INJE08: Programming Paradigms
04IN1024: Programming Language Theory

Final WS 2014/15

Universität Koblenz-Landau, FB4
Prof. Dr. Ralf Lämmel
20.02.2015

Name, Vorname	
Matrikel-Nr.	
Studiengang	
ECTS (8 oder 6)	

Grading scheme

There are 10 questions with 0-2 points each: 2 points for a (mostly) correct and complete solution; 1 point for a somewhat reasonable solution with significant omissions or defects; 0 points otherwise. About 50 % of the points are needed to pass the exam. If you are a student enrolled according to the previous exam rules, then about (8/6)*50 % of the points are needed to pass the exam.

Contents

The topics of the exam are published with the dry run and reused for the actual final and the resit as well. Of course, the topics are chosen to cover broad subjects areas. The actual questions are focused on specific competences that were emphasized in the lecture and exercised in the lab—in the edition of the course at hand.

1 Topic: *Abstract syntax*

Consider an imperative language with statements as follows:

- assign a number literal a variable,
- increment a variable,
- sequence of two statements,

Notation: Define the syntax in an appropriate grammar notation. You may use Prolog or Haskell, too.

Reference solution

If BNF-like notation was used:

statement	=	*variable* '=' *number*
	|	*variable* '--'
	|	*statement* ';' *statement*
number		— not further defined here
variable		— not further defined here

2 Topic: *Operational semantics*

Define a big-step operational semantics for the language of the previous task. You do not need to define routine helpers for memory look-up and modification; just assume them. *Notation: Define the semantics in an appropriate operational semantics notation. You may use Prolog or Haskell, too.*

Reference solution

```
execute(assign(V, N), M1, M2) :−
  update(M1, V, N, M2).
execute(inc(V), M1, M2) :−
  lookup(M1, V, N0), N1 is N0 + 1, update(M1, V, N1, M2).
execute(seq(S1, S2), M1, M3) :−
  execute(S1, M1, M2), execute(S2, M2, M3).
```

3 Topic: *Type systems*

Here is a language. It has types *Number* and *String*. There are basic expression forms for literals (constants) for numbers and strings. There are operations for binary addition both on numbers and strings. However, binary subtraction is only defined on numbers. Define a type system for this sort of language. *Notation: Define the judgment in a notation appropriate for well-formedness or type systems. You may use Prolog or Haskell, too.*

Reference solution

```
typeOf(number(_), numberType).
typeOf(string(_), stringType).
typeOf(add(E1, E2), T) :−
  typeOf(E1, T), typeOf(E2, T).
typeOf(sub(E1, E2, numberType) :−
  typeOf(E1, numberType), typeOf(E2, numberType).

% A more flexible scheme for add is feasible as well, e.g.:
typeOf(add(E1, E2), numberType) :−
  typeOf(E1, numberType), typeOf(E2, numberType).
typeOf(add(E1, E2), stringType) :−
  typeOf(E1, numberType), typeOf(E2, stringType).
typeOf(add(E1, E2), stringType) :−
  typeOf(E1, stringType), typeOf(E2, numberType).
typeOf(add(E1, E2), stringType) :−
  typeOf(E1, stringType), typeOf(E2, stringType).
```

4 Topic: *Lambda calculi*

Present a simple example of substitution $[N/x]M$ (with M the term in which to substitute, x the variable to substitute, and N the term to replace x) that requires alpha conversion.

Reference solution

$[z/x](\lambda\ z.\ x\ z)$

5 Topic: *Type safety*

Explain the notion of preservation. *Please, be concise: 140 characters or less.*

Reference solution

The notion defines a property for a combination of type system and operational semantics. Preservation holds if any reduction result (big step) or any transition result (small step) is of the 'same' type as the initial term.

6 Topic: *Polymorphism*

Consider the syntax of System F.

$$t ::= x \mid v \mid t\ t \mid t[T]$$
$$v ::= \lambda x : T.t \mid \Lambda X.t$$
$$T ::= X \mid T \to T \mid \forall X.T$$

The small lambda abstracts over lambda terms. The big lambda abstract over lambda-term types. What is a noticable difference otherwise between the two abstraction forms? *Please, be concise: 140 characters or less.*

Reference solution

Small lambda abstractions are constrained by the type of terms. Big lambda abstractions are not constrained like this; they can bind any type.

7 Topic: *Object orientation*

Consider the following rules of FJ's dynamic semantics:

$$\frac{e_0 \mapsto e_0'}{e_0.f \mapsto e_0'.f}$$

$$\frac{e_0 \mapsto e_0'}{e_0.m(\underline{e}) \mapsto e_0'.m(\underline{e})}$$

$$\frac{e_0 \text{ value} \quad \underline{e} \mapsto \underline{e'}}{e_0.m(\underline{e}) \mapsto e_0.m(\underline{e'})}$$

What do they express? *Please, be concise: 140 characters or less.*

Reference solution

The rules 'simplify' receivers of field access and method invocation as well as arguments of method invocation. (The rules do not say what the actual semantics of field access and method invocation is.)

8 Topic: *Denotational semantics*

Consider the following equation of an if-statement of the imperative language 'While':

$$\mathcal{S}_{ds}[\texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2] = \mathsf{cond}(\mathcal{B}[b],\, \mathcal{S}_{ds}[S_1],\, \mathcal{S}_{ds}[S_2])$$

Explain how this equation meets 'compositionality'. *Please, be concise: 140 characters or less.*

Reference solution

Compositionality is met as the meaning only refers to the meaning of direct constituents b, S_1, and S_2 as opposed to (the meaning of) any made-up syntactic phrases.

9 Topic: *Axiomatic semantics*

Consider the following formulae as they may appear in pre- and postconditions of triples:

- $a > b \;\&\&\; b > c$
- $b < a \;\&\&\; b > c$

These formulae are logically but not syntactically equivalent. Describe a predicate that finds indeed that these formulae are equivalent. You may use Prolog or otherwise relatively formal explanations.

Reference solution

One could use a predicate like this:

```
% Syntactic equality
equiv(X, X).
% '>' versus '<'
equiv(X>Y, Y<X).
% Distribution of equiv/2 over conjunction
equiv((X1,Y1), (X2,Y2)) :- equiv(X1, X2), equiv(Y1, Y2).

/*

% Illustration

?- equiv((a>b,b>c),(b<a,b>c)).
true .

*/
```

10 Topic: *Concurrency*

Consider the transition rules for CCS' composition operator:

- Com_1 $\quad \dfrac{E \xrightarrow{\alpha} E'}{E|F \xrightarrow{\alpha} E'|F}$
- Com_2 $\quad \dfrac{F \xrightarrow{\alpha} F'}{E|F \xrightarrow{\alpha} E|F'}$
- Com_3 $\quad \dfrac{E \xrightarrow{l} E' \quad F \xrightarrow{\bar{l}} F'}{E|F \xrightarrow{\tau} E'|F'}$

The completed (perfect) action τ clearly may happen in accordance with the last rule. However, it could also happen in accordance to the first or second rule. How? *Please, be concise: 140 characters or less.*

Reference solution

E in the first rule could be of the form $E'|F'$ such that the third rule applies to E.