1 Syntax

1.1 V-AST Syntax

An abstract syntax tree with variability (V-AST) can be represented using the syntax given in (1). A tree vertex (or an object) *obj* can be either a presence condition (*pc*), a language element (*lang*) or a generic node (*node*). All V-AST types have corresponding types in the Java/XTend code as shown in Tbl. 1.

$$\begin{array}{rcl} obj & ::= & pc \mid lang \mid node \\ node & ::= & (name, pair) \\ pair & ::= & \epsilon \mid obj :: pair \end{array} \tag{1}$$

Objects. Objects are represented only as identifiers $o_{1?}$, $o_{2?}$.

Nodes. A *node* is a tuple of two elements: a string *name* and a *pair* which contains the children. It can be represented as an identifier $n_{1\circ}$, $n_{2\circ}$, as a tuple $(\mathsf{Conditional},p_{()})$ or by using the *name* $\mathsf{Conditional}_\circ$. The *name* of the *node* can be accessed through the function name (e.g. $\mathsf{Conditional}_\circ.\mathsf{name}$). The *node* type also has an index accessor for its children (e.g. $(\mathsf{Conditional},p_{()})$[i] is accessing child i as ordered in the *pair* $p_{()}$).

Pair. A *pair* can either be empty (ϵ) or it can be formed of a head of type *obj* and a tail of type *pair*. The head and tail can be accessed through their respective functions: $p_{1()}.\mathsf{head}$ and $p_{1()}.\mathsf{tail}$.

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

Table 1: Mapping from V-AST vertices to Java classes.

type	representations
obj	$o_{1?}$, $o_{2?}$
pc	$pc_{1\phi}$, $pc_{2\phi}$, ϕ_1, ψ_1, true_ϕ, false_ϕ, $\phi \wedge \mathsf{false}$
lang	$ln_@$
node	$n_\circ$, $(\mathsf{Conditional},p_{()})$, $\mathsf{Conditional}_\circ$
pair	$p_{0()}$

Table 2: Various representations for vertices.

1.2 Rule Syntax

A rule has a name, an input pattern, an output patten and an algorithm.

RemOneRule		
Input $pair_{()}$	Output $\_{()}$	Algorithm
Conditional$_{\circ}$ true$_{\phi}$ $children_{()}$ $tail_{()}$	$children_{()}$ $tail_{()}$	preconditions: $pair_{()}$!= ϵ $pair_{()}$.head : (Conditional, _) $pair_{()}$.head.filter(pc).size = 1 $pair_{()}$.head[0] = true$_{\phi}$ do: return $pair_{()}$.head .getChildrenGuardedBy(true$_{\phi}$) .append($pair_{()}$.tail)

RemZeroRule		
Input $pair_{()}$	Output $\_{()}$	Algorithm
Conditional$_{\circ}$ false$_{\phi}$ $children_{()}$ $tail_{()}$	$tail_{()}$	preconditions: $pair_{()}$!= ϵ $pair_{()}$.head : (Conditional, _) $pair_{()}$.head.filter(pc).size = 1 $pair_{()}$.head[0] = false$_{\phi}$ do: return $pair_{()}$.tail

SplitConditionalRule		
Input $pair_{()}$	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: $pair_{()}$!= ϵ $pair_{()}$.head : (Conditional, _) $pair_{()}$.head.filter(pc).size >= 2 do: $newPair_{()} = \epsilon$ for (ϕ_i in $[\phi_1..\phi_n]$) $newPair_{()} = newPair_{()}$.append((Conditional, ϕ_i::$pair_{()}$.head.getChildrenGuardedBy(ϕ_i))) return $pair_{()}$.tail

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

ancestors :
(Conditional,ψ_1::$\_{()}$)
(Conditional,ψ_2::$\_{()}$)
(Conditional,ψ_n::$\_{()}$)
 	
Conditional$_{\circ}$
├ ϕ_1
└ $children_{()}$
	Conditional$_{\circ}$
	├ constrain(ϕ_1, $\psi_1 \wedge \psi_2 \wedge \psi_n$)
	└ $children_{()}$
		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,$simpl_{\phi}$::$node_{\circ}$.toPair.tail)
		else
		  return $node_{\circ}$

