



POTSDAM INSTITUTE FOR CLIMATE IMPACT RESEARCH **PIK**

MEMBER OF THE LEIBNIZ ASSOCIATION

Generic Libraries in C++ with Concepts from High-Level Domain Descriptions in Haskell

A Domain-Specific Library for Vulnerability Assessment

Daniel Lincke, Patrik Jansson, Marcin Zalewski and Cezar Ionescu

July 15th, 2009



Leibniz
Gemeinschaft



Outline

Domains in scientific programs

Vulnerability modelling in Haskell

Domain encoding in C++

Outlook

From Haskell to C++

Haskell notion	C++ concept
<code>Arr :: a -> b</code> general function type	<code>Arrow<class Arr></code>
<code>T :: f x</code> the type <code>T</code> is a constructor application	<code>ConstructedType<class T></code>
<code>T1 :: f x</code> , <code>T2 :: f y</code> the types <code>T1</code> and <code>T2</code> are constructed by the same type constructor	<code>SameTypeConstr<class T1, class T2></code>
<code>T1 :: x</code> , <code>T2 :: x</code> the types <code>T1</code> and <code>T2</code> are the same type	<code>SameType<class T1, class T2></code>
<code>Arr :: a -> b</code> , <code>Type :: f a</code> the type <code>Type</code> is constructed by a functor, <code>Arr</code> is a function which can be fmapped to <code>Type</code>	<code>FMappable<class Arr, class Type></code>

The problem in scientific computing

Problem from a scientific domain



hacking

Computer program to solve this problem

- ▶ Big gap between the problem and the program
- ▶ Domain scientists: knowledge in form of domain specific notions
- ▶ Computer scientists: not much domain knowledge
- ▶ Implementations often bear little apparent resemblance to the domain
- ▶ Practitioners accept low-level implementations to guarantee performance

What to do?

Problem from a scientific domain



specification

Formal description of the problem



implementation

Computer program to solve the problem

- ▶ Include an intermediate layer between the problem and the program
- ▶ Should serve as a common base of communication
- ▶ Should therefore be easy to understand for both sides
- ▶ Domain specific notions should somehow be visible in both the specification language and the programming language

Our approach

- ▶ We start from a specification of a scientific domain
- ▶ This specification is done in Haskell
- ▶ We provide a domain specific generic C++ library which allows structured use of existing, efficient code
- ▶ The library is derived from the Haskell specification
- ▶ We use *generic programming* to increase the level of abstraction in the C++ code, without losing efficiency
- ▶ Our application: the problem of vulnerability assessment
- ▶ The specification of vulnerability computations was done by Ionescu [2008]

Our approach

- ▶ Our technique preserves the high-level specification notions from the domain in the implementation library
- ▶ High-level specification notions are expressed as concepts
- ▶ Efficient implementations can be easily plugged in the specification framework
- ▶ After plugging them in, the library allows to combine implementation components

Research questions

- ▶ Starting from a given Haskell specification of vulnerability assessment, how are domain notions represented there?
- ▶ How can we encode them in C++? What framework do we have to provide to express them?
- ▶ Can we find some common pattern in our translation which makes the approach applicable to other domains?

Specification vs. Implementation

- ▶ Specification can be done in different ways: informal mathematics or a specification language
- ▶ Ionescu's approach: the functional programming language Haskell
 - ▶ clean notation
 - ▶ expressive type system
 - ▶ first-class support for generic programming
 - ▶ definitions are operational (John Hughes: “animated mathematics”)
- ▶ But why don't we go with our “animated mathematics?”
- ▶ Because finally we want to do software engineering, not “animated mathematics”
- ▶ Implementation language is C++, amongst others because of legacy code

So, what is vulnerability?

