ABSTRACT

IN MY THESIS I TRY TO DERIVE SOME THEORETICAL AND TECHNICAL CONSEQUENCES FROM SOME ASPECTS OF DENOTATIONAL SEMANTICS OF PROGRAMMING LANGUAGES. MORE PRECISELY I ANALYZE THE IMPACT OF THE STRUCTURE OF THE D.S. META-LANGUAGE (WHICH IS ESSENTIALLY APPLICATIVE) ON THE STRUCTURE AND THE IMPLEMENTATION OF LANGUAGES.

MOST OF THE EXAMPLES IN D.S. DEAL WITH IMPERATIVE LANGUAGES, PERAPHS FOR HISTORICAL REASONS (PEOPLE WANTED TO SHOW THAT D.S. IS POWERFUL ENOUGH FOR THOSE LANGUAGES TOO). BUT I THINK THAT THE IMPACT OF D.S. ON LANGUAGES IS SOMETHING MORE THAN MERELY (!) GIVING THEM A SEMANTICS. D.S. GIVES US NEW INSIGHTS ON HOW A LANGUAGE SHOULD BE DESIGNED (THE LET AND LETREC CONSTRUCTS ARE EXAMPLES OF WHAT I MEAN) AND IMPLEMENTED (CONTINUATIONS ARE A POWERFUL IMPLEMENTATION TOOL).

I NOTICE THAT SCOTT'S LAMBDA LANGUAGE IS A USEFUL SAMPLE LANGUAGE (AND META-LANGUAGE) TO EXEMPLIFY MANIPULATIONS THAT CAN BE CARRIED ON MORE COMPLEX LANGUAGES. SO I START WITH AN EXPOSITION OF SCOTT'S PW MODEL OF LAMBDA-CALCULUS, IN THE STILE OF THE "DATA TYPES AS LATTICES" PAPER. THEN I WRITE A META-CIRCULAR LAMBDA-INTERPRETER AND DEVELOP SOME OF THE D.S. STANDARD TECNIQUES (NAMELY ENVIRONMENTS, CONTINUATIONS AND STORES) WITH SLIGHT MODIFICATIONS TO THE LAMBDA LANGUAGE. THE FRAME IS AN APPLICATIVE ONE, SO I DON'T WORK WITH A STATUS TO STATUS SEMANTICS, BUT WITH AN EVALUATION SEMANTICS.

IT COMES OUT VERY EASILY THAT REYNOLD'S TRANSFORMATIONS OF

INTERPRETERS MAY BE APPLIED TO A LAMBDA-LIKE LANGUAGE, TO OBTAIN AN ITERATIVE INTERPRETER. YOU MAY FOLLOW (I HOPE) THE PHASES OF THIS TRANSFORMATIONS FROM FORMULAS ON MY THESIS (LAMBDA META-CIRCULAR INTERPRETER (WITH STRICT CONDITIONAL) [I.4.1]; ELIMINATION OF FUNCTIONAL ARGUMENTS [I.4.2]; ELIMINATION OF FUNCTIONAL VALUES BY INTRODUCING CLOSURES [I.4.3]; INTRODUCTION OF SUSPENSIONS (A SPECIAL KIND OF CLOSURES) TO HANDLE CALL-BY-NAME [I.4.4]).

