive Theorem Proving (ITP), University of Cambridge, August 2009

# Typecheckers, Refinements, Provers

- Refinement (aka subset) types are a Hot Topic in PL design
  - **type** pos = x:**int** {x > 0}
    **val** sqrt: x:**real** {x >= 0} -> r:**real** {x = r*r}
  - Unifies behavioural types, security types, patterns, Hoare logic, etc
  - Typechecker generates logical goals, passes to automatic prover
  - But what if ATP fails, can we somehow appeal to ITP?

1. F7: Refinement Types for a Concurrent ML
2. Minim: Refinement Types for a Database Query Language
3. Some Observations, Some Challenges

J. Bengtson, K. Bhargavan, C. Fournet, A. Gordon, S. Maffeis,
*Refinement Types for Secure Implementations*, IEEE CSF 2008.

K. Bhargavan, C. Fournet, A. Gordon,
*Modular Verification of Security Protocol Code by Typing*, under review.

# F7 – REFINEMENT TYPES FOR ML WITH CONCURRENCY (F#/OCAML)

# Problem of Verifying Protocol Code

- The problem of vulnerabilities in security protocols is remarkably resistant to the success of formal methods
- Perhaps, tools for verifying the actual protocol code will help
  - Csur (VMCAI'05), fs2pv (CSF'06), F7 (CSF'08), Aspier (CSF'09), etc etc
- Currently, fs2pv most developed, but hitting a wall
  - Translates libraries and protocol code from F#/OCaml to ProVerif
  - ProVerif does whole-program analysis of code versus symbolic attacker
  - Long, unpredictable run times on Cardspace (ASIACCS'08), TLS (CCS'08)

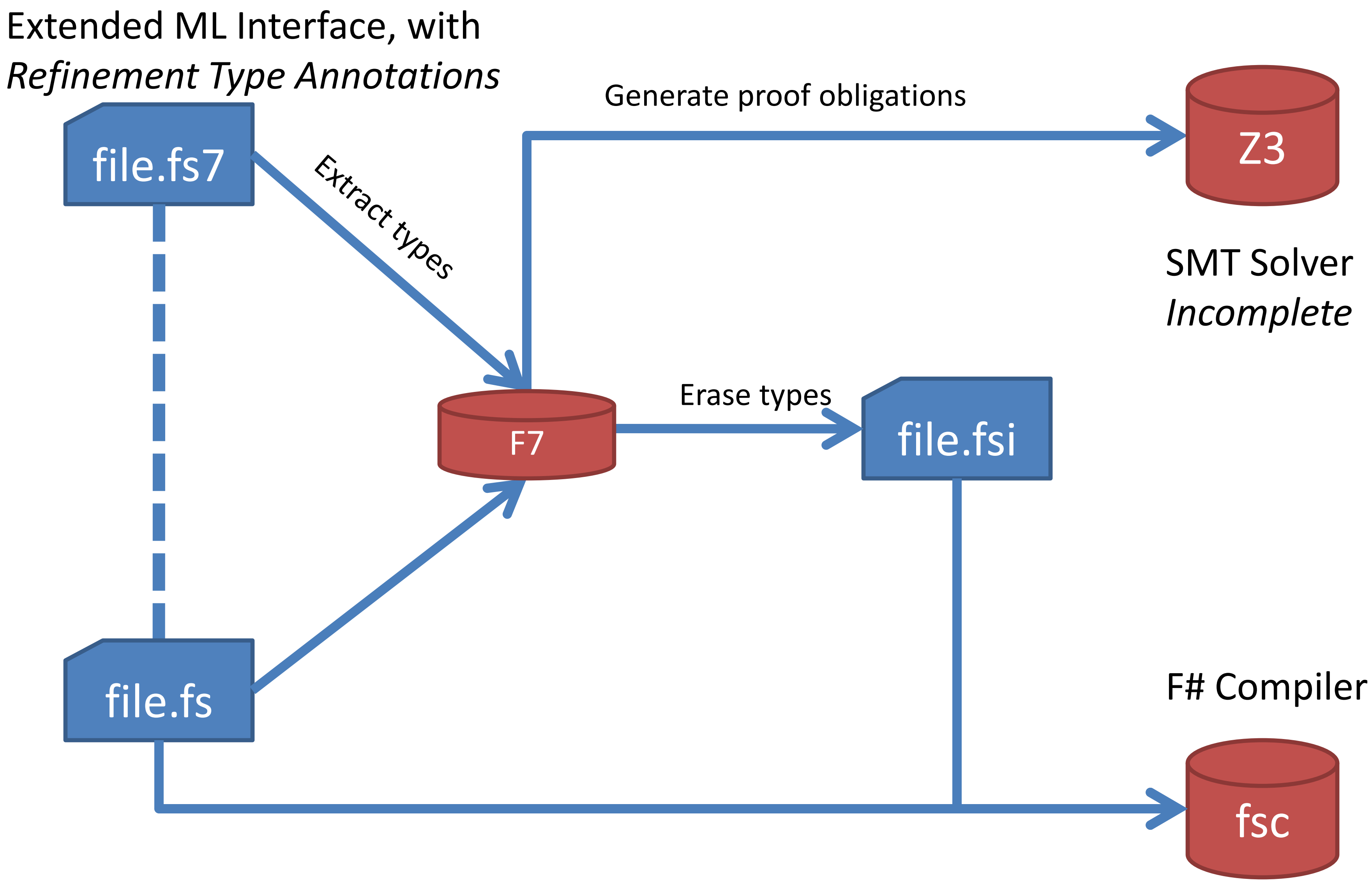
- Instead, we're developing a compositional analysis for the fs2pv libraries and code, based on refinement types

