0.1 Transition Rules

In this section some of the transition rules in SPLAD will be explained. The complete list of all the rules can be seen in appendix **??**. In the following text we use the following names to represent different syntactic categories.

- $n \in \mathbf{Num}$ - Numerals
- $x \in \mathbf{Var}$ - Variables
- $r \in \mathbf{Arrays}$ - Array names
- $a \in \mathbf{A_{exp}}$ - Arithmetic expression
- $b \in \mathbf{B_{exp}}$ - Boolean expression
- $S \in \mathbf{Stm}$ - Statements

0.1.1 Environment-Store Model

In our project we use the *environment-store model* to represent how a variable is bound to a storage cell (called a *location*), in the computer, and that the value of the variable is the content of the bound location. All the possible locations are denoted by **Loc** and a single location as $l \in \mathbf{Loc}$. We assume all locations are integer, and therefore $\mathbf{Loc} = \mathbb{Z}$. Since all locations are integers we can define a function to find the next location: $\mathbf{Loc} \to \mathbf{Loc}$, where $l = l + 1$.

We define the set of stores to be the mappings from locations to values $\mathbf{Sto} = \mathbf{Loc} \rightharpoonup \mathbb{Z}$, where *sto* is an single element in **Sto**.

An variable-environment is like a symbol table containing each variable and store the variables address. The store then describe which values that is on each address.

The following names represent the different environments.

- $env_V \in Env_V$ - Variable environment
- $env_A \in Env_A$ - Array environment
- $env_P \in Env_P$ - Procedure environment

0.1.2 Big-step-semantic and small-step-semantic

FiXme Fatal: Fo
hvad det er, og a
bruger big-step.

0.1.3 Abstract Syntax

FiXme Fatal: bla

Here the abstract syntax of SPLAD can be seen. An abstract syntax is bla bla bla...

$$
\begin{aligned}
S &::= x := a \mid r[a_1] := a_2 \mid S_1;\ S_2 \mid \text{if } b \text{ do } S \mid \text{if } b \text{ do } S_1 \text{ else do } S_2 \mid \text{while } b \text{ do } S \\
&\quad \mid \text{from } x := a_1 \text{ to } a_2 \text{ step } a_3 \text{ do } S \mid \text{call } p(\vec{x}) \mid \text{begin } D_V\ D_P\ S \text{ end} \\
a &::= n \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \mid a_1/a_2 \mid (a) \\
b &::= a_1 = a_2 \mid a_1 > a_2 \mid a_1 < a_2 \mid \neg b \mid b_1\ \wedge\ b_2 \mid b_1\ \vee\ b_2 \mid (b) \\
D_V &::= \text{var } x := a;\ D_V \mid \varepsilon \\
D_P &::= \text{proc } p \text{ is } S;\ D_P \mid \varepsilon \\
D_A &::= \text{array } r[a_1] := a_2;\ D_A \mid \varepsilon
\end{aligned}
$$

0.1.4 Statements

The transition rules for the statements are on the form: $env_V, env_P \vdash \langle S, sto\rangle \rightarrow sto'$.

The transition rule for variable assignment in SPLAD can be seen on table 0.1. When a variable is assigned the contents of l is updated to v, where l is the location of x found in the env_V and v is the result of the arithmetic expression a.

$$\text{[VAR-ASS]} \quad env_V, env_P \vdash \langle x <-- a, sto\rangle \rightarrow sto[l \mapsto v]$$

where $env_V, sto \vdash a \rightarrow_a v$
and $env_V\ x = l$

Table 0.1: Transition rule for variable assignment.

0.1.5 Arithmetic Expressions

The transition rules for the arithmetic expressions are on the form: $env_V, sto \vdash a \rightarrow_a v$.

The transition rule for multiplication in SPLAD can be seen on table 0.2. The rule states, that if a_1 evaluates to v_1 and a_2 evaluates to v_2, using any of the rules from the arithmetic expressions, then $a_1 \cdot a_2$ evaluates to v where $v = v_1 \cdot v_2$.

0.1.6 Boolean Expression

The transition rules for boolean expressions are on the form: $env_V, sto \vdash b \rightarrow_b t$.

The transition rule for logical-or in SPLAD can be seen on table 0.3. The rules have two parts: [OR-TRUE] and [OR-FALSE]. The [OR-TRUE] rule states that either b_1 or b_2 evaluates to *TRUE*, using any of the rules from the boolean expressions, then the expression b_1 OR b_2 evaluates to *TRUE*. [OR-FALSE] states that if both b_1 and b_2 evaluate to *FALSE* then the expression b_1 OR b_2 evaluates to *FALSE*.

$$\text{[OR-TRUE]} \quad \frac{env_V, sto \vdash b_i \rightarrow_b \text{TRUE}}{env_V, sto \vdash b_1 \vee b_2 \rightarrow_b \text{TRUE}}$$

where $i \in 1, 2$

$$\text{[OR-FALSE]} \quad \frac{env_V, sto \vdash b_1 \rightarrow_b \text{FALSE} \quad env_V, sto \vdash b_2 \rightarrow_b \text{FALSE}}{env_V, sto \vdash b_1 \vee b_2 \rightarrow_b \text{FALSE}}$$

Table 0.3: Transition rule for the boolean expression logical-or.

0.1.7 Variable Declaration

The transition rules for the variable declarations are on the form: $\langle D_V, env_V, sto\rangle \rightarrow_{DV} (env'_V, sto')$

atal: Bla bla bla

$$\text{[VAR-DECL]} \quad \frac{\langle D_V, env''_V, sto[l \mapsto v]\rangle \rightarrow_{DV} (env'_V, sto'}{\text{var } x <-- a; D_V, env_V, sto\rangle \rightarrow_{DV} (env'_V, sto')}$$

where $env_V, sto \vdash a \rightarrow_a v$
and $l = env_V$ next

Continued on the next page

$$\text{and } env''_V = env_V[x \mapsto l][\text{next} \mapsto \text{new } l]$$

Table 0.4: Transition rules for the variable declarations.

0.1.8 Procedure Declaration

The transition rules for the procedure declarations are on the form: $env_V \vdash \langle D_P, env_P \rangle \rightarrow_{DP} env'_P$ FiXme Fatal: bla

$$\text{[PROC-DECL]} \quad \frac{env_V \vdash \langle D_P, env_P[p \mapsto (S, env_V, env_P)] \rangle \rightarrow_{DP} env'_P}{env_V \vdash \langle \text{proc } p \text{ is } S; D_P, env_P \rangle \rightarrow_{DP} env'_P}$$

Table 0.5: Transition rules for the procedure declarations.

0.1.9 Array Declaration

The transition rules for the variable declarations are on the form: ??? FiXme Fatal: bla

[MULT] $$\frac{env_V, sto \vdash a_1 \rightarrow_a v_1 \quad env_V, sto \vdash a_2 \rightarrow_a v_2}{env_V, sto \vdash a_1 \cdot a_2 \rightarrow_a v}$$

where $v = v_1 \cdot v_2$

Table 0.2: The transition rule for the arithmetic multiplication expression.