Software transformation and generation

Paulo Borba
Informatics Center
Federal University of Pernambuco

phmb@cin.ufpe.br ◈ twitter.com/pauloborba

Language definition, terms as strategies

Paulo Borba
Informatics Center
Federal University of Pernambuco

phmb@cin.ufpe.br ◈ twitter.com/pauloborba

Choice versus disjunction

```
where (<eq>(e1,|[true]|) ; <eq>(e2,|[false]|)) +
      (<eq>(e1,|[false]|) ; <eq>(e2,|[true]|))
```

```
where or(and(<eq>(e1,|[true]|),<eq>(e2,|[false]|)),
         and(<eq>(e1,|[false]|),<eq>(e2,|[true]|)))
```

Predefined strategy operators so far

- `id`
- `?(_)`
- `not(_)`
- `topdown(_)`
- `try(_)`
- `iojava2java-wrap(_)`
- `_;_`
- `_+_`
- `and(_,_)`
- `or(_,_)`
- `eq`
- `_=>_`

Arguments to strategy operators are functions from terms to terms

- strategies
- rules
- terms!

Terms as functions

```
|[ balance = 0; ]|
```

```
f(x) = |[ balance = 0; ]|
```

```
|[ balance = 0; ]|(x) = |[ balance = 0; ]|
```

Operator receiving strategy or term...

```
  |[ if (e) stm ]| -> |[ if (e) {stm} ]|
where
  <not(block)> stm
```

```
  |[ if (e) stm ]| -> |[ if (e) {stm} ]|
where
  not(<block> stm)
```

```
  |[ if (e) stm ]| -> |[ if (e) {stm} ]|
where
  not(<<block> stm> stm)
```

unreadable version!

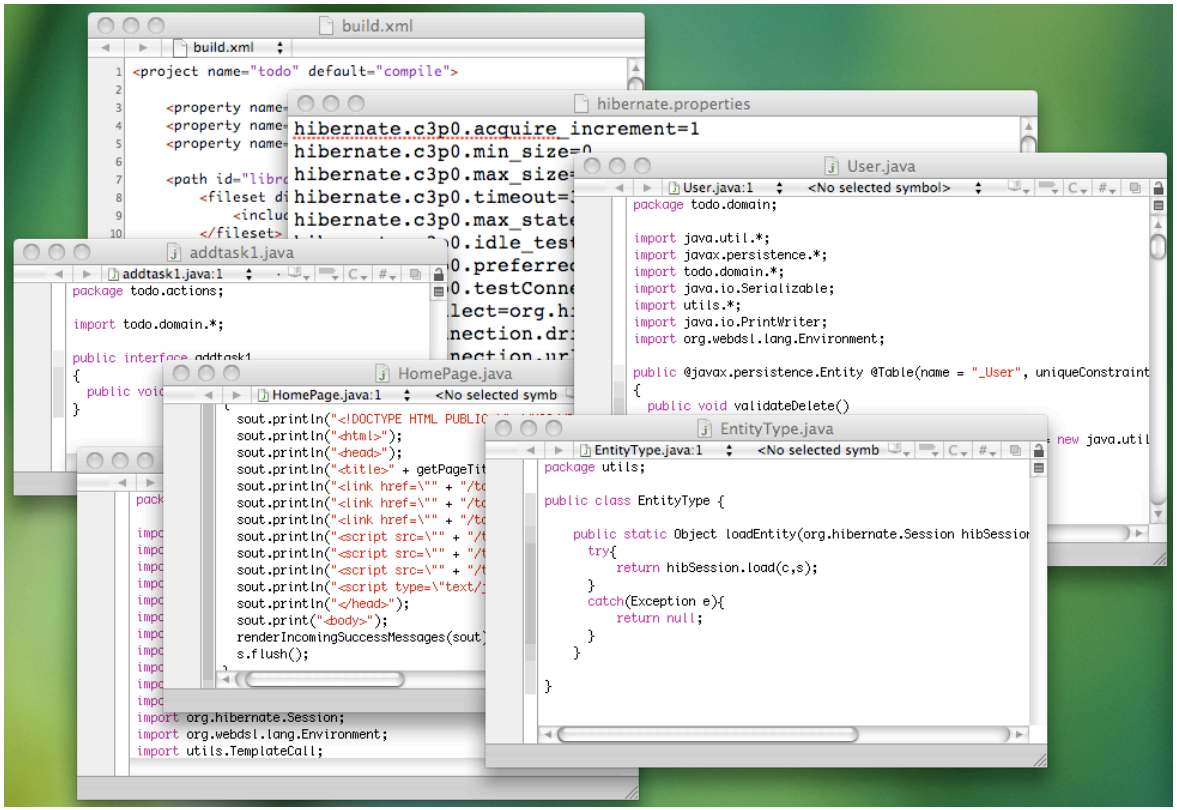
```
  |[ if (e) stm ]|-> |[ if (e) {stm} ]|
where
  < not(?(stm |[ {bstm*} ]|)) > stm
```

```
  |[ if (e) stm ]|  -> |[ if (e) {stm} ]|
where
  not(<?(stm |[ {bstm*} ]|)> stm)
```

```
  |[ if (e) stm ]|  -> |[ if (e) {stm} ]|
where
  not(<<?(stm |[ {bstm*} ]|)> stm> stm)
```

That's all nice for transforming Java programs...

but can we generate Java programs?



