Inter-Office Memorandum

To	Cedar Users	Date	December 18, 1981
From	Ed Satterthwaite	Location	Palo Alto
Subject	Cedar 7T10 Language and Compiler Update	Organization	CSL

XEROX

Filed on: [Indigo]<CedarDocs>Lang>Cedar7T10Update.Bravo

DRAFT

This memo summarizes the differences between the 7T7 and 7T10 versions of the Cedar language and compiler. It consists of extracts from the document [Indigo]<CedarDocs>Lang>Cedar7T10.press.

LANGUAGE CHANGES

Predeclared Types

To support the currently recommended Cedar standards, the types *BOOL*, *INT* and *CHAR* are predeclared, with the following definitions:

BOOL: TYPE = *BOOLEAN*;
CHAR: TYPE = *CHARACTER*;
INT: TYPE = LONG *INTEGER*;

Also, the definition of the predeclared type *CONDITION* has been changed. The default value for the timeout interval now is effectively infinite; i.e., a WAIT on a condition variable with default initialization will never time out. (The previous default provided a timeout after 100 ticks.) Use a runtime procedure such as *Process.SetTimeout* to change the default setting.

Rope Literals

The Cedar language now provides rope literals. Such a literal is denoted by a quoted string, e.g., "This is a rope literal". Its value is a reference to a rope object in the standard (counted) zone provided by the Cedar system.

The target type established by the context in which a quoted string literal appears determines the interpretation of that literal. There are three cases:

If the target type is *Rope.ROPE*, *Rope.Ref* or *Rope.Text*, the quoted string denotes a rope literal and has type *Rope.Ref*.

If the target type is any other REF type, the literal has type REF TEXT.

Otherwise, the literal has type STRING.

In the first case, the test is actually for equivalence between the target type and either REF *Rope.RopeRep* or REF *Rope.TextRep*. The matching is performed on the names of the interface (*Rope*) and referent type (*RopeRep* or *TextRep*), not on the structure of the referent type. Since this is a loophole in the type checking, use nonstandard versions of the *Rope* interface very cautiously.

Escape Convention for Literals

Cedar provides an escape convention to allow denotations of nonprinting characters in character and string literals (cf. the escape convention for the language C). The escape character is \, and the following codes are recognized:

Code	Interpretation	
\n, \N, \r, \R	*Ascii.CR*	
\t, \T	*Ascii.TAB*	
\b, \B	*Ascii.BS*	
\f, \F	*Ascii.FF*	
\l, \L	*Ascii.LF*	-- note that \n = LF in C
ddd	*ddd*C	-- where *d* is an octal digit, *ddd* < 377B
\\	\	
\’	’	
\"	"	

Anything else following a \ is an error.

You can use the escape convention in character literals (e.g., ’\n or ’\032) or string literals (e.g., "abc\ndef").

APPLY and RETURN

Cedar is based upon a model of interprocedural control transfer in which the construction of an argument record is clearly separated from the actual transfer of control. In the usual forms for specifying call or return, however, these operations are syntactically indivisible. There are now alternative syntactic forms that allow you to invoke transfer operations using already constructed argument records.

This extension is not fully general. The existing record must have a type compatible with the type required by the transfer operation, and the only types compatible with argument record types are other argument record types. Such types are defined implicitly by the definitions of transfer types, and they are always anonymous. Thus you cannot declare variables having such types, nor can you construct values with such types unless the target type is established by a transfer operation of some sort.

The operator APPLY is used to apply a value with some transfer type to an argument record. The syntactic form is

Call	::=	...
	|	APPLY [**Expression** , **Expression**]
	|	APPLY [**Expression** , **Expression** ! **CatchSeries**]

The type of the first **Expression** must be some transfer type (i.e., a type built using PROC, SIGNAL, ERROR, PROCESS, PORT or PROGRAM), and the second **Expression** must have a record type as good as the argument type required for the transfer (see below). The effect is to invoke the transfer operation appropriate to the type of the first **Expression**, i.e., to call a procedure, raise a signal, join a process, etc. The scope of the optional catch phrase is just the transfer itself.

Note that the first **Expression** implies a target type for the second, which can be (but normally would not be) a constructor. For example,

p[*x*, *y*]	can be written as	APPLY[*p*, [*x*, *y*]]	
q[*x*]	can be written as	APPLY[*q*, [*x*]]	-- not APPLY[*q*, *x*]

The corresponding forms for returning an existing record are

ReturnStmt	::=	...
	|	RETURN **Call**
	|	RETURN (**Expression**)
ResumeStmt	::=	...
	|	RESUME **Call**
	|	RESUME (**Expression**)

In these forms, the required type is established by the context in which the statement appears. The type of the **Call** or **Expression** must be a record type as good as the result type of the procedure body in which the **ReturnStmt** appears (or of the catch phrase in which the **ResumeStmt** appears).

An argument record type T_1 is as good as an argument record type T_2 if both of the following conditions are satisfied:

T_1 and T_2 have the same number of fields, say *n*.

For each *i*, $1 < i < n$, the type of the *i*-th component of T_1 is as good as the type of the *i*-th component of T_2; in addition, if both these components are named, the names are identical (i.e., names of field selectors must match, but an anonymous component matches any named component).

Note that this rule is more liberal than the rule for explicitly declared record types.

In the terminology of the Mesa 5 manual, T_1 is as good as T_2 iff T_1 conforms freely to T_2; e.g., [0..10) is as good as [0..100). In the new view of types, we would say that T_1 is as good as T_2 iff the predicate for T_1 implies the predicate for T_2.

In Cedar 7T10, the constructs described above do not work for empty argument records; i.e., you cannot nest applications of procedures taking/returning nothing.

Examples:

```
P1: PROC [x, y: INT] RETURNS [m, n: INT] = {...};

P2: PROC [m, n: INT] RETURNS [u, v: INT] = {...};

P3: PROC [a, b: INT] RETURNS [u, v: INT] = {
  RETURN APPLY[P2, P1[a, b]]};

i, j: INT;
. . .
[i, j] _ APPLY[P2, IF i < j THEN P1[i, j] ELSE [j, i]];
[i, j] _ APPLY[P3, [0, 0] ! s => {GOTO L}];          -- [i, j] _ P3[0, 0 ! s => {GOTO L}]
```

COMPILER CHANGES

Tioga Source Files

The compiler and binder ignore text in Tioga trailers. Any occurrence of a pair of NUL characters (characters with value 0C) in a source file marks the logical end of that source file.

File Locking

The Cedar 7T10 compiler is designed to be run under control of the system modeller. It also exports an interface allowing it to be run from Tajo or from the temporary Cedar executive. When it is run in this mode, the (rather minimal) facilities in PreCascade for obtaining exclusive access to a file are bypassed. Use caution.