Possible Answers for Final Exam (June 6, 2017)

Compilers course

Masters in Informatics and Computing Engineering (MIEIC), 3rd Year

João M. P. Cardoso

Dep. de Engenharia Informática
Faculdade de Engenharia (FEUP), Universidade do Porto,
Porto, Portugal
Email:jmpc@acm.org

PART I (10 PTS)

Group 1. (5 pts)

- Considering the following grammar (parentheses and numbers on the right identify the productions):
  - $S \rightarrow xB \mid yC$ (1, 2)
  - $B \rightarrow Aa$ (3)
  - $C \rightarrow Aba$ (4)
  - $A \rightarrow b \mid \varepsilon$ (5, 6)
- **1a) [1pt] Indicate the First and Follow sets for the grammar variables.**

First(S)={x, y}
First(B)={a,b}
First{C}={b}
First(A)={b} // {b, ε}

Follow(S)={}
Follow(B)={}
Follow{C}={}
Follow(A)={b,a}

Group 1. (5 pts)

- Considering the following grammar (parentheses and numbers on the right identify the productions):
  - $S \rightarrow xB \mid yC$ (1, 2)
  - $B \rightarrow Aa$ (3)
  - $C \rightarrow Aba$ (4)
  - $A \rightarrow b \mid \varepsilon$ (5, 6)
- **1b) [1pt] Indicate if this grammar is LL(1) and show the LL(1) parsing table.**

	x	y	a	b
S	$S \rightarrow xB$	$S \rightarrow yC$		
B			$B \rightarrow Aa$	$B \rightarrow Aa$
C				$C \rightarrow Aba$
A			$A \rightarrow \varepsilon$	$A \rightarrow b$ $A \rightarrow \varepsilon$

The gramar is not LL(1) as there is at least a cell in the LL(1) parsing table with conflicts

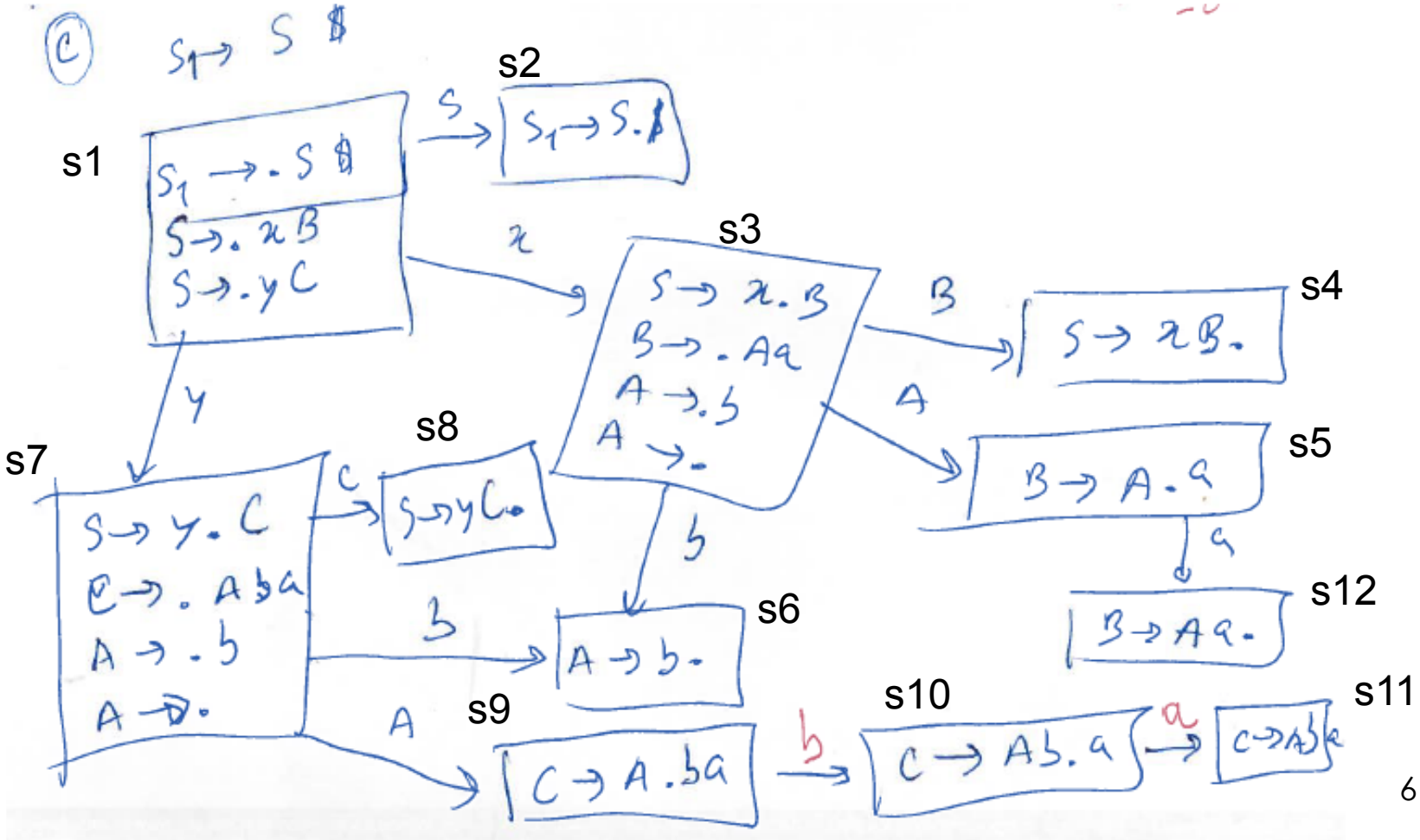
Group 1. (5 pts)

