Balsa Language Summary

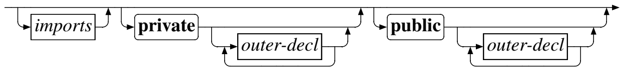
Andrew Bardsley

1. Top level and declarations

1.1. balsa-description

Each .balsa file contains a single balsa-description. That description may import definitions from other files and make a number of definitions of its own. Definitions in the ‘private’ part of the file are visible only to subsequent definitions in this file. Definitions in the ‘public’ part are exported to the .breeze file by balsa-c. Names become bound at point of definition and are visible to subsequent definitions in the file. Balsa has separate namespaces for procedure/function/shared procedures names, variables and channel, and types.

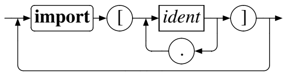
balsa-declaration



1.2. imports

Imports read in definitions from other files by parsing their (previously compiled) .breeze files. The dot separated paths represent file system paths from the root of the directories listed in the include search path defined by balsa-c. The first file to be found by matching against those path is the first to be included. Name conflicts between imported files are considered errors.

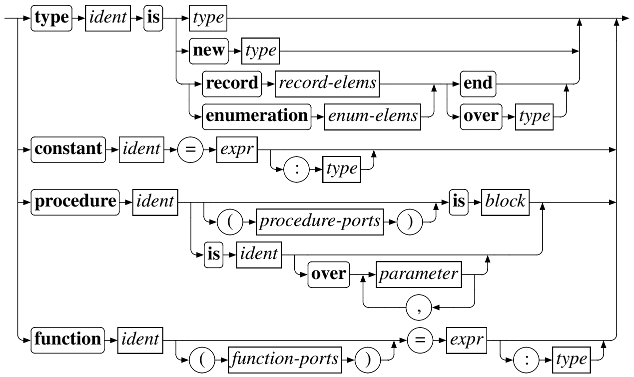
imports



1.3. outer-decl

At the top level of a balsa-description only the declaration/definition flavours given by outer-decl are allowed. The ‘over’ sections in record and enumeration declarations allow a bounding type to be given which determines (and restricts) the size of those types.

outer-decl

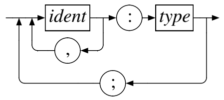


1.4. record-elems and enum-elems

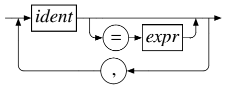
Record elements are packed in the given order into record typed values. The first element will occupy the least significant bits of a record typed value.

Enumeration elements get values starting from 0 and increasing by one from left to right. Elements with explicit values reset this counter to that value and so for a pair of elements: elem1 = 5, elem2; the value of element elem2 is 6.

record-elems



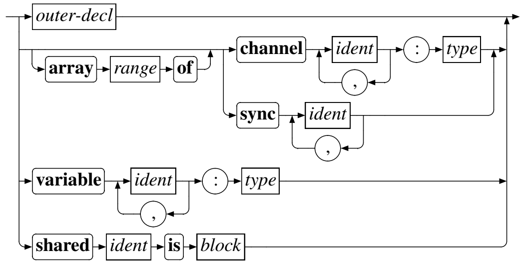
enum-elems



1.5. inner-decl

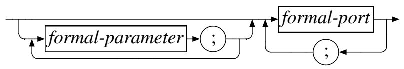
Channels, variables and shared procedures may only be declared inside procedures.

inner-decl

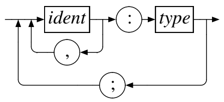


1.6. procedure-ports and function-ports

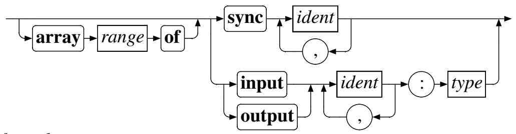
procedure-ports



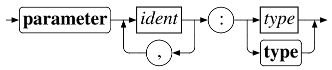
function-ports



formal-port



formal-parameter



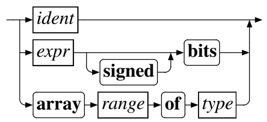
2. Expressions, types, ranges and lvalues

2.1. type and range

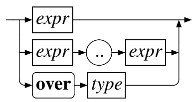
Balsa has two classes of anonymous types: numeric types (declared with the ‘bits’ keyboard) and arrays of other types. Numeric types can be either signed or unsigned (without the ‘signed’ keyboard). Signedness has an effect on expression operators and casting. Enumeration types and record types must be bound to names by a type declaration before use. Type equivalence in Balsa is usually by comparing points of declaration for named types, by size and signedness for numeric types and by size of range and base type for arrays. Only numeric types and arrays of other types may be used without first binding a name to those types.

