TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI

VICTORIA

UNIVERSITY OF WELLINGTON

EXAMINATIONS — 2010

END-OF-YEAR

SWEN 224
Formal Foundations of Programming
WITH ANSWERS

Time Allowed: 3 Hours

Instructions:

- Answer all seven questions.
- The exam will be marked out of one hundred and eighty (180).
- Calculators ARE NOT ALLOWED.
- Non-electronic Foreign language dictionaries are allowed.
- No other reference material is allowed.

Question 1. Static Alloy Modelling [20 marks]

Consider the following Alloy specification, which models family relations.

```
abstract sig Person {
  father: lone Man,
  mother: lone Woman
}

sig Man extends Person {
  wife: lone Woman
}

sig Woman extends Person {
  husband: lone Man
}

fun parent: Person -> Person {
  mother + father + father.wife + mother.husband
}

fun grandparent: Person -> Person {
  parent.parent
}

pred p {
  some p: Person | p in p.grandparent
}
```

(a) Understanding an Instance

Consider the following instance of this model:

$$\texttt{Man} = \{Adam, Bob\}$$

$$\texttt{Woman} = \{Ana, Jane\}$$

`husband` =

Ana	*Bob*
Jane	*Adam*

`wife` =

Bob	*Ana*
Adam	*Jane*

`father` =

Adam	*Bob*

`mother` =

Ana	*Jane*

(i) [1 mark] Compute `Adam.wife`

Jane

(ii) [1 mark] Compute `~wife`

Ana	*Bob*
Jane	*Adam*

(iii) [2 marks] Compute `parent`

Ana	*Adam*
Ana	*Jane*
Adam	*Bob*
Adam	*Ana*

(iv) [2 marks] Compute `Adam.grandparent`

Adam
Jane

(v) [2 marks] In your own words, describe what predicate p expresses.

There is a person who is their own grandparent.

(vi) [2 marks] Is the predicate `p` true for this instance? Briefly explain why or why not.

Yes, the predicate is true since Adam is his own grandparent.

(b) Writing Alloy

(i) [1 mark] Provide a run command that shows instances with at least one woman.

```
run { some Woman }
```

(ii) [2 marks] Provide a run command that shows instances with at least one woman who is the wife of someone.

```
run { some wife }
```

(iii) [2 marks] Write a function called `spouse` that takes a person as argument and returns the spouse (that is, the wife if the person is male or the husband if the person is female) of the given person.

```
fun spouse[p: Person]: lone Person { p.(husband+wife) }
```

(iv) [3 marks] Write a check command to check whether every person who is a spouse of someone has a spouse. Provide a counter-example that could have been generated if your check command was added to the above model and executed.

```
check { all p: Person | (some s: Person | p = spouse[s]) implies some spouse[p] }
```

(v) [2 marks] Add a fact to the specification to ensure that every person with a spouse is the spouse of his or her spouse.

```
fact { husband = ~wife }
```

Question 2. Dynamic Alloy Modelling [25 marks]

The following Alloy provides a dynamic model for the marriage relation.

```
abstract sig Person {}
sig Man, Woman extends Person {}

sig State {
  husband: Woman -> Man,
  wife: Man -> Woman
}

pred divorce(s,s’: State, m: Man, w: Woman) {
  s’.husband = (Woman-w) <: s.husband
  s’.wife = (Man-m) <: s.wife
}
```

(a) Understanding an Instance

Consider the following instance of this model:

$$\texttt{Man} = \{Adam, Bob\}$$

$$\texttt{Woman} = \{Ana, Jane\}$$

$$\texttt{State} = \{S0, S1, S2\}$$

husband =

S1	Ana	Adam
S2	Ana	Adam
S2	Ana	Bob

wife =

S2	Adam	Ana
S2	Bob	Ana

(i) [3 marks] For each of the states $S0$, $S1$, and $S2$ explain who is husband or wife to whom.