AS FOR THE PRACTICAL DESIGN AND IMPLEMENTATION PROBLEMS, I HAVE FOUND MANY SUGGESTIONS FROM THE PAPERS BY FRIEDMAN AND WISE ABOUT SUSPENSIONS (IMPLEMENTATION OF NOT-STRICT FUNCTIONS) AND BY SUSSMAN AND STEELE ABOUT THE PROGRAMMING LANGUAGE SCHEME. IN THE SECOND PART OF MY THESIS I DEFINE A LANGUAGE (TAU), THAT CAN BE CONSIDERED AS AN EXTENSION OF SCHEME (OR OF STRICT LAMBDA, IF YOU PREFER). I TRY TO GIVE A D.S. TO IT AS SIMPLE AS POSSIBLE. THIS IMPLIES THAT THE OPERATIONAL SEMANTICS IS MORE COMPLEX, AND I HAVE TO WORRY ABOUT CALL-BY-NAME IMPLEMENTATIONS (CALL-BY-NEED), FULL LETREC AND MINIMAL FIXED POINTS OF DATA-STRUCTURES (WITH THE POSSIBILITY OF CIRCULAR AND INFINITE STRUCTURES, LIKE LANDIN'S STREAMS), NOT EVALUATING (SUSPENDING) DATA-CONSTRUCTORS AND SO ON. BUT I THINK THAT A CLEAR D.S. IS BETTER THAN A CLEAR O.S., ALSO BECAUSE THE FORMER OFTEN IMPLIES THE LATTER (EVEN IF CLEARER DOESN'T IMPLY EASIER TO IMPLEMENT).

TWO ISSUES ARE EXPLORED IN GREATER DETAIL. THE FIRST ONE IS SCOPING. I COMPARE STATIC AND DYNAMIC SCOPING IN RELATION WITH THE BEHAVIOUR OF ENVIRONMENTS AT RUN TIME AND SHOW HOW VALUES CAN BE RETRIEVED IN VARIOUS IMPLEMENTATIONS

OF VARIABLE EVALUATION. THE DISCUSSION BRINGS TO THE CONCLUSION THAT STATIC SCOPING HAS GREAT ADVANTAGES OVER DINAMIC SCOPING FROM THE POINT OF VIEW OF BOTH THE USER AND THE IMPLEMENTER, SO I CHOOSE THE STATIC SCOPING FOR THE SEMANTICS OF TAU.

THEN I TRY TO DEFINE THE CONCEPT OF SCOPING INDEPENDENTLY OF ANY PARTICULAR LANGUAGE. I DEFINE A FORMALISM TO EXPRESS COMPUTABLE FUNCTIONS WHICH IS AN AXIOMATIZATION OF THE SCOTT-STRACHEY NOTATION, SO THAT I CAN TALK ABOUT LANGUAGES AND INTERPRETERS IN A VERY GENERAL WAY.

IN THIS FRAME I DEFINE THE SCOPING OF A VARIABLE AS THE OCCURRENCES EVALUATED IN AN ENVIRONMENT EXTENDED BY A BINDER FOR THAT VARIABLE. THEN I CALL "DECIDABLE SCOPING" THE ORDINARY (SYNTACTIC) CONCEPT OF STATIC SCOPING. A SEMANTIC CONCEPT OF STATIC SCOPING IS ALSO DEFINED, CORRISPONDING TO THE LAW WHEN ANY VARIABLE OCCURRENCE IS WITHIN THE SCOPE OF ONLY ONE BINDER (SEMANTIC BINDER). I SHOW SOME PROPOSITIONS STATING THAT A FREE VARIABLE WILL BE EVALUATED IN AN ENVIRONMENT WITHOUT BINDINGS FOR IT, SOMETIMES OR ALWAYS DEPENDING ON THE SCOPING OF THE LANGUAGES. THE MOST INTERESTING FACT IS A THEOREM STATING THAT A LANGUAGE WHATSOEVER (WITH THE ONLY RESTRICTION THAT IT MUST BE ABLE TO EXPRESS LAMBDA-ABSTRACTION) HAS STATIC SCOPING (IN THE SEMANTIC SENSE) IFF ANY SINGLE VARIABLE OCCURRENCE IN A PROGRAM IS ALWAYS EVALUATED IN SIMILAR ENVIRONMENTS (I.E. ENVIRONMENTS WITH THE SAME OPERATIONAL (LIST) STRUCTURE BUT WITH POSSIBLY DIFFERENT VALUES FOR VARIABLES). THIS THEOREM JUSTIFY THE USUAL TECHNIQUES OF ACCESS IN STATIC (AND DECIDABLE) SCOPING, AND SHOWS THAT DYNAMIC SCOPING IS ALWAYS

UNDECIDABLE, AND THAT DECIDABLE SCOPING IS ALWAYS STATIC.