Ranges are used for specifying the indices of elements of arrays, array slicing/element extraction, arrayed channels and in 'case' command patterns and 'for' loops. A single 'expr' value is equivalent to $0..expr-1$ in array and arrayed channels and to just the value of 'expr' in slice/array extract operations, 'case' patterns and 'for' loops. Ranges over a type are equivalent to a range $0..2^{sizeof\,type}$. For array elements and 'for' loops ranges are always ascending even when specified with the greater value on the left hand side (the two values are swapped). Ranges may be applied over numeric types and enumerations. All slice operations must have constant indices.

type



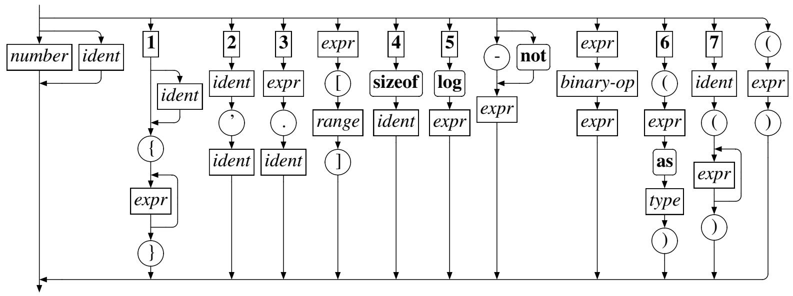
range



2.2. expr

1. Record/array construction. The optional ident specifies a type where this isn't obvious from context
2. Enumeration element choice: e.g. Colours'Red. Where the enumeration type is obvious from context, the type name can be omitted: e.g. Red
3. Record element extraction
4. Number of bits in type ident. Must be resolvable at compile time.
5. $\lceil \log_2 ''expr \rceil$. Must be resolvable at compile time.
6. Type cast operator, the parentheses are required. All value except signed numeric values are cast by truncation or zero padding. Signed numeric values cast into wider signed numeric types are sign extended. The only implicit type coercion in Balsa is the extension of numeric literals and constant expressions.
7. Function call expression.

expr



2.3. binary/unary-operators

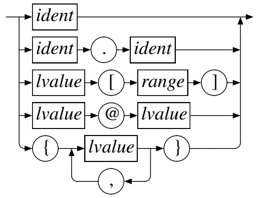
Operator are shown in order of decreasing precedence.

Symbol	Operation	Valid types	Notes
[ ]	array indexing	array	non-const index possible, can generate **lots** of hardware
.	record indexing	record	
not, log, unary -	unary ops.	numeric	log only works on constants, - makes returns a result one bit wider than the argument
+, -	add, subtract	numeric	results are between one and two bits longer than largest argument (two bits for signed +/- unsigned with a larger or equally sizes unsigned argument
*, /, %	multiply, divide, remainder	numeric	only applicable to constants
<, >, <=, >=	inequalities	numeric, enumeration	
=, /=	equals, not equals	all	comparison is by sign extended value for signed numeric types
and	logical/bitwise and	numeric	Balsa uses type '1 bits' for 'if'/'while' guards so bitwise and logical operators are the same
or	logical/bitwise or	numeric	

2.4. lvalue

The variable naming clause on the left hand side of assignments, the channel naming clauses on either side of channel I/O commands and the formal arguments to procedures are grouped together under the term lvalues. Lvalues name variables, channels, record elements of variables, slices/elements of variables of array types and the slices, construction and concatenation (with the '@' symbol) of arrayed channels used in procedure calls. The left hand array will occupy the low index portion of resulting array.

