

École nationale supérieure des mines de Nantes
ASCOLA Research Group



pyAAL User Guide

Version 0.1

May 27, 2015

Walid BENGHABRIT

Supervisor : **Prof. Jean-Claude Royer**
Co-Supervisor : **A/Prof. Hervé Grall**



Contents

Table of contents	1
1 Getting started	2
1.1 Install AccLab	2
1.2 Using AAL compiler "aalc"	2
1.3 Writing your first AAL program	2
1.4 Using core libraries	4
1.5 Using the shell	5
2 AAL language	8

1 Getting started

1.1 Install AccLab

- To use the ltl prover, you need to put the following executable files (tspass, fotl-translate) in tools/your_platform/ (linux/mac/win)
- Basic run : python aalc.py (you need python3.4.0 or greater)
- Run an AAL file : python aalc.py -i testfile.aal

1.2 Using AAL compiler "aalc"

Listing 1: aalc options

```
aalc
-h  --help          display this help and exit
-i  --input         the input file
-o  --output        the output file
-c  --compile       compile the file, that can be loaded after using -l
-m  --monodic       apply monodic check on aal file
-s  --shell         run a shell after handling aal program
-k  --check         perform a verbose check
-l  --load          load a compiled aal file (.aalc) and run a shell
-t  --ltl           translate the aal program into FOTL
-r  --reparse       reparse tspass file
-r  --recompile     recompile the external files
-b  --no-colors     disable colors in output
-x  --compile-stdlib compile the standard library
-d  --hotswap       enable hotswaping (for development only)
-a  --ast           show ast tree
```

1.3 Writing your first AAL program

Let consider the following senario : we have three actors :

- cloud storage service : let call it **css** which is a cloud service provider
- alice and bob : an end users that uses css service

The **css** offers the following services : read (a user reads some data form css server), store (a user stores some data into css server), delete (a user deletes some data from css server). **css** allows users to read/store/delete only their data on his server, and don't allow them to read other customers data. **css** can also read and delete any data from his server.

Alice want to check if **css** policy respect her privacy. Typically she want to know if she is allowed to performs some actions and if bob can read here data.

a. **Declaring services** The services are the

```
SERVICE read
SERVICE store
SERVICE delete
```

b. **Declaring actors** : first we need to declare our actors

1. GETTING STARTED

```
// Agents declaration
AGENT alice
AGENT bob
AGENT css
```

- c. **Linking services and actors** First we need to declare our actors

```
AGENT alice TYPES() REQUIRED(read store delete) PROVIDED()
AGENT bob   TYPES() REQUIRED(read store delete) PROVIDED()
AGENT css   TYPES() REQUIRED() PROVIDED(read store delete)
```

- d. **Defining policies** First we need to declare our actors

```
/*
 *
 */
CLAUSE css_policy (

)

/*
 *
 */
CLAUSE alice_policy (

)
```

- e. **Writing checks** First we need to declare our actors

```
CALL validate("'css_policy'" "'alice_pref'")
```

1.4 Running the program

- Run an AAL file (by default, it)

```
aalc -i tests/tuto0.aal
```

```
[WARNING] Agents declarations missing !
-> cloudS at line 20
-> cloudY at line 33
-> leslie at line 20
[WARNING] Services declarations missing !
-> read at line 13
-> notify at line 19
-> sensors at line 17
-> write at line 15
-> log at line 36
-> audit at line 20
-> sanction at line 21
-> delete at line 16
-> storage at line 3
[WARNING] Types declarations missing !
-> data at line 11
-> DataSubject at line 2
-> agent at line 12
-> DataProcessor at line 3
```

```
Execution time : 0.26976990699768066
```

1. GETTING STARTED

- Perform an advanced check (need to be completed)

```
aalc -i tests/tuto1.aal -k

----- Checking Monodic test -----

** DECLARATIONS
[DECLARED AGENTS] : 5
[DECLARED SERVICES] : 13
[DECLARED DATA] : 0
[DECLARED TYPES] : 4

*** Forwards references check...
[AGENTS] : 3
[SERVICES] : 9
[DATA] : 0
[TYPES] : 4

** CLAUSES
[CLAUSES] : 2
Monodic test :
Formula is monodic !
----- Checking Monodic End -----
```

- Perform monodic test on all clauses :

```
aalc -i tests/tuto1.aal -m

----- Checking Monodic test -----
Formula is monodic !
----- Checking Monodic End -----
```

