**Applied Automata Theory** (WS 2012/2013) Technische Universität Kaiserslautern

# Exercise Sheet 2

Jun.-Prof. Roland Meyer, Reiner Hüchting, Georgel Călin Due: Tue, Oct 30 (**10:00**)

### Exercise 2.1 WMSO Expressiveness

(a) Show that any WMSO[$<$, suc]-definable language is already WMSO[suc]-definable.

(b) Show that any WMSO[suc]-definable language is already $\text{WMSO}_0$-definable.

### Exercise 2.2 Sing Elimination

You showed that, with respect to languages, $\text{WMSO}_0$ is as expressive as WMSO[$<$, suc]. Since one often wants a logic with as few predicates as possible, you should eliminate the Sing-predicate. Assume that $|\Sigma| \geq 2$ and that we only consider word structures.

(a) Present a formula $\varphi$ in $\text{WMSO}_0$ that does not utilize the Sing-predicate and that expresses emptiness of a set (i.e., $\varphi$ has a free second order variable $X$ and is satisfied by an interpretation $I$ if and only if $I$ assigns $X$ to the empty set).

(b) Present a $\text{WMSO}_0$-formula $\varphi$ that does not use the Sing-predicate and expresses the property of being a singleton. (Hint: Singleton sets have exactly two subsets.)

### Exercise 2.3 WMSO[$<$, suc]-defined Languages

(a) Present a WMSO[$<$, suc]-formula that defines the language $\{w \in \{a, b\}^* \mid |w| \in 3\mathbb{N}\}$.

(b) Present a WMSO[$<$, suc]-formula that defines the language $\{aaa, bbb\}^*$.

(c) Show that, for any alphabet $\Sigma$, the language defined by

$$\begin{aligned}\exists X :&(\forall x : \forall y : \forall z : (X(x) \wedge X(y) \wedge x < z \wedge z < y) \rightarrow X(z)) \\ &\wedge (\exists x : \exists y : (x < y \wedge X(x) \wedge X(y))) \\ &\wedge (\forall x : X(x) \rightarrow P_a(x))\end{aligned}$$

is regular.

### Exercise 2.4 From WMSO to Finite Automata

Using the method presented in the lecture, construct a finite automaton that accepts the language defined by the formula $\varphi = \forall x : (P_a(x) \rightarrow \forall y : (x < y \rightarrow P_b(y)))$. Give a regular representation of this language.

**Exercise 2.5 (Parallel) Programs as Automata** **optional**

We extend the syntax of Boolean programs. As before, there is a finite set of shared memory locations $\mathtt{x}_1, \ldots, \mathtt{x}_n$. This time, each thread $t \in \{t_1, \ldots, t_k\}$ additionally has its own set of registers $\mathtt{r}_1^t, \ldots, \mathtt{r}_m^t$. Initially, all registers and memory locations hold value 0. Programs operate on registers and memory locations using the following assembler-like instruction set:

| Statement | Meaning |
|---|---|
| $\mathtt{r} \leftarrow \mathtt{x}$ | Load the value stored at address $\mathtt{x}$ into register $\mathtt{r}$. |
| $\mathtt{x} \leftarrow \mathtt{r}$ | Store the value of register $\mathtt{r}$ at address $\mathtt{x}$. |
| $\mathtt{if\ r\ then\ l_1\ else\ l_2}$ | Test if $\mathtt{r} = 1$. Continue with the corresponding label. |
| $\mathtt{r_1} := \neg \mathtt{r_2}$, $\mathtt{r_1} := \mathtt{r_2} \wedge \mathtt{r_3}$ | Perform basic ($\neg$, $\wedge$) Boolean operations on registers. |
| $\mathtt{goto\ l}$ | Continue with label $\mathtt{l}$. |

Consider the following programs, implementing Dekker's protocol:

```
P1:                                          P2:
  l0 :   r1 := ¬r1                             l0' :  r1' := ¬r1'
         x ← r1                                      y ← r1'
         r2 ← y                                      r2' ← x
         if r2 then l1 else l2                       if r2' then l1' else l2'
  l1 :   r1 := ¬r1                             l1' :  r1' := ¬r1'
         x ← r1                                      y ← r1'
         goto l0                                     goto l0
  l2 :                                         l2' :
```

Give a translation of these programs into the syntax from Exercise 1.2. Notice that, as a consequence of your translation, one can interpret assembler programs as above in terms of finite automata.