In state 0, nobody is married. In state 1, Ana is married to Adam but Adam is not married so this is not a valid state. In state 2, Ana is married both to Adam and Bob, which is not valid either.

(ii) [3 marks] Compute `(Man-Bob) <: S2.wife`

Adam	Ana

(iii) [3 marks] Explain the role of the `divorce` predicate and its arguments.

(iv) [2 marks] Can the `divorce` predicate be satisfied in the given instance? If it can, identify the arguments for which the `divorce` predicate is true.

```
divorce[S1,S0,Adam,Ana]
```

(b) Extending the Alloy Model

(i) [4 marks] Provide a predicate called `inv` that takes a state as argument and is true if, for the given state, every man has at most one wife and every woman has at most one husband.

```
pred inv[s: State] {
  all m: Man | lone s.wife[m]
  all w: Woman | lone s.husband[w]
}
```

(ii) [6 marks] Provide an operation `marry` that models a man and a woman getting married and preserves the invariant.

```
pred marry[s,s': State, m: Man, w: Woman] {
  no s.husband.m + s.husband[w] + s.wife.w + s.wife[m]
  s'.husband = s.husband + w->m
  s'.wife = s.wife + m->w
}
```

(iii) [4 marks] Write an Alloy command to check whether the `marry` operation you provided in **(ii)** preserves invariant `inv`.

```
check { all s,s': State, m: Man, w: Woman |
  inv[s] and marry[s,s',m,w] implies inv[s']
} for 10
```

Question 3. JML [25 marks]

(a) Understanding JML

Do the following methods correctly implement their specification? Give a brief explanation why you think they do or do not.

(i) [2 marks]

```
//@ requires true;
//@ ensures 0 <= \result && \result <= 50;
int magicNumber1() { return 42; }
```

Yes, since the method always satisfies the postcondition by returning a value within the range 0 upto 50.

(ii) [2 marks]

```
//@ requires x == 0;
//@ ensures \result == 41 || \result == 42;
int magicNumber2(int x) { return 42; }
```

Yes, since the method always satisfies the postcondition by returning a value that is equivalent to 41 or 42.

(iii) [2 marks]

```
//@ requires false;
//@ ensures false;
int magicNumber3() { return 42; }
```

Yes, since the method only needs to satisfy the postcondition if the precondition has been satisfied by the caller but the precondition cannot be satisfied.

(iv) [2 marks]

```
//@ requires a != null;
//@ ensures 0 <= \result && \result < a.length;
//@ ensures a[\result] == value;
int find1(int[] a, int value) {
  a[0] = value;
  return 0;
}
```

No, since if the array a is empty, an exception will be thrown.

(v) [2 marks]

```
//@ requires a != null;
//@ ensures 0 <= \result && \result <= a.length;
//@ ensures a[\result] == value;
int find2(int[] a, int value) {
  for (int i = 0; i < a.length; i++) {
      if (a[i] == value) return i;
    }
    return a.length;
  }
```

No, since if the value is not in a, the second postcondition cannot be satisfied.

(b) Writing JML

Write a JML specification for each of the following methods.

(i) [3 marks] `int max(int x, int y)`

Method `max` returns the maximum of the two given integers.

```
//@ ensures \result == x || \result == y;
//@ ensures \result >= x;
//@ ensures \result >= y;
```

(ii) [5 marks] `int[] replace(int[] a, int x, int y)`

Given integers `x` and `y`, and a non-null integer array `a`, the array returned by method `replace` is the same as `a` with all occurrences of `x` replaced by `y`.

```
//@ requires a != null;
//@ ensures \result != null && \result.length == a.length;
/*@ ensures (\forall int i; 0 <= i && i<a.length;
       (a[i] == x && \result[i] == y) || (a[i] != x && \result[i]==a[i])); */
```

(c) Class Invariants

Consider the following Java class to represent a bank account.

```
public class Account
{
  //@ invariant amount >= 0;
  private int amount;

  public void deposit(int value) {
    amount += value;
  }

  public void withdraw(int value) {
    amount -= value;
  }
}
```

(i) [3 marks] Explain what the JML invariant clause means.