- Translate aal program into FOTL (in tspass syntax):

```
aalc -i tests/tuto1.aal -t

----- FOTL Translation start -----
! [d] ! [a] (((((((PERMIT(read, kim, cloudX, d) & PERMIT(read, kim, cloudX, d)) & PERMIT(write, kim
, cloudX, d)) & PERMIT(delete, kim, cloudX, d)) & PERMIT(sensors, cloudX, kim, d)) & (~
threeYears => Adelete(cloudX, d))) & ( Aread(a, cloudX, d) => sometime(Anotify(cloudX, kim
)))))) ! [a] ! [d] ! [b] (((((((PERMIT(read, a, cloudX, d) & PERMIT(read, a, cloudX, d)) &
PERMIT(write, a, cloudX, d)) & PERMIT(delete, a, cloudX, d)) & PERMIT(sensors, cloudX, a, d
)) & PERMIT(storage, cloudX, cloudY, d)) & (~twoYears => Adelete(cloudX, d))) & (Aread(b,
cloudX, d) => sometime(Anotify(cloudX, a)))) & (Astorage(cloudX, cloudY, d) => sometime(
Alog(cloudX, cloudX))))))
----- FOTL Translation end -----
```

1.5 Using core libraries

You can load external AAL files using `LOAD "aal_file"`(without the extension)

core.macros Contains the basic types declarations (DataSubject, DataController, DataProcessor, ...)

```
// Loading libraries
LOAD "core.types"
```

1. GETTING STARTED

core.macros Contains some basic macros.

```
// Loading libraries
LOAD "core.macros"

/** ltl check**/
CALL ltl_check()

// Checking validity c1 => c2
CALL validate(c1 c2) (

//
CALL resolve(c1 c2)
'NOT_YET_IMPLEMENTED'

// Show the loaded libraries used in the current AAL program
CALL show_libs()

// Translate
CALL ltl(c)

CALL show_clause(c)

CALL to_natural(c)
```

core.eu Contains the basic types

```
// Loading libraries
LOAD "core.eu"

/** Show all obligations **/
CALL obligations()

/* Result :

Obligations list : (L) Legal / (C) Contractual / (E) Ethical

L Obligation 1-3      : Informing about processing, purposes and recipients.
L Obligation 4       : Informing about rights.
L Obligation 5       : Data collection purposes.
L Obligation 6       : The right to access, correct and delete personal data.
L Obligation 7       : Data storage period.
L Obligation 8,11-12 : Security and privacy measures.
L Obligation 9-10    : Rules for data processing by providers.
L Obligation 13-15   : Consent to processing.
L Obligation 16      : Informing DPAs.
C Obligation 17      : Informing about the use of sub-processors.
C Obligation 18      : Security breach notification.
C Obligation 19-20   : Evidence on data processing and data deletion.
C Obligation 21      : Data location.
E Obligation 22      : Informing about personal data processing.
E Obligation 23      : Personal data minimization.
E Obligation 24      : Privacy-by-default.
E Obligation 25      : Specifying user preferences.
E Obligation 26      : Monitoring of data practices.
E Obligation 27      : Compliance with user preferences.
E Obligation 28      : Compliance with privacy policies.
E Obligation 29-30   : Informing about policy violations and privacy preferences violations.
E Obligation 31      : Remediation in case of incidents.
*/
```

1. GETTING STARTED

```
/** Checking if the clauses respect **/  
CALL obligation18()  
  
/* Obligation 18: Security breach notification. */  
-> No notification in clause kim_policy s rectification at line 15  
-> No notification in clause cloudX_policy s rectification at line 30
```

1.6 Using the shell

The shell is a useful tool for developing

- Run the shell.

```
aalc -i tests/tuto2.aal -s
```

```
/* Result :  
shell >
```

```
*/
```

- Type help to show the shell help.

```
Shell Help  
- call(macro, args)  call a macro where /  
                      *macro : is the name of the macro  
                      *args : a list of string; << ex : ["'args1'", "'args2'", ..."'argsN'"] >>  
- clauses()          show all declared clauses in the loaded aal program  
- macros()           show all declared macros in the loaded aal program  
- quit / q           exit the shell  
- help / h / man()   show this help  
- self               the current compiler instance of the loaded aal program  
- aalprog            the current loaded aal program  
- man(arg)           print the help for the given arg  
- hs(module)         hotswaping : reload the module  
- r()                hot-swaping the shell
```

