A PCR Infrastructure for Distributed Computations in an ML-like Language

Felipe Zipitría

Grupo de Seguridad Informática - Instituto de Computación
Facultad de Ingeniería - UdelaR

RESECO 2007

Outline

1. Introduction
   - Languages
   - Motivation
2. Proof Carrying Result
3. Hands on
   - Enhancing our language
   - PCR Infrastructure

Outline

1. Introduction
   - Languages
   - Motivation
2. Proof Carrying Result
3. Hands on
   - Enhancing our language
   - PCR Infrastructure

Topics we will cover

- Safety in distributed languages
- ML-like languages, focusing on a particular language
- Serialization/Marshaling in a distributed environment

ML-like languages?

Brief comments

- OCaml
  - Well-known, lots of developers and well maintained

Warning

marshalling is currently not type-safe (from the documentation)

- GHC (Haskell) has primitive operations and types for .NET interop
- Standard ML
- Acute
  - Research language
- HashCaml
  - Ocaml based, but derived from previous work on Acute language

ML-like languages?

Brief comments

- OCaml
  - Well-known, lots of developers and well maintained

Warning

marshalling is currently not type-safe (from the documentation)

- GHC (Haskell) has primitive operations and types for .NET interop
- Standard ML
- Acute
  - Research language
- HashCaml
  - Ocaml based, but derived from previous work on Acute language

ML-like languages?

Brief comments

- OCaml
  - Well-known, lots of developers and well maintained

Warning

marshalling is currently not type-safe (from the documentation)

- GHC (Haskell) has primitive operations and types for .NET interop
- Standard ML
- Acute
  - Research language
- HashCaml
  - Ocaml based, but derived from previous work on Acute language

ML-like languages?

Brief comments

- OCaml
  - Well-known, lots of developers and well maintained

Warning

marshalling is currently not type-safe (from the documentation)

- GHC (Haskell) has primitive operations and types for .NET interop
- Standard ML
- Acute
  - Research language
- HashCaml
  - Ocaml based, but derived from previous work on Acute language

ML-like languages?

Brief comments

- OCaml
  - Well-known, lots of developers and well maintained

Warning

marshalling is currently not type-safe (from the documentation)

- GHC (Haskell) has primitive operations and types for .NET interop
- Standard ML
- Acute
  - Research language
- HashCaml
  - Ocaml based, but derived from previous work on Acute language

Introducing ACute

- Primitives for type-safe (un)marshalling

e ::= ... | marshal e1 e2 : T | unmarshal e as T | ...

- Provides safe and robust mechanisms to develop and execute separately-built programs
- Supports distributed computation of values providing (un)marshalling procedures
- Allows cooperating programs to send and receive values through (untyped) communication channels

Particular Acute features

Type-safe (un)marshalling

- Types are hashed to be used by the type checker
- Dynamic type-check at unmarshal time
- Guarantees both type-safety and abstraction-safety
- Type equality is defined simply by equality on hashes

Particular Acute features

Type-safe (un)marshalling

- Types are hashed to be used by the type checker
- Dynamic type-check at unmarshal time
- Guarantees both type-safety and abstraction-safety
- Type equality is defined simply by equality on hashes

Particular Acute features

Type-safe (un)marshalling

- Types are hashed to be used by the type checker
- Dynamic type-check at unmarshal time
- Guarantees both type-safety and abstraction-safety
- Type equality is defined simply by equality on hashes

Particular Acute features

Type-safe (un)marshalling

- Types are hashed to be used by the type checker
- Dynamic type-check at unmarshal time
- Guarantees both type-safety and abstraction-safety
- Type equality is defined simply by equality on hashes

Some language contructs

T ::= int | bool | string | unit | char | void | $T_1 * .. * T_n$ | $T_1 + .. + T_n$ |
$T \to T'$ | T list | T option | T ref | exn | $M_M.t$ | t |
$\forall t.T$ | $\exists t.T$ | T name | T tie | thread | mutex |
cvar | thunkifymode | thunkkey | thunklet | h.t | n

e ::= ... | marshal e1 e2 : T | unmarshal e as T | freshT |
cfreshT | hash(M_m.x)t | hash(T,e2)T' | hash(T, e2,e1)T' |
h.x | ... | create_thread | thunkify | ...

Respecting abstractions

Example

P3a = send (marshal(5:int))

P3b = module EvenCounter =

```
struct                  sig
  type t=int              type t
  let start=0       :     val start:t
  let get x = x           val get:t->int
  let up x = x+2          val up:t->t
end                     end

print_int (EvenCounter.get
   (unmarshal (receive ()):EvenCounter.t))
```

This computation should fail

machineA[P3a] | machineB[P3b]

Respecting abstractions

Example

P3a = send (marshal(5:int))

P3b = module EvenCounter =

```
struct                      sig
  type t=int                  type t
  let start=0           :     val start:t
  let get x = x               val get:t->int
  let up x = x+2              val up:t->t
end                         end

print_int (EvenCounter.get
   (unmarshal (receive ()):EvenCounter.t))
```

This computation should fail

machineA[P3a] | machineB[P3b]

Hashing

- A notion of type equality that makes sense is needed across the entire distributed system
- Type *EvenCounter*.t is compiled to *h*.t, where the hash *h* is (roughly)

Example

```
hash ( module EvenCounter =
        struct                  sig
          type t=int              type t
          let start=0         :   val start:t
          let get x = x           val get:t->int
          let up x = x+2          val up:t->t
        end                     end

        )
```

Outline

1. Introduction
   - Languages
   - Motivation

2. Proof Carrying Result

3. Hands on
   - Enhancing our language
   - PCR Infrastructure

Who to trust...

