

# ASL Reference Progress

## DDI 0623

Arm Architecture Technology Group

September 25, 2024



# Contents

<b>1</b>	<b>Non-Confidential Proprietary Notice</b>	<b>5</b>
<b>2</b>	<b>Disclaimer</b>	<b>7</b>
<b>3</b>	<b>Not Implemented by ASLRef</b>	<b>9</b>
3.1	Syntax . . . . .	9
3.1.1	Reserved Keyword . . . . .	9
3.1.2	Pragmas . . . . .	9
3.1.3	Declaring Multiple Identifiers Without Initialization . . . . .	9
3.1.4	Annotations . . . . .	9
3.1.5	Recursion Limits . . . . .	9
3.1.6	Concatenation Declarations . . . . .	10
3.1.7	Guards . . . . .	10
3.1.8	Catching Operator Precedence Errors . . . . .	10
3.2	Semantics . . . . .	10
3.2.1	Enforcing Loop Limits . . . . .	10
3.2.2	Non-main Entry Point . . . . .	10
3.3	Typing . . . . .	11
3.3.1	Throwing Exceptions without Braces . . . . .	11
3.3.2	Checking that All Paths in a Function Return a Value . . . . .	11
3.3.3	Side-effect-free Subprograms . . . . .	11
3.3.4	Statically evaluable programs . . . . .	12
3.3.5	Restriction on Use of Parameterized Integer Types . . . . .	12
<b>4</b>	<b>Issues Not Yet Addressed by the References</b>	<b>13</b>
4.1	Semantics . . . . .	13
4.1.1	Standard Library and Primitives . . . . .	13
4.2	Typing . . . . .	13
4.2.1	Checking Type Annotations for Absence of Side Effects . . . . .	13
4.2.2	Enumeration Labels . . . . .	13
4.2.3	Calls to Setters and Getters . . . . .	13
4.3	Side-Effects . . . . .	14



# Chapter 1

## Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, any patents, copyrights, trade secrets, trademarks, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof

is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of this document shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm’s trademark usage guidelines at

<https://www.arm.com/company/policies/trademarks>.

Copyright © [2023,2024] Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England. 110 Fulbourn Road, Cambridge, England CB1 9NJ. (LES-PRE-20349)

## Chapter 2

# Disclaimer

This document is part of the ASLRef material.

This material covers ASLv1, a new, experimental, and as yet unreleased version of ASL.

The development version of ASLRef can be found here:

<https://github.com/herd/herdtools7>.

A list of open items being worked on can be found here:

<https://github.com/herd/herdtools7/blob/master/asllib/doc/ASLRefProgress.tex>.

This material is work in progress, more precisely at Alpha quality as per Arm's quality standards. In particular, this means that it would be premature to base any production tool development on this material.

However, any feedback, question, query and feature request would be most welcome; those can be sent to Arm's Architecture Formal Team Lead Jade Alglave ([jade.alglave@arm.com](mailto:jade.alglave@arm.com)) or by raising issues or PRs to the herdtools7 github repository.





## Chapter 3

# Not Implemented by ASLRef

This chapter describes what is not yet present in the executable version of ASLRef (Build #215 from Feb 22, 2024).

### 3.1 Syntax

#### 3.1.1 Reserved Keyword

`pattern` should be a reserved keyword.

#### 3.1.2 Pragmas

ASLRef does not currently parse pragmas:

```
pragma asl_pragma1;
```

#### 3.1.3 Declaring Multiple Identifiers Without Initialization

The following simultaneous declaration of three global variables does not currently parse with ASLRef.

```
var x, y, z : integer;
```

The same line does parse and correctly handled inside a subprogram.

#### 3.1.4 Annotations

ASLRef does not yet support annotations in general. Loop limit annotations are supported, but recursion limit annotations are not yet supported.

#### 3.1.5 Recursion Limits

ASLRef does not yet parse and support `@recurselimit(<LIMIT>)` annotations.

### 3.1.6 Concatenation Declarations

Declarations of multiple bitvectors via concatenation as in the program

```
func main() => integer
begin
  var [ a[7:0], b, c[3:0] ] = Zeros(13);
  return 0;
end
```

do not currently parse.

