Principles and Applications of Refinement Types

Andrew D. Gordon (MSR)
ISS AiPL Summer School, August 2009

A Type of Positive Numbers: Why Not?

```
fun MyFun (x:pos, y:pos): pos = if x>y then x-y else 42
```

- **Q:** No currently popular or hip language has these – why not?
- **A:** The typechecker would need to know $\forall x.\ \forall y.\ x>y \Rightarrow x-y>0$ and computers don't do arithmetic reasoning, do they?

- This is an example **refinement type** Integer where value>0
- Known since the 1980s, but typechecking impractical, because automated reasoning is hard, inefficient, and unreliable

Objectives

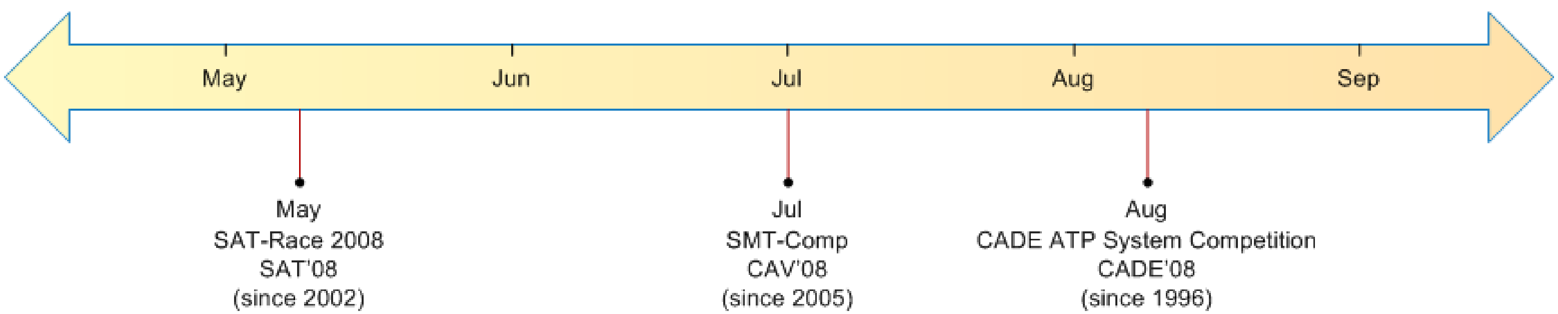
- This lecture is a primer on refinement types
- I'm assuming you know about types in standard languages like C, Java, C#, etc, but not that you're a type theory geek
- Why learn about refinement types?
- What's on offer in this lecture?
- How do I find out more?

- **Q:** How did the typechecker decide $\boxed{\forall x.\ \forall y.\ x>y \Rightarrow x-y>0}$?
- **A:** It didn't. It didn't even try. It asked an SMT solver.

An Opportunity: Logic as a Platform

"Satisfiability Modulo Theory (SMT) solvers decide logical satisfiability (or dually, validity) with respect to a background theory expressed in classical first-order logic with equality. These theories include: real or integer arithmetic, and theories of program or hardware structures such as bitvectors, arrays, and recursive datatypes."

- Dramatic advances in theorem proving this decade
  - Contenders include Simplify (HPL), Yices (SRI), Z3 (MSR)



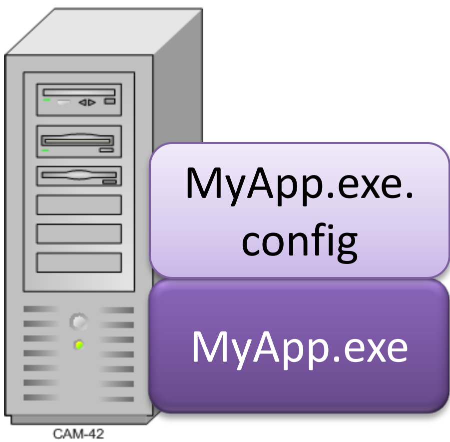
Annual competitions, standard formats for logical goals – a platform

How typechecking based on an external solver makes type-safe systems modeling practical, and helps extend the Microsoft platform