- Considering the following grammar (parentheses and numbers on the right identify the productions):
  - $S \rightarrow xB \mid yC$ (1, 2)
  - $B \rightarrow Aa$ (3)
  - $C \rightarrow Aba$ (4)
  - $A \rightarrow b \mid \varepsilon$ (5, 6)
- **1c) [1pt] Determine and show the LR(0) automaton.**

Group 1. (5 pts)

- **1c) [1pt] Determine and show the LR(0) automaton.**

Add:
S1 → S $
To the gramar (other option would be to add $ to end of every rule in S)



Group 1. (5 pts)

- Considering the following grammar (parentheses and numbers on the right identify the productions):
  - $S \rightarrow xB \mid yC$ (1, 2)
  - $B \rightarrow Aa$ (3)
  - $C \rightarrow Aba$ (4)
  - $A \rightarrow b \mid \varepsilon$ (5, 6)
- **1d) [1pt] Is the grammar LR(0)? Justify your answer indicating the LR(0) parsing table.**

Group 1. (5 pts)

- **1d) [1pt] Is the grammar LR(0)? Justify your answer indicating the LR(0) parsing table.**

State	x	y	a	b	$	S	B	C	A
s1	shift s3	shift s7	error	error	error	goto s2			
s2	error	error	error	error	accept				
s3	red. (6)	red. (6)	red. (6)	red. (6) shift s6	red. (6)		Goto s4		Goto s5
s4	red. (1)	red. (1)	red. (1)	red. (1)	red. (1)				
s5	error	error	shift s12	error	error				
s6	red. (5)	red. (5)	red. (5)	red. (5)	red. (5)				
s7	red. (6)	red. (6)	red. (6)	shift s6 red. (6)	red. (6)			Goto s8	Goto s9
s8	red. (2)	red. (2)	red. (2)	red. (2)	red. (2)				
s9				shift s10					
s10			Shift s11						
s11	red. (4)	red. (4)	red. (4)	red. (4)	red. (4)				
s12	red. (3)	red. (3)	red. (3)	red. (3)	red. (3)				

The grammar is not LR(0). There are shift/reduce conflicts in the LR(0) parsing table.

Group 1. (5 pts)

- **1e) [1pt] The following grammar is not LL(k). Indicate changes to the grammar that make it LL(k).**
  - ID = [a-z][0-9a-z]*
  - S → E = E ;
  - S → E ;
  - E → E * E
  - E → E + E
  - E → E (E)
  - E → ID

New grammar:
S → E S1
S1 → ; | = E;
E → ID E1
E1 → ε | + E E1 | * E E1 | (E) E1

Group 2. (3 pts)

- The goal is to define a programming language with a grammar that forces applying certain semantic rules. A team designing the language finds itself discussing the possibilities to verify if the data type defined as return in a function header coincides with the data type in the *return* instructions used in the code of the same function.
  - **2a) [1pt] Indicate the necessary aspects and possible restrictions that need to be taken into account in terms of such programming language to make the semantic rules mentioned above implemented by the grammar;**
  - **2b) [2pt] Indicate possible grammar rules that illustrate the way those semantic rules can be implemented considering the int, float, and double data types. Notice that it is only necessary to indicate sections of the grammar that illustrate the way the semantic rules can be implemented by the grammar.**

Group 2. (3 pts)

- **2a) [1pt] Indicate the necessary aspects and possible restrictions that need to be taken into account in terms of such programming language to make the semantic rules mentioned above implemented by the grammar;**