(ii) [2 marks] Does the `deposit` method preserve the given JML invariant? Give a brief explanation why you think it does or does not.

No, it does not preserve the invariant since `value` could be negative and $|amount| < |value|$. In this case, `amount + value < 0`.

(iii) [2 marks] The `withdraw` method does not preserve the given JML invariant. Provide a JML annotation that ensures that the `withdraw` method preserves the given JML invariant.

```
//@ requires amount - value >= 0;
```

Question 4. Loop Invariants and Variants [25 marks]

Consider the following Java method:

```
boolean test(String s) {
  int k = 0;
  int n = s.length()/2;
  while ( k < n && s.charAt(k) == s.charAt(n+k) ) {
    k = k+1;
  }
  return ( k == n );
}
```

Given a string, s, of even length, this method determines whether s is the concatenation of two copies of the same string. For example, `test` will return `true` if s is `"abcabc"` or `"xx"` and false if s is `"abba"`.

(a) [5 marks] Give a JML specification (precondition and postcondition) that formalises the above description of the `test` method.

```
//@ requires s != null;
//@ requires s.length()%2 == 0;
/*@ ensures
    \result <==>
    (\forall int i; 0 <= i && i < s.length()/2;
                    s.charAt(i) == s.charAt(s.length()/2+i)); @*/
```

Note that `==` can be used in place of `<==>`. The postcondition can also be expressed as `\result <==> s.substring(0, s.length()/2).equals(s.substring(s.length()/2))`.

(b) [12 marks] Give a loop invariant and use it to give an argument (informal proof) that the method satisfies its specification.

```
//@ loop_invariant s != null;
//@ loop_invariant n == s.length()/2;
//@ loop_invariant 0 <= k && k <= n;
//@ loop_invariant (\forall int i; 0 <= i && i < k; s.charAt(i) == s.charAt(n+i));
```

We must show (i) that the loop invariant holds on entry to the loop, (ii) that the loop invariant is preserved by the loop body when the loop test holds, and (iii) that the loop invariant implies the postcondition when the loop exits (i.e. when the loop test fails).

(i) At the beginning of the loop, we know that `s != null` from the method's precondition, and that `k == 0` and `n == s.length()/2`, from the effects of the assignments to `k` and `n`.

Thus, `s != null` and `n == s.length()/2` obviously hold.

`0 <= k && k <= n` holds, because `k == 0` and `n == s.length()/2`, since `s.length() >= 0` for any string `s`.

`\forall int i; 0 <= i \&\& i < k; s.charAt(i) == s.charAt(n+i))` holds, because `k == 0`, since `(\forall int i; 0 <= i && i < k; C)` holds for any condition `C`.

(ii) `s != null` and `n == s.length()/2` are obviously preserved since `s` and `n` are not changed in the loop.

`0 <= k` is preserved, since `0 <= k` holds at the beginning of the loop body and `k` is increased (more formally, `0 <= k` implies `0 <= k+1`).

`k <= n` is preserved, since `k < n` when the loop test succeeds (more formally, `k < n` implies `k+1 <= n`).

`(\forall int i; 0 <= i && i < k; s.charAt(i) == s.charAt(n+i))` is preserved since `s.charAt(k) == s.charAt(n+k)` holds when the loop test succeeds (more formally, `(\forall int i; 0 <= i && i < k; s.charAt(i) == s.charAt(n+i))` and `s.charAt(k) == s.charAt(n+k)` imply `(\forall int i; 0 <= i && i < k+1; s.charAt(i) == s.charAt(n+i))`).

(iii) We need to show that `k == n <==> (\forall int i; 0 <= i \&\& i < s.length()/2; s.charAt(i) == s.charAt(s.length()/2+i))` holds when the loop exits, `k == n` is the value returned by the method.

When the loop exits, we know that the loop invariant holds, and, because the loop test fails, either `!(k < n)` or `!(s.charAt(k) == s.charAt(n+k))`.