CONDITIONPUSHDOWNRULE		
Input $pair_{()}$	Output $\_{()}$	Algorithm

┌ Conditional$_{\circ}$
  ├ ψ_1
  ├ Conditional$_{\circ}$
    ├ ϕ_{11}
    └ $children_{11\,()}$
  ├ ψ_2
  ├ Conditional$_{\circ}$
    ├ ϕ_{21}
    ├ $children_{21\,()}$
    ├ ϕ_{22}
    └ $children_{22\,()}$
  ├ Conditional$_{\circ}$
    ├ ϕ_{31}
    └ $children_{31\,()}$
  ├ ψ_n
  └ Conditional$_{\circ}$
    ├ ϕ_{n1}
    └ $children_{n1\,()}$
└ $tail_{()}$
	┌ Conditional$_{\circ}$
	  ├ $\phi_{11} \wedge \psi_1$
	  └ $children_{11\,()}$
	├ Conditional$_{\circ}$
	  ├ $\phi_{21} \wedge \psi_2$
	  ├ $children_{21\,()}$
	  ├ $\phi_{22} \wedge \psi_2$
	  └ $children_{22\,()}$
	├ Conditional$_{\circ}$
	  ├ $\phi_{31} \wedge \psi_2$
	  └ $children_{31\,()}$
	├ Conditional$_{\circ}$
	  ├ $\phi_{n1} \wedge \psi_n$
	  └ $children_{n1\,()}$
	└ $tail_{()}$
		preconditions :
		$pair_{()}$!= ϵ
		$pair_{()}$.head : (Conditional,_)
		$pair_{()}$.head . forall [it : $\_{\phi}$ $\vee$ it : (Conditional,_)]
		
		do :
		$pair_{()}$.head . filter ($cond$) . map [$node_{\circ}$ |
		  (Conditional,$node_{\circ}$. map [$child_{?}$ |
		    if ($child_{?}$: $\_{\phi}$) $child_{\phi}$ $\wedge$ pcOf($node_{\circ}$)
		    else $child_{?}$
		  ])
		] . append($tail_{()}$)

<table>
<tr><td colspan="3">MergeSequentialMutexConditionalRule</td></tr>
<tr><td>Input $pair_{()}$</td><td>Output $\_{()}$</td><td>Algorithm</td></tr>
<tr><td>Conditional$_\circ$
ϕ_1
$children_{1\,()}$
Conditional$_\circ$
ϕ_2
$children_{2\,()}$
$tail_{()}$</td><td>Conditional$_\circ$
$\phi_1 \lor \phi_2$
$children_{1\,()}$
$tail_{()}$</td><td>preconditions:
$pair_{()}$!= ϵ
$pair_{()}$.size >= 2
$pair_{()}$.head : (Conditional, _)
$pair_{()}$.head . filter (cond) . size == 1
$pair_{()}$.tail.head : (Conditional, _)
$pair_{()}$.tail.head . filter (cond) . size == 1
areMutex(ϕ_1,ϕ_2)
structurallyEquals($children_{1\,()}$,$children_{2\,()}$)

do:
(Conditional, $\phi_1 \lor \phi_2$ :: $children_{1\,()}$) :: $tail_{()}$</td></tr>
</table>

1.3 Functions

Constrain / Generalized cofactor. The constrain function (2), a.k.a. the generalized cofactor, constrains a presence condition ϕ by another presence condition ψ that has already been decided to be true and eliminates any redundant variables.

$$\begin{aligned}&\mathsf{models}(\phi) = \{M \mid M \implies \phi\}\\ &\mathsf{constrain}(\phi,\psi) = \phi' \left| \begin{array}{l}\phi' \land \psi \implies \phi\\ \mathsf{models}(\phi') = \mathsf{models}(\phi) \backslash \mathsf{models}(\psi)\end{array}\right. \qquad \textit{and}\end{aligned} \tag{2}$$

For example, $\mathsf{constrain}(A \land C, A \land B) = C$ shows that, having decided that $A \land B$ holds, C is the minimal proposition required to show that $A \land C$ also holds: $C \land A \land B \to A \land C$.

Another example: $\mathsf{constrain}(B \lor C, A \land B) = \mathsf{true}$ shows that, having decided that $A \land B$ holds, true is the minimal proposition required to show that $B \lor C$ also holds: $\mathsf{true} \land A \land B \to B \lor C$.