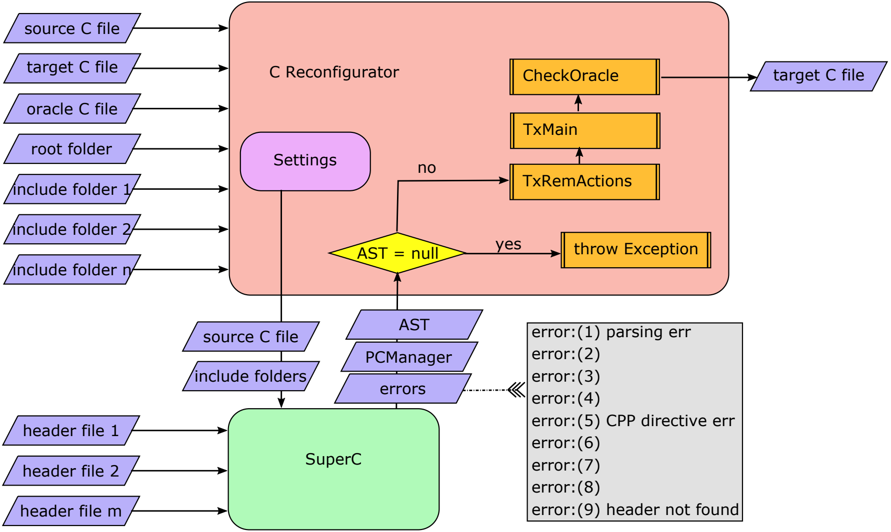
C Reconfigurator

Alexandru F. Iosif-Lazăr

ITU Copenhagen

26-04-2017

C Reconfigurator Overview



V-AST Syntax

$$\begin{array}{rcl} node & ::= & (name, list) \\ list & ::= & \epsilon \mid obj :: list \\ obj & ::= & pc \mid lang \mid node \end{array}$$

- A *node* is a tuple of two elements: a string *name* and a *pair* which contains the children.
- A *list* can either be empty (ϵ) or it can be formed of a head of type *obj* and a tail of type *list*.
- An *obj* can be a presence condition *pc*, a language element *lang* or a *node*.

V-AST vs. Java/Xtend

V-AST	Java/Xtend
obj	Object
pc	PresenceCondition
lang	Language<CTag>
node	GNode
list	Pair<Object>

Table: Mapping from V-AST vertices to Java classes.

Rule structure

```
abstract class Rule {

  def init() {
    this
  }

  def dispatch PresenceCondition transform(PresenceCondition cond) {
    throw new UnsupportedOperationException("TODO: auto-generated method stub")
  }

  def dispatch Language<CTag> transform(Language<CTag> lang) {
    throw new UnsupportedOperationException("TODO: auto-generated method stub")
  }

  def dispatch Pair<Object> transform(Pair<Object> pair) {
    throw new UnsupportedOperationException("TODO: auto-generated method stub")
  }

  def dispatch Object transform(GNode node) {
    throw new UnsupportedOperationException("TODO: auto-generated method stub")
  }
}
```

Other rule types

```
abstract class Rule {

}

//----------------------------------------------------------------------

abstract class AncestorGuaranteedRule extends Rule {

  protected var ArrayList<GNode> ancestors

  // Returns the presence condition guarding the node from the bottom-most
      Conditional ancestor.
  def protected PresenceCondition guard(Node node) { ... }

  // Computes the conjunction of all ancestor PresenceConditions of a Node.
  def protected PresenceCondition presenceCondition(Node node) { .. }

}

//----------------------------------------------------------------------

abstract class ScopingRule extends AncestorGuaranteedRule {

  // Collects declarations as it traverses the AST top-bottom until the place
      where it can be applied.
  protected val DeclarationScopeStack variableDeclarations
  protected val DeclarationPCMap typeDeclarations
  protected val DeclarationPCMap functionDeclarations

}
```

Strategy

```
abstract class Strategy {

  protected val ArrayList<Rule> rules

  new() {
    this.rules = new ArrayList<Rule>
  }

  def register(Rule rule) {
    rules.add(rule.init())
  }

  def dispatch PresenceCondition transform(PresenceCondition cond) {
    throw new UnsupportedOperationException("TODO:␣auto−generated␣method␣stub")
  }

  def dispatch Language<CTag> transform(Language<CTag> lang) {
    throw new UnsupportedOperationException("TODO:␣auto−generated␣method␣stub")
  }

  def dispatch Pair<Object> transform(Pair<Object> pair) {
    throw new UnsupportedOperationException("TODO:␣auto−generated␣method␣stub")
  }

  def dispatch Object transform(GNode node) {
    throw new UnsupportedOperationException("TODO:␣auto−generated␣method␣stub")
  }

}
```

