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.

- $n \in$ **Num** - Numerals
- $x \in$ **Var** - Variables
- $r \in$ **Arrays** - Array names
- $a \in \mathbf{A_{exp}}$ - Arithmetic expression
- $b \in \mathbf{B_{exp}}$ - Boolean expression
- $S \in$ **Stm** - Statements
- $p \in$ **Pnames** - Procedure names
- $D_V \in$ **DecV** - Variable declarations
- $D_P \in$ **DecP** - Procedure declarations
- $D_A \in$ **DecA** - Array declarations

0.1.1 Abstract Syntax

In order to describe the behavior of a program, one must first account for its syntax. We use the notion of abstract syntax, since it will allow us to describe the essential structure of the program. An abstract syntax is defined as follows. We assume a collection of syntactic categories and for each syntactic category we give a finite set of formation rules which defines how the the inhabitants of the category can be build [**?**]. Here the abstract syntax of SPLAD can be seen.

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

0.1.2 Transition Systems

0.1.3 Big-step-semantic and small-step-semantic

0.1.4 Environment-Store Model

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

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 $\mathbf{Loc} = \mathbb{N}$. 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

0.1.5 Statements

The transition rules for the statements are on the form: $env_V, env_P \vdash \langle S, sto\rangle \to sto'$. The transition system are defined by: $(\Gamma_{\mathbf{Stm}}, \to, \mathrm{T}_{\mathbf{Stm}})$ and the configurations are defined by $\Gamma_{\mathbf{Stm}} = \mathbf{Stm} \times \mathbf{Sto} \cup \mathbf{Sto}$. The end configurations are defined by $\mathrm{T}_{\mathbf{Stm}} = \mathbf{Sto}$.

On table 0.1 the assignment rule for variables can be seen. The rule states, that if a evaluates to v, and x points to the location l, then v is stored in the l.

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

$$\text{where } env_V, sto \vdash a \to_a v$$
$$\text{and } env_V\ x = l$$

Table 0.1: Transition rule for variable assignment.

0.1.6 Arithmetic Expressions

The transition rules for the arithmetic expressions are on the form: $env_V, sto \vdash a \to_a v$. The transition system are defined by: $(\Gamma_{\mathbf{Aexp}}, \to_a, \mathrm{T}_{\mathbf{Aexp}})$ and the configurations are defined by $\Gamma_{\mathbf{Aexp}} = \mathbf{Aexp} \cup \mathbb{Z}$. The end configurations are defined by $\mathrm{T}_{\mathbf{Aexp}} = \mathbb{Z}$.

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 $v = v_1 \cdot v_2$.

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

$$\text{where } v = v_1 \cdot v_2$$

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

0.1.7 Boolean Expression

The transition rules for boolean expressions are on the form: $env_V, sto \vdash b \rightarrow_b t$. The transition system are defined by: $(\Gamma_{\mathbf{Bexp}}, \rightarrow_b, \mathrm{T}_{\mathbf{Bexp}})$ and the configurations are defined by $\Gamma_{\mathbf{Bexp}} = \mathbf{Bexp} \cup \{tt, ff\}$. The end configurations are defined by $\mathrm{T}_{\mathbf{Bexp}} = \{tt, ff\}$.

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*.

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

$$\text{where } i \in 1, 2$$

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

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

0.1.8 Variable Declaration

The transition rules for the variable declarations are on the form: $\langle D_V, env_V, sto\rangle \rightarrow_{DV} (env'_V, sto')$. The transition system are defined by: $\Gamma_{\mathbf{ErkV}}, \rightarrow_{DV}, \mathrm{T}_{\mathbf{ErkV}}$ and the configurations are defined by $\Gamma_{DV} = (\mathbf{ErkV} \times \mathbf{EnvV} \times \mathbf{Sto}) \cup (\mathbf{EnvV} \times \mathbf{Sto})$ and $\mathrm{T}_{DV} = (\mathbf{EnvV} \times \mathbf{Sto})$. The end configurations are defined by $\mathrm{T}_{\mathbf{ErkV}} = \mathbf{EnvV} \times \mathbf{Sto}$.

On table 0.4 the transition rules for variable declaration can be seen. It is done by binding l to the next available location and binding x to this location. The function *new* is then used to point at the next available location. Then env_V is updated to include the new variable, while the store remains unchanged.

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

$$\begin{aligned} &\text{where } env_V, sto \vdash a \rightarrow_a v \\ &\text{and } l = env_V \text{ next} \\ &\text{and } env''_V = env_V[x \mapsto l][\text{next} \mapsto \text{new } l] \end{aligned}$$

Table 0.4: Transition rules for the variable declarations.

0.1.9 Procedure Declaration

The transition rules for the procedure declarations are on the form: $env_V \vdash \langle D_P, env_P\rangle \rightarrow_{DP} env'_P$. The transition system are defined by: $(\Gamma_{\mathbf{ErkP}}, \rightarrow_{DP}, \mathrm{T}_{\mathbf{ErkP}})$ and the configurations are defined by $\Gamma_{DP} = (\mathbf{ErkP} \times \mathbf{EnvP}) \cup \mathbf{EnvP}$ and $\mathrm{T}_{DP} = \mathbf{EnvP}$. The end configurations are defined by $\mathrm{T}_{\mathbf{ErkP}} = \mathbf{EnvP}$.

On table 0.5 the transitions rules for the procedure declaration with none or multiple parameters can be seen. The rule states that the new procedure is stored in the procedure

environment along with the statement, formal parameters, and procedure- and variable-bindings from the time of declaration.

$$\text{[PROC-PARA-DEC]} \quad \frac{env_V \vdash \langle D_P, env_P[p \mapsto (S, \vec{x}, env_V, env_P)]\rangle \rightarrow_{DP} \ env'_P}{env_V \vdash \langle \text{func } p(\text{var } \vec{x}) \text{ is begin } S \text{ end}, \ env_P\rangle \rightarrow_{DP} env'_P}$$

Table 0.5: Transition rules for the procedure declarations.

0.1.10 Array Declaration

The transition rules for the variable declarations are on the form: $\langle D_A, env_V, sto\rangle \rightarrow_{DA} (env'_V, sto')$. The transition system are defined by: $(\Gamma_{\mathbf{ErkA}}, \rightarrow_{DA}, \mathrm{T}_{\mathbf{ErkA}})$ and the configurations are defined by $\Gamma_{DA} = (\mathbf{ErkA} \times \mathbf{EnvV} \times \mathbf{Sto}) \cup (\mathbf{EnvV} \times \mathbf{Sto})$ and $\mathrm{T}_{DA} = \mathbf{EnvV} \times \mathbf{Sto}$. The end configurations are defined by $\mathrm{T}_{\mathbf{ErkA}} = \mathbf{EnvV} \times \mathbf{Sto}$.

On table 0.6 the transitions rules for the array declaration can be seen. The rule states that a number of location equal to the length of the array plus one is allocated, and the pointer is set to point at the next available location. The length of the array are then stored in the first location of the array.

$$\text{[ARRAY-DEC]} \quad \frac{\langle D_A, env_V[r \mapsto l, \text{next} \mapsto l + v + 1], sto[l \mapsto v]\rangle \rightarrow_{DA} (env'_V, sto')}{\langle \text{array } r[a_1], env_V, sto\rangle \rightarrow_{DA} (env'_V, sto')}$$

where $env_V, sto \vdash a_1 \rightarrow_a v$
and $l = env_V \text{next}$
and $v > 0$

Table 0.6: Transition rules for the array declarations.