Term Annotation in Stratego

Martin Bravenboer, Eelco Visser

mbravenb@cs.uu.nl

Institute of Information and Computing Sciences
University Utrecht
The Netherlands
Introduction
What is an annotation?

*term attached to a term without being part of the structure*

Rules and strategies can be applied to a structure without knowing about the annotation.

general usage: store terms that

- don’t fit into a signature
- you don’t want in a signature
Example usage (1)

maintaining semantic information on terms during a transformation

- type (for example in type-checker)
- scope of variables and functions
- escape information of variables
- mode of a tile in instruction selection
Example usage (2)

cooperation with other datatypes
- XML (limited attributes)
- transformation to XHTML

data-oriented applications:
- data not being part of the structure
Syntax and basic strategies
Annotations in match and build

match:

?BinOp(op, e1{Int}, e2{Int}) ?e1{Int} ?__{__}

build:

!BinOp(PLUS, e1, e2){Int}
Annotations in rules

rule example:

\[
\text{TcExp:} \\
\quad \text{BinOp} (op, \text{Typed}(e1, \text{Int}), \text{Typed}(e2, \text{Int})) \rightarrow \\
\quad \text{Typed} \left( \text{BinOp}(op, e1, e2), \text{Int} \right)
\]

before annotations:

\[
\text{TcExp:} \\
\quad \text{BinOp} (op, \text{Typed}(e1, \text{Int}), \text{Typed}(e2, \text{Int})) \rightarrow \\
\quad \text{Typed} (\text{BinOp}(op, e1, e2), \text{Int})
\]
What is an annotation?

options:

- Term has a list of annotations (ATerm).
- Term has one annotation, which is one term, which might be a list (Stratego).

list-approach:

- list-matching

  \[ \text{!Var("a")\{[Int, Float]\} should become } \text{!Var("a")\{Int, Float\}} \]
Basic strategies for annotation

Annotations module provides some basic strategies:

get-annotations = ?t; prim(...)
set-annotations = ?(t, a); prim(...)
rm-annotations = ?t; prim(...)
has-annos = ?_=__
strip-annos = bottomup(rm-annotations)
How do annotations fit into existing Stratego constructs?
Annotation construction in overlays

An annotation can be attached in an overlay.

overlays

\[ \text{IntBinOp}(\text{op}, \ x, \ y) = \text{BinOp}(\text{op}, \ x, \ y)\{\text{Int}\} \]
Annotations are preserved on the application of a congruence.

\[
<\text{Call}(\text{Var}(\text{is-string}), \text{list(exp)})> \\
\quad \text{Call}(\text{Var}("f"), [])\{\text{Scope}(\text{Var}("g"))\} \\
=> \text{Call}(\text{Var}("f"), [])\{\text{Scope}(\text{Var}("g"))\}
\]

Congruences can apply strategies to annotations.

\[
\text{Call}(\text{Var}(\text{is-string}), \text{list(exp)}) \\
\quad \{\text{Scope}(\text{Var}(\text{is-string}))\}
\]
Deconstruction with annotation

pattern:

?p1#(p2){anno}

deconstruct:

   p1#(p2){anno}  ->  (p1, p2, anno)

<deconstruct>  Plus(e1, e2){Int}

=>  ("Plus", [e1, e2], Int)
all, one, some preserve annotations

test28 =
  apply-test(!"test28"
  , all(id); get-annotations
  , !Var("a"){Int}
  , Int
  )

  → simple-traversals preserve annotations
Don’t loose your annotation
Transparency of annotation

Annotation is not part of the structure of a term.

\[ !\text{Plus}(e_1, e_2)\{\text{Int}\} \Rightarrow \text{Plus}(e_1, e_2) \]

Desugar: \( \text{Plus}(e_1, e_2) \rightarrow \text{BinOp}(\text{PLUS}, e_1, e_2) \)

\[ <\text{Desugar}> \ \text{Plus}(e_1, e_2)\{\text{Int}\} \]
\[ \Rightarrow \text{BinOp}(\text{PLUS}, e_1, e_2) \]
Problem

Annotations are *not* preserved over the application of a classic rule.

Desugar:

\[
\text{Plus}(e_1, e_2) \rightarrow \text{BinOp}(\text{PLUS}, e_1, e_2)
\]

application on term with annotation:

\[
\langle \text{Desugar} \rangle \quad \text{Plus}(e_1, e_2)\{\text{Int}\}
\rightarrow \text{BinOp}(\text{PLUS}, e_1, e_2)\{\}
\]
Preserving annotations

preserve-anno(s)

preserves the annotation of the current term over the application of a strategy

<preserve-anno(Desugar)>

Plus(e1, e2){Int} => BinOp(PLUS, e1, e2){Int}

future: attributes for rules and strategies?
More annotations
Properties

annoproperties: [(key, value)].

currently nospecialsyntax, just strategies

- has-prop(k)
- has-prop(k, c)
- get-prop(k)

- apply-prop(k, s)

- replace-prop(k, v)
- add-prop(k, v)
- set-prop(k, v)
More and more annotations

If annotations are passed between applications, namespaces for properties can be useful.

signature

constructors

    Tiger: Namespace       Type: Property

overlays

    TigerType = (Tiger, Type)

strategies

    get-type    = get-prop(!TigerType)
    set-type(t) = set-prop(!TigerType, t)
Discussion and status
Drawbacks

- Annotations easily get lost.
Drawbacks

- Annotations easily get lost.

```haskell
main = ! (1, 2) \{ Int \}; Swap; ? (2, 1) \{ Int \}
```

→ Although annotations are transparent, some generic strategies must handle annotations:
  - standard library
  - build-in primitives like all, one
Drawbacks

- Annotations easily get lost.
- Annotations with semantic information must be kept up-to-date.
- danger of variants
Current implementation problems

Implementation must be considered alpha.

- list versus single term
- Anno in overlay is not allowed.
- Anno gets lost in congruence $\text{Term}(\ldots)$.
- $?\{\_\}$ matches term without anno (has-annos).
- $?\{\text{Term}\}$ results in seg-fault on term without anno.
- most likely a lot of library problems
Questions or remarks?