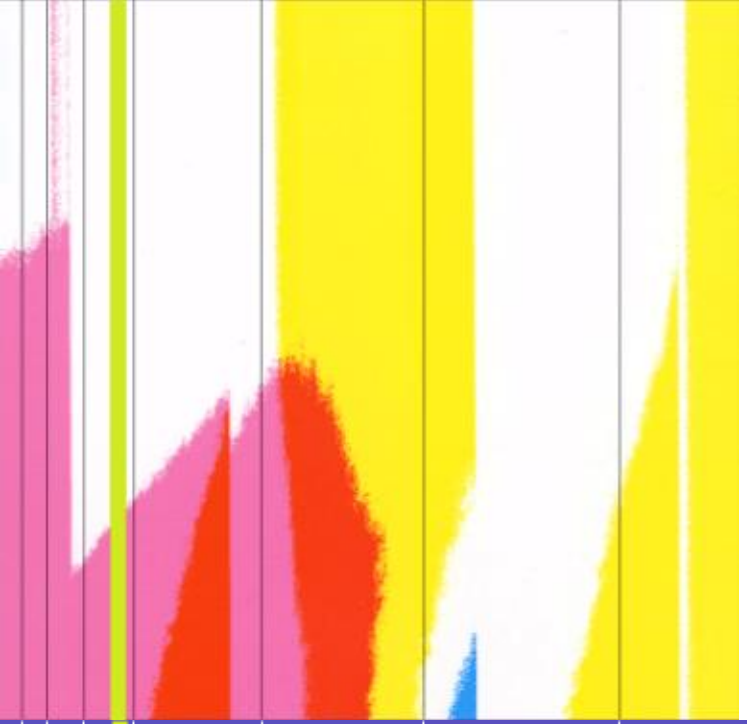


Implementing Program Transformations with Tom and Java

Pierre-Etienne Moreau

GTTSE, July 2, 2007

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

Context

- Problem:
  - plus(x,0) = x
  - plus(x, suc(y)) = suc(plus(x,y))
  - plus(suc(suc(0)), suc(0)) =? Plus(suc(0), suc(suc(0)))
- In 1975, Michael J. O'Donnell and Joseph A. Goguen introduced the notion of Equational Programming
  - plus(x,0) $\rightarrow$ x
  - plus(x, suc(y)) $\rightarrow$ suc(plus(x,y))
- Since, the notion of Term Rewriting has been studied
  - Many interesting theoretical properties (termination, confluence)
  - Many practical properties (high level, executable, efficient)
  - Many implementations (OBJ, ASF+SDF, ELAN, Maude, Stratego, Tom)

In this technology presentation

- Tom
- Extension of Java
- Based on Term Rewriting
- Hybrid Approach
  - Transformations are easy to describe
  - They are integrated in a Java environment
  - The code is more maintainable
    - Transformations are expressed in a purely functional style
    - Glue and imperative parts are expressed in Java
- Application to program transformation and compilation

Short presentation of Tom

- A Java program is a Tom program

```
import pil.term.types.*;
import java.util.*;
public class Pil {
  …
  public final static void main(String[] args) {
    Expr p = …;
    System.out.println("p = " + p);
    …
  }
}
```

Tom adds algebraic data-types to Java

- Gom supports many-sorted first order signature

```
import pil.term.types.*;
import java.util.*;
public class Pil {
  …
  public final static void main(String[] args) {
    Expr p = …;
    System.out.println("p = " + p);
    …
  }
}
```

```
%gom {
  module Term
    imports int String
    abstract syntax
    Expr =
         | Var(name:String)
         | Let(var:Expr, e:Expr, body:Expr)
         | a()
         | …
    Bool =
         | True()
         | False()
         | Eq(e1:Expr, e2:Expr)
}
```

Think to an AST,
a grammar,
a meta-model

An algebraic term is a Java object

- Back-quote (`) to build a term

```
import pil.term.types.*;
import java.util.*;
public class Pil {
  …
  public final static void main(String[] args) {
    Expr p =  `Let(Var("x"),a(),Var("x"));
    System.out.println("p = " + p);
    …
  }
}
```

```
%gom {
  module Term
    imports int String
    abstract syntax
    Expr =
         | Var(name:String)
         | Let(var:Expr, e:Expr, body:Expr)
         | a()
         | …
    Bool =
         | True()
         | False()
         | Eq(e1:Expr, e2:Expr)
}
```

Tom adds pattern matching to Java

- %match supports syntactic and associative pattern matching

```
import pil.term.types.*;
import java.util.*;
public class Pil {
  …
  public final static void main(String[] args) {
    Expr p = …;
    System.out.println("p = " + p);
    …(pretty(p));
  }
  …
}
```