If `!(k < n}`, we must have `k == n`, since the loop invariant implies `k <= n`. The loop invariant also tells us that `n == s.length()/2`, which means that `k == s.length()/2`. Substituting this in `(\forall int i; 0 <= i \&\& i < k; s.charAt(i) == s.charAt(n+i))`, gives `(\forall int i; 0 <= i && i < s.length()/2; s.charAt(i) == s.charAt(s.length()/2+i))`. Thus `k == n <==> (\forall int i; 0 <= i &&i < s.length()/2; s.charAt(i) == s.charAt(s.` holds, since both sides are `true`.

If `!(s.charAt(k) == s.charAt(n+k))` holds, then `k < n` must hold (otherwise `s.charAt(k) == s.charAt(n+k))` would not have been evaluated), and so `k == n` is `false`. Also, `(\forall int i; 0 <= i && i < s.length()/2; s.charAt(i) == s.charAt(s.length()/2+i))` must be `false`, since `s.charAt(k) == s.charAt(n+k)`. Thus, `k == n <==> (\forall int i; 0 <= i \&\& i < s.length()/2; s.charAt(i) == s.charAt(s.length()/2+i))` holds, since both sides are `false`.

(c) [8 marks] Give a loop variant and an argument (informal proof) to show that the method terminates.

```
//@ decreasing n - k;
```

Since `0 <= k && k <= n` and `k` is increased by the loop body, `n-k` must decrease each time the loop body is executed, and the loop must exit when `n-k` because `0` (because of the condition `k < n`) if not before (because of the condition `s.charAt(k) == s.charAt(n+k)`.

Also note that the precondition `s != null` ensures that `s.length()%2` is well defined, and that the conjuncts `s != null`, `s.length()%2 == 0`, `n == s.length()/2` and `0 <= k && k <= n` of the loop invariant ensure that the operations `s.charAt(k)` and s.charAt(n+k) are well defined, so the method does not give a run-time error or exception.

(Proper termination also relies on the fact that in Java, when evaluating the loop test, when `k < n` evaluates to `false`, the second conjunct, `s.charAt(k) == s.charAt(n+k))` is not evaluated. Otherwise, evaluating `s.charAt(n+k))` would give an error when `k == n`.)

Note that the reasoning shown here (for all parts of the question) is more detailed that required as an exam answer.

Question 5. Properties of Strings and Languages [30 marks]

(a) [4 marks] A string s is a *prefix* of another string t, written $s \preceq t$, if t is the concatenation of s and some other string, say u; i.e. $s \preceq t \equiv \exists u : t = s \frown u$.

Show that if x is a prefix of y and y is a prefix of z, then x is a prefix of z (i.e. $x \preceq y$ and $y \preceq z$ implies $x \preceq z$).

If x is a prefix of y and y is a prefix of z, then there are strings, say u and v, such that $y = x \frown u$ and $z = y \frown v$. But this implies that $z = x \frown u \frown v$, so x is a prefix of z. That is, we can show that there is a string, say w, such that $z = x \frown w$, by taking $w = u \frown v$.

(b) [10 marks] Give a proof for each of the following equalities, where X and Y are languages over some alphabet A, and X^R is the *reflection* of X, i.e. $X^R = \{\alpha^R \mid \alpha \in X\}$:

(i) $(X \cup Y)^R = X^R \cup Y^R$

$$\begin{array}{ll}
(X \cup Y)^R & \\
= \{\alpha^R \mid \alpha \in X \cup Y\} & \text{Defn of } X^R \\
= \{\alpha^R \mid \alpha \in X\} \cup \{\alpha^R \mid \alpha \cup Y\} & \text{Defn of } \cup \\
= X^R \cup Y^R & \text{Defn of } X^R
\end{array}$$