# Refined Types for Crypto APIs

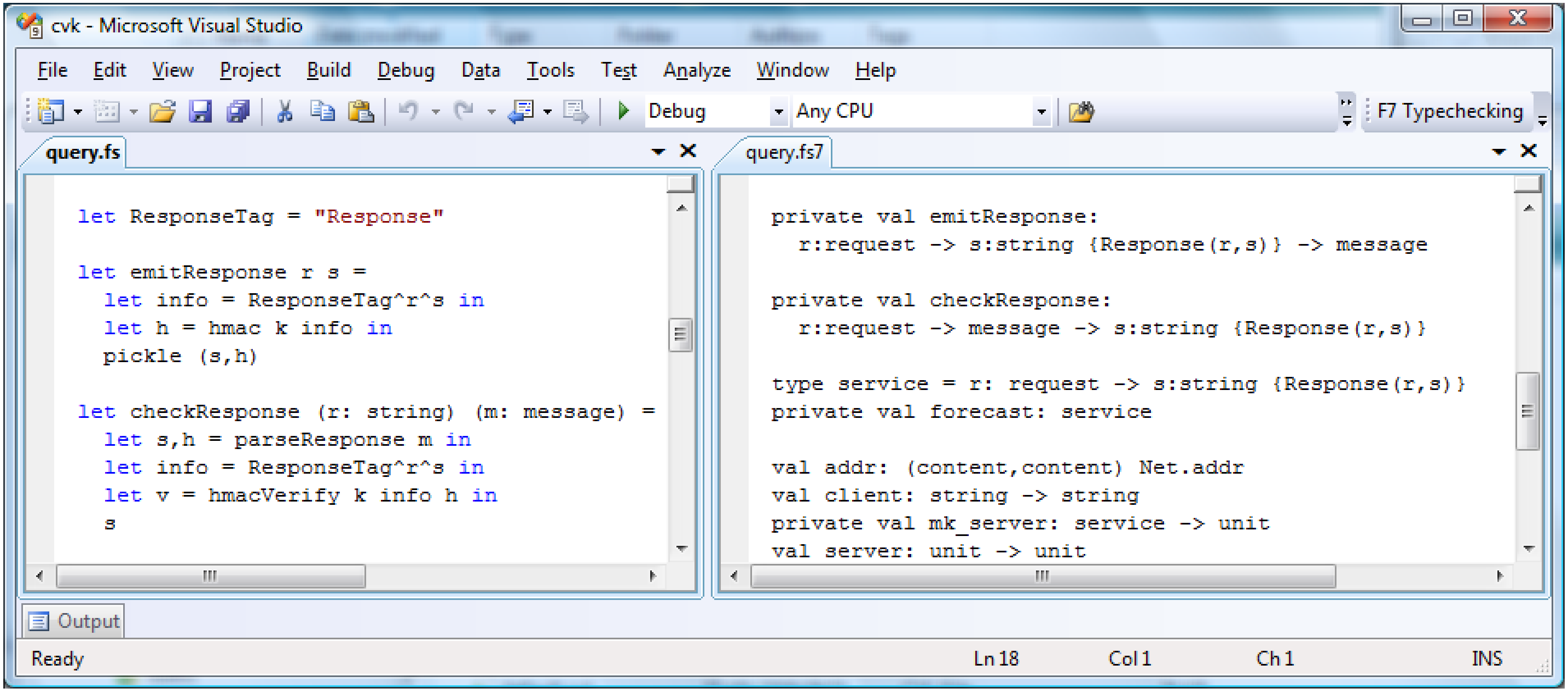
```
val aes_encrypt: (* AES CBC *)
  k:key →
  b:bytes{(SKey(k) ∧ CanSymEncrypt(k,b)) ∨ (Pub(k) ∧ Pub(b))} →
  e:bytes{IsEncryption(e,k,b)}

val aes_decrypt: (* AES CBC *)
  k:key{SKey(k) ∨ Pub(k)} →
  e:bytes →
  b:bytes{(∀p. IsEncryption(e,k,p) ⇒ b = p) ∧ (Pub(k) ⇒ Pub(b))}
```