- ▶ The concept “vulnerability” is important in sustainability science
- ▶ Vulnerability in ordinary language: exposed to the possibility of being attacked or harmed: *small fish are vulnerable to predators* (Soanes and Stevenson [2005])
- ▶ Vulnerability in the context of climate change: *Bangladesh is vulnerable to sea level rise*
- ▶ Vulnerability assessments: (computational) studies of vulnerability in socio-ecologic systems
- ▶ Many different, partly ambiguous, definitions (Thywissen [2006] lists 35 different definitions!)
- ▶ There is, among the definitions, a certain similarity
- ▶ This similarity is made explicit by the formal specification of Ionescu [2008]: Vulnerability is about **possible future harm**

A vulnerability model in Haskell

- ▶ Vulnerability is about a **possible future** of a system, usually with a finite horizon

```
model :: (Functor f) => state -> f state
```

```
possible :: (Functor f) => state -> f evolution
```

- ▶ f determines the type of the model
- ▶ **harm** is a measure on evolutions

```
harm :: evolution -> v
```

- ▶ A **measure** is used to collapse harm values

```
measure :: (Functor f) => f v -> w
```

A vulnerability model in Haskell

- ▶ Then vulnerability is possible harm:

```
vulnerability :: state -> w
vulnerability = measure . fmap harm . possible
```

- ▶ A more flexible version takes it parts (possible, harm and measure) as higher-order inputs

```
vulnerability :: (Functor f)
=> (f v -> w) -> (evolution -> v)
-> (state -> f evolution) -> state -> w
vulnerability m h poss = m . fmap h . poss
```

Domain notions in Haskell model

- ▶ Domain notions are expressed by Haskell's expressive type system
- ▶ Polymorphic function signatures express notions like `possible`
- ▶ Even if function signatures do not explicitly contain domain notions, they could be made explicit by introducing new types

C++ Implementation

How to express generic function signatures (our domain notions) in C++? There are some important differences between Haskell and C++ which make direct translation difficult:

- ▶ Haskell offers a built in type for functions, C++ does not
- ▶ Type constructors (parametrised types) are well supported in Haskell, in C++ there is no such first class support

However, we want to use generic function signatures to provide domain notions so we have to:

- ▶ Use templates for genericity and concepts to constrain template parameters
- ▶ Implement a conceptual framework specifying a system of “types of types”
- ▶ This system provides, more or less explicit, domain notions

The C++ Library

Our library consist of several layers:

- ▶ concepts: specifying the type system of our library
- ▶ constrained template functions: realize algorithmic computations
- ▶ combinators (constrained template classes): combine components into new ones

The library is not that complex in terms of numbers:

- ▶ about 20 concepts
- ▶ about 15 constrained template functions
- ▶ about 15 combinators classes

Some concepts we provide

Haskell notion	C++ concept
<code>Arr :: a -> b</code> general function type	<code>Arrow<class Arr></code>
<code>T :: f x</code> the type <code>T</code> is a constructor application	<code>ConstructedType<class T></code>
<code>T1 :: f x</code> , <code>T2 :: f y</code> the types <code>T1</code> and <code>T2</code> are constructed by the same type constructor	<code>SameTypeConstr<class T1, class T2></code>
<code>T1 :: x</code> , <code>T2 :: x</code> the types <code>T1</code> and <code>T2</code> are the same type	<code>SameType<class T1, class T2></code>
<code>Arr :: a -> b</code> , <code>Type :: f a</code> the type <code>Type</code> is constructed by a functor, <code>Arr</code> is a function which can be mapped to <code>Type</code>	<code>FMappable<class Arr, class Type></code>

Basic concepts: Arrow

- ▶ Different ways of encoding function types in C++ (Lincke and Schupp [2009])

- ▶ We encoded them by the Arrow concept:

```
// F is of type a -> b
concept Arrow<class F> {
    typename Domain;
    typename Codomain;

    Codomain operator () (F, Domain);
};
```

- ▶ A concept specifies an interface including associated types, member functions and operators
- ▶ Concepts are closely related to Haskell type classes (Bernardy et al. [2008])

Concept framework: SameType

- ▶ In Haskell, type identities are explicit in signatures

```
compose :: (b -> c) -> (a -> b) -> a -> c
```

- ▶ In our approach we have to require them explicitly, as member types are not visible in template parameters

```
template<class G, class F>  
requires  
    Arrow<G>, Arrow<F>,  
    SameType<G::Domain,F::Codomain>  
class Compose{ ... };
```