- Possibilities:
  - Declaration of the return variable
  - Use of a reserved word for the variable to be return and forcing by grammar rules the adequate type
  - Identifiers with the identification of the type
  - Enforcing the use of casts associated with the return type

Group 2. (3 pts)

- **2b) [2pt] Indicate possible grammar rules that illustrate the way those semantic rules can be implemented considering the int, float, and double data types. Notice that it is only necessary to indicate sections of the grammar that illustrate the way the semantic rules can be implemented by the grammar.**

Example using the declaration of a specific return variable:
FunctionHeader int → FunctionName(…) { … DeclareRetVar OtherStatements Return }
DeclareRetVar → int ret;
OtherStatements → …
Return → return ret;
…
FunctionHeader double → FunctionName(…) {… DeclareRetVar OtherStatements Return }
DeclareRetVar → double ret;

Group 3. (2 pts)

- **3a) [2pt] Comment the following sentence: "The high level intermediate representation obtained right after semantic analysis is just an AST with variable identifiers replaced by the access type (e.g., *load* array, *store* local variable, *load* local variable, *load* function parameter)."**

- The sentence is generally false as the IR after semantic analysis may have, according to the programming language, operations regarding the conversion between types and thus might be very different from the AST (even in the cases this IR is based on tree of expressions).

PART II (10 PTS)

Group 4. (8 pts)

- **4a) [2pt] Indicate the low level intermediate representation (LLIR) for the section of the code in the example below (where N represents a 32-bit *int* constant) based on expression trees, but considering the *Jouette+* processor (instruction set presented in annex), and the type and storage of the variables given by the following table.**

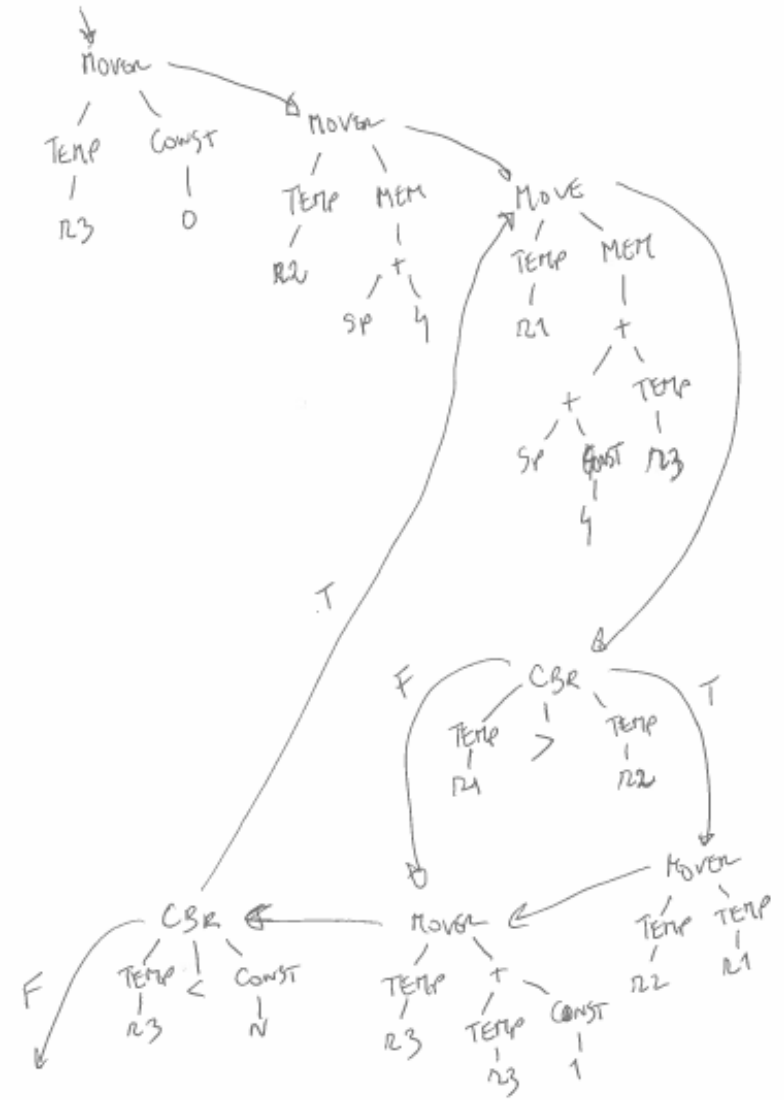
```
//in={}
1i = 0;
2m = A[0];
do {
3    i=i+1;
4    x = A[i];
5    if (x > m)
6        m = x;
7} while(i<N);
// out = {m}
```