TYPES ARE THE OTHER ISSUE EXPLORED IN THE SECOND PART. I MEAN PROCEDURAL (I.E. NOT ALGEBRAIC, A' LA SCOTT), TYPES AND CLASSES. THE PROBLEM IS PARTICULARLY IMPORTANT IN THE VIEW OF GIVING A TAU META-CIRCULAR INTERPRETER, WITHOUT FALLING IN CAR(CDR(CDR(CDR(TROUBLES)))). I MUST SAY I AM NOT FULLY SATISFIED WITH THE TYPE STRUCTURE OF TAU. IT IS VERY NICE FROM A PRACTICAL POINT OF VIEW (AS YOU MAY SEE IN [III.1]), BUT SEMANTICS DOES NOT LOOK PERFECTLY CLEAN. THE PROBLEM IS ABOUT CLASSES (DISJOINT UNION IS NOT ENOUGH) AND I WAS NOT ABLE TO FIND (IN LAMBDA CALCULUS MODELS) SOMETHING CORRISPONDING TO CLASSES OTHER THAN LABELING OF DOMAINS (MUST WE TRY WITH AIGEBRAS?).

IN THE THIRD PART OF MY THESIS I SHOW A TAU LAMBDA-INTERPRETER (THE STANDARD SEMANTICS), A TAU META-CIRCULAR INTERPRETER, AND A TAU ITERATIVE INTERPRETER. ACTUALLY IT IS NOT MY INTENTION TO IMPLEMENT TAU (I CONSIDER IT AS AN ATTEMPT TO CLARIFY SOME PROBLEMS) BUT I HAVE LEARNED MANY IMPLEMENTATION TRICKS (TRICKS?) ALONG THIS WORK ON D.S. AND I HAVE LEARNED WHAT ONE SHOULD LIKE TO HAVE.

I THINK THAT THE BASIC STEP TOWARD EASY HIGH LEVEL LANGUAGE IMPLEMENTATION IS TO HIDE THE HARD-MEMORY BY SIMULATING A HIGH LEVEL MEMORY SYSTEM. I AM CURRENTLY WRITING DOWN THE CODE OF AN INCREMENTAL ALL-PURPOSE GARBAGE COLLECTION SYSTEM BASED ON THE CHENEY (HEWITT-BAKER) ALGORITHM. THIS MAKES IT POSSIBLE TO DEFINE FEW BASIC DATA-FORMATS (IMMEDIATE VALUE, POINTER, RECORD, ARRAY OF IMMEDIATES, OF POINTERS, OF RECORDS) WHICH THE GARBAGE COLLECTOR KNOWS HOW TO PICK UP. A DATA TYPE MAY BE DECLARED AS HAVING SOME FORMAT. ONE

IMMEDIATELY GETS CONSTRUCTORS, SELECTORS AND DISCRIMINATORS FOR THAT DATA TYPE, WITH ALL THE CHECKS ON BOUNDS AND CONSISTENCE OF SUBSTRUCTURES. SO THE PROGRAMMER IS FREE FROM ANY MEMORY HANDLING FOR WHATEVER SYSTEM ONE WOULD CODE, AND THE PROGRAMMING EFFORT IS DRAMATICALLY REDUCED. THE CODE OF THE PROTOTYPE VERSION IS VERY COMPACT (SOMETHING LIKE 1K BYTES) AND IT IS EXPECTED TO COLLECT 8K PER SECOND, IN THE WORST CONDITIONS. IT WILL SOON RUN ON A ZILOG Z-80 MICROCOMPUTER AT I.S.I..

CONTENTS