- ▶ The same holds for type constructors and type constructor identities

Functorial types

```
concept FMappable<Arrow Arr, ConstructedType Type> {  
    // Arr = A -> B; Type = F(X)  
  
    ConstructedType ReturnType;  
    requires  
    CopyConstructible<ReturnType>,  
    SameTypeConstructor<Type,ReturnType>,  
    SameType<Type::Inner, Arr::Domain>,  
    SameType<ReturnType::Inner, Arr::Codomain>;  
    // ReturnType = G(Y); F = G; X = A; Y = B  
  
    ReturnType FMap(Arr const&, Type const&);  
};
```

Generic Vulnerability

```
template<class Poss, class Harm, class Meas>
requires
  Arrow<Poss>, Arrow<Harm>, Arrow<Meas>,
  FMappable<Harm, Poss::Codomain>,
  ConstructedType<Meas::Domain>,
  SameType<Meas::Domain::Inner, Harm::Codomain>,
  SameTypeConstructor<Poss::Codomain, Meas::Domain>
Meas::Codomain
vulnerability (Poss poss, Harm harm, Meas meas,
              Poss::Domain state) {
  return meas(FMap(harm, poss(state)));
};
```

Vulnerability DSL

- ▶ What makes the library described an embedded domain specific language?
- ▶ Library can be used by programmers who do not have deep domain knowledge
- ▶ We reduce domain notions to type signatures, which should be understandable by every programmer
- ▶ The programmer just provides objects of appropriate types to “plug them in”
- ▶ We bring the programmer closer to the domain

Example

```
struct Evolutions_computation {
    typedef Climate_state  Domain;
    typedef Id<Evolution>  Codomain;

    Codomain operator(Domain c_state) {
        // very complex implementation, legacy code, ...
    }
};
```

- Implementations are plugged in with concept maps

```
concept_map Arrow<Evolutions_computation> {
    ...
}
```

Example

- ▶ Assuming that we have the following function objects

```
Evolutions_computation possible;  
Harm_computation      harm;  
Id_Measure<float>     measure;
```

- ▶ We define an initial climate state

```
Climate_state initial_climate;  
// ... set initial climate
```

- ▶ Finally we plug the parts into the vulnerability computation

```
float result =  
vulnerability(possible, harm, measure, initial);
```

Work already done or in progress

- ▶ The specification of Ionescu [2008] provides more; we have also implemented in our library
 - ▶ **Monadic Dynamical Systems** as an advanced way of generic modelling future evolutions of systems, which for instance allow to model external input to the system
 - ▶ Specification of other notions related to vulnerability: resilience, sensitivity and adaptive capacity
- ▶ Informal description of algorithms for partly automatized translation of Haskell declarations in C++ templates (constrained with concepts)
- ▶ Therefore our approach is in general applicable to other domains

Thanks!

Thank you for your attention.

For further information we refer to the paper (Lincke et al. [2009]),
to the literature and to `www.pik-potsdam.de/favaia`

References

- C. Ionescu. *Vulnerability Modelling and Monadic Dynamical Systems*. PhD thesis, Freie Universität Berlin, 2008.
- C. Soanes and A. Stevenson. *Oxford Dictionary of English (Dictionary)*. Oxford University Press, August 2005.
- K. Thywissen. *Core terminology of disaster reduction a comparative glossary*. UNU Press, Tokyo, 2006.
- D. Lincke and S. Schupp. The Function Concept in C++ - An Empirical Study. In *WGP '09: Proceedings of the ACM SIGPLAN workshop on Generic programming*, New York, NY, USA, 2009. ACM.
- J. Bernardy, P. Jansson, M. Zalewski, S. Schupp, and A. Priesnitz. A Comparison of C++ Concepts and Haskell Type Classes. In *WGP '08: Proceedings of the ACM SIGPLAN workshop on Generic programming*, pages 37–48, New York, NY, USA, 2008. ACM.
- D. Lincke, P. Jansson, M. Zalewski, and C. Ionescu. Generic Libraries in C++ with Concepts from High-Level Domain Descriptions in Haskell. A Domain-Specific Library for Computational Vulnerability Assessment. In *Proceedings of the IFIP Working Conference on Domain Specific Languages*, 2009.