Other Strategy types

```
abstract class Strategy {

  protected val ArrayList<Rule> rules

  new() { this.rules = new ArrayList<Rule> }

  def register(Rule rule) {
    rules.add(rule.init())
  }
}

//-----------------------------------------------------------------

abstract class AncestorGuaranteedStrategy extends Strategy {

  protected val ArrayList<GNode> ancestors

  new() {
    super()
    this.ancestors = new ArrayList<GNode> }

  public def register(AncestorGuaranteedRule rule) {
    rules.add(rule.init(ancestors))
  }
}

//-----------------------------------------------------------------

class TopDownStrategy extends AncestorGuaranteedStrategy {

}
```

Immutable AST

DeclarationOrStatementList$_\circ$
- Conditional$_\circ$
  - ψ
  - Conditional$_\circ$
    - ϕ_1
    - *children*$_{1::}$
  - Conditional$_\circ$
    - ϕ_2
    - *children*$_{2::}$
- *tail*$_{::}$

DeclarationOrStatementList$_\circ$
- Conditional$_\circ$
  - $\phi_1 \wedge \psi$
  - *children*$_{1::}$
- Conditional$_\circ$
  - $\phi_2 \wedge \psi$
  - *children*$_{2::}$
- *tail*$_{::}$

assuming

$\phi_2 \wedge \psi = true_\phi$

DeclarationOrStatementList$_\circ$
- Conditional$_\circ$
  - $\phi_1 \wedge \psi$
  - *children*$_{1::}$
- *children*$_{2::}$
- *tail*$_{::}$

Top-Down Strategy I

```
pc transform (pc cond_φ)
  newCond_φ = cond_φ
  rules.foreach[rule |
    newCond_φ = rule.transform(newCond_φ)
  ]
  return newCond_φ
```

```
lang transform (lang lang_@)
  newLang_@ = lang_@
  rules.foreach[rule |
    newLang_@ = rule.transform(newLang_@)
  ]
  return newLang_φ
```

Top-Down Strategy II

```
obj transform (node node○)
  newNode○ = node○
  do prev○ = newNode○
    rules.foreach[rule |
      newNode○ = rule.transform(newNode○)
    ]

    if (newNode○ : (NodeName,children::))
      ancestor○ = newNode○
      ancestors.add(ancestor○)
      newNode○ = (NodeName, transform(
            children::))
      ancestors.remove(ancestor○)
      if (ancestor○ != newNode○)
        return newNode○
  while (newNode○ != prev○)
  return newNode○
```

```
list transform (list list::)
  if (list:: != ϵ)
    newList:: = list::
    do prev:: = newList::
      rules.foreach[rule |
        newList:: = rule.transform(newList::)
      ]

    if (newList:: != prev::)
      return newList::

    if (newList:: : head?::tail::)
      newHead? = transform(head?)
      newList:: = transform(tail::)

      if (newHead? != head? || newList:: !=
            tail::)
        return newHead?::newList::
      else
        return newList::
```

RemOneRule

Input $list_{::}$	Output $\_{::}$	Algorithm

Conditional$_{\circ}$
  true$_{\phi}$
  children$_{::}$
tail$_{::}$
	children$_{::}$
	tail$_{::}$
		preconditions:
		  $list_{::}$!= ϵ
		  $list_{::}$.head : (Conditional, _)
		  $list_{::}$.head.filter(pc).size = 1
		  $list_{::}$.head[0] = true$_{\phi}$
		
		do:
		  return $list_{::}$.head
		    .getChildrenGuardedBy(true$_{\phi}$)
		    .append($list_{::}$.tail)

RemZeroRule

Input $list_{::}$	Output $\_{::}$	Algorithm

┌ Conditional$_{\circ}$
│ ├ false$_{\phi}$
│ └ *children*$_{::}$
└ *tail*$_{::}$
	tail$_{::}$	preconditions:
		  $list_{::}$!= ϵ
		  $list_{::}$.head : (Conditional, _)
		  $list_{::}$.head.filter(pc).size = 1
		  $list_{::}$.head[0] = false$_{\phi}$
		
		do:
		  return $list_{::}$.tail

SplitConditionalRule

Input $list_{::}$	Output $\_{::}$	Algorithm
Conditional$_\circ$ ϕ_1 $children_{1::}$ ϕ_2 $children_{2::}$ ϕ_n $children_{n::}$ $tail_{::}$	Conditional$_\circ$ ϕ_1 $children_{1::}$ Conditional$_\circ$ ϕ_2 $children_{2::}$ Conditional$_\circ$ ϕ_n $children_{n::}$ $tail_{::}$	preconditions: $list_{::}$!= ϵ $list_{::}$.head : (Conditional, _) $list_{::}$.head.filter(pc).size >= 2 do: $newList_{::}$ = ϵ for (ϕ_i in [ϕ_1..ϕ_n]) $newList_{::}$ = $newList_{::}$.append((Conditional, ϕ_i::$list_{::}$.head.getChildrenGuardedBy(ϕ_i))) return $list_{::}$.tail