REFINEMENT TYPES AND M

Based on joint work with Gavin Bierman and David Langworthy

M The Oslo Modeling Language




```xml
<?xml version="1.0" encoding="utf-8"?>
<policies xmlns="http://schemas.microsoft.com/wse/2005/06/policy">
  <policy name="policy-CAM-42">
    <mutualCertificate10Security
      establishSecurityContext="false"
      messageProtectionOrder="EncryptBeforeSign">
    </mutualCertificate10Security>
  </policy>
</policies>
```

- Server stacks (eg .NET) allow post-deployment configuration
  - But as server farms scale, manual configuration becomes problematic
  - Better to drive server configurations from a central repository
- M is a new modeling language for such configuration data
  - Ad hoc modeling languages remarkably successful in Unix/Linux world
  - M is in development (first CTP at PDC’08, most recent May 2009)
  - Next, Oslo in their own words...

The Core of the M Language

- A **value** may be a **general value** (integer, text, boolean, null)
- Or a **collection** (an unordered list of values),
- Or an **entity** (a finite map from string labels to values)

- The expression

```
( from n in { 5, 4, 0, 9, 6, 7, 10}
  where n < 5
  select {Num=>n, Flag=>(n>0)} )
```

has the type

```
{Num:Integer; Flag:Logical;}*
```

and evaluates to

```
{{Num=>4,Flag=>true},
 {Num=>0, Flag=>false}}
```

- Semantic domain of values (in F# syntax)

```
type General = G_Integer of int | G_Logical of bool | G_Text of string | G_Null
type Value = G of General | C of Value list | E of (string * Value) list
```

Interdependent Types and Expressions

- A **refinement** type *T* where *e* consists of the values of type *T* such that boolean expression *e* holds
- A **typecase** expression *e* in *T* returns a boolean to indicate whether the value of *e* belongs to type *T*
  - {x=>1, y=>2} in {x:Any;} returns true (due to subtyping)
- A **type ascription** *e* : *T* requires that *e* have type *T*
  - Verify statically if possible
  - Compile to (*e* in *T*) ? *e* : throw "type error" if necessary

Some Examples in M

- Example: type-safe unions
- Demo: comparison of M/MiniM
- Case study: how static typing may help Dynamic IT

Some Derived Types

- Empty type

  Empty $\equiv$ Any where false

- Singleton type

  $\{e\} \equiv$ Any where value==e

- Null type

  Null $\equiv$ {null}

- Union type

  $T \mid U \equiv$ Any where (value in T || value in U)

- Nullable type

  Nullable $T \equiv T \mid$ {null}

Example: Type-Safe Union Types

- Given source

```
type NullableInt : Integer | {null}
from x in ({1, null, 42, null } : NullableInt*)
where x!=null
select (x:Integer)
```

our typechecker calls the solver as follows:

```
(x!=null), x:NullableInt |- x in Integer
===
Asked Z3:
  (BG_PUSH (FORALL (x) (IFF ($NullableInt x) (OR (In_Integer x) (EQ x (v_null))))))
  (IMPLIES (AND (NOT (EQ $x (v_null))) ($NullableInt $x)) (In_Integer $x))
Z3 said : True
```

Interlude: Implementation Notes

- Expressions typed by “bidirectional rules” as in eg C#
  - But no constraint inference
- Subtyping decided semantically, by external solver
  - Term T(*e*) for each expression *e*, formula F(*T*)(*x*) for each type *T*

F([42])(x) = (x=42)
F(Integer where value < 100)(x) = (x<100)

  - Subtyping is implication: T <: U iff $\forall x.$ F(T)(x) $\Rightarrow$ F(U)(x)

[42] <: (Integer where value < 100) iff $\forall$x. (x=42) $\Rightarrow$ (x<100)

