AspectMATLAB Specification (Syntax)

June 4, 2016

Preliminarys

Reserved Keywords

AspectMatlab has all keywords for MATLAB and expands it with the following keywords:

aspect	Denote the beginning of a new aspect
patterns	Start the pattern section within an aspect
actions	Start the action section within an aspect
set	The set pattern
get	The get pattern
call	The call pattern
execution	The execution pattern
mainexecution	The mainexecution pattern
loop	The loop pattern
loopbody	The loopbody pattern
loophead	The loophead pattern
annotate	The annotate pattern
op	The operator pattern
within	The within pattern
dimension	The dimension pattern
istype	The istype pattern

Wildcards

⟨*wildcards*⟩	::= '*****' | '**..**'

*****	Will match to exactly one of elements suitable in its position
..	Will match any number (zero or more) of elements suitable in its position

Type Signature (type information)

⟨*typeSignature*⟩	::=	[A-Za-z0-9]+
	|	⟨*wildcards*⟩

AspectMatlab use type signature to capture the types for set/get pattern, call pattern and execution. The type signature should be a valid string literal in MATLAB, which will be compared to the return value of built-in function *class*. For example, "function_handle" is a valid type type signature. AspectMatlab will not check wether the type signature is a valid signature at compile-time nor run-time (i.e. AspectMatlab will not check whether such type exist). One possible implementation is simply use the type signature compare with the return value of *class* function.

Dimension Signature (Shape information)

Inspired by the dimension pattern we add dimension signature to capture the input arguments and variables.

⟨*dimensionList*⟩	::=	[⟨*wildcards*⟩ | ⟨*integerLiteral*⟩]
	|	⟨*dimensionList*⟩ ',' [⟨*wildcards*⟩ | ⟨*integerLiteral*⟩]
⟨*dimensionSignature*⟩	::=	'[' ⟨*dimensionList*⟩ ']'

For example, $[1, 2]$ will match to any matrix with shape 1×2. Wildcards are allowed in the shape signature, e.g. $[.., 3]$ will match to matrix with dimension of 1×3 and $2 \times 3 \times 3$. Wildcards are allowed in any place in the Dimension Signature. AspectMatlab Compiler should raise a warning for any redundant signature, e.g. $[1, .., .., 2]$ which could simply write as $[1, .., 2]$.

Variable Signature

⟨*variableSignature*⟩	::=	⟨*dimensionSignature*⟩ ⟨*typeSignature*⟩
	|	⟨*dimensionSignature*⟩
	|	⟨*typeSignature*⟩

For example, $[1, 2]$uint32 will match to any argument or variable with dimension of 1×2 and type of unsigned 32-bit integer, and $[1, 3]\star$ will match to any argument or variable with shape of 1×3 (the star will match to any shape).

Patterns

Set Patterns (set)

A set pattern will match to the action of "writing" a matrix in MATLAB code.

⟨*setPattern*⟩	::=	'set' '(' ⟨*identifier*⟩ ')'
	|	'set' '(' ⟨*wildcards*⟩ ')'
	|	'set' '(' ⟨*identifier*⟩ ':' ⟨*variableSignature*⟩ ')'
	|	'set' '(' ⟨*wildcards*⟩ ':' ⟨*variableSignature*⟩ ')'

The first two type of set pattern is used to support old AspectMatlab, and they will only match on the name of variable (ignoring the shape and type information). The last two type of set pattern will not only match on the name of the variable but also on the shape and type of the variable. Pattern *set(a)* is equivalent to *set(a:⋆)*.

Get Pattern (get)

Similar with a set pattern, a get pattern will match to the action of "reading" from a matrix in MATLAB code.