What's the syntax of the source language?

```
entity Person {
  fullname : String
  email : String
  homepage : String
}
```

```
entity Publication {
  title : String
  author : Person
  year : Int
  abstract : String
  pdf : String
}
```

Full example from Eelco Visser, *Domain-Specific Language Engineering*, GTTSE 2007.

Defining the syntax of the source language, sentences...

```
context-free syntax
  Definition* -> Model
  Entity -> Definition
  "entity" Id "{" Property* "}" -> Entity
  Id ":" Sort -> Property
  Id -> Sort
```

with constructors

```
context-free syntax
  Definition* -> Model {cons("Model")}
  Entity -> Definition
  "entity" Id "{" Property* "}"
     -> Entity {cons("Entity")}
  Id ":" Sort -> Property {cons("Property")}
  Id -> Sort {cons("SimpleSort")}
```

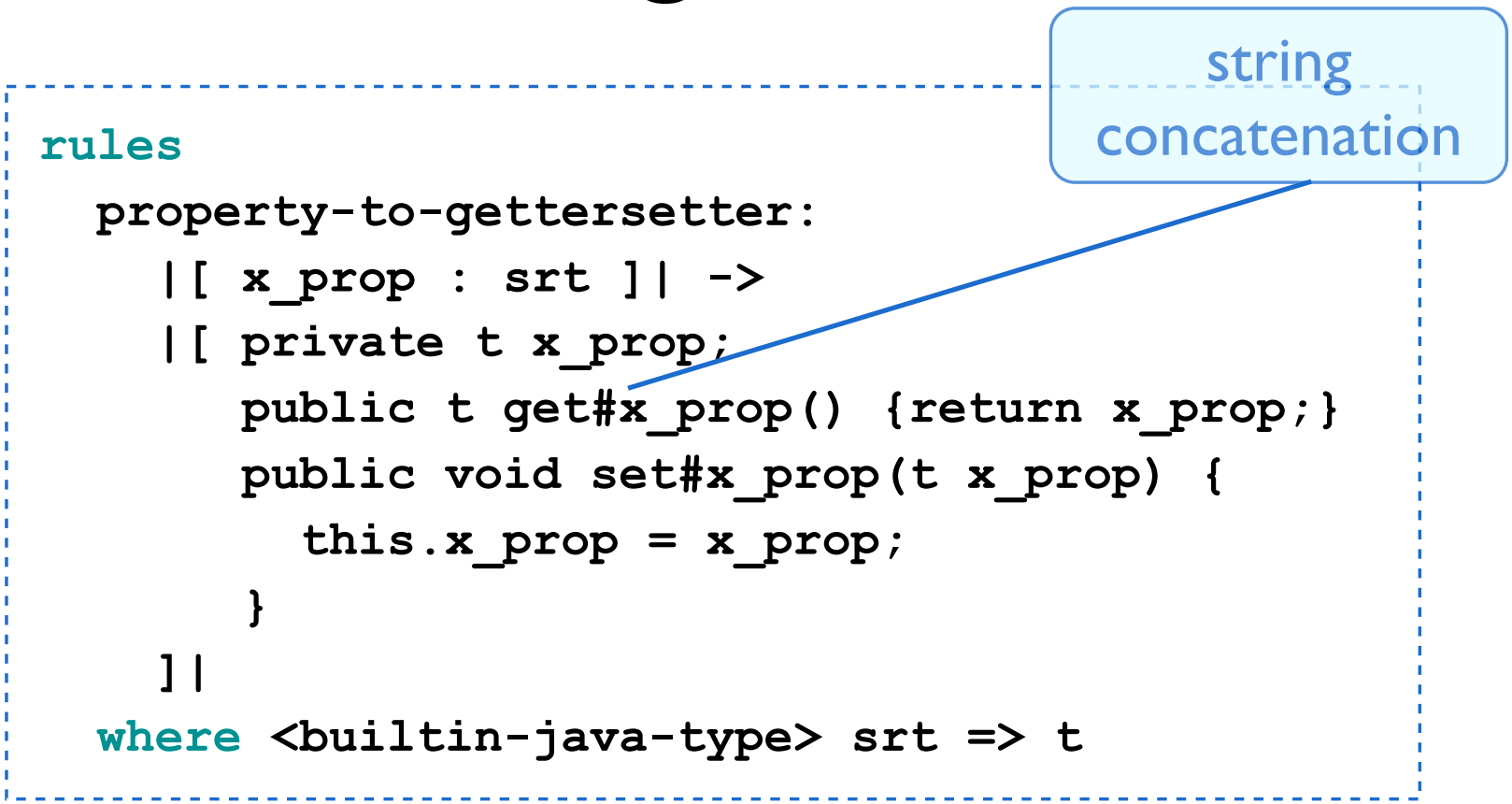
Defining the syntax of the source language, words

```
lexical syntax
   [a-zA-Z][a-zA-Z0-9\_]* -> Id
   [0-9]+ -> Int
   "\"" ~[\"\n]* "\"" -> String
   [\ \t\n\r] -> LAYOUT
   "//" ~[\n\r]* [\n\r] -> LAYOUT
```

