COSC 4780 Name: Jacob Tuttle
Principles of Programming Langauges

Final Exam

May 13, 2020

The following document describes the language of **Bumps+**, an extension of the language **Bumps** presented in Transitions and Trees[1]. **Bumps+** describes the semantics for a variation of **Bumps** where the Variable Declarations and Procedure Declarations have been combined so that the Variable Environments, **EnvV**, and the Procedure Environments, **EnvP**, are combined into a single environment record, **Env**.

6.1 Abstract Syntax of Bumps+

In order to facilitate the bundling of environment records into a single record, we modify the abstract syntax rules to describe only two categories.

- **Pnames -** the category of procedure names (unchanged from **Bips**)
- **Dec -** the category of declarations, both variable and procedure

We continue to denote elements of **Pnames** by $p, q, ...$ and now denote elements of **Dec** by V. The formation rules of **Aexp** and **BExp** remain unchanged from their description in the book, but the formation rules below now describe the new rules for **Stm** and **Dec**. Furthermore, ϵ continues to denote the empty declaration.

$$
\begin{aligned}
S ::= {} & x := a \mid \texttt{skip} \mid S_1; S_2 \mid \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \mid \\
& \texttt{while } b \texttt{ do } S \mid \texttt{begin } V\ S \texttt{ end} \mid \texttt{call } p(y) \mid \texttt{call } p(\&y) \\
V ::= {} & \texttt{var } x := a \mid \texttt{proc } p(\texttt{var } x) \texttt{ is } S \mid \texttt{proc } p(x) \texttt{ is } S \mid V_1; V_2 \mid \epsilon
\end{aligned}
$$

[1] Huttel, 2010

Walking through the changes to the syntax of **Bumps** to get **Bumps+** we see the following changes:

- The `begin...end` command has changed to combine the D_V D_P representation of the environments into a single representation V to be consistent with our new declarations for the environment.
- Two `call...` options are present, one with an ampersand preceding the parameter and one without. These correspond to calling the procedure p by reference and by value respectively.
- D_V and D_P have been combined into a single environment record, V.
- There are two declarations for procedures, one containing the keyword `var` and one without. These correspond to call by reference and call by value respectively.
- V no longer contains a recursive description at the end of a declaration, but contains the sequencing like operation $V_1; V_2$. This functions in the same manner as sequencing for statements and allows declarations to be sequenced. It also corresponds to the implementation of **Bumps** in `Bumps_Final.hs`.[2]

6.2 Changes to the Environment Model in Bumps+

The changes to the environment system allow us to eliminate the complexity of passing both a $EnvV$ and $EnvP$ argument around as we evaluate code written in **Bumps+**. It also has the beneficial side effect of allowing us to declare procedures and variables in any order, a change from Huttel's descriptions that forced variables to be declared before procedures. This simplicity will become apparent in further sections of this paper where the semantic rules are updated to conform to this new paradigm.

In the implementation of these rules, the haskell code makes use of two constructors in the declaration data type, `Decl`, namely `DSemi` and `DEmpty`. Through comparison, we can see that the constructor `DSemi` is identical to the constructor `Seq` for sequences. `DSemi Decl Decl` and `Seq Stm Stm` both sequence two objects of their parent data type, `Decl` and `Stm` respectively which correspond to our abstract syntax of V and S. `DEmpty` is a constructor of `Decl` as well and takes no arguments. Because of this, we know that it acts

[2]This construction and implementation will be discussed in more detail below.

as a terminal character or the implementation of ϵ. Without it, it would be impossible to have 0 declarations, but since it is present, it is therefore possible to have any number of declarations in any combination of variables and procedures.

The final addition that makes this bundling possible is the addition of the `‘unionE‘` operator on environments. It functions in the way one would expect a union operator to work, returning the environment made up of the union of variable declarations and the union of procedure declarations. Something worth note is that due to implementation restrictions, the values stored in the first environment take precedence over those stored in the second when using the `‘unionE‘` operation.

6.3 Evaluation of expressions in the new environment paradigm

As noted by Huttel, the semantics of statements and declarations both depend on the semantics of **Aexp** and those expressions may contain variables. Since the way in which variables are processed and stored has been completely renovated, the rules for evaluating these epxressions will be redefined. The transition system remains the same, but all all transitions will now be of the form $env,\ sto \vdash a \rightarrow_a v$ and can be read as ”given the variables and procedures known by env and the storage content of sto, the arithmetic expression a evaluates to the value v.”

The following rules now represent the new Big Step Semantic rules[3]:

$$[\text{PLUS-BUMPS+}]\ \frac{env,\ sto \vdash a_1 \rightarrow_a v_1 \mid env,\ sto \vdash a_2 \rightarrow_a v_2}{env,\ sto \vdash a_1 + a_2 \rightarrow_a v} \quad \text{where } v = v_1 + v_2$$

$$[\text{MINUS-BUMPS+}]\ \frac{env,\ sto \vdash a_1 \rightarrow_a v_1 \mid env,\ sto \vdash a_2 \rightarrow_a v_2}{env,\ sto \vdash a_1 - a_2 \rightarrow_a v} \quad \text{where } v = v_1 - v_2$$

[3]A change from the textbook, the BSS is omitted from the name of rules for simplicity and since small step semantic rules will not be defined in this paper.

$$\frac{env,\ sto \vdash a_1 \rightarrow_a v_1 \mid env,\ sto \vdash a_2 \rightarrow_a v_2}{env,\ sto \vdash a_1 * a_2 \rightarrow_a v} \text{ [MULT-BUMPS+]}$$
where $v = v_1 \cdot v_2$

$$\frac{env,\ sto \vdash a_1 \rightarrow_a v_1}{env,\ sto \vdash (a_1) \rightarrow_a v_1} \text{ [PARENT-BUMPS+]}$$

[NUM-BUMPS+] $env,\ sto \vdash n \rightarrow_a v$ if $\aleph[|n|] = v$[4]

[VAR-BUMPS+] $env,\ sto \vdash x \rightarrow_a v$ if $env\ x = l$ and $sto\ l = v$

6.4 Changes to Declaration Big Step Semantics

With the changes to declarations presented in this paper, the declaration operational semantics presented by Huttel must be modified. The execution of a `begin...end` block will evaluate all types of declarations before executing statements. Now these environment operational semantic changes will be described.