⟨*getPattern*⟩ ::= 'get' '(' ⟨*identifier*⟩ ')'
| 'get' '(' ⟨*wildcards*⟩ ')'
| 'get' '(' ⟨*identifier*⟩ ':' ⟨*variableSignature*⟩ ')'
| 'get' '(' ⟨*wildcards*⟩ ':' ⟨*variableSignature*⟩ ')'

Note A weeding phase need to be add to this pattern, we only allow to use ⋆ pattern in the *wildcards* section. The AspectMatlab should reject any pattern using .. in the *wildcards* section.

Call Pattern (call)

Call patter will match to function or subroutine calling within MATLAB code. Unlike *execution* pattern, call pattern match the calling, and weaved code should be inserted in the caller.

⟨*callArgSignature*⟩ ::= ⟨*empty*⟩
| ⟨*callArgumentSignature*⟩ ',' ⟨*variableSignature*⟩

⟨*callPattern*⟩ ::= 'call' '(' ⟨*identifier*⟩ '(' ⟨*callArgSignature*⟩ ')' ')'
| 'call' '(' ⟨*wildcards*⟩ '(' ⟨*callArgSignature*⟩ ')' ')'
| 'call' '(' ⟨*identifier*⟩ '(' ⟨*callArgSignature*⟩ ')' ':' ⟨*variableSignature*⟩ ')'
| 'call' '(' ⟨*wildcards*⟩ '(' ⟨*callArgSignature*⟩ ')' ':' ⟨*variableSignature*⟩ ')'

Note A weeding phase need to be add to this pattern, we only allow to use ⋆ pattern in the *wildcards* section. The AspectMatlab should reject any pattern using .. in the *wildcards* section.

Note A weeding/analysis phase need to be add to the third or fourth variation of call pattern. Instead of being able of weaving before, after, around, the last two variation of pattern can only use after pattern. (As we can not decide the return type nor the return shape of a function or a subroutine)

Execution Pattern (execution)

Like the call pattern, the execution match the execution body in the function or subroutine. Unlike the call pattern which insert the weaved code in the caller body, the execution should insert the weaved code in the callee body. The execution pattern should support matching depend on the shape or type information of input parameters.

⟨*execArgSign*⟩ ::= ⟨*empty*⟩
| ⟨*execArgSign*⟩ ',' ⟨*variableSignature*⟩

⟨*executionPattern*⟩ ::= 'execution' '(' ⟨*identifier*⟩ '(' ⟨*execArgSign*⟩ ')' ')'
| 'execution' '(' ⟨*wildcards*⟩ '(' ⟨*execArgSign*⟩ ')' ')'
| 'execution' '(' ⟨*identifier*⟩ '(' ⟨*execArgSign*⟩ ')' ':' ⟨*variableSignature*⟩ ')'
| 'execution' '(' ⟨*wildcards*⟩ '(' ⟨*execArgSign*⟩ ')' ':' ⟨*variableSignature*⟩ ')'

Note A weeding phase need to add to the execution pattern, we only allow the *wildcards* to be ⋆, the AspectMATLAB Compiler should reject the .. wildcard.

Note Like the third and fourth variation of the call pattern, the third and fourth variation of the execution pattern can only be weaved in after mode, the compiler should reject any attempt to weave the code in before and around mode. (Same reason as call pattern, we cannot decide the return type and shape information without executing the whole function)

Main Execution Pattern (mainexecution)

The *mainexecution* pattern will capture the main function or script of a set of MATLAB file. It cannot be a part of a compound pattern.

⟨*mainexecPattern*⟩ ::= '`mainexecution`' '`(`' '`)`'

Note A weeding phase need to be added to this pattern, in order to make sure that it should not be a part of a compound pattern.

Loop Pattern (loop)

The old AspectMatlab Compiler has loop pattern designed for for loop only (it matches the for loop by the name of the loop control variable). However, we would like to extends the AspectMatlab to capture *while* loop. Unlike the for loop which has a named control variable, the while loop may become anonymous.

