PRIFYSGOL CYMRU; UNIVERSITY OF WALES

DEGREE EXAMINATIONS JANUARY 2003

SWANSEA

Computer Science

CS 423 Algebraic Specification of Software and Hardware

Attempt 2 questions out of 3

Time allowed: 2 hours

Students are permitted to use the dictionaries provided by the University

Students are NOT permitted to use calculators

CS _423
ALGEBRAIC SPECIFICATION OF SOFTWARE AND HARDWARE

(Attempt 2 questions out of 3)

Question 1

(a) Let ϕ: X $\rightarrow$ Y and f: A $\rightarrow$ B be functions. Using equations and commutative diagrams, give *four* ways of to formalise the idea that

"ϕ is an abstract or high-level version of f"

using mappings α and β that represent the data in X and Y by data in A and B, respectively. Show there are precisely four ways.

[5 marks]

(b) Define the ideas of *right inverse* and *left inverse* to a map and explain their connection with injections and surjections.

[4 marks]

(c) Define what is meant by a *retiming* of two discrete clocks T and R.

Which of the following maps λ: T $\rightarrow$ R is a retiming? For any map that is a retiming give its *kernel* $\equiv_\lambda$, its *factor set* T/$\equiv_\lambda$, and a section λ*:

$\lambda(t) = 10t$;
$\lambda(t) = \lfloor \log_2(t) \rfloor$;
$\lambda(t) = 2^t$;
$\lambda(t) = \lfloor t/25 \rfloor$.

[7 marks]

(d) Explain the ideas of (i) *digital data* and (ii) *analogue data*.

Show how to define the concept of an *digital data type* using homomorphisms.

[6 marks]

(e) Use one of the equational methods of formalising hierarchical structure in part (a) to define the correctness of a compiler

$$\text{Compile:Prog}_{\text{Source}} \rightarrow \text{Prog}_{\text{Target}}$$

under input-output semantics over state spaces $\text{State}_{\text{Source}}$ and $\text{State}_{\text{Target}}$.

[3 marks]

Question 2

(a) Consider the following two retimings:

(1) $\lambda : \mathbf{N}^+ \rightarrow Ret(S,T)$

$\lambda(a)(s) = \lfloor s/a \rfloor$

(2) $\kappa : \mathbf{N}^+ \rightarrow Ret(S,T)$

$\kappa(a)(s) = log_a(s)$

Why is retiming λuniform but κ not? Write down a definition of λ in terms of a *duration function*. **[5 marks]**

(b) Consider the following simple initialised iterated map, intended to enumerate odd numbers*:*

$F : T \times \mathbf{N} \rightarrow \mathbf{N}$

$F(0,a) = h(a)$

$F(t+1,a) = F(t,a) + 2$

If we define *h* as $h(a)=1$, then *F* is not time-consistent. If we define *h* as

$$h(a) = \begin{cases} a, & \text{if } a \text{ odd;} \\ a+1, & \text{otherwise} \end{cases}$$

then *F* is time-consistent. Why is this? **[5 marks]**

(c) Formally state the *one-step theorems*, that can eliminate induction from the verification of microprocessors under specific conditions. Your answer should also include a statement of the conditions. **[10 marks]**

(d) Explain informally why the one-step theorems work. **[5 marks]**

Question 3

(a) Term rewriting is the basis of Maude and our methods for modelling and verifying microprocessor descriptions. Explain what it means for a set of rewrite rules to be:

- Terminating and Confluent.
- Terminating but not Confluent.
- Neither Terminating nor Confluent.

We commonly wish to translate sets of equations into rewrite rules. Give examples of simple and common properties, easily represented using equations, which require special treatment with rewrite rules. **[5 marks]**

(b) Explain how *membership axioms* in Maude make it possible to define a range of sorts representing fixed-length sequences of bits, given an algebra defining the behaviour of arbitrary bit strings. Give examples in your answer. **[5 marks]**

(c) The following Maude code is used to verify the correctness of the implementation of a microprocessor against some specification, in conjunction with another module `GET-CASE` (omitted) that includes the operations getCase and AddEquation. In addition, each case operator is equationally defined.

- Explain how the code works, paying particular attention to `CheckEquals` and its subfunctions.
- Explain how *conditional statements* must be written in the description of *the implementation* of a microprocessor in order for the code to work.
- Explain the major weakness of this method.

[15 marks]

```
fmod META-CASE is
    protecting META-LEVEL .
    protecting GET-CASE .

    op CORR : -> Module .
    eq CORR = (fmod 'CORR is
              including 'CORRECT .
              sorts none .
              none none none none none
         endfm) .

    vars T1 T2 C1 C2 C3 : Term .
    var TL : TermList .
    var MOD : Module .

    op CheckEquals : Module Term Term -> Bool .
    op CheckEqualsAux : Module Term Term -> Bool .
    op CheckEqualsAux2 : Module Term Term Term -> Bool .

    eq CheckEquals(MOD, T1, T2) =
         CheckEqualsAux(MOD, T1, meta-reduce(MOD, T2)) .

    eq CheckEqualsAux(MOD, T1, T2) =
         CheckEqualsAux2(MOD, T1, T2, getCase(T2)) .

    eq CheckEqualsAux2(MOD, T1, T2, error*) =
         meta-reduce(MOD, T1) == T2 .

    eq CheckEqualsAux2(MOD, T1, T2,
      'case[C1,'_:_[C2,C3]])
         = CheckEquals(AddEquation(MOD,C1,C2), T1, T2) .

    eq CheckEqualsAux2(MOD, T1, T2,
      'case[C1,'_:_[C2,C3],TL])
         = CheckEquals(AddEquation(MOD,C1,C2), T1, T2)
       and CheckEqualsAux2(MOD, T1, T2, 'case[C1,TL]) .
endfm
```