

Exercise 1 (Circularity of attribute grammars):

(6 Points)

Consider the following grammar $G = (N, \Sigma, P, S)$ with inherited attributes $i1, i2$ and synthesised attributes $s1, s2$.

$$\begin{aligned}
 S' &\rightarrow S & i1.0 &= 1 \\
 & & i2.0 &= 2 \\
 & & i1.1 &= s1.1 \\
 & & i2.1 &= i1.0 \\
 & & s2.0 &= s2.1 \\
 S &\rightarrow AA & i1.1 &= s1.1 \\
 & & i2.1 &= i1.0 \\
 & & i1.2 &= 0 \\
 & & i2.2 &= i2.0 \\
 & & s1.0 &= s2.1 \\
 & & s2.0 &= s2.2 \\
 S &\rightarrow A & i1.1 &= 0 \\
 & & i2.1 &= i2.0 \\
 & & s2.0 &= s2.1 \\
 A &\rightarrow a & s1.0 &= 0 \\
 & & s2.0 &= i1.0 \\
 A &\rightarrow b & s2.0 &= 0 \\
 & & s1.0 &= i2.0
 \end{aligned}$$

- Provide the dependency graph for each production in G .
- Apply the circularity test from the lecture to G .
 - Calculate the set $IS(A)$ for all $A \in N$.
 - Is G circular? Justify your answer.
- To simplify the circularity test we want to consider so-called strong circularity. To this aim we modify our circularity test in a way, that attribute dependencies caused by different syntax trees are not distinguished anymore. Thus $IS(A)$, $A \in N$, is now defined as follows:

$$IS(A) = \{(\beta, \alpha) | \beta \xrightarrow{A} \alpha \text{ in some syntax tree } t \text{ with root label } A\} \subseteq Inh \times Syn$$

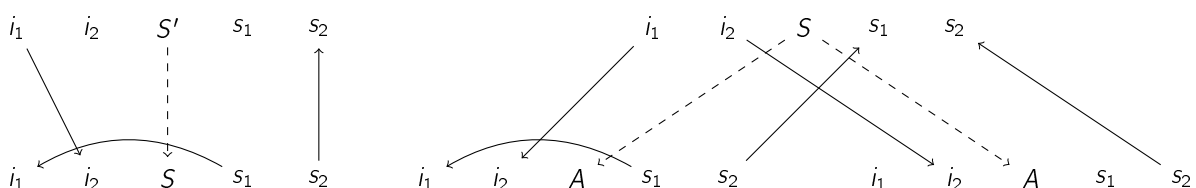
with $\beta \in inh(A), \alpha \in syn(A)$.

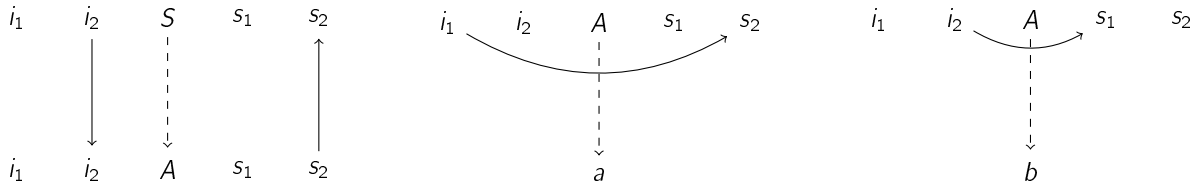
Hint: $IS(A)$ is not a system of attribute dependence sets anymore, but a union!

If we adapt the circularity test from the lecture to this definition of $IS(A)$, does it provide the result that G is strongly non-circular? Argue why!