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

```
module Constraints
{
  type Person : { Name:Text; Age:Integer32; };
  type EligiblePerson : Person where value.Age > 17;
  type Marriage : { SpouseA: EligiblePerson; SpouseB: EligiblePerson; };

  PatChris(): Marriage
  {
   {SpouseA => {Name => "Pat", Age => 24},
    SpouseB => {Name => "Chris", Age => 32}}
  }

  BillySam(): Marriage
  {
   {SpouseA => {Name => "Billy", Age => 4},
    SpouseB => {Name => "Sam", Age => 5}}
  }
}
```

```
module TaggedUnions
{
  type T1 : {tag: {42}; bar: Integer32;};
  type T2 : {tag: {43}; foo: Text;};
  type U : T1 | T2;

// this fails to typecheck, because it makes insufficient checks
// Test1(xs:U*) : Text* { from x in xs select x.foo  }

  Test2(xs : U*) : Text*
  {
    from x in xs select ( x.tag==42 ? "Hello" : x.foo )
  }

  Test3(xs : U*) : Text*
  {
    from x in xs where (x.tag==43) select x.foo
  }
}
```

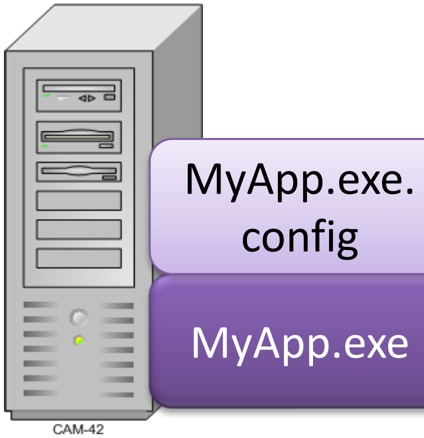
```
//typeful
module Points
{
  type Nat : Integer32 where value==0 || value>0;
  type Byte : Nat where value<256;
  type Color : {Red:Byte; Green:Byte; Blue:Byte;};
  type Point : {X: Integer32; Y:Integer32;};
  type ColorPoint : Point & {Color:Color;};
  type Points : Point*;
  type ColorPoints : ColorPoint*;

  f(x:Point) : ColorPoint { x }
}
```

```
module MinimTests
{
  type Operator : Text where
    value=="plus" || value=="minus" ||
    value=="times" || value=="div";

  type Expression :
  {kind:{"variable"}; name: Text;} |
  {kind:{"integer"}; val: Integer32;} |
  {kind:{"binary app"}; operator: Operator;arg1: Expression; arg2: Expression;};

  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"};};

  FirstExp(E:Expression) : Text
  {
     (E.kind=="variable") ? E.name : (
     (E.kind=="integer") ? "integer" :
      E.operator)
  }

  FirstStatement(S:Statement) : Expression
  {
     (S.kind=="assignment") ? S.rhs : (
     (S.kind=="while" || S.kind=="if") ? S.test :
     {kind=>"integer", val=>42})
  }

  //Test(S:Statement) : Expression { S.rhs }  // this correctly fails to typecheck
}
```

Comparing the MiniM typechecker with the May CTP M typechecker; MiniM focuses on types, lacks significant features like extents

DEMO

Better Dynamic IT by Typing



- Many systems errors arise from misconfigurations
  - Formats often too flexible; operators make mistakes
- Numerous ad hoc tools advise on config "safety"
  - Find misconfigurations in firewalls, routers, protocol stacks, etc; check that adequate security patches have been applied
  - Tools package specialist expertise; more accessible than best practice papers; easy to update as new issues arise
- M is a general purpose platform for systems modeling
  - User-defined types can express advisories, subsuming ad hoc tools
  - Let's look at a concrete example: WSE Policy Advisor

Microsoft Research Cambridge

A Typical Config-Based Advisor

Aftermath:

Servers and Tools customers love this sort of tool

Promoted by the Patterns and Practices group

But, no good platform for writing such tools,

and XSLT not a great programming experience

ements
guration,
isor XSLT
Explorer

Risks and advice for an endpoint policy & config

Advice: Do not use test keys in production: set the attribute **allowTestRoot="false"** in the **<x509>** element of the WSE configuration file.

