Dr. Matthias Heizmann
Dominik Klumpp

Hand in until 10:00 on June 08, 2020
Discussion: June 08, 2020

Tutorial for Program Verification
Exercise Sheet 6

In this exercise we work with the programming & verification language Boogie, and our reduced variant of this language, called Boostan.

Submit your solution by uploading it as PDF in ILIAS.

Exercise 1: Boogie — 2 Points

Using the Boogie[1] language, implement a procedure with signature

```
procedure gcd(x : int, y : int) returns (z : int)
```

that takes two (mathematical) integers x, y and, if they are both not equal to 0, computes their greatest common divisor z. The algorithm may only make use of addition and subtraction, but not use multiplication, division or modulo.[2]

You can use the Boogie interpreter Boogaloo[3] to test your program. A user manual is available online[4].

Exercise 2: Boostan Grammar — 2 Points

In this exercise you should propose a syntax for the Boostan programming language. State a context-free grammar $\mathcal{G}_{\mathsf{Boo}} = (\Sigma_{\mathsf{Boo}}, N_{\mathsf{Boo}}, P_{\mathsf{Boo}}, S_{\mathsf{Boo}})$ such that a word of the generated language is a program of (your version of) the Boostan language.

In the lecture slides we propose the grammar $\mathcal{G}_{\mathsf{I}} = (\Sigma_{\mathsf{I}}, N_{\mathsf{I}}, P_{\mathsf{I}}, S_{\mathsf{I}})$ for integer expressions, with $\Sigma_{\mathsf{I}} = \{-, +, *, /, \%, (,), 0, \ldots, 9, a, \ldots, z, A, \ldots Z\}$,
$N_{\mathsf{I}} = \{X_{iexpr}, X_{num}, X_{num'}, X_{var}, X_{var'}\}$, $S_{\mathsf{I}} = X_{iexpr}$ and the following derivation rules:

$$
\begin{aligned}
P_{\mathsf{I}} = \{ X_{iexpr} &\to \texttt{(}X_{iexpr}\texttt{)} \\
X_{iexpr} &\to \texttt{-}X_{iexpr} \\
X_{iexpr} &\to X_{iexpr}\texttt{+}X_{iexpr} \mid X_{iexpr}\texttt{-}X_{iexpr} \mid X_{iexpr}\texttt{*}X_{iexpr} \mid X_{iexpr}\texttt{/}X_{iexpr} \mid X_{iexpr}\texttt{\%}X_{iexpr} \\
X_{iexpr} &\to X_{var} \\
X_{iexpr} &\to X_{num} \\
X_{num} &\to \texttt{0}X_{num'} \mid \ldots \mid \texttt{9}X_{num'} \\
X_{num'} &\to \texttt{0}X_{num'} \mid \ldots \mid \texttt{9}X_{num'} \mid \varepsilon \\
X_{var} &\to \texttt{a}X_{var'} \mid \ldots \mid \texttt{z}X_{var'} \mid \texttt{A}X_{var'} \mid \ldots \mid \texttt{Z}X_{var'} \\
X_{var'} &\to \texttt{a}X_{var'} \mid \ldots \mid \texttt{z}X_{var'} \mid \texttt{A}X_{var'} \mid \ldots \mid \texttt{Z}X_{var'} \mid \texttt{0}X_{var'} \mid \ldots \mid \texttt{9}X_{var'} \mid \varepsilon \}
\end{aligned}
$$

[1] https://www.microsoft.com/en-us/research/wp-content/uploads/2016/12/krml178.pdf
[2] Hint: https://en.wikipedia.org/wiki/Euclidean_algorithm
[3] http://comcom.csail.mit.edu/comcom/#Boogaloo
[4] https://bitbucket.org/nadiapolikarpova/boogaloo/wiki/User%20Manual