Variable	Type	Storage
A	int8 A[N] (array of integers (8-bit))	array stored in the stack starting at SP + 4
i	int i (32-bit scalar variable)	Variable stored in register r3
m	int m (32-bit scalar variable)	Variable stored in register r2
x	int x (32-bit scalar variable)	Variable stored in register r1

Group 4. (8 pts)

- **4a)**

```
//in={}
1i = 0;
2m = A[0];
do {
3    i=i+1;
4    x = A[i];
5    if (x > m)
6        m = x;
7} while(i<N);
// out = {m}
```



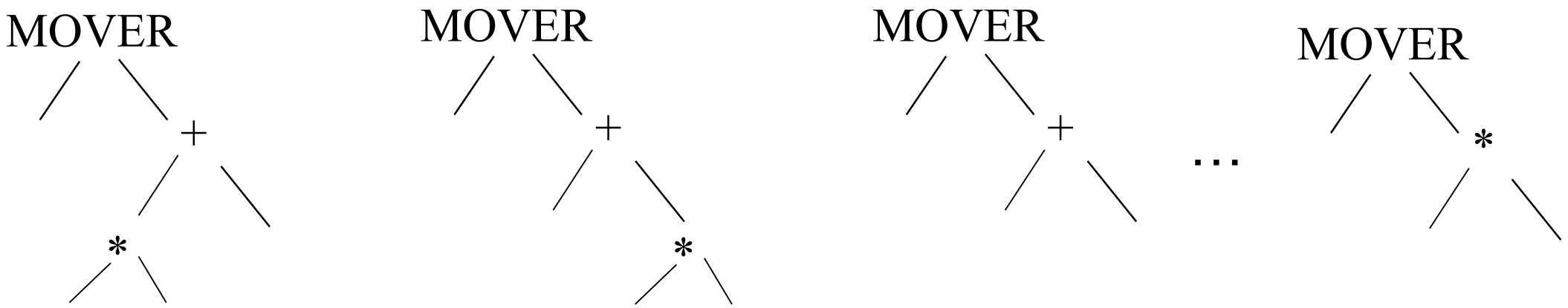
Group 4. (8 pts)

- **4b) [2pt] In this LLIR for the *Jouette*+ processor, code of the type A = B + C, supposing that R2, R3, and R4 are the registers that store A, B and C, respectively, is represented by two instructions, such as illustrated by the following example: MOVER: R2 ← R5; ADD: R5 ← R3 + R4. However, it is possible to make the selection of instructions resulting in the direct generation (i.e., without optimizations and/or elimination of instructions after selection) of the code ADD: R2 ← R3 + R4. In order to do this, indicate the pattern trees of the IR that make the selection of instructions more efficient in the context of the *Jouette*+ processor.**

Group 4. (8 pts)

- **4b)**

  For all instructions which result can be used by a MOVER instruction:



Group 4. (8 pts)

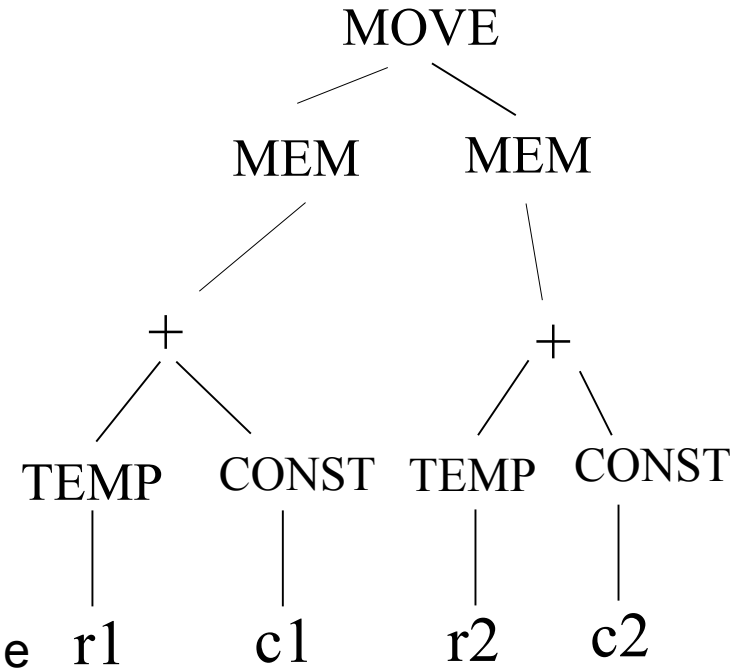
- **4c) [2pt] Taking into account the costs per instruction of the *Jouette*+ processor presented in the table below, indicate an example in which the Maximal Munch algorithm is unable to achieve a selection of instructions that corresponds to the minimum global cost. Indicate which would be the minimum global cost for that example and indicate how the use of dynamic programming can obtain a selection of instructions with that cost.**