Inspired by the annotation pattern, we add a reserved annotation pattern in order to allow the user to mark the name of the while loop.

```
%@ loopname <identifier>
```

⟨*loopType*⟩ ::= '`for`'
| '`while`'
| ⟨*wildcards*⟩

⟨*loopPattern*⟩ ::= '`loop`' '`(`' [⟨*wildcards*⟩ | ⟨*identifier*⟩] '`)`'
| '`loop`' '`(`' ⟨*loopType*⟩ '`:`' [⟨*wildcards*⟩ | ⟨*identifier*⟩] '`)`'

The first alternation of loop pattern is used to support backward compatibility. It will only match on the identifier and ignore the loop type information. In the new AspectMATLAB, we introduce the "full" loop pattern which allow the user to specific which type of loop pattern they would like to match.

Loop Body pattern (loopbody)

Similar with the loop pattern, we extends the loopbody pattern as following.

⟨*loopBodyPattern*⟩ ::= '`loopbody`' '`(`' [⟨*wildcards*⟩ | ⟨*identifier*⟩] '`)`'
| '`loopbody`' '`(`' ⟨*loopType*⟩ '`:`' [⟨*wildcards*⟩ | ⟨*identifier*⟩] '`)`'

Both type of loop support body pattern match, however, the content exposure may be different for two different, which will be discussed later.

Loop Head pattern (loophead)

We also extands the *loophead* pattern.

⟨*loopHeadPattern*⟩ ::= '`loophead`' '`(`' [⟨*wildcards*⟩ | ⟨*identifier*⟩] '`)`'
| '`loophead`' '`(`' ⟨*loopType*⟩ '`:`' [⟨*wildcards*⟩ | ⟨*identifier*⟩] '`)`'

Annotate Pattern (annotate)

⟨*annotationSelectorPrimitive*⟩ ::= '`var`'
| '`str`'
| '`num`'
| ⟨*wildcard*⟩

⟨*annotationSelectorListPrimitive*⟩ ::= ⟨*annotationSelectorPrimitive*⟩
| ⟨*annotationSelectorListPrimitive*⟩ '`,`' ⟨*annotationSelectorPrimitive*⟩

⟨*annotationSelector*⟩ ::= ⟨*annotationSelectorPrimitive*⟩
| '[' ⟨*annotationSelectorListPrimitive*⟩ ']'

⟨*annotationSelectorList*⟩ ::= ⟨*empty*⟩
| ⟨*annotationSelector*⟩ ',' ⟨*annotationSelectorList*⟩

⟨*annotatePattern*⟩ ::= 'annotate' '(' ⟨*identifier*⟩ '(' ⟨*annotationSelectorList*⟩ ')' ')'
| 'annotate' '(' ⟨*wildcard(*)*⟩ '(' ⟨*annotationSelectorList*⟩ ')' ')'

We will leave the annotation in the source code as a raw string, and parse it when it needed (i.e. dynamic parsing?) [TODO HERE]

Operator Pattern (op)

⟨*supportOperator*⟩ ::= ⟨*plus(+)*⟩
| ⟨*minus(-)*⟩
| ⟨*times(.*)*⟩
| ⟨*rdivide(./)*⟩
| ⟨*ldivide(.“)*⟩
| ⟨*power(.^)*⟩
| ⟨*mtimes(*)*⟩
| ⟨*mrdivide(/)*⟩
| ⟨*mldivide(“)*⟩

⟨*opPattern*⟩ ::= 'op' '(' ⟨*supportOperator*⟩ ')'
| 'op' '(' ⟨*variableSignature*⟩ ':' ⟨*supportOperator*⟩ ')'
| 'op' '(' ⟨*supportOperator*⟩ ':' ⟨*variableSignature*⟩ ')'
| 'op' '(' ⟨*variableSignature*⟩ ':' ⟨*supportOperator*⟩ ':' ⟨*variableSignature*⟩ ')'

In the new AspectMATLAB, we would like to extend the operator pattern by allowing the user to specify the type and shape information on the both side of the operator. The first alternation of operator pattern is used for backward compatibility, which only match on the operator while the second and third alternation is used to match on side of the operator (second alternation to match the left side and the third alternation to match the right side). The last alternation is considered the "full" operator pattern, which allow the user to specify both side of the operator.