Alternatively, let α be a string in A^*, then:

$$\begin{array}{ll}
\alpha \in (X \cup Y)^R & \\
\text{iff } \alpha^R \in X \cup Y & \text{Defn of } X^R \\
\text{iff } \alpha^R \in X \vee \alpha^R \in Y & \text{Defn of } \cup \\
\text{iff } \alpha \in X^R \vee \alpha \in Y^R & \text{Defn of } \cup \\
\text{iff } \alpha \in X^R \cup Y^R & \text{Defn of } X^R
\end{array}$$

(ii) $(X \frown Y)^R = Y^R \frown X^R$

$$\begin{array}{ll}
(X \frown Y)^R & \\
= \{\alpha^R \mid \alpha \in X \frown Y\} & \text{Defn of } X^R \\
= \{(x \frown y)^R \mid x \in X \wedge y \in Y\} & \text{Defn of } X \frown Y \\
= \{y^R \frown x^R \mid x \in X \wedge y \in Y\} & \text{Property of } \alpha^R \\
= \{y^R \mid y \in Y\} \frown \{x^R \mid x \in X\} & \text{Defn of } X \frown Y \\
= \{y \mid y \in Y^R\} \frown \{x \mid x \in X^R\} & \text{Defn of } X^R \\
= Y^R \frown X^R & \text{Defn of } X^R
\end{array}$$

Alternatively, let α be a string in A^*, then:

$$\begin{array}{ll}
\alpha \in (X \frown Y)^R & \\
\text{iff } \alpha^R \in X \frown Y & \text{Defn of } X^R \\
\text{iff } \alpha^R = x \frown y \text{ for some } x \in X \text{ and } y \in Y & \text{Defn of } X \frown Y \\
\text{iff } \alpha = y^R \frown x^R \text{ for some } y \in Y \text{ and } x \in X & \text{Property of } \alpha^R \\
\text{iff } \alpha \in Y^R \frown X^R & \text{Defn of } \frown
\end{array}$$

(iii) $(X^*)^R = (X^R)^*$

$$\begin{array}{ll}
(X^*)^R & \\
(\bigcup_{n\geq 0} X^n)^R & \text{Defn of } X^* \\
\bigcup_{n\geq 0}(X^n)^R & \text{By (i)} \\
\bigcup_{n\geq 0}(X^R)^n & \text{By (ii)} \\
= (X^R)^* & \text{Defn of } X^*
\end{array}$$

Alternatively, let α be a string in A^*, then:

$$\begin{array}{ll}
\alpha \in (X^*)^R & \\
\text{iff } \alpha^R \in X^* & \text{Defn of } X^R \\
\text{iff } \alpha^R \in X^n\text{, for some } n \geq 0 & \text{Defn of } X^* \\
\text{iff } \alpha^R = \alpha_1 \frown \cdots \frown \alpha_n\text{, for some } n \geq 0 \text{ and } \alpha_1, \cdots, \alpha_n \in X & \text{Defn of } X^n \\
\text{iff } \alpha = \alpha_n^R \frown \cdots \frown \alpha_1^R\text{, for some } n \geq 0 \text{ and } \alpha_1, \cdots, \alpha_n \in X & \text{By (ii)} \\
\text{iff } \alpha \in (X^R)^n\text{, for some } n \geq 0 \in (X^n)^R\text{, for some } n \geq 0 & \text{Defn of } X^n \\
\text{iff } \alpha \in (X^R)^* & \text{Defn of } X^*
\end{array}$$

In these proofs, you should explain each step and state the properties of sets and strings that they rely on, but you do not need to prove these properties.

(c) [5 marks] Explain how the results in part **(b)** above can be used to show that the class of regular languages is closed under reflection.

We need to show that a language X is regular iff its reflection X^R is regular. If X is regular, it can be defined by a regular expression, x. We will show by induction that X^R can also be defined by a regular expression.

For the base cases, if x is λ, ϕ or $a \in A$, then $x^R = x$, since $\lambda^R = \lambda$, $\phi^R = \phi$ and $a^R = a$.

