PRIFYSGOL CYMRU; UNIVERSITY OF WALES

DEGREE EXAMINATIONS MAY/JUNE 2002

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 $\phi: X \to Y$ and $f: A \to B$ be functions. Give *four* ways of using equations to formalise the idea that "ϕ is an abstract or high-level version of f".

[8 marks]

(b) Define a *retiming* of two discrete clocks T and R. Which of the following maps $\lambda: T \to R$ is a retiming? For each retiming derive its *kernel* $\equiv_\lambda$, its *factor set* $T/\equiv_\lambda$, and a section:

(i) $\lambda(t) = 2t$;
(ii) $\lambda(t) = \lfloor t/5 \rfloor$;
(iii) $\lambda(t) = \lfloor \sqrt{t} \rfloor$;
(iv) $\lambda(t) = t^2$

[10 marks]

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

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

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

[2 marks]

(d) *Sketch briefly* a recursive compiler that translates a simple parallel language

$$S ::= x_1, \ldots, x_k := e_1, \ldots, e_k \mid S_1; S_2 \mid \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi},$$

based on the concurrent assignment, into a language for a sequential virtual machine. In what sense is the compiler an "algebraic" transformation?

[5 marks]

Question 2

(a) Formally state what it means for an iterated map to be *time-consistent*. What condition must the initialisation function of an initialised iterated map G satisfy if G is to be time-consistent? **[5 marks]**

(b) What is a *uniform retiming?* Show how we would typically define a uniform retiming using a *duration function* . **[5 marks]**

(c) Formally state the *one-step theorems*, that eliminate induction from the verification of microprocessors under certain circumstances. **[10 marks]**

(d) Informally, explain why the one-step theorems work. You may find diagrams helpful. **[5 marks]**

Question 3

(a) State what it means for a term rewriting system to be (1) confluent and (2) terminating. If we convert systems of equations into rewrite rules, (1) and (2) often do not hold. Explain why violations of (1) and (2) can occur in this situation. **[5 marks]**

(b) *Membership axioms* in Maude allow us to conveniently define sorts consisting of fixed-length strings of bits. Write the appropriate fragments of Maude code to define 8-bit bytes. You can assume the presence of a module defining *Bit* (single bit words) and *Bits* (arbitrary length bit strings). Why would it be a mistake to use the following conditional membership axiom:

```
cmb A : Byte if | B | == 8 .
```

[5 marks]

continued overleaf

(c) The following is part of the meta-level code used to verify simple example processors. Explain how this code works. **[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) .

    op AddEquation : FModule Term Term -> FModule .

    var QI : Qid .
    var IL : ImportList .
    var SD : SortDecl .
    var SSDS : SubsortDeclSet .
    var ODS : OpDeclSet .
    var VDS : VarDeclSet .
    var MAS : MembAxSet .
    var EqS : EquationSet .
    vars T1 T2 C1 C2 C3 : Term .
    var TL : TermList .
    var MOD : Module .

    eq AddEquation((fmod QI is IL SD SSDS ODS VDS MAS
       EqS endfm), T1,T2) =
         fmod QI is IL SD SSDS ODS VDS MAS
         ((eq T1 = T2 .) EqS) endfm .

    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
```