- APIs enriched with pre- and post-conditions in FOL
- Predicates declared by “equational” or “inductive” definitions
- Typechecker F7 relies on external SMT solver

# F7 Typechecker Implementation



# F7 in Action


cvk - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Test Analyze Window Help
Debug
Any CPU
F7 Typechecking
query.fs
let ResponseTag = "Response"
let emitResponse r s =
let info = ResponseTag^r^s in
let h = hmac k info in
pickle (s,h)
let checkResponse (r: string) (m: message) =
let s,h = parseResponse m in
let info = ResponseTag^r^s in
let v = hmacVerify k info h in
s
query.fs7
private val emitResponse:
r:request -> s:string {Response(r,s)} -> message
private val checkResponse:
r:request -> message -> s:string {Response(r,s)}
type service = r: request -> s:string {Response(r,s)}
private val forecast: service
val addr: (content,content) Net.addr
val client: string -> string
private val mk_server: service -> unit
val server: unit -> unit
Output
Ready
Ln 18
Col 1
Ch 1
INS

# Performance on Larger Protocols

| Example | F# Program | | F7 Typechecking | | Fs2pv Verification | |
|---|---|---|---|---|---|---|
| | Modules | Lines of Code | Interface | Checking Time | Queries | Verifying Time |
| Cryptographic Patterns | 1 | 158 lines | 100 lines | 17.1s | 4 | 3.8s |
| Basic Protocol (Section 2) | 1 | 76 lines | 141 lines | 8s | 4 | 4.1s |
| Otway-Rees (Section 4.2) | 1 | 265 lines | 233 lines | 1m.29.9s | 10 | 8m 2.2s |
| Otway-Rees (No MACs) | 1 | 265 lines | - | (Type Incorrect) | 10 | 2m 19.2s |
| Secure Conversations (Section 4.3) | 1 | 123 lines | 111 lines | 29.64s | - | (Not Verified) |
| Web Services Security Library | 5 | 1702 | 475 | 48.81s | (Not Verified Separately) | |
| X.509-based Client Auth (Section 5.1) | + 1 | + 88 lines | + 22 lines | + 10.8s | 2 | 20.2s |
| Password-X.509 Mutual Auth(Section 5.2) | + 1 | + 129 lines | + 44 lines | + 12s | 15 | 44m |
| X.509-based Mutual Auth | + 1 | + 111 lines | + 53 lines | + 10.9s | 18 | 51m |
| Windows Cardspace (Section 5.3) | 1 | 1429 lines | 309 lines | 6m3s | 6 | 66m 21s |

**Table 1.** Verification Times and Comparison with ProVerif

- F7's compositional type-checking is scaling better than ProVerif's whole-program analysis on these examples
- Still, ProVerif can find attack traces;
  maybe ProVerif's analysis can be modularized?