For the inductive cases, assume that $y^R = y'$ and $z^R = z'$, for regular expressions y, y', z and z'. The above results show that if $x = y|z$, then $x^R = y'|z'$; if $x = y \frown z$, the $x^R = z' \frown y'$; and if $x = y^*$, then $x^R = y'^*$

(d) [6 marks] Explain how the set of strings represented by a *trie* can be obtained by solving a set of equations. Illustrate this result using a small example.

We determine the set of strings represented by a trie as follows. We associate a set of strings, $L(i)$, with each node, i, in the trie, such that $L(i)$ is the set of strings formed by collecting all of the symbols on the edges of a path from node i to a final node. Then $L(root)$ is the required set.

The values of L for adjacent nodes are related by a set of equations of the form:

$L(i) = \Lambda(i) \cup (\bigcup_{a \in A} \{a\} \frown L(N(i,a)))$

where: $\Lambda(i) = \{\lambda\}$ if i is a final node, and $\emptyset$ otherwise.

We can solve these equations to find $L(i)$ as follows. We first note that $L(i) = \{\lambda\}$ for any leaf node i (we assume that leaves are always final). Then repeatedly substitute the value of some $L(i)$ for which we have a value. If there are n nodes in the trie, this process will terminate after (at most) n steps, with the value of $L(root)$.

For example, if the trie has edges $\{(1,a,2),(1,b,4),(2,n,3),(4,e,5),(4,y,6)\}$, were 1 is the root and nodes 2, 3, 5 and 6 are final, we get equations (writing L_i instead of $L(i)$ for convenience):

$$L_1=\{a\} \frown L_2 \cup \{b\} \frown L_4$$
$$L_2=\{\lambda\} \cup \{n\} \frown L_3$$
$$L_3=\{\lambda\}$$
$$L_4=\{e\} \frown L_5 \cup \{y\} \frown L_6$$
$$L_5=\{\lambda\}$$
$$L_6=\{\lambda\}$$

Substituting for L_3, L_5 and L_6 gives:

$$L_2=\{\lambda\} \cup \{n\} \frown \{\lambda\} = \{\lambda, n\}$$
$$L_4=\{e\} \frown \{\lambda\} \cup \{y\} \frown \{\lambda\} = \{e, y\}$$

Substituting for L_2 and L_4 gives:

$$L_1=\{a\} \frown \{\lambda, n\} \cup \{b\} \frown \{e, y\} = \{a, an, be, by\}$$

(e) [5 marks] How is the result in part **(d)** above affected if we represent the set of strings by an NFA rather than a trie? Illustrate your answer using a small example.

If we have an NFA, rather than a trie, we can have multiple edges out of a node with the same label. This makes it a bit harder to give a generic description of the set of equations, but doesn't make them any harder to solve.

We can also have cycles, which does make the equations harder to solve, because they can be recursive, which means we can't just substitute them all out the way we can for a trie. However, we can always reduce our equations to a form:

$$L_i = X \frown L_i \cup Y$$

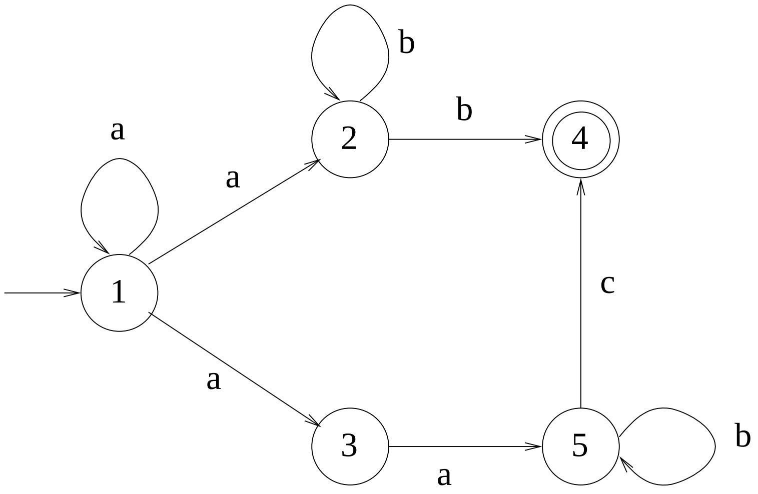
where X is the set of strings that ca be accepted on a path from i back to itself and Y is the set of strings that ca be accepted on a path from i to a final node. Any such equation has the solution:

$$L_i = X^* \cup Y$$

Example: See lecture notes.

