INJE08: Programming Paradigms
04IN1024: Programming Language Theory

Dry run WS 2014/15

Universität Koblenz-Landau, FB4
Prof. Dr. Ralf Lämmel
21.01.2015

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 a simple expression language with constructs as follows:

- number literals (whose structure can be ignored here),
- variable identifiers (whose structure can be ignored here),
- binary, infix additions such as '$41 + 1$',
- unary, prefix negation such as '-42'.

Define the syntax of such expressions using BNF-like notation or Prolog.

Reference solution

If BNF-like notation was used:

$$
\begin{array}{lcl}
\textit{expression} & = & \textit{number} \\
& | & \textit{variable} \\
& | & \textit{expression}\ '+'\ \textit{expression} \\
& | & '-'\ \textit{expression} \\
\textit{number} & & \text{— not further defined here} \\
\textit{variable} & & \text{— not further defined here}
\end{array}
$$

2 Topic: *Operational semantics*

Identify the normal forms (values) for the expression forms of the previous question, if you were to assume a small-step semantics.

Reference solution

The only normal form is the number form.

3 Topic: *Type systems*

Imagine you are providing a type system for a given programming language. Obviously, you need to provide the typing rules, eventually. How can you decompose the process of providing the type system? Name 2+ activities to be performed or entities to be defined.

Reference solution

For example:

- The syntax of types needs to be defined.
- The context of the typing rules needs to be defined.
- One judgment per syntactic category has to be defined.
- One rule per syntactic construct has to be provided.
- The proof of type safety has to be carried out.

4 Topic: *Lambda calculi*

Use a concise argument to show that self-application is not typeable in the simply-typed lambda calculus, i.e., no T can be found so that $\lambda\, x : T.\ x\, x$. Your argument should refer to the following typing rule of application:

$$\textsf{T-Application}\quad \frac{\Gamma \vdash t : U \to T \qquad \Gamma \vdash u : U}{\Gamma \vdash t\ u : T}$$

Reference solution

No T can be found because x would need to be of two different (nonunifiable) types: according to the first premise, x (i.e., t in the rule) would need to be of a function type; according to the second premise, x (i.e., u in the rule) would need to be of the argument type of said function type.

5 Topic: *Type safety*

Describe a very simple scenario in which a type system and semantics fail to collaborate in type safety.

Reference solution

For instance, the type system could claim type *bool* for the result of an operator application, whereas the semantics would compute a value of type *nat* for the same application.

6 Topic: *Polymorphism*

Consider the following typing rule of System F.

$$\frac{}{(\lambda X.t)[T] \to [T/X]t}\ \text{E-TypeAppAbs}$$

What does it express? (Please, be very brief.)

Reference solution

A type abstraction $\Lambda X.t$ is applied to an actual type T. A substitution is performed so that the variable X of the type abstraction is replaced by T within the lambda term t, where X may occur within types of lambda abstractions.

7 Topic: *Object orientation*

Consider the following rules of FJ's type system:

$$\texttt{C} <: \texttt{C} \qquad \frac{\texttt{C} <: \texttt{D} \quad \texttt{D} <: \texttt{E}}{\texttt{C} <: \texttt{E}} \qquad \frac{\texttt{class C extends D \{...\}}}{\texttt{C} <: \texttt{D}}$$

What do they express? (Please, be very brief.)

Reference solution

The rules define the subtyping relation. The rule on the left models reflexivity of subtyping: each type is a subtype of itself. The rule in the middle models transitivity. The rule on the right models the fact that a class C being declared explicitly to be a subclass of another class D implies that C is indeed a subtype of D.

8 Topic: *Denotational semantics*

Consider the following equation, as if it was part of a denotational semantics:

$$\mathit{Meaning}[\mathsf{while}\ b\ \mathsf{do}\ S] = \mathit{Meaning}[\mathsf{if}\ b\ \mathsf{then}\ S;\ \mathsf{while}\ b\ \mathsf{do}\ S\ \mathsf{else}\ \mathsf{skip}]$$

Explain how this equation fails to meet 'compositionality', which is a requirement for any equation of a denotational semantics description.

Reference solution

The meaning of a while loop is *not* defined in terms of the meanings of the constituents of the while loop. Instead, the meaning is defined in terms of a new syntactic phrase that also repeats the while loop as part of it. A denotational semantics is supposed to define an equational system over the syntactical constituents of a given program. When compositionally is violated, then there is no guarantee that the equational system can be solved (uniquely).

9 Topic: *Axiomatic semantics*

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

- $a > b \;\&\&\; b > c$
- $!(a <= b) \;\&\&\; b > c$

These formulae are logically equivalent. Describe (informally) a rewrite rule so that both formulae would be syntactically equivalent by means of normalization.

Reference solution

The rewrite rule should eliminate the use of negation ("!") at hand. Formally, the rewrite rule looks like this: $!(x <= y) \rightarrow x > y$. That is, negation on top of '<=' can be replaced by '>' without negation.

10 Topic: *Concurrency*

Consider again 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 $\quad \dfrac{E \xrightarrow{l} E' \quad F \xrightarrow{\bar{l}} F'}{E|F \xrightarrow{\tau} E'|F'}$

Even if two processes E and F could communicate (see the third rule), then transitions without communication (see first and second rules) are still feasible. Why is that?

Reference solution

The communication between E and F, if possible, is just an option. The composition of E and F may be part of a 'bigger' process, where other options for communication may exist. CCS' restriction though may rule out the actions for the transitions of E and F to escape.