Variables – Free or Bound?

in expressions in functional programming, a free variable is a variable, which has not been bound

- through a *let binding*, or
- as a *formal parameter* of a function definition

for example:

input	*free*
x + 1	$\{ x \}$
let (x = 5 + z) **in** y + x	$\{ y, z \}$
let (x = x + 1) **in** y + x	$\{ x, y \}$
if y **then** x **else** z	$\{ x, y, z \}$
fun (x y) −> x + z	$\{ z \}$
f (f 1 x)	$\{ f, x \}$

Attribute Grammar for Free Variables in a (toy) Functional Language

The following grammar represents a fraction of expressions in a functional language.

rule			production	attribute system
1	*expr*	$\rightarrow$	*expr* `+` *expr*	
2		|	`let` `(` `var` `=` *expr* `)` `in` *expr*	
3		|	`if` *expr* `then` *expr* `else` *expr*	
4		|	`const`	
5		|	`var`	
6		|	`fun` `(` *varseq* `)` `->` *expr*	
7		|	*expr* `(` *expseq* `)`	
8	*varseq*	$\rightarrow$	`var` *varseq*	
9		|	`var`	
10	*expseq*	$\rightarrow$	*expr* *expseq*	
11		|	*expr*	

Attribute Grammar for Free Variables in a (toy) Functional Language

The following grammar represents a fraction of expressions in a functional language.

rule			production	attribute system
1	*expr*	$\rightarrow$	*expr* `+` *expr*	$free[0] := free[1] \cup free[3]$
2		$\mid$	`let` `(` `var` `=` *expr* `)` `in` *expr*	$free[0] := (free[8] \setminus \{n[3]\}) \cup free[5]$
3		$\mid$	`if` *expr* `then` *expr* `else` *expr*	$free[0] := free[2] \cup free[4] \cup free[6]$
4		$\mid$	`const`	$free[0] := \{\}$
5		$\mid$	`var`	$free[0] := \{n[1]\}$
6		$\mid$	`fun` `(` *varseq* `)` `->` *expr*	$free[0] := free[6] \setminus free[3]$
7		$\mid$	*expr* `(` *expseq* `)`	$free[0] := free[1] \cup free[3]$
8	*varseq*	$\rightarrow$	`var` *varseq*	$free[0] := \{n[1]\} \cup free[2]$
9		$\mid$	`var`	$free[0] := \{n[1]\}$
10	*expseq*	$\rightarrow$	*expr* *expseq*	$free[0] := free[1] \cup free[2]$
11		$\mid$	*expr*	$free[0] := free[1]$

Correctly Nested Expression Statements

A few specific properties of languages are usually not treated with syntactical rules, instead they are addressed via the semantical analysis.

(Artificial) Example:

Expressions may comprise

- binary operators
- value assignment
- (multi-dimensional) array access.

input	*nesting*
x = y+1	*valid*
x = y = z[3]+1	*valid*
x[1] = y	*valid*
y+1 = z	*invalid*
5 = z+1	*invalid*
z++ = 42	*invalid*

Correctly Nested Expression Statements – Attribute Grammar

The following grammar represents the fraction of a language that treats expression statements.

rule	production			attribute system
1	*S'*	$\rightarrow$	{ *stmts* }	
2	*stmts*	$\rightarrow$	*expr* ; *stmts*	
3		|	ε	
4	*expr*	$\rightarrow$	*expr* `binop` *expr*	
5		|	*expr* = *expr*	
6		|	*expr* [*elist*]	
7		|	`const`	
8		|	`var`	
9	*elist*	$\rightarrow$	*expr* , *elist*	
10		|	*expr*	
12				
11				

Correctly Nested Expression Statements – Attribute Grammar

The following grammar represents the fraction of a language that treats expression statements.

rule	production			attribute system
1	*S'*	$\rightarrow$	{ *stmts* }	$valid[0] := valid[2]$
2	*stmts*	$\rightarrow$	*expr* ; *stmts*	$valid[0] := valid[1] \wedge valid[3]$
3		|	ε	$valid[0] :=$ `true`
4	*expr*	$\rightarrow$	*expr* `binop` *expr*	$lhs[0] :=$ `false` $valid[0] := valid[1] \wedge valid[3]$
5		|	*expr* = *expr*	$lhs[0] :=$ `false` $valid[0] := lhs[1] \wedge valid[3]$
6		|	*expr* [*elist*]	$lhs[0] :=$ `true` $valid[0] := valid[1] \wedge valid[3]$
7		|	`const`	$lhs[0] :=$ `false` $valid[0] :=$ `true`
8		|	`var`	$lhs[0] :=$ `true` $valid[0] :=$ `true`
9	*elist*	$\rightarrow$	*expr* , *elist*	$valid[0] := valid[1] \wedge valid[3]$
10		|	*expr*	$valid[0] := valid[1]$
12	*expr*	$\rightarrow$	*expr* `incop`	$lhs[0] :=$ `false` $valid[0] := lhs[1]$
11	*expr*	$\rightarrow$	`incop` *expr*	$lhs[0] :=$ `false` $valid[0] := lhs[2]$

Cycles in Attribute Grammars

Consider the following Attribute Grammar G, with the start symbol R:

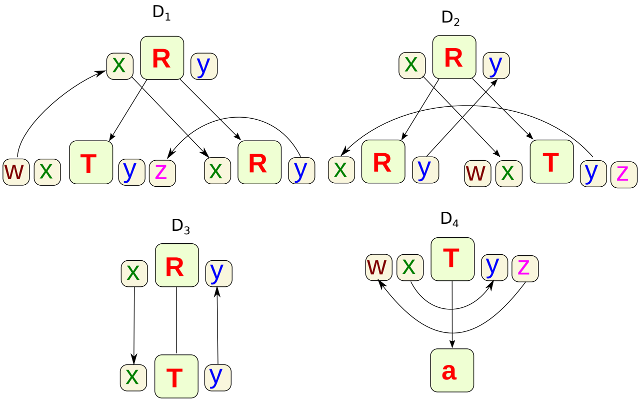
$$\begin{array}{llllll}
R \to T\,R & : & x[0] = w[1], & x[2] = 2 \cdot x[0], & z[1] = min(y[2], 1) \\
R \to R\,T & : & y[0] = min(y[1], 0), & x[1] = y[2] + 1, & x[2] = x[0] \\
R \to T & : & y[0] = y[1], & x[1] = x[0] & \\
T \to a & : & y[0] = x[0] + 2, & w[0] = z[0] &
\end{array}$$

Compute the overapproximation of the global dependencies $\mathcal{R}^\star$ of all non-terminals from G, according to the method from the strongly acyclicity test.

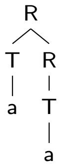
Please give reasons for/against:

1. Is G L-attributed?
2. Is G strongly acyclic?
3. Is G acyclic?

$$
\begin{array}{llllll}
R \to T\,R & : & x[0] = w[1], & x[2] = 2 \cdot x[0], & z[1] = min(y[2], 1) \\
R \to R\,T & : & y[0] = min(y[1], 0), & x[1] = y[2] + 1, & x[2] = x[0] \\
R \to T & : & y[0] = y[1], & x[1] = x[0] & \\
T \to a & : & y[0] = x[0] + 2, & w[0] = z[0] &
\end{array}
$$



1. G is not L-attributed; in production $R \to T\ R$, we have $y[2] \to z[1]$
2. $\mathcal{R}^\star(R)$ is cyclic with $x \to x$
3. G is actually cyclic: We get a concrete cycle for



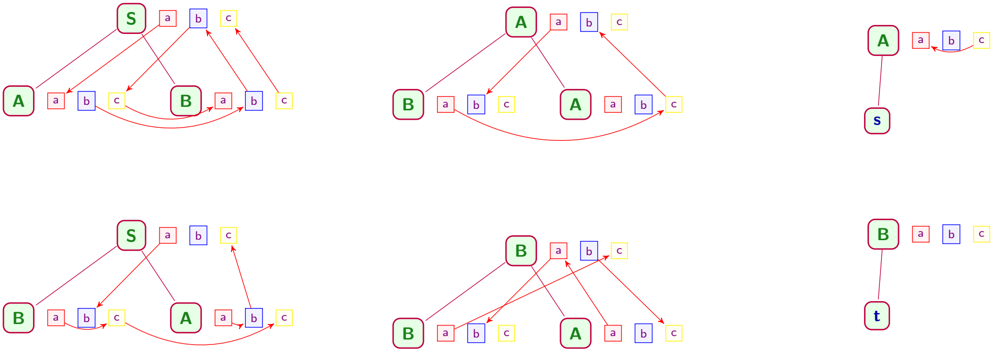
non-terminal	1st iteration	2nd iteration	3rd iteration
$\mathcal{R}(T)$	$x \to y,\ z \to w$	$x \to y,\ z \to w$	$x \to y,\ z \to w$
$\mathcal{R}(R)$		$x \to y$	$x \to y,\ x \to x$

Another one

Consider Attribute Grammar G:

rule	production	attribute system
1	$S \rightarrow A\,B$	$a[1] ::= a[0]$ $b[2] ::= b[1]$ $c[1] ::= b[0]$ $a[2] ::= c[1]$ $b[0] ::= b[2]$ $c[0] ::= c[2]$
2	$S \rightarrow B\,A$	$c[0] ::= b[2]$ $b[1] ::= a[0]$ $c[2] ::= c[1]$ $c[1] ::= a[1]$ $b[2] ::= a[2]$
3	$A \rightarrow B\,A$	$b[0] ::= c[2]$ $c[2] ::= a[1]$ $b[1] ::= a[0]$
4	$B \rightarrow B\,A$	$c[2] ::= b[0]$ $c[0] ::= a[1]$ $b[1] ::= a[0]$ $a[0] ::= a[2]$
5	$A \rightarrow s$	$a[0] ::= c[0]$
6	$B \rightarrow t$	

1. Provide the local dependency graphs for , S, A, B
2. Compute the least solution of $\mathcal{R}^\star$ for all nonterminals , S, A, B
3. Is G strongly acyclic?



iteration	0	1	2	3	4
$\mathcal{R}(S)$			$a \rightarrow c$	$a \rightarrow b \quad a \rightarrow c \quad b \rightarrow b \quad b \rightarrow c$	$a \rightarrow b \quad a \rightarrow c \quad b \rightarrow b \quad b \rightarrow c$
$\mathcal{R}(A)$	$c \rightarrow a$	$c \rightarrow a$	$a \rightarrow b \quad c \rightarrow a$	$a \rightarrow b \quad c \rightarrow a$	$a \rightarrow b \quad c \rightarrow a$
$\mathcal{R}(B)$		$b \rightarrow a$	$a \rightarrow c \quad b \rightarrow a \quad b \rightarrow c$	$a \rightarrow c \quad b \rightarrow a \quad b \rightarrow c$	$a \rightarrow c \quad b \rightarrow a \quad b \rightarrow c$