# Three Observations

- We need some way to justify our assumptions
- ATP is mono-tactical ITP
  - Input via obscure parameters, patterns, repetition
  - Output partly via timing channel
- A lesson learnt from crypto formalisms is that it’s better to start from code and extract logical model, than the converse
  - Think of the C++ “don’t pay if you don’t use” principle
  - F# is in-the-box with Visual Studio 2010 – what will happen?
  - But with some exceptions, this is reverse of tooling I’ve seen for ITP

G. Bierman, A. Gordon, D. Langworthy,
*Semantic Subtyping with an SMT Solver*, under review.

# MINIM – REFINEMENT TYPES FOR A DATABASE QUERY LANGUAGE

# Semantic Subtyping with an SMT Solver

- Since summer 2008, we've been collaborating on the design and implementation of typing for a new database language, M
- M is a data-oriented first-order functional language, combining refinement types (*T where e*) and typecase (*e in T*)
  - A novel combination, useful eg for database integrity constraints
- Our research contributions include:
  - Semantics for M in first-order logic: expressions are terms; types are predicates; (semantic) subtyping is valid implication
  - MSRC Minim checker relies on SMT solver (Z3) to decide subtyping
- Semantic subtyping adds value in key Oslo scenarios (eg DSLs)
  - So, engaging to enhance Oslo codebase with Minim algorithms
  - And, building reference implementation for post-PDC version of M

# Accessing Tagged Unions

File Edit View Window Help Build 3.0.1803.1

untitled1* 146% Minim Mode

```
module M {
  type U : {tag: Logical; data: Any;}
           where (value.tag) ? (value.data in Integer32) : (value.data in Text);

  SomeTaggedData(): U*
  {
    { {tag=>true, data=>42},
      {tag=>false, data=>"freddy"} }
  }

  UnsafeGetText(y:U): Text { y.data }

  SafeGetText(y:U): Text
  {
    (y.tag ? "not text" : y.data)
  }
```

Can't convert y to type {data:Text;}

U is the type of tagged data, where the tag determines the type of the data

A notorious problem is forgetting to check the tag, but Minim catches this

To type-check the else-branch y.data,
we know !(y.tag), and must show the type of y,
which is (U where value==y),
is a **subtype** of the record type {data:Text;}

We check subtyping via a semantics of types in logic, and ask Z3 the following:
"if !(y.tag) and y satisfies (U where value==y), does y satisfy {data: Text;}"

File Edit View Window Help — Build 3.0.1803.1

untitled1* — 146% Minim Mode

```
module M {
    F() : Integer32 where value == 2 { 3 }
}
```

Can't convert 3 to type (value:Integer where (value==2))

The standard M typechecker relies on standard **structural subtyping**;
Structural rules do not work well for the rich type system of M and fail to catch even simple errors like this one, caught by Minim's **semantic subtyping**

File Edit View Window Help — Build 3.0.1803.1

untitled1* — 146% Minim Mode

```
type Statement : {kind:{"assignment"}; var: Text; rhs: Expression;} |
                 {kind:{"while"}; test:Expression; body:Statement;} |
                 {kind:{"if"}; test:Expression; tt:Statement; ff:Statement;} |
                 {kind:{"seq"}; s1:Statement; s2:Statement;} |
                 {kind:{"skip"};};

FindExpr(S:Statement) : (Expression | {null}) {
    (S.kind=="assignment") ? S.rhs :
    ((S.kind=="while" || S.kind=="if") ? S.test : null) }
```

Semantic subtyping effectively checks code manipulating the syntax trees of Domain Specific Language, an important application area for M

# Three Challenges

In the context of Fancy Type Systems, three reasons to use ITP:

1. To Mechanize the Metatheory for the Masses (the **POPLmark Challenge**)
2. To check that FOL theories used in refinement formulas are sound
3. To help out the ATP during type-checking

