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** \( \text{Integer where value}>0 \)

• 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 $\forall x. \forall y. x>y \Rightarrow x-y>0$?
• 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

REFINEMENT TYPES AND M

Based on joint work with Gavin Bierman and David Langworthy

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

- Server stacks (e.g., .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...

```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>
```
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)

![Image](image.png)

• The expression

```fsharp
(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)

```fsharp
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 \( \{x: \text{Any};\} \) 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
  \[ \text{Empty} \equiv \text{Any where } \text{false} \]

- Singleton type
  \[ \{e\} \equiv \text{Any where } \text{value}==e \]

- Null type
  \[ \text{Null} \equiv \{\text{null}\} \]

- Union type
  \[ T \mid U \equiv \text{Any where } \text{value in } T \mid \text{value in } U \]

- Nullable type
  \[ \text{Nullable } T \equiv T \mid \{\text{null}\} \]
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 $T$
    
    $$
    \begin{align*}
    F([42])(x) &= (x=42) \\
    F(\text{Integer where value < 100})(x) &= (x<100)
    \end{align*}
    $$

  - Subtyping is implication: $T <: U$ iff $\forall x. F(T)(x) \Rightarrow F(U)(x)$
    
    $$
    [42] <: (\text{Integer where value < 100}) \text{ iff } \forall x. (x=42) \Rightarrow (x<100)
    $$
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
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.

---

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 body was encrypted.