Scope Pattern (within)

⟨*scopeType*⟩ ::= 'function'
| 'script'
| 'class'
| 'aspect'
| 'loop'
| ⟨*wildcards(*)*⟩

⟨*scopePattern*⟩ ::= 'within' '(' ⟨*scopeType*⟩ ':' [⟨*wildcards(*)*⟩ | ⟨*identifier*⟩] ')'

The function, class and aspect can be specify via the name identifier, the script can be specify via file name and the loop can be specify via the iterate variable name or the name annotation.

Dimension Pattern (dimension)

Inherent from the old AspectMatlab Compiler.

⟨*dimensionPattern*⟩ ::= '`dimension`' '`(`' ⟨*dimensionSignature*⟩ '`)`'

Type Pattern (istype)

Inherent from the old AspectMatlab Compiler.

⟨*typePattern*⟩ ::= '`istype`' '`(`' ⟨*typeSignature*⟩ '`)`'

Compound Pattern

A compound pattern is a pattern built on a set of patterns linked with logic operator.

⟨*compoundPattern*⟩ ::= ⟨*setPattern*⟩
| ⟨*getPattern*⟩
| ⟨*callPattern*⟩
| ⟨*executionPattern*⟩
| ⟨*mainexecPattern*⟩
| ⟨*loopPattern*⟩
| ⟨*loopBodyPattern*⟩
| ⟨*loopHeadPattern*⟩
| ⟨*annotatePattern*⟩
| ⟨*opPattern*⟩
| ⟨*scopePattern*⟩
| ⟨*typePattern*⟩
| ⟨*dimensionPattern*⟩
| ⟨*identifier*⟩
| '`(`' ⟨*compundPattern*⟩ '`)`'
| ⟨*compoundPattern*⟩ '`&`' ⟨*compoundPattern*⟩
| ⟨*compoundPattern*⟩ '`|`' ⟨*compoundPattern*⟩
| '`~`' ⟨*compoundPattern*⟩

Note not all compound patterns are valid pattern, we need add additional weaving and analysis procedure before accept the input. (e.g. boundness check, weavablity check)

Actions

⟨*weavedType*⟩ ::= '`before`'
| '`after`'
| '`around`'

⟨*exposureList*⟩ ::= ⟨*empty*⟩
| ⟨*identifier*⟩ '`,`' ⟨*exposureList*⟩

⟨*action*⟩ ::= ⟨*identifier*⟩ '`:`' ⟨*compoundPattern*⟩ '`:`' '`(`' ⟨*exposureList*⟩ '`)`'
⟨*MATLABCode*⟩
'`end`'

Unlike the old AspectMATLAB, we allow both anonymous patterns and defined patterns. Instead of check the exposure keyword at parsing time, we leave the check to the weeding part (i.e. we need additional analysis information on the pattern in order to decide the available selection). Also, not all patterns support all weave type, analysis and weeding are needed here.

Aspect

⟨*patternEntry*⟩ ::= ⟨*identifier*⟩ '`:`' ⟨*compoundPattern*⟩ ⟨*statementTerminator*⟩

⟨*patternSection*⟩ ::= '`patterns`'
⟨*patternEntry*⟩*
'`end`'

⟨*actionSection*⟩ ::= '`actions`'
⟨*action*⟩*
'`end`'

⟨*aspectContent*⟩ ::= ⟨*patternSection*⟩
| ⟨*actionSection*⟩
| ⟨*propertiesSection*⟩
| ⟨*methodsSection*⟩
| ⟨*eventsSection*⟩
| ⟨*enumerationSection*⟩

⟨*aspect*⟩ ::= '`aspect`' ⟨*identifier*⟩
⟨*aspectCotent*⟩* '`end`'