This policy enables a dictionary attack on an encrypted request, response, or fault whose message

- *StockService* (policy: *MySecurityPolicy*) (SOAP: *request*) [policyCache]
- *StockService* (policy: *MySecurityPolicy*) (SOAP: *response*) [policyCache]
- *StockService* (policy: *MySecurityPolicy*) (SOAP: *fault*) [policyCache]

Risk: The message body is encrypted, but the cryptographic hash of the plaintext message body is also included in the signature. Hence, an attacker that intercepts the message may obtain this hash and compare it to the hash of a large number of potential message bodies. Once two hashes match, the attacker has broken confidentiality of the message body.

Advice: If the body cannot be guaranteed to have high entropy (that is, if the body does not always include some fresh, secret cryptographic value), use either **messageProtectionOrder="EncryptBeforeSign"** or **messageProtectionOrder="SignBeforeEncryptAndEncryptSignature"**.

Microsoft Research Cambridge

1: Representing XML Data

```
<?xml version="1.0" encoding="utf-8"?>
<policies xmlns="http://schemas.microsoft.com/wse/2005/06/policy">
  <policy name="policy-CAM-42">
    <mutualCertificate10Security
      establishSecurityContext="false"
      messageProtectionOrder="EncryptBeforeSign">
    </mutualCertificate10Security>
  </policy>
</policies>
```

```
{tag="policies",
 xmlns="http://schemas.microsoft.com/wse/2005/06/policy",
 body={{tag=>"policy",
          name=>"policy-CAM-42",
          body={{tag=>"mutualCertificate10Security",
                   establishSecurityContext=>"false",
                   messageProtectionOrder=>"EncryptBeforeSign" }}}}}
```

2: Types for Schema-Correct Configs

```
type bool : {"true"} | {"false"};
type messageProtectionOrder : {"EncryptBeforeSign"}|{"SignBeforeEncrypt"};
type mutualCertificate10Security :
   {tag:{"mutualCertificate10Security"};
     establishSecurityContext:bool;
     messageProtectionOrder:messageProtectionOrder; } ;
```

```
Policy = mutualCertificate10Security  | ...
Config = {tag:{"policies"}; body:{tag:{"policy"}; body:Policy*; }*; } ;
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<policies xmlns="http://schemas.microsoft.com/wse/2005/06/policy">
  <policy name="policy-CAM-42">
    <mutualCertificate10Security
      establishSecurityContext="false"
      messageProtectionOrder="EncryptBeforeSign">
    </mutualCertificate10Security>
  </policy>
</policies>
```

has type Config

3: Types for **Safe** Configs

```
type q_credit_taking_attack_10 :
  (mutualCertificate10Security
    where value.messageProtectionOrder == "EncryptBeforeSign") ;
type Advisory = q_credit_taking_attack_10  | ...
```

```
type SafePolicy : Policy & (!Advisory)
type SafeConfig : {tag:{"policies"}; body:{tag:{"policy"}; body:SafePolicy*; }*; } ;
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<policies xmlns="http://schemas.microsoft.com/wse/2005/06/policy">
  <policy name="policy-CAM-42">
    <mutualCertificate10Security
      establishSecurityContext="false"
      messageProtectionOrder="EncryptBeforeSign">
    </mutualCertificate10Security>
  </policy>
</policies>
```

has type Config
but **not** type SafeConfig

Microsoft Research Cambridge

Related Work

			Refinement	Typecase	Subtyping
1983	Nordström/Petersson	**Subset types**	{x:A | B(x)}	no	no
1986	Rushby/Owre/Shankar	**Predicate subtyping**	predicate subtype	no	limited
1989	Cardelli et al	**Modula-3 Report**	no	on references	structural
1991	Pfenning/Freeman	**Refinement types**	refined sorts	no	no
1993	Aiken and Wimmers	**Type inclusion...**	no	no	semantic
1999	Pfenning/Xi	**DML**	{x: General | e}	no	no
1999	Buneman/Pierce	**Unions for SSD**	no	yes, as pattern	structural
2000	Hosoya/Pierce	**XDuce**	no	yes, as pattern	semantic, ad hoc
2006	Flanagan et al	**SAGE**	{x: T | e}	no (but has cast)	structural, SMT
2006	Fisher et al	**PADS**	{x:T | e}	no	structural
2007	Frisch/Castagna	**CDuce**	no	e in T	semantic, ad hoc
2007	Sozeau	**Russell**	{x:T | e}	no	structural
2008	Bhargavan/Fournet/G	**F7/RCF**	{x: T | C} (formula C)	no	structural, SMT
2008	Rondon/Jhala	**Liquid Types**	{x: General | e}	no	structural, SMT
2009	Bierman/G/Langworthy	**M/MiniM**	{x: T | e}	e in T	semantic, SMT