- Here an example, we print all clauses in the AAL program.

```
shell> clauses()  
  
/* Result :  
kim_policy cloudX_policy  
  
*/
```

- self variable represent the co

```
shell> self  
  
/* Result :  
<AALCompiler.AALCompilerListener object at 0x7f8b00ce8630>  
  
*/  
  
shell> man(self)
```

1. GETTING STARTED

```
/* Result :
printing manual for <class 'AALCompiler.AALCompilerListener'>
Manual for aal compiler visitor
- Attributes
  - aalprog      Get the AAL program instance
  - file         The AAL source file
  - libs         Show the loaded libraries
  - libsPath     Print the standard lib path
- Methods
  - load_lib(lib_name)  Load an aal file
  - clause(clauseId)    Get a clause
  - show_clauses()      Print all clauses
  - get_clauses()       Get all clauses (array format)
  - get_macros()        Get all macros

*/

shell> man(aalprog)

/* Result :
printing manual for <class 'AALMetaModel.m_aalprog'>

AAL program class.
Note that clauses and macros extends a declarable type, but are not in the declarations dict

Attributes
  - clauses: a list that contains all program clauses
  - declarations: a dictionary that contains lists of typed declarations
  - comments: a list that contains program's comment
  - macros: a list that contains program's macros declarations
  - macroCalls: a list that contains program's comment

*/
```

- hotswaping commands are used for debugging purpose only. `r()` command allows you to reload the shell after
`hs(module)` reloading other modules after ! IMORTANT : to use hotswaping properly you must enable it explicitly in aalc arguments `-d / -hotswap`,

2 AAL language

Listing 2: AAL Syntax

```
// AAL CORE
AALprogram ::= (Declaration | Clause | Comment | Macro | MacroCall | EXEC)*
Declaration ::= AgentDec | ServiceDec | DataDec | TypesDec
AgentDec ::= AGENT Id TYPE('Type*') REQUIRED('service*') PROVIDED('service*')
ServiceDec ::= SERVICE Id TYPE('Type*') [PURPOSE 'Id*']
DataDec ::= DATA Id TYPE('Type*') [REQUIRED('service*') PROVIDED('service*')] SUBJECT
  agent
Clause ::= CLAUSE Id '(' [Usage] [Audit Rectification] ')'
Usage ::= ActionExp
Audit ::= AUDITING [ActionExp THEN] agent.audit['agent'] '(')
Rectification ::= IF_VIOLATED_THEN ActionExp (??Usage)
ActionExp ::= Action | NOT ActionExp | Modality ActionExp | Condition
  | ActionExp (AND|OR|ONLYWHEN) ActionExp | Author | Quant*
  | IF ActionExp THEN ActionExp
Exp ::= Variable | Constant | Variable.Attribute
Condition ::= [NOT] Exp | Exp ['=='] | Exp ['!='] Exp | Condition (AND|OR) Condition
Author ::= (PERMIT | DENY) Action
Action ::= agent.service ['['[agent]]'] '('Exp')' [Time] [Purpose]
Quant ::= (FORALL | EXISTS) Var [WHERE Condition]
Variable ::= Var ':' Type
Modality ::= MUST | MUSTNOT | ALWAYS | NEVER | SOMETIME
Time ::= (AFTER | BEFORE) Date | Time (AND | OR) Time
Date ::= hh ':' mm ':' ss DD '/' MM '/' YYYY (use string)
Type, var, val, attr Id, agent, Constant, Purpose ::= literal

// AAL Type extension
TypesDec ::= TYPE Id [EXTENDS '(' Type* ')'] ATTRIBUTES '(' AttributeDec* ')' ACTIONS '('
  ActionDec* ')'
AttributeDec ::= Id ':' Type
ActionDec ::= Id
Type, Id ::= literal
Affectation ::= var.id '=' val

// Reflexion extension
Macro ::= MACRO Id '(' param* ')' '(' mcode ')'
MCode ::= Meta model api + Python3 code (subset)
MCall ::= CALL Id '(' param* ')'
LoadLib ::= LOAD STRING;
EXEC : M_exec MCODE;

// LTL checking extension
Modified version of LTL
ltlCheck ::= M_check ID args? h_lpar check h_rpar;
check ::= formula;
checkApply ::= M_apply ID h_lpar STRING* h_rpar;
atom ::= C_clause h_lpar h_clauseId h_rpar (h_dot (C_usage | C_audit | C_rectification))?
;
```