ConstrainNestedConditionalsRule

Input $node_{\circ}$	Output $\_{\circ}$	Algorithm

ancestors:
(Conditional,$\psi_{\mathbf{1}}$::$\_{::}$)
(Conditional,$\psi_{\mathbf{2}}$::$\_{::}$)
(Conditional,ψ_{n}::$\_{::}$)
 	
Conditional$_{\circ}$
├ $\phi_{\mathbf{1}}$
└ $children_{::}$
	Conditional$_{\circ}$
	├ constrain($\phi_{\mathbf{1}}$, $\psi_{\mathbf{1}} \wedge \psi_{\mathbf{2}} \wedge \psi_{n}$)
	└ $children_{::}$
		preconditions:
		$node_{\circ}$.name = Conditional
		$node_{\circ}$.filter(pc).size = 1
		
		do:
		$simpl_{\phi}$ = constrain(ϕ, $node_{\circ}$.presenceCondition)
		
		if ($simpl_{\phi}$!= ϕ)
		  return (Conditional,$simpl_{\phi}$::$node_{\circ}$.toList.tail)
		else
		  return $node_{\circ}$

ConditionPushDownRule

Input $list_{::}$

- Conditional$_{\circ}$
  - $\psi_{\mathbf{1}}$
  - Conditional$_{\circ}$
    - $\phi_{\mathbf{11}}$
    - *children*$_{\mathbf{11}::}$
  - $\psi_{\mathbf{2}}$
  - Conditional$_{\circ}$
    - $\phi_{\mathbf{21}}$
    - *children*$_{\mathbf{21}::}$
    - $\phi_{\mathbf{22}}$
    - *children*$_{\mathbf{22}::}$
  - Conditional$_{\circ}$
    - $\phi_{\mathbf{31}}$
    - *children*$_{\mathbf{31}::}$
  - ψ_{n}
  - Conditional$_{\circ}$
    - $\phi_{n\mathbf{1}}$
    - *children*$_{n\mathbf{1}::}$
- *tail*$_{::}$

Output $\_{::}$

- Conditional$_{\circ}$
  - $\phi_{\mathbf{11}} \wedge \psi_{\mathbf{1}}$
  - *children*$_{\mathbf{11}::}$
- Conditional$_{\circ}$
  - $\phi_{\mathbf{21}} \wedge \psi_{\mathbf{2}}$
  - *children*$_{\mathbf{21}::}$
  - $\phi_{\mathbf{22}} \wedge \psi_{\mathbf{2}}$
  - *children*$_{\mathbf{22}::}$
- Conditional$_{\circ}$
  - $\phi_{\mathbf{31}} \wedge \psi_{\mathbf{2}}$
  - *children*$_{\mathbf{31}::}$
- Conditional$_{\circ}$
  - $\phi_{n\mathbf{1}} \wedge \psi_{n}$
  - *children*$_{n\mathbf{1}::}$
- *tail*$_{::}$

Algorithm

preconditions:

$list_{::}$!= ϵ

$list_{::}$.head : (Conditional, _)

$list_{::}$.head . **forall** [**it** : $\_{\phi}$ $\vee$ **it** : (Conditional, _)]

do:

$list_{::}$.head . **filter** (*cond*) . **map** [$node_{\circ}$ |
  (Conditional, $node_{\circ}$. **map** [$child_{?}$ |
    if ($child_{?}$: $\_{\phi}$) $child_{\phi} \wedge$ pcOf($node_{\circ}$)
    else $child_{?}$
  ])
] . append($tail_{::}$)

MergeSequentialMutexConditionalRule

Input $list_{::}$	Output $\_{::}$	Algorithm
$\text{Conditional}_{\circ}$ $\phi_{\mathbf{1}}$ $children_{\mathbf{1}::}$ $\text{Conditional}_{\circ}$ $\phi_{\mathbf{2}}$ $children_{\mathbf{2}::}$ $tail_{::}$	$\text{Conditional}_{\circ}$ $\phi_{\mathbf{1}} \vee \phi_{\mathbf{2}}$ $children_{\mathbf{1}::}$ $tail_{::}$	preconditions: $list_{::}$!= ϵ $list_{::}$.size >= 2 $list_{::}$.head : (Conditional, _) $list_{::}$.head.filter(*cond*).size == 1 $list_{::}$.tail.head : (Conditional, _) $list_{::}$.tail.head.filter(*cond*).size == 1 areMutex($\phi_{\mathbf{1}}$,$\phi_{\mathbf{2}}$) structurallyEquals($children_{\mathbf{1}::}$,$children_{\mathbf{2}::}$) do: (Conditional, $\phi_{\mathbf{1}} \vee \phi_{\mathbf{2}}$ :: $children_{\mathbf{1}::}$) :: $tail_{::}$