Formal Languages and Compilers
Proff. Breveglieri, Crespi Reghizzi, Morzenti
Written exam[1]: laboratory question
04/09/2015

SURNAME: ....
NAME: .... Student ID: ....
Course: ○ Laurea Specialistica ○ V. O. ○ Laurea Triennale ○ Other: ...
Instructor: ○ Prof. Breveglieri ○ Prof Morzenti

The laboratory question must be answered taking into account the implementation of the `Acse` compiler given with the exam text.

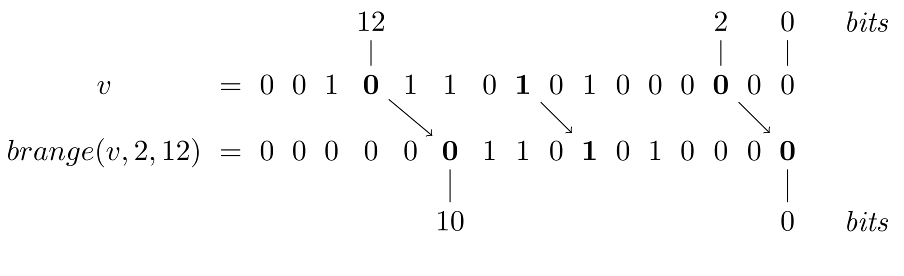
Modify the specification of the lexical analyser (`flex` input) and the syntactic analyser (`bison` input) and any other source file required to extend the `Lance` language with the operator **brange**. The semantic of this operation is to extract any bit-sequence specified by the two indices *lo* and *hi* that respectively indicate the index of the lowest and the highest bits to be extracted during the operation. Note that the least significant bit is indexed with 0, while the most significant bit is indexed with 31.
Indeed if it **does not hold** that $lo \leq hi \wedge lo \geq 0 \wedge hi \leq 31$, the result must be **zero**.
An example is provided in the following.

```
int v, r;

// let assume that v = 11584
r = 42 + brange(v, 2, 12);

// r = 42 + 848 = 890
write(r);
```

The operator syntax is `brange(value, lo, hi)`. In the example, `brange(v, 2, 12)` extracts the bits in the range $[2, 12]$ from the value v. Assuming that $v = 11584$ we have that



12
2
0
bits
v = 0 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0
brange(v, 2, 12) = 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 0
10
0
bits

Note that **brange** operator can take a generic expression for any of its operands.

[1]Time 60'. Textbooks and notes can be used.
Pencil writing is allowed. Write your name on any additional sheet.

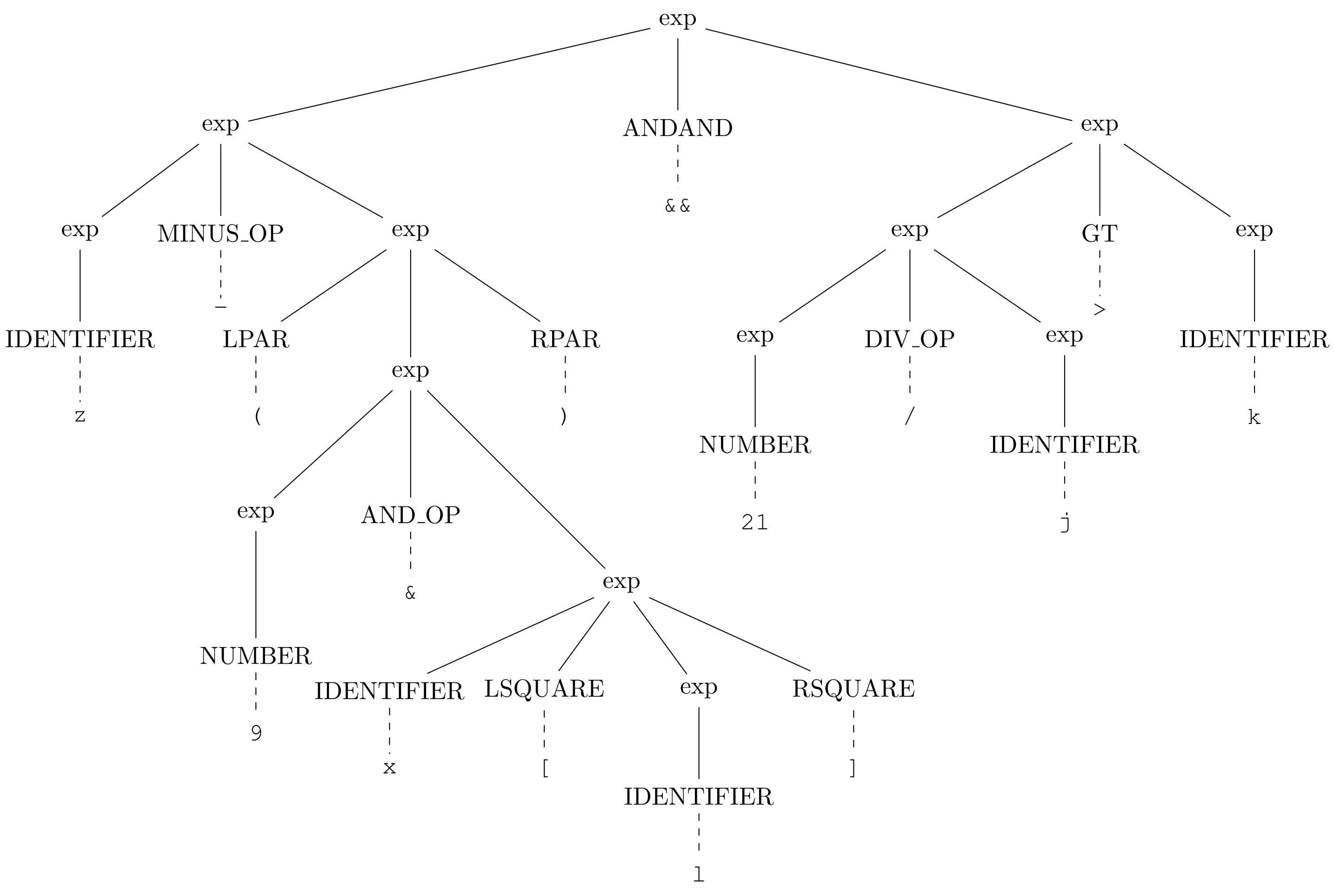
1. Define the tokens (and the related declarations in **Acse.lex** and **Acse.y**). (1 points)
2. Define the syntactic rules or the modifications required to the existing ones. (4 points)
3. Define the semantic actions needed to implement the required functionality. **In particular, take care of implementing the minimum number of runtime checks needed to validate the range of bits, depending of the knowledge of the compile-time constants. Indeed whenever possible, fold the computation at compile-time.** (20 points)

   The solution is in the attached patch.

4. Given the following `Lance` code snippet:

```
z - (9 & x[l]) && 21 / j > k
```

write down the syntactic tree generated during the parsing with the Bison grammar described in Acse.y *starting from the farthest* `exp` *nonterminal.* (5 points)



5. (**Bonus**) Describe how to modify your solution to extend the **brange** operator in the case of $lo > hi \wedge lo >= 0 \wedge hi <= 31$, to produce the as result the sequence of bits from lo to hi wrapping-around when the most significant bit is reached.

```
int v, r;

// let
assume v = 2^31 + 2^25 + 2^2 + 2^0
r = brange(v, 31, 0)

// r = 2^1 + 2^0 = 3
write(r);
```