### 3.1.7 Guards

Guards are used on **case** and **catch** statements, to restrict matching on the evaluation of a boolean expression. They are not yet implemented in ASLRef.

### 3.1.8 Catching Operator Precedence Errors

ASLRef does not yet catch the following types of errors nor are they defined in the syntax reference.

Operator precedence is used to disambiguate binary expressions.

Given two binary operators *op1* and *op2*, an expression of the form *x op1 y op2 z*, is interpreted as (*x op1 y*) *op2 z* if *op1* has higher precedence than *op2* or as *x op1 (y op2 z)* if *op1* has lower precedence than *op2*. If *op1* is associative and *op1* = *op2* then the expression may be interpreted as either (*x op1 y*) *op2 z* or as *x op1 (y op2 z)* since there is no difference. Otherwise it is a static operator precedence error.

The following operators are associative: + \* && || AND OR XOR

Precedence Class Operators		
1 (Highest)	Membership	IN
2	Unary	- ! NOT
3	Power	^
4	Mul-Div-Shift	* / DIV DIVRM MOD << >>
5	Add-Sub-Logic	+ - AND OR XOR
6	Comparison	== != > >= < <=
7 (Lowest)	Boolean	&&    --> <->

## 3.2 Semantics

### 3.2.1 Enforcing Loop Limits

@looplimit annotations are type-checked but not enforced for **while** and **repeat** loops.

### 3.2.2 Non-main Entry Point

Currently ASLRef only supports **main** as an entry point.

## 3.3 Typing

### 3.3.1 Throwing Exceptions without Braces

In the following example, the commented out `throw` statement should type-check, but it currently fails.

```
type except of exception;

func main() => integer
begin
  // throw except; // Should type-check
  throw except{}; // Okay

  return 0;
end
```

### 3.3.2 Checking that All Paths in a Function Return a Value

The following function currently passes type-checking even though it does not return a value when `a <= 7`.

```
func foo(a : integer) => integer
begin
  if (a > 7) then
    return 0;
  end
end
```

This requires a control-flow analysis.

A call to `Unreachable` needs to indicate to the control-flow analysis that it is okay for the `else` path to not return a value.

```
func foo(a : integer) => integer
begin
  if (a > 7) then
    return 0;
  else
    Unreachable();
  end
end
```

### 3.3.3 Side-effect-free Subprograms

ASLRef does yet infer whether a subprogram is side-effect-free. Therefore, there are no checks that expressions are side-effect-free when those are expected, for example, in `for` loop ranges.

### 3.3.4 Statically evaluable programs

Side effects analysis has not been implemented yet. This makes detection of statically evaluable subprograms impossible.

Furthermore, non-execution time subprograms, expressions, and types have not been implemented.

### 3.3.5 Restriction on Use of Parameterized Integer Types

#### As storage types

Restrictions on the use of parameterized integer types as storage element types are not implemented.

#### as Expression With a Constrained Type

Restriction on the use of parameterized integer types as left-hand-side of a Asserted Typed Conversion is not implemented in ASLRef. For example, the following will not raise a type-error:

```
func foo {N} (x: bits(N)) => integer {0..2*N}
begin
  return N as integer {0..2*N};
end
```

## Chapter 4

# Issues Not Yet Addressed by the References

### 4.1 Semantics

#### 4.1.1 Standard Library and Primitives

The standard library is not yet defined by a reference.

### 4.2 Typing

#### 4.2.1 Checking Type Annotations for Absence of Side Effects

Type annotations that contain expressions must ensure that those expressions are side-effect-free. This is currently ensured by disallowing such expressions to contain call expressions. Allowing side-effect-free calls is being considered.

#### 4.2.2 Enumeration Labels

Enumeration labels are not yet treated as first-class literals, but rather as integer literals. The domain of an enumeration type should be a set of enumeration labels.

#### 4.2.3 Calls to Setters and Getters

The replacement of implicit calls to getters and setters (written for example as slices) to explicit calls to subprograms has not been transliterated. In terms of abstract syntax, this corresponds to the translation between a `E.Slice` and a `E.Call`.

### 4.3 Side-Effects

Side-Effects are not yet defined by the references.