Next, we proposed the grammar $\mathcal{G}_\mathsf{B} = (\Sigma_\mathsf{B}, N_\mathsf{B}, P_\mathsf{B}, S_\mathsf{B})$ for Boolean expressions, with $\Sigma_\mathsf{B} = \Sigma_\mathsf{I} \cup \{!, \&\&, \|, ==>, ==, <, >, <=, >=\}$, $N_\mathsf{B} = N_\mathsf{I} \cup \{X_{bexpr}\}$, $S_\mathsf{B} = X_{bexpr}$ and the following derivation rules:

$$
\begin{aligned}
P_\mathsf{B} = \{ X_{bexpr} &\to \texttt{(} X_{bexpr} \texttt{)} \\
X_{bexpr} &\to \texttt{!} X_{bexpr} \\
X_{bexpr} &\to X_{bexpr} \texttt{\&\&} X_{bexpr} | X_{bexpr} \texttt{||} X_{bexpr} | X_{bexpr} \texttt{==} X_{bexpr} \\
X_{bexpr} &\to X_{iexpr} \texttt{==} X_{iexpr} | X_{iexpr} \texttt{<} X_{iexpr} | X_{iexpr} \texttt{>} X_{iexpr} | X_{iexpr} \texttt{<=} X_{iexpr} | X_{iexpr} \texttt{=>} X_{iexpr} \\
X_{bexpr} &\to X_{var} \\
X_{bexpr} &\to \textbf{true} | \textbf{false} \} \cup P_\mathsf{I}
\end{aligned}
$$

We propose that you use $\Sigma_{\mathsf{Boo}} = \Sigma_\mathsf{B} \cup \{\texttt{while}, \texttt{if}, \texttt{else}, \{, \}, \texttt{;}, \texttt{:=}\}$ and re-use the grammars for integer and boolean expressions.

Your language should have the following properties:

- There should be a while statement, an if-then-else statement and an assignment statement.
- The concatenation of statements should be a statement.
- A program should be a statement.
- We do not need statements for declaring variables.

Exercise 3: Derivation Tree 1 Point

Give a derivation tree for the grammar $\mathcal{G}_\mathsf{I}$ and the word $15 + a + 4$.

In the next part of the lecture, we will define the semantics of Boostan formally. For this purpose, we will use the *reflexive transitive closure* of a relation. The exercises below should make you familiar with that term.

Given a set X, a *binary relation* over X is a subset of the Cartesian product $X \times X$. We call a binary relation R *reflexive* if for all $x \in X$ the pair (x, x) is an element of R. We call a binary relation R *transitive* if for all $x, y, z \in X$ the following property holds:

$$\text{If } (x, y) \in R \text{ and } (y, z) \in R \text{ then } (x, z) \in R.$$

Exercise 4: Reflexivity and Transitivity 2 Points

State for each of the following relations if the relation is reflexive and if the relation is transitive.

(a) The "strictly smaller" relation over integers, $R_a = \{(x, y) \in \mathbb{Z} \times \mathbb{Z} \mid x < y\}$

(b) The relation $\mathsf{win}_{\mathsf{RPS}}$ over the set $\{\mathsf{Rock}, \mathsf{Paper}, \mathsf{Scissors}\}$.
$R_b = \{(\mathsf{Rock}, \mathsf{Scissors}), (\mathsf{Scissors}, \mathsf{Paper}), (\mathsf{Paper}, \mathsf{Rock})\}$

(c) $R_c = \{(x, y) \in \mathbb{Z} \times \mathbb{Z} \mid (x - y) \bmod 42 = 0\}$

(d) $R_d = \{(x, y) \in \mathbb{Z} \times \mathbb{Z} \mid (x - y) = 2 \text{ or } (y - x) = 2\}$

Given a binary relation R over the set X, the *reflexive transitive closure*, denoted R^*, is the smallest relation such that $R \subseteq R^*$, R^* is reflexive and R^* is transitive.
We note that in this context "smallest" means that there is no strict subset that has the same properties and that this minimum is indeed unique since reflexive and transitive relations are closed under intersection.

Exercise 5: Reflexive Transitive Closure 2 Points

Compute for each of the four relations above the reflexive transitive closure.