- `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`. 
1: Representing XML Data

```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>
```

```json
{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 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>
3: Types for Safe Configs

type q_credit_taking_attack_10 : (mutualCertificate10Security
   where value.messageProtectionOrder == "EncryptBeforeSign")

has type Config
but not type SafeConfig

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

<table>
<thead>
<tr>
<th>Year</th>
<th>Authors</th>
<th>Types</th>
<th>Refinement</th>
<th>Typecase</th>
<th>Subtyping</th>
</tr>
</thead>
<tbody>
<tr>
<td>1983</td>
<td>Nordström/Petersson</td>
<td>Subset types</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>1986</td>
<td>Rushby/Owre/Shankar</td>
<td>Predicate subtyping</td>
<td>no</td>
<td>no</td>
<td>limited</td>
</tr>
<tr>
<td>1989</td>
<td>Cardelli et al</td>
<td>Modula-3 Report</td>
<td>no</td>
<td>no</td>
<td>structural</td>
</tr>
<tr>
<td>1991</td>
<td>Pfenning/Freeman</td>
<td>Refinement types</td>
<td>no</td>
<td>no</td>
<td>structural</td>
</tr>
<tr>
<td>1993</td>
<td>Aiken and Wimmers</td>
<td>Type inclusion...</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>1999</td>
<td>Pfenning/Xi</td>
<td>DML</td>
<td>no</td>
<td>yes, as pattern</td>
<td>no</td>
</tr>
<tr>
<td>1999</td>
<td>Buneman/Pierce</td>
<td>Unions for SSD</td>
<td>no</td>
<td>yes, as pattern</td>
<td>structural</td>
</tr>
<tr>
<td>2000</td>
<td>Hosoya/Pierce</td>
<td>XDuce</td>
<td>no</td>
<td>no (but has cast)</td>
<td>structural, SMT</td>
</tr>
<tr>
<td>2006</td>
<td>Flanagan et al</td>
<td>SAGE</td>
<td>no</td>
<td>no</td>
<td>structural</td>
</tr>
<tr>
<td>2006</td>
<td>Fisher et al</td>
<td>PADS</td>
<td>no</td>
<td>yes, as pattern</td>
<td>semantic, ad hoc</td>
</tr>
<tr>
<td>2007</td>
<td>Frisch/Castagna</td>
<td>CDuce</td>
<td>no</td>
<td>no</td>
<td>SMT</td>
</tr>
<tr>
<td>2007</td>
<td>Sozeau</td>
<td>Russell</td>
<td>no</td>
<td>no</td>
<td>structural</td>
</tr>
<tr>
<td>2008</td>
<td>Bhargavan/Fournet/G</td>
<td>F7/RCF</td>
<td>no (but has cast)</td>
<td>no</td>
<td>structural</td>
</tr>
<tr>
<td>2008</td>
<td>Rondon/Jhala</td>
<td>Liquid Types</td>
<td>e in T</td>
<td>no</td>
<td>structural, SMT</td>
</tr>
<tr>
<td>2009</td>
<td>Bierman/G/Langworthy</td>
<td>M/MiniM</td>
<td>e in T</td>
<td>no</td>
<td>semantic, SMT</td>
</tr>
</tbody>
</table>
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
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
Our goal is a toolkit to verify reference implementations of standardized and custom cryptographic protocols.

Verification Tools for F#
- Statically verify security assertions
- Different techniques, cryptographic models

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]
# F7: Refinements for Security

Check out our site [http://research.microsoft.com/cvk](http://research.microsoft.com/cvk)

<table>
<thead>
<tr>
<th>Example</th>
<th>F# Modules</th>
<th>Lines of Code</th>
<th>F7 Typechecking Checking Time</th>
<th>FS2PV Verification Verifying Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cryptographic Patterns</td>
<td>1</td>
<td>158 lines</td>
<td>100 lines</td>
<td>17.1s</td>
</tr>
<tr>
<td>Basic Protocol (Section 2)</td>
<td>1</td>
<td>76 lines</td>
<td>141 lines</td>
<td>8s</td>
</tr>
<tr>
<td>Otway-Rees (Section 4.2)</td>
<td>1</td>
<td>265 lines</td>
<td>233 lines</td>
<td>1m.29.9s (Type Incorrect)</td>
</tr>
<tr>
<td>Otway-Rees (No MACs)</td>
<td>1</td>
<td>265 lines</td>
<td>-</td>
<td>29.64s</td>
</tr>
<tr>
<td>Secure Conversations (Section 4.3)</td>
<td>1</td>
<td>123 lines</td>
<td>111 lines</td>
<td>3.8s</td>
</tr>
<tr>
<td>Web Services Security Library</td>
<td>5</td>
<td>1702 lines</td>
<td>475</td>
<td>48.81s</td>
</tr>
<tr>
<td>X.509-based Client Auth (Section 5.1)</td>
<td>1</td>
<td>+ 88 lines</td>
<td>+ 22 lines</td>
<td>+ 10.8s</td>
</tr>
<tr>
<td>Password-X.509 Mutual Auth (Section 5.2)</td>
<td>1</td>
<td>+ 129 lines</td>
<td>+ 44 lines</td>
<td>+ 12s</td>
</tr>
<tr>
<td>X.509-based Mutual Auth</td>
<td>1</td>
<td>+ 111 lines</td>
<td>+ 53 lines</td>
<td>+ 10.9s</td>
</tr>
<tr>
<td>Windows Cardspace (Section 5.3)</td>
<td>1</td>
<td>1429 lines</td>
<td>309 lines</td>
<td>6m3s</td>
</tr>
</tbody>
</table>

Table 1. Verification Times and Comparison with ProVerif
A Good Year for Refinements

Access control, crypto protocols

OO refinements, array bounds

Systems models

Automatic inference for refinement types
Ideas to Take Away

• Remember the riddle
  – **Q:** How did the typechecker decide $\forall x. \forall y. x>y \implies x-y>0$? 
  – **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
- Oslo and its modeling language, M
  [http://msdn.microsoft.com/oslo](http://msdn.microsoft.com/oslo)
- Refinement types for security in F#
  [http://research.microsoft.com/f7](http://research.microsoft.com/f7)
- Liquid types (including online demo)
  [http://pho.ucsd.edu/liquid/](http://pho.ucsd.edu/liquid/)
- This lecture
THE END