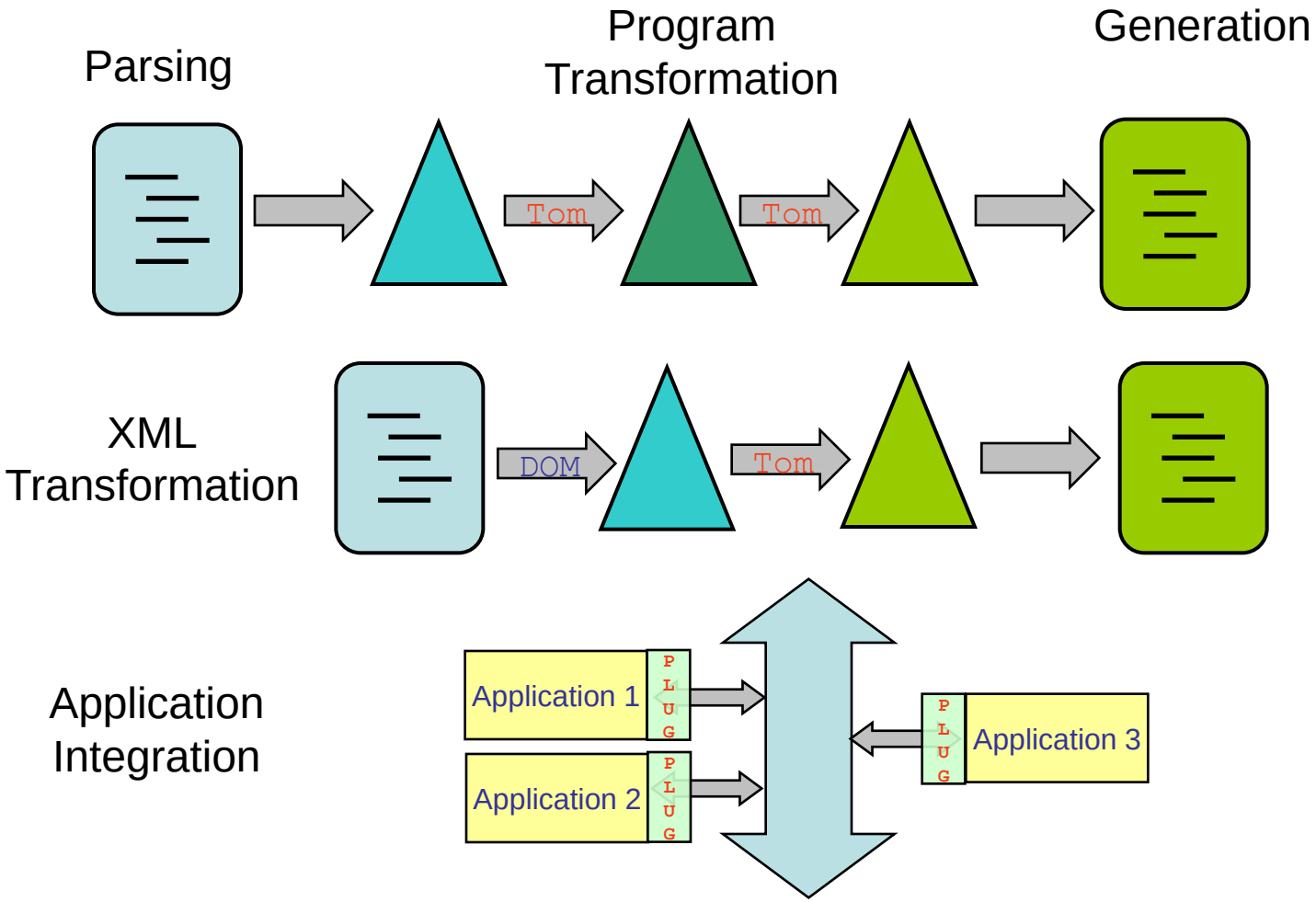
```
public static String pretty(Object o) {
    %match(o) {
      Var(name) -> { return `name; }

      Let(var,expr,body) -> {
        return "let " + pretty(`var) +
               " <- " + pretty(`expr) +
               " in " + pretty(`body);
      }

      …

    }
    return o.toString();
  }
```

It can be used for



Bytecode analysis and transformation

What you will see

- An interactive demo of Tom
- Introduction to the notion of rules and strategies
- How they are integrated into Java
- Application to program transformation and compilation

Tom is available at http://tom.loria.fr/