[ASS-BUMPS+] $env \vdash \langle x := a, sto\rangle \rightarrow sto[l \mapsto v]$
where $env,\ sto \vdash a \rightarrow_a v$ and $env\ x = l$

[SKIP-BUMPS+] $env \vdash \langle \texttt{skip}, sto\rangle \rightarrow sto$

$$\frac{env \vdash \langle S_1, sto\rangle \rightarrow sto'' \quad env \vdash \langle S_2, sto''\rangle \rightarrow sto'}{env \vdash \langle S_1; S_2, sto\rangle \rightarrow sto'} \text{ [COMP-BUMPS+]}$$

[4] $\aleph$ will stand in for the scripty N in Huttel's book and [| will stand in for the double bar bracket.

$$\frac{env \vdash \langle S_1, sto\rangle \to sto'}{env \vdash \langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2, sto\rangle \to sto'} \text{ [IF-TRUE-BUMPS+]}$$
where $env,\ sto \vdash b \to_b tt$

$$\frac{env \vdash \langle S_2, sto\rangle \to sto'}{env \vdash \langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2, sto\rangle \to sto'} \text{ [IF-TRUE-BUMPS+]}$$
where $env,\ sto \vdash b \to_b ff$

$$\frac{env \vdash \langle S, sto\rangle \to sto'' \quad env \vdash \langle \texttt{while } b \texttt{ do } S, sto''\rangle \to sto'}{env \vdash \langle \texttt{while } b \texttt{ do } S, sto\rangle \to sto'} \text{ [WHILE-TRUE-BUMPS+]}$$
where $env,\ sto \vdash b \to_b tt$

[WHILE-FALSE-BUMPS+] $env \vdash \langle \texttt{while } b \texttt{ do } S, sto\rangle \to sto$
where $env,\ sto \vdash b \to_b ff$

$$\frac{\langle V, env, sto\rangle \to_V (env' sto'') \quad env' \vdash \langle S, sto''\rangle \to sto'}{env \vdash \langle \texttt{begin } V\ S \texttt{ end}, sto\rangle \to sto'} \text{ [BLOCK-BUMPS+]}$$

To complete this new set of Big Step Semantic rules for everything except procedure calls, a new transition is declared, $\to_V$. This transtion system is defined as a union between the transitions $\to_{DP}$ and $\to_{DV}$ presented in the Huttel text. Furthermore, this transition system, allows the big step semantics for procedures as:

$$\frac{env \vdash \langle V, env[p \mapsto (S, x, env)]\rangle \to_V env'}{env \vdash \langle \texttt{proc } p(\texttt{var } x) \texttt{ is } S; V, env\rangle \to_V env'} \text{ [PROC-REF-BUMPS+]}$$

$$\text{[PROC-VAL-BUMPS+]}\ \frac{env\ \vdash \langle V, env[p \mapsto (S, x, env)]\rangle \rightarrow_V env'}{env\ \vdash \langle \texttt{proc}\ \ p(x)\ \ \texttt{is}\ \ S; V, env\rangle \rightarrow_V env'}$$

$$\text{[PROC-EMPTY-BUMPS+]}\ env\ \vdash \langle \epsilon, env\rangle \rightarrow_V env$$

These new definitions of the big step semantic rules for **BUMPS+** fully define the language in terms of the new bundled environment record. The only missing operation is the `call` operation that allows us to call procedures.

6.5 Call-by-reference and Call-by-value Syntax

The code in `Bumps_Final.hs` supports both call by reference and call by value, as a result, rules for both will be defined using the syntax notation presented in the haskell file and defined in the abstract syntax rules presented in **Section 6.1**.

$$\text{[CALL-REF-BUMPS+]}\ \frac{env'[x \mapsto l]\ \vdash \langle S, sto\rangle \rightarrow sto'}{env\ \vdash \langle \texttt{call}\ \ p(\&y), sto\rangle \rightarrow sto'}$$

where $env\ p = (S, x, env')$ and $l = env\ y$

$$\text{[CALL-VAL-BUMPS+]}\ \frac{env'[x \mapsto l]\ \vdash \langle S, sto[l \mapsto v]\rangle \rightarrow sto'}{env\ \vdash \langle \texttt{call}\ \ p(y), sto\rangle \rightarrow sto'}$$

where $env\ p = (S, x, env')$ and $env,\ sto\ \vdash a \rightarrow_a v$

An observant reader will notice that nowhere is the value of l updated to the new/next free storage location. This is because the implementation of **BUMPS+** in `Bumps_Final.hs` abstracts this operation away from the environment and places it within the **Sto**. This abstraction simplifies the abstract syntax rules and tasks the `alloc` function with automatically updating it whenever a new variable is placed within the **Sto**.