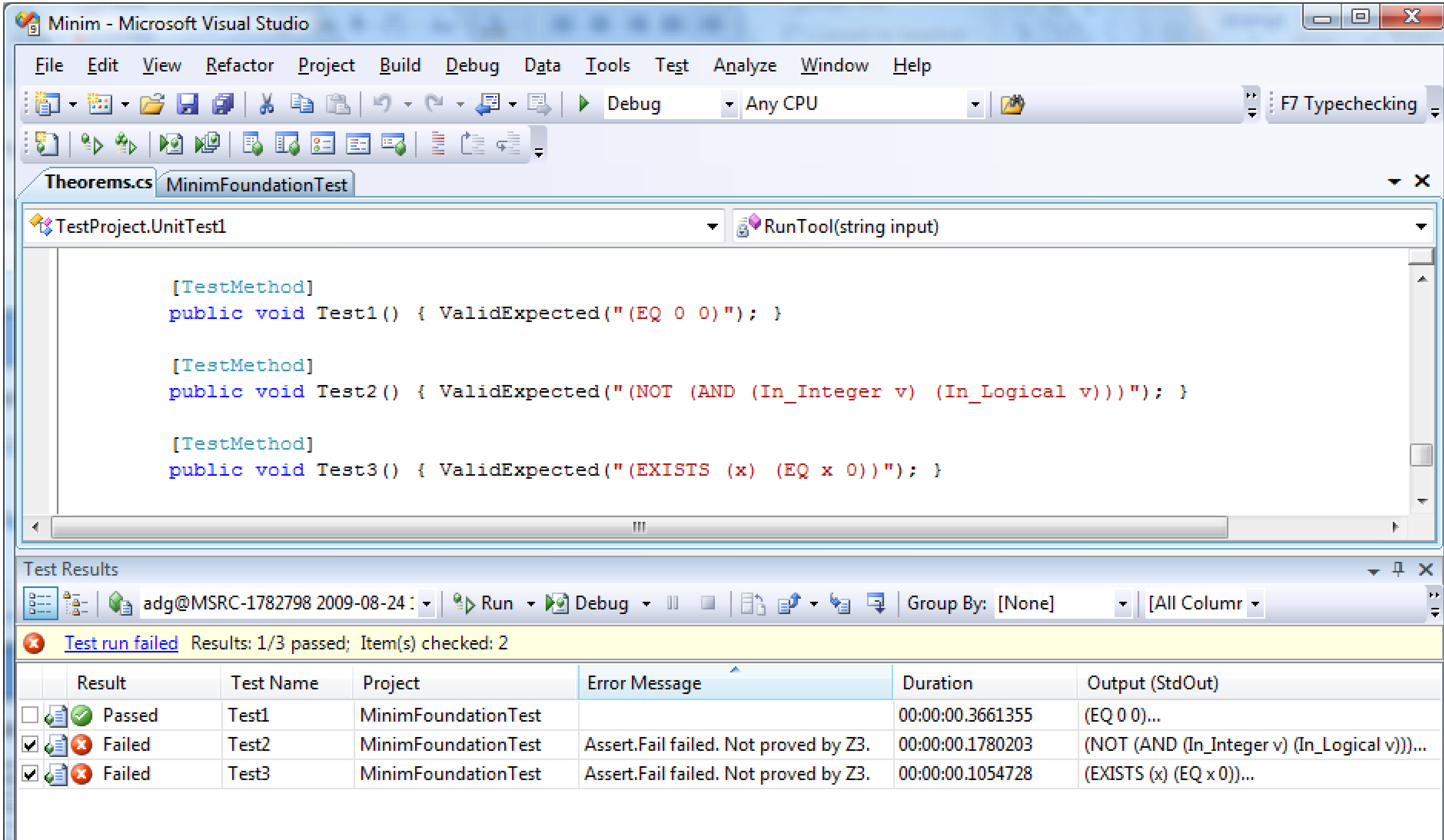
- **Challenge 2:** Steal UI ideas from modern programming and testing environments (as if proofs were programs!)
  - Hover, Pause, F5
- **Challenge 3:** Conversely, can typecheckers steal ideas from ITP to "make the common case easy, and the rare case possible"
  - Annotate code with tactics to help typechecker (cf Why/Caduceus and HOL-Boogie)
  - Least common denominator tactic language? How about an ITP Systems Comp?

File Edit View Window Help Build 3.0.1803.1

untitled1* 236% Simplify Mode

```
(DEFPRED (Man x))
(DEFPRED (Mortal x))

(BG_PUSH (Man Socrates))  ; add to background theory

(Man Socrates)        ; purple formulas proved by Z3
(Mortal Socrates)     ; red formulas not proved

(BG_PUSH (FORALL (x) (IMPLIES (Man x) (Mortal x))))

(Man Socrates)
(Mortal Socrates)

; squiggles updated behind scenes by running Z3
```

# Proof by Testing



# Resources

- Umbrella project, Cryptographic Verification Kit
  http://research.microsoft.com/cvk
- F7: refinement types for F#
  http://research.microsoft.com/F7
- Lectures on Principles and Applications of Refinement Types
  http://research.microsoft.com/en-us/people/adg/part.aspx
- Microsoft "Oslo" Developer Center
  http://msdn.microsoft.com/oslo
- Z3: an efficient SMT solver
  http://research.microsoft.com/en-us/um/redmond/projects/z3/