Question 6. Finite Acceptors [30 marks]

(a) Consider the following NFA, M_1:



(i) [3 marks] Show the sequence of configurations that M_1 passes through in accepting the input *aaabc*. Note that you should show *all* states that M may be in after accepting part of the input.

States	*Input*
1	*aaabc*
$1,2,3$	*aabc*
$1,2,3,5$	*abc*
$1,2,3,5$	*bc*
$2,4,5$	*c*
4	λ

(ii) [3 marks] Write a regular expression describing the language accepted by M.

$a^*(ab^*b|aab^*c)$

(iii) [8 marks] Draw a transition diagram for the DFA obtained by applying the *subset construction* to M. You only need to show reachable states.

The edges are: $(\{1\},a,\{1,2,3\}),(\{1,2,3\},a,\{1,2,3,5\}),$
$(\{1,2,3\},b,\{2,4\}),(\{1,2,3,5\},a,\{1,2,3,5\}),$
$(\{1,2,3,5\},b,\{2,4,5\}),(\{2,4,5\},b,\{2,4,5\}),$
$(\{2,4,5\},c,\{4\}),(\{2,4\},b,\{2,4\})$

States $\{4\}$, $\{2,4\}$ and $\{2,4,5\}$ are final.

(b) [10 marks] Given an NFA $M = (Q, q_I, A, N, F)$, show how to construct an NFA, $M' = (Q', q'_I, A', N', F')$, which accepts the reflection of the language accepted by M (i.e. $\mathcal{L}(M') = \mathcal{L}(M)^R$).

Give a brief argument to show that the resulting NFA does in fact accept the required language.

We just need to reverse all edges, and the initial state of M the final state, and add a new initial state with null transistions to the final states of M'.

For formally, $M' = (Q', q'_I, A', N', F')$, where:

- $Q' = Q \cup \{q_N\}$, where q_N is a new state, not in Q.
- $q'_I = q_N$
- $A' = A$
- $N' = \{(q', a, q \mid (q, a, q') \in N\} \cup \{(q_N, \epsilon, q) \mid q \in F\}$
- $F' = \{q_I\}$

If a string $\alpha = a_0 \cdots a_n$ is accepted on a path $\langle q_0, \cdots, q_n \rangle$ in M, α^R is accepted on a path $\langle q_N, q_n, \cdots, q_1 \rangle$ in M', and vice versa.

(c) [6 marks] Given an NFA$_\epsilon$ $M = (Q, q_I, A, N, F)$ (i.e. an NFA with null transitions), describe an algorithm to determine whether M accepts the empty string.

We need to determine whether there is a path from q_I to a final state consisting only of null transitions. The required algorithm is a simple graph traversal, following only null transitions and looking for final states. We'll use two sets called V, to record the nodes that have been visited, and W, to record the nodes that are waiting to be visited.

```
$W := \{q_i\}$
While $W \neq \emptyset$ do
  Select a node $n$ from $W$ and remove it
  If $n$ is a final node, then return $true$
  else add $n$ to $V$ and add to $W$ any states in $N(n, \epsilon)$ but not in $V$
od
return $false$
```

Question 7. Context Free Grammars [25 marks]

(a) [5 marks] Given a grammar, $G = (V_N, V_T, S, P)$, explain what it means for a tree to be a *parse tree* for a string w from G.

A tree T is a parse tree for w from G if:

- The root of T is labelled with S, and every other non-leaf node is labelled with a nonterminal in V_N.
- The leaf nodes are labelled with terminals in V_T, and the fringe is w.
- For every non-leaf node with label N and children labelled $x_1, \cdots x_n$, $N \to x_1 \cdots x_n$ is a rule in P.