Refinement Types and M

- The interdependence between typecase expressions and refinement types in M is a novel source of great expressivity
- Relying on an external solver achieves type safety for union and dependent types without complex, arbitrary rules
- Security and error checking expressible within M type system
  - Helps M extend the Microsoft platform

- Our Z3-based typechecker Minim was jointly developed with the Oslo team in parallel with the mainline typechecker
  - We hope to merge the code-bases this year

Microsoft Research Cambridge

Applying refinement types to the verification of cryptographic protocols and APIs

REFINEMENT TYPES AND F7

Based on joint work with Karthikeyan Bhargavan and Cédric Fournet

Cryptographic Verification Kit

MICROSOFT SECURITY DEVELOPMENT LIFECYCLE

SECURITY ENGINEERING & COMMUNITY

OCTOBER 1, 2008

Version 4.1

Innovative use of cryptographic constructs often results in subtle (or not so subtle) mistakes. Using standard algorithms in standard ways, or getting expert advice from the crypto board greatly reduces the odds of a problem.

Our goal is a toolkit to verify reference implementations of standardized and custom cryptographic protocols

Case Studies

WS-Security
1750 lines
fs2pv [MSRC'06]

CardSpace
1420 lines
fs2pv [MSRC'08]

TLS 1.0
2940 lines
fs2pv, fs2cv
[MSR-INRIA'08]

Multi-party Sessions
2180 lines
f7 [MSR-INRIA'08]

Application

Typed Interface

Protocol

Typed Interface

Crypto Library	Computational Crypto	Symbolic Crypto
Network Library	Poly-time Adversary	Active Adversary

fsc — *Concrete runs and interop tests over .NET Runtime*

fs2cv — *Computational crypto proof using CryptoVerif*

fs2pv — *Symbolic proof or attack trace*

f7 — ***Symbolic proof by typing using Z3***

```
// a refined type for strings that are genuine requests
type request = s:string {Request(s)}

// a key trusted for authenticating requests
val k: request hkey
```

```
let verify message =
  let payload, mac = unpickle message in
  // is this the right cryptographic check?
  let request = hmacVerify k payload mac in
  request

let server() =
  let socket = Net.listen addr in
  let message = Net.recv socket in
  let request = verify message in
  // can I be sure this request is authentic?
  assert Request(request);
  let response = handle request in
  let msg2 = emitResponse request response in
  Net.send socket msg2
```

Verification Tools for F#

- Statically verify security assertions
- Different techniques, cryptographic models

Microsoft Research

K Bhargavan, C Fournet, AD Gordon (MSR Cambridge), R Corin, P-M Deniélou, JJ Leifer, E Zalinescu (MSR-INRIA)

