Abstract. This chapter describes the structuring tool ISTST in further detail, as well as several of the smaller Fortran 77 transformation tools from Toolpack/1 Release 2. These smaller tools are oriented towards performing simple regular transformations on user programs which would be prone to error if done manually.

1. Introduction

The first part of this chapter describes the internal operation of the structuring tool ISTST. This tool was introduced in the Chapter “Fortran 77 Transformers”. The rest of this chapter describes several of the smaller Fortran 77 transformation tools included in the second release of Toolpack/1.

The smaller tools are:

1. a tool for making references to specific intrinsic functions into references to generic intrinsics (ISTGI),
2. a tool for rearranging expressions so that compilers need less stack space to compile them (ISTME),
3. a tool for joining and splitting strings in FORMAT statements (ISTJS), and
4. a tool for ensuring PARAMETER consistency throughout a whole program.
2. Internal Operation of ISTST

The processing by ISTST of each program-unit is split into the following three passes.

2.1. First Pass – Parse Tree Canonicalisation

The first pass of ISTST is performed by the library routine ZFCAPU (from the ACCESS supplementary library). This routine performs the following conversions:

1. Arithmetic IF statements are converted into logical IF statements (possibly with a following GOTO) except in the case where all three labels are different.
2. All DO loops are made to end on unique CONTINUE statements.
3. COMMENT statements are added to hold comment information immediately prior to statements which shall be removed during flowgraphing: that is, CONTINUE, unconditional GOTO, ENDIF and ELSE. (This is only done where there are actually some comments).
4. The extended data (see ZYSTXF and ZYGTXF) for each statement-level node in the parse tree is set to the original statement number for that statement.

2.2. Second Pass – Flowgraph Creation

- Flowgraphing Principles

A flowgraph is a representation of the executable part of a program. It is a directed graph with each node representing a single executable statement and arcs representing transfers of control. The arcs from a particular node are its outarcs and the arcs going into a particular node are its inarcs.

There are three types of statement which are not represented by flowgraph nodes: declarative statements, (unconditional) control-flow statements and null statements. These control-flow statements are the unconditional GOTO, ELSE and RETURN (without an alternate specifier). The null statements are CONTINUE and ENDIF.

There are four types of flowgraph node. These are:

1. The straight-line code node. This node has exactly one outarc. It represents any executable statement (except for END) which does not branch.
2. The exit node. This node has no outarcs. It represents the END statement.
3. The if node. This node has exactly two outarcs. It represents either an IF-conditional (except arithmetic IF) or a DO statement.
4. The case node. This node has at least two outarcs. It represents an arithmetic IF, computed GOTO, I/O statement with ‘ERR=’ and/or ‘END=’ specifiers, a subroutine call with alternate return specifiers or a dummy node inserted at the