Instruction	Cost	Instruction	Cost	Instruction	Cost	Instruction	Cost
MADD	5	LOAD	3	ADD	2	MOVEM	4
MUL	3	STORE	2	ADDI	1	Other instructions	1

Group 4. (8 pts)

- **4c)**

The example corresponding to: M[r1+c1] = M[r2+c2]
(another example: M[0] = M[r1])



Maximal Munch ⇒ 4+1+1 = 6 (cost)
Dynamic Programming ⇒ 2+3 = 5 (cost)

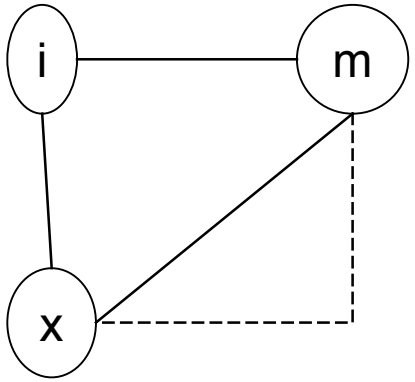
Dynamic Programming:
Starting at the bottom and keeping the best selection for each node allows dynamic programming to select a LOAD and a STORE instead of the MOVEM and two ADDI's. (solution can use the example and show the steps of the algorithm)

Instruction	Cost	Instruction	Cost	Instruction	Cost	Instruction	Cost
MADD	5	LOAD	3	ADD	2	MOVEM	4
MUL	3	STORE	2	ADDI	1	Other instructions	1

Group 4. (8 pts)

- **4d) [2pt] Draw the interference graph for the local scalar variables used in the section of code presented above by direct inspection. Indicate a possible allocation of registers to variables using the graph coloring algorithm explained in class and supposing the utilization of 2 registers (*R1*, e *R2*). Show the content of the stack immediately after the simplification of the interference graph. In the case of having to perform *spilling*, specify an efficient criterion that you suggest for the selection of variables for *spilling* and the order of selection determined by that criterion. Show the result of the first graph coloring (i.e., without repeating the process).**

4d)



Top of stack
i
x
may-spill m

Possible cost function to select the variable(s) to spill:

Variable	#use	#def	#load	#store	Total (#load+#store)	Total/(#def+#use)
i	3	2	3N	1+N	1+4N	(1+4N)/5
x	2	1	2N	N	3N	N
m	1	2	N	1+N	1+2N	(1+2N)/3

Ordering to spill m,
then i, then x

R1={i}
R2={x}
Spill m

Group 5. (2 pts)

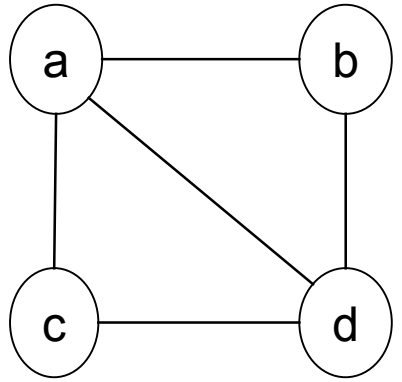
- **5a) [2pt] Suppose there is a need to pack variables in the same register, even if the lifetime of those variables interferes (for instance, 32-bits register can store the value of two 16-bits variables). Assuming 32-bits registers and variables with 8, 16 e de 32-bits data types, indicate what would have to be changed in the register allocations based in graph coloring so that packaging is used to reduce the number of registers required. Use the examples that you think are adequate to illustrate that process.**

- One possibility is to keep the simplification process and to modify the assignment of variables to registers based on the possible slots of contiguous bytes free in a 32-bit register. Now two variables with interference can be assigned to the same register if there are slots available in the register to store the two variables. Example:

Group 5. (2 pts)

- **5a)**
- One possibility is to keep the simplification process and to modify the assignment of variables to registers based on the possible slots of contiguous bytes free in a 32-bit register. Now two variables with interference can be assigned to the same register if there are slots available in the register to store the two variables. Example:

Assume:
a, c: 8-bit variables
b: 16-bit variable
d: 32-bit variable
and the interference graph on the right.
Let's do register allocation considering 2 32-bit registers.
The stack is obtained after the simplification process.
We now start coloring.
Pop d and assign it to R1=1111 (1 identifies a byte occupied and 0 a byte free in the register)
Pop c and assign it to R2=0001
Pop b and assign it to R2=0011
Pop a and assign it to R2=0100



Top of stack
d
c
b
mayspill-a