Centre de Recherche Commun INRIA Microsoft Research

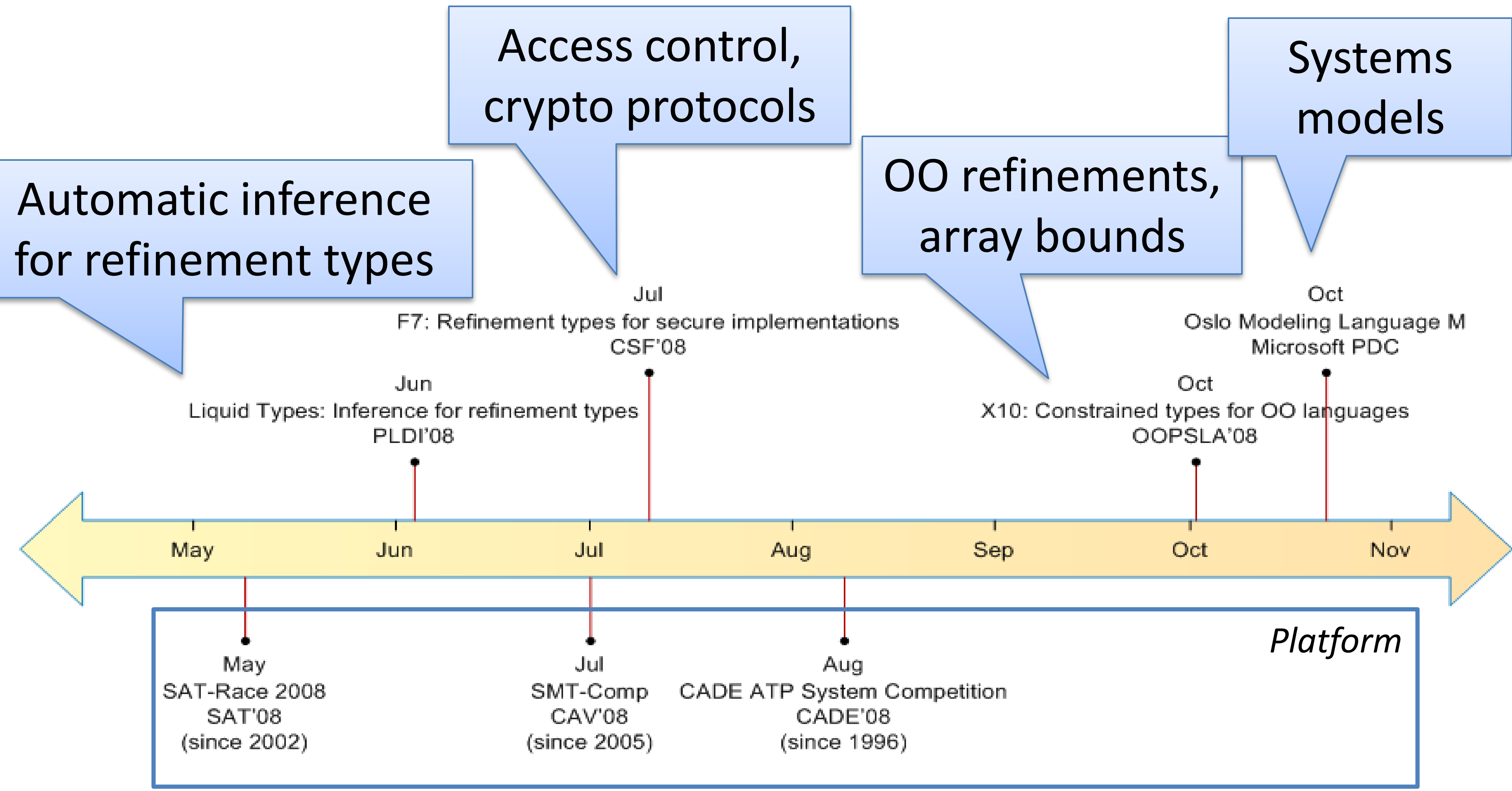
F7: Refinements for Security

Check out our site http://research.microsoft.com/cvk

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

A Good Year for Refinements



Ideas to Take Away

- Remember the riddle
  - **Q:** How did the typechecker decide $\forall x.\ \forall y.\ x>y \Rightarrow x-y>0$?
  - **A:** It didn't. It didn't even try. It asked an SMT solver.
- Remember that boundaries are blurring
  - Between types, predicates, policies, patterns, schemas
  - Between typechecking and verification
- Still, SMT solvers are incomplete, often amazingly so
  - So dealing with typing errors remains a challenge

Resources

- The Microsoft Research SMT solver, Z3
  http://research.microsoft.com/en-us/um/redmond/projects/z3/
- Oslo and its modeling language, M
  http://msdn.microsoft.com/oslo
- Refinement types for security in F#
  http://research.microsoft.com/f7
- Liquid types (including online demo)
  http://pho.ucsd.edu/liquid/
- This lecture
  http://research.microsoft.com/en-us/people/adg/part.aspx

THE END