What happens if some attacker steals our hash ?

Problem

P3a = send ({5,hash(EvenCounter)})
P3b = (same as before)

Invariant broken !

print_int (EvenCounter.get (unmarshal (receive ()):EvenCounter.t))

Who to trust...

What happens if some attacker steals our hash ?

Problem

P3a = send ({5,hash(EvenCounter)})
P3b = (same as before)

Invariant broken !

print_int (EvenCounter.get (unmarshal (receive ()):EvenCounter.t))

Proof Carrying Result
General Scheme

- Distributed computation among untrusted hosts

Example

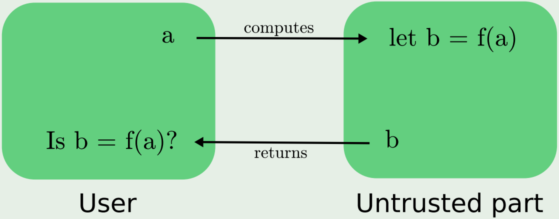

Figure: Result checking scheme

Fundamental approach

- Based on verification
- Transparent for end users
- General, flexible and configurable
- *Resource-aware*

Fundamental approach

- Based on verification
- Transparent for end users
- General, flexible and configurable
- *Resource-aware*

Fundamental approach

- Based on verification
- Transparent for end users
- General, flexible and configurable
- *Resource-aware*

Fundamental approach

- Based on verification
- Transparent for end users
- General, flexible and configurable
- *Resource-aware*

More Formally

Some Definitions

$f \in A \rightarrow B, a \in A$

$f(a)$ is delegated to an untrusted party

User must have a function

$check_f \in AxB \rightarrow bool \mid \forall (a,b) \in AxB, check_f(a,b) = true \Rightarrow b = f(a)$

More Formally

Some Definitions

$f \in A \rightarrow B, a \in A$

$f(a)$ is delegated to an untrusted party

User must have a function

$check_f \in AxB \rightarrow bool \mid \forall(a, b) \in AxB, check_f(a, b) = true \Rightarrow b = f(a)$

Certification Algorithms

- Fundamental part of PCR
- Integration with the infrastructure
- Must be a general solution

Outline

1. Introduction
   - Languages
   - Motivation

2. Proof Carrying Result

3. Hands on
   - Enhancing our language
   - PCR Infrastructure

Adding functionality to ACute

Objective

To allow verifying correctness of marshalled values

- Acute language has been extended
  - Parser, Lexer, AST, etc.
- Main problems
  - lack of coding standards (sorry, but true :)
  - a little bit rusty with my functional background

Adding functionality to ACute

Objective

To allow verifying correctness of marshalled values

- Acute language has been extended
  - Parser, Lexer, AST, etc.
- Main problems
  - lack of coding standards (sorry, but true :)
  - a little bit rusty with my functional background

Adding functionality to ACute

Objective

To allow verifying correctness of marshalled values

- Acute language has been extended
  - Parser, Lexer, AST, etc.
- Main problems
  - lack of coding standards (sorry, but true :)
  - a little bit rusty with my functional background

New semantics

Particular expression

Old

$marshal\, e_1\, e_2 : T$

New

$marshal\, e_1\, e_2 : T < certificate >$

Outline

Using Proof Carrying Results

- Defined infrastructure
  - Independent oracles
- Certifying algorithms
  - Which algoritm could be better in this case?

Which certificate applies to this case?

- We don't have too much information in this case, for an abstract type
- For any value of this type, we only know *how* it was constructed
- … or, we could use additional information

Using construction of values

Module example

module EvenCounter
: sig
type t
val start:t (projection 1 *)*
val get:t->int (projection 2 *)*
val up:t->t (projection 3 *)*
end

- So, in this case, our certificate will be

$$C_4 = [(1,1);(3,2)]$$

More generally

$$C_m = [(c_i, n_i, [lparam_i]);(c_{i+1}, n_{i+1}, [lparam_{i+1}]);\ldots] \quad (1)$$

Using additional information

- We can add a new way of defining our invariants

New expression

invariant P

$\rightarrow$where the property P will be used as a certificate

For EvenCounter case

invariant (forall x:t, x mod 2 = 0)

Summary

- Acute language is being enhanced with certificates for using proof carrying results
- Definition of certificates for this case is a good starting point
- But finding certificates for general use could be a difficult task to achieve

Summary

- Acute language is being enhanced with certificates for using proof carrying results
- Definition of certificates for this case is a good starting point
- But finding certificates for general use could be a difficult task to achieve

Summary

- Acute language is being enhanced with certificates for using proof carrying results
- Definition of certificates for this case is a good starting point
- But finding certificates for general use could be a difficult task to achieve

For Further Reading I

P. Sewell et al.
Acute: High-Level Programming Language Design for Distributed Computation.
University of Cambridge TR 605, 2005.

Various people
Notes on Proof Carrying Results.
August 29, 2006.

J.Leifer et al.
Global Abstraction-Safe Marshalling with Hash Types.
ICPF'03, August 25-29, 2003.