Putting it all together...

```
module DataModel
exports
  sorts Id Int Entity Property Sort...
context-free syntax
  "entity" Id "{" Property* "}"
     -> Entity {cons("Entity")}
  Id ":" Sort -> Property {cons("Property")}
  ...
lexical syntax
  [a-zA-Z][a-zA-Z0-9\_]* -> Id
  ...
```

with rules and strategies too!



no # operator

```
rules
  property-to-gettersetter:
    |[ x_prop : srt ]| ->
    |[ private t x_prop;
       public t x1() {return x_prop;}
       public void set#x_prop(t x_prop) {
         this.x_prop = x_prop;
       }
    ]|
  where <builtin-java-type> srt => t ;
        concatString(|"get") x_prop => x1
```

Which variables to use?

```
variables
    "ent" [0-9]*       -> Entity        {prefer}
    "prop"[0-9]*       -> Property      {prefer}
    "prop"[0-9]* "*"   -> Property*     {prefer}
    "srt" [0-9]*       -> Sort          {prefer}
    [xyz][0-9]*           -> Id         {prefer}
    [xyz]"_"[A-Za-z0-9]+  -> Id         {prefer}
```

Part of quotation syntax, meta-syntax

```
context-free syntax
  "webdsl" "|[" Entity "]|" -> E
                                 {cons("ToMetaExpr")}
           "|[" Entity "]|" -> E
                                 {cons("ToMetaExpr")}
```

Syntax definition language...

- Literals: **`"while"`**
- Zero or more : **`Stm*`**
- One or more: **`TypeDec+`**
- Zero or more, with separator: **`{Exp ","}*`**
- One or more, with separator: **`{Id "."}+`**
- Optional: **`Expr?`**
- Alternative: **`{Expr ","}* | LocalVarDec`**
- Sequence: **`(A0...An)`**

Lexical

- Digits: `[0-9]`
- Hexa: `[0-9a-fA-F]`
- Whitespace: `[\ \t\12\r\n]`
- Strings: `~[\"\\\n\r]`

Reject and follow restriction

```
lexical syntax

   "goto" -> Id {reject}

lexical restrictions

   Id -/- [A-Za-z0-9]

   "goto" -/- [A-Za-z0-9]

context-free syntax

   "goto" Id -> Stm {cons("Goto")}
```

Multi-line comments

```
lexical syntax
  BlockComment -> LAYOUT
  "/*" CommentPart* "*/" -> BlockComment
  ~[\/\*]      -> CommentPart
  Asterisk     -> CommentPart
  Slash        -> CommentPart
  BlockComment -> CommentPart
  [\/] -> Slash
  [\*] -> Asterisk

lexical restrictions
  Asterisk -/- [\/]
  Slash    -/- [\*]
```

Start symbol and brackets

```
exports
  context-free start-symbols Exp
  context-free syntax
    Id              -> Exp {cons("Var")}
    IntConst        -> Exp {cons("Int")}
    "(" Exp ")"     -> Exp {bracket}
```

Priorities

```
context-free syntax
  Exp "*" Exp -> Exp {left, cons("Times")}
  Exp "/" Exp -> Exp {left, cons("Div")}
  Exp "+" Exp -> Exp {left, cons("Plus")}
  Exp "-" Exp -> Exp {left, cons("Minus")}
  Exp ">" Exp -> Exp {cons("Gt"), non-assoc}
context-free priorities
  {left: Exp "*" Exp -> Exp
         Exp "/" Exp -> Exp
  }
> {left: Exp "+" Exp -> Exp
         Exp "-" Exp -> Exp
  }
> {non-assoc:
         Exp ">" Exp -> Exp
  }
```

Language definition, terms as strategies

Paulo Borba
Informatics Center
Federal University of Pernambuco

phmb@cin.ufpe.br ◈ twitter.com/pauloborba

Software transformation and generation

Paulo Borba
Informatics Center
Federal University of Pernambuco

phmb@cin.ufpe.br ◈ twitter.com/pauloborba