(b) Consider the grammar $G_P = (\{S\}, \{a, b\}, S, \{S \to \lambda \mid aSa \mid bSb\})$.

(i) [2 marks] Write all of the strings of length less than or equal to 4 that can be produced from G_P.

(Note that a string s can be produced from G iff s is in the language defined by G, i.e. $s \in \mathcal{L}(G)$.)

λ, aa, bb, $aaaa$, $abba$, $baab$, $bbbb$

(ii) [4 marks] Show that every string produced from G_P is a *palindrome*. (A string s is a palindrome if it is equal to its own reflection, i.e. if $s^R = s$.)

The proof is by induction on the number rule applications required to produce s.

The base case is when one rule application is required, and $s = \lambda$. Clearly $\lambda^R = \lambda$, so λ is a palindraome.

For the step case, assume that and string s that can be produced with n rule applications (for $n \geq 1$) is a palindrome. Then the strings that can be produced with $n = 1$ rule applications are asa and bsb. But if s is a palidrome, then so are asa and bsb. Thus all strings that can be produced with $n + 1$ rule applications are palindromes.

Therefore all strings that can be produced from G_P are palindromes.

(iii) [6 marks] Can every palindrome over $\{a, b\}$ be produced by G_P? If so, prove that this is the case. If not, modify the grammar so that it does produce all palindromes over $\{a, b\}$ and prove that the resulting grammar does indeed produce all palindromes over $\{a, b\}$.

No, the grammar only produces strings of even length (since the first rule produces λ and the others produce a string 2 longer than another string), so cannot produce a, b, aaa, aba, or any other odd length palindrome.

We extend the definitoin of S so that it can produce a or b as well as the other cases:

$S \to \lambda \mid a \mid b \mid aSa \mid bSb$

...

(c) [8 marks] Given two grammars, $G = (V_N, V_T, S, P)$ and $G' = (V'_N, V'_T, S', P')$, defining languages X and Y (i.e. $X = \mathcal{L}(G)$ and $Y = \mathcal{L}(G')$), show how you can construct a grammar $G'' = (V''_N, V''_T, S'', P'')$ which defines the concatenation of X and Y (i.e. $\mathcal{L}(G'') = V_N \frown V'_N = \{\}$).

Give a brief argument to show that the resulting grammar does in fact define the required language.

Combine the terminals, nonterminals and productions of G and G', and add a new start symbol S_i and a rule $S_i \rightarrow S\ S'$; i.e. $G'' = (V''_N, V''_T, S'', P'')$ where

- $S'' \notin V_N \cup V_T \cup V'_N \cup V'_T$
- $V''_N = V_N \cup V'_N \cup \{S_i\}$
- $V''_T = V_T \cup V'_T$
- $P'' = P \cup P' \cup \{S'' \rightarrow S\ S'\}$

If $\alpha \in X \frown Y$, there are strings $\beta \in X$ and $\gamma \in Y$, such that $\alpha = \beta \frown \gamma$. Now, since $\beta \in X$ and $\gamma \in Y$, there is a parse tree, say T, for β from G and a parse tree, say T', for γ from G'. Thus, we can construct a parse tree for α from G'' by applying the rule $S'' \rightarrow S\ S'$ and taking T and T' as its subtrees, as shown on the right. Thus, every string in $X \frown Y$ is in $\mathcal{L}(G'')$.

If $\alpha \in \mathcal{L}(G'')$, there is a parse tree T'' for α from G. But since $S'' \rightarrow S\ S'$ is the only rule for S'', T'' must have two subtrees, say T and T', whose roots are labelled S and S', as shown on the right. Let β be the fringe of T and γ be the fringe of T', then T is a parse tree for β from G, and T' is a parse tree for β from G', so $\beta \in X$ and $\gamma \in Y$. Thus every string in $\mathcal{L}(G'')$ is in $X \frown Y$.

****