lvalue

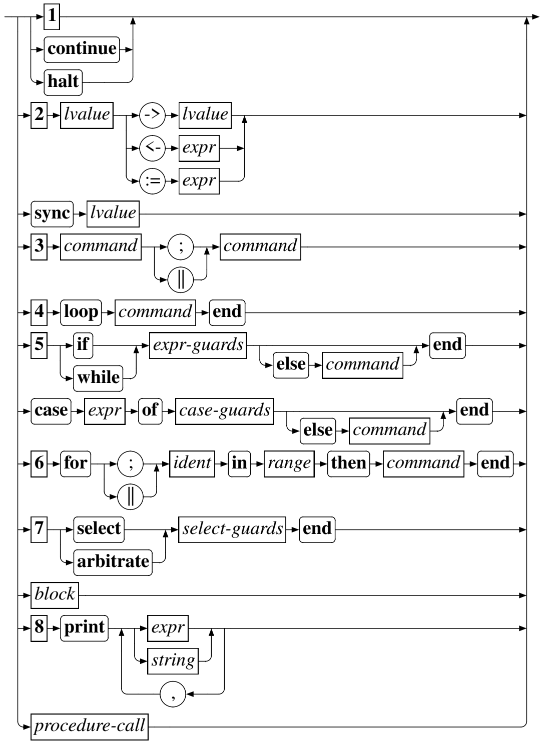


3. Commands

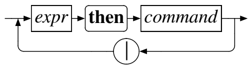
3.1. command

1. 'continue' is the null command, 'halt' causes deadlock.
2. Channel I/O and variable assignment. '->' is channel input, '<-' is channel output.
3. '||' binds tighter than ';', use blocks to override precedence.
4. Infinite loop.
5. 'if' and 'while' commands may have multiple guarded commands, only one of which is executed when its guard is met.
6. 'for' performs structural iteration/composition similar to 'for generate' in VHDL. The iteration range must be resolvable at compile time.
7. 'select' and 'arbitrate' perform passive input selection, 'select' without arbitration, 'arbitrate' with arbitration. A 'select' may apply to any number of channel sets, 'arbitrate' must be applied to exactly two sets. Selected channels are available as read-only variables within the guarded commands. Where a channel is an element of an arrayed channel, it must be given an ident alias for use within the guarded command. This is done using the lvalue -> ident aliasing syntax.
8. 'print' can be used to give diagnostic messages during compilation. Any Balsa value can be passed to print but if the first element is one of the values of the built-in enumerated type BalsaError, this is used to specify what type of message is given. BalsaError is defined as {fatal, error, warning, report}.

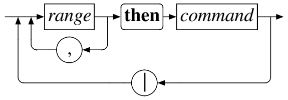
command



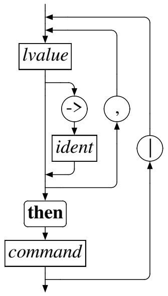
expr-guards



case-guards



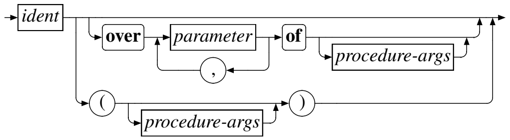
select-guards



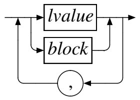
3.2. procedure-call

The ‘over’ procedure call syntax is used to call parameterised procedures. Pre-parameterised versions of such procedures can be bound to names using a ‘procedure’ declaration (see outer-decl). A block can be given in place of a sync channel name allowing the called procedure to activate that block with a handshake on that port.

procedure-call



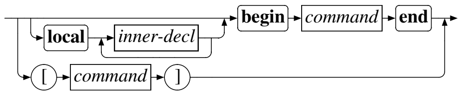
procedure-args



3.3. block

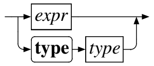
Blocks allow inclusion of local definitions around a command and the overriding of the precedence of command composition.

block



3.4. parameter

parameter

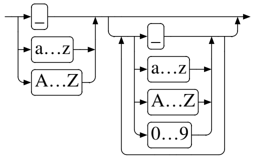


4. Terminals and comments

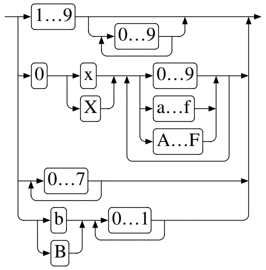
The terminals may have no extra whitespace inserted in them. Numbers can be given in decimal (starting with one of 1…9), hexadecimal (0x prefix), octal (0 prefix) and binary (0b prefix). Identifiers follow rules similar to C, they are case sensitive and may not be identical to any of the predefined keywords.

There are two forms of comments: line comments begun by ‘--’ and continuing to the end of the current line, and block comments enclosed the symbols ‘(--’ and ‘--)’. Block comments are nestable.

ident



number



string