Lösung:

- For every production π in the attributed grammar, we draw the dependency graph $\rightarrow_{\pi}$.





b) $IS(A) = \{is_1, is_2\}$ where

- $is_1 = is[A \rightarrow a] = \{(i_1, s_2)\}$
- $is_2 = is[A \rightarrow b] = \{(i_2, s_1)\}$

$IS(S) = \{is_3, \dots, is_8\}$ where

- $is_3 = is[S \rightarrow A, is_1] = \emptyset$
- $is_4 = is[S \rightarrow A, is_2] = \emptyset$
- $is_5 = is[S \rightarrow AA, is_1, is_1] = \emptyset$
- $is_6 = is[S \rightarrow AA, is_1, is_2] = \emptyset$
- $is_7 = is[S \rightarrow AA, is_2, is_1] = \emptyset$
- $is_8 = is[S \rightarrow AA, is_2, is_2] = \emptyset$

Now observe that

$$is[S' \rightarrow S, is_3] = \dots = is[S' \rightarrow S, is_8] = is[S' \rightarrow S, \emptyset] .$$

Hence, $IS(S') = \{is_9\}$ where

- $is_9 = is[S' \rightarrow S, \emptyset] = \emptyset$

G is not circular! There are two possible cover productions. In both productions $S' \rightarrow S$ and $S \rightarrow AA$, we have that i_1 depends on s_1 . However we have just shown that there is no subtree of S nor a subtree of A which has any dependencies. Therefore it is impossible to construct a dependency graph with a cycle.

c) Strong **non**circularity

Unfortunately, the definition on the exercise sheet only considers productions of the form $A \rightarrow w$. But the text below indicates what is actually meant. We give a precise definition here.

$IS'(A)$ for strong noncircularity of attribute grammars $IS'(A)$ is a weakened version of $IS(A)$ where we take the union of computed dependencies as follows. $IS'(A)$ is the least subset of $Inh \times Syn$ with

$$\forall \pi = A \rightarrow w_0 A_1 w_1 \dots A_k w_k. is[\pi, IS'(A_1), \dots, IS'(A_k)] \subseteq IS'(A) .$$

□

With that we compute $IS'(A) = \{(i_1, s_2), (i_2, s_1)\}$

$IS'(S) = is'_3 \cup is'_4 = \{(i_1, s_1)\}$ where

- $is'_3 = is[S \rightarrow A, IS'(A)] = \emptyset$
- $is'_4 = is[S \rightarrow AA, IS'(A), IS'(A)] = \{(i_1, s_1)\}$

$IS'(S') = is[S' \rightarrow S, IS'(S)] = \emptyset$

The grammar is *not* strongly noncircular as we can construct a circle, by taking the production to S and using the dependencies in $IS'(S)$.

Exercise 2 (Jasmin):

(4 Points)

Before we turn our attention to code generation, we need to understand our target language, the language of Jasmin.

“Jasmin is an assembler for the Java Virtual Machine. It takes ASCII descriptions of Java classes, written in a simple assembler-like syntax using the Java Virtual Machine instruction set. It converts them into binary Java class files, suitable for loading by a Java runtime system.”

Resources:

- Download Jasmin (<http://jasmin.sourceforge.net/>)
- Make sure you are able to execute it, try to compile and execute the example files
- For further information consult the Jasmin guide (<http://jasmin.sourceforge.net/guide.html>).
- An overview of all JVM op codes can be found here: (<http://homepages.inf.ed.ac.uk/kwxm/JVM/codeByNm.html>).

Hints:

- If you wonder what the I and L in `invokestatic java/lang/String/valueOf(I)Ljava/lang/String;` stand for, check out <http://www.cs.sjsu.edu/~pearce/modules/lectures/co/jvm/jasmin/data.html>
- In Jasmin comments are started with a semicolon, however sometimes the semicolon is part of a method call so mind the difference between:
 - `bipush 10 ;cmt` (*the space before the comment “;cmt” is mandatory!*)
 - `invokestatic java/lang/String/valueOf(I)Ljava/lang/String;` (*the semicolon is part of the statement and must follow it without spaces!*)

There is no framework for this exercise.

Tasks:

1. Write a Jasmin program that prints “Hello World” to the command line.
2. Write a Jasmin program that reads an integer from the command line, increases it by one and prints the result on the command line (no error handling required).
3. Write a Jasmin program that evaluates $3 + (5 * 2)$ and prints the result on the command line.
4. In Jasmin, implement a loop that increases a counter from 0 to 10. On every iteration, if the current value is odd, print the value on the command line.

Write a single file for each subtask, hand in all files packed in one zip file via L2P.

Lösung: _____