

Module Interface Specification for Kaplan

Jen Garner

November 21, 2018

1 Revision History

Date		Version	Notes
November 20, 2018 (Tuesday)		1.0	Initial draft

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at [\[give url —SS\]](#)

[\[Also add any additional symbols, abbreviations or acronyms —SS\]](#)

Contents

1	Revision History	i
2	Symbols, Abbreviations and Acronyms	ii
3	Introduction	1
4	Notation	1
5	Module Decomposition	1
6	MIS of GA Input	3
6.1	Module	3
6.2	Uses	3
6.3	Syntax	3
6.3.1	Exported Constants	3
6.3.2	Exported Access Programs	3
6.4	Semantics	3
6.4.1	State Variables	3
6.4.2	Environment Variables	4
6.4.3	Assumptions	4
6.4.4	Access Routine Semantics	4
6.4.5	Local Functions	4
7	MIS of GA Control	5
7.1	Module	5
7.2	Uses	5
7.3	Syntax	5
7.3.1	Exported Constants	5
7.3.2	Exported Access Programs	5
7.4	Semantics	5
7.4.1	State Variables	5
7.4.2	Environment Variables	5
7.4.3	Assumptions	5
7.4.4	Access Routine Semantics	5
7.4.5	Local Functions	6
8	MIS of Fit_G	7
8.1	Module	7
8.2	Uses	7
8.3	Syntax	7
8.3.1	Exported Constants	7
8.3.2	Exported Access Programs	7

8.4	Semantics	7
8.4.1	State Variables	7
8.4.2	Environment Variables	7
8.4.3	Assumptions	7
8.4.4	Access Routine Semantics	7
8.4.5	Local Functions	8
9	MIS of Tournament	9
9.1	Module	9
9.2	Uses	9
9.3	Syntax	9
9.3.1	Exported Constants	9
9.3.2	Exported Access Programs	9
9.4	Semantics	9
9.4.1	State Variables	9
9.4.2	Environment Variables	9
9.4.3	Assumptions	9
9.4.4	Access Routine Semantics	9
9.4.5	Local Functions	10
10	MIS of Crossover & Mutation	11
10.1	Module	11
10.2	Uses	11
10.3	Syntax	11
10.3.1	Exported Constants	11
10.3.2	Exported Access Programs	11
10.4	Semantics	11
10.4.1	State Variables	11
10.4.2	Environment Variables	11
10.4.3	Assumptions	11
10.4.4	Access Routine Semantics	11
10.4.5	Local Functions	12
11	MIS of Ring	13
11.1	Module	13
11.2	Uses	13
11.3	Syntax	13
11.3.1	Exported Constants	13
11.3.2	Exported Access Programs	13
11.4	Semantics	13
11.4.1	State Variables	13
11.4.2	Environment Variables	13
11.4.3	Assumptions	13

11.4.4	Access Routine Semantics	13
11.4.5	Local Functions	14
12	MIS of Output	15
12.1	Module	15
12.2	Uses	15
12.3	Syntax	15
12.3.1	Exported Constants	15
12.3.2	Exported Access Programs	15
12.4	Semantics	15
12.4.1	State Variables	15
12.4.2	Environment Variables	15
12.4.3	Assumptions	15
12.4.4	Access Routine Semantics	15
12.4.5	Local Functions	16
13	MIS of Molecule Input	17
13.1	Module	17
13.2	Uses	17
13.3	Syntax	17
13.3.1	Exported Constants	17
13.3.2	Exported Access Programs	17
13.4	Semantics	17
13.4.1	State Variables	17
13.4.2	Environment Variables	17
13.4.3	Assumptions	17
13.4.4	Access Routine Semantics	17
13.4.5	Local Functions	18
14	MIS of Energies	19
14.1	Module	19
14.2	Uses	19
14.3	Syntax	19
14.3.1	Exported Constants	19
14.3.2	Exported Access Programs	19
14.4	Semantics	19
14.4.1	State Variables	19
14.4.2	Environment Variables	19
14.4.3	Assumptions	19
14.4.4	Access Routine Semantics	19
14.4.5	Local Functions	20

15 MIS of RMSD	21
15.1 Module	21
15.2 Uses	21
15.3 Syntax	21
15.3.1 Exported Constants	21
15.3.2 Exported Access Programs	21
15.4 Semantics	21
15.4.1 State Variables	21
15.4.2 Environment Variables	21
15.4.3 Assumptions	21
15.4.4 Access Routine Semantics	21
15.4.5 Local Functions	22
16 Appendix	24

3 Introduction

The following document details the Module Interface Specifications for [Fill in your project name and description —SS]

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at [provide the url for your repo —SS]

4 Notation

[You should describe your notation. You can use what is below as a starting point. —SS]

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol $:=$ is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | \dots | c_n \Rightarrow r_n)$.

Jen is also using [this link](#) for now.

The following table summarizes the primitive data types used by Kaplan.

Data Type	Notation	Description
character	char	a single symbol or digit
integer	\mathbb{Z}	a number without a fractional component in $(-\infty, \infty)$
natural number	\mathbb{N}	a number without a fractional component in $[1, \infty)$
real	\mathbb{R}	any number in $(-\infty, \infty)$

The specification of Kaplan uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, Kaplan uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	Level 2
Hardware-Hiding Module	
	GA Input
	GA Control
	Fit_G
Behaviour-Hiding Module	Tournament
	Crossover & Mutation
	Ring
	Output
Software Decision Module	Molecule Input
	Energies
	RMSD

Table 1: Module Hierarchy

6 MIS of GA Input

[You can reference SRS labels, such as R1. —SS]

[It is also possible to use L^AT_EX for hypperlinks to external documents. —SS]

6.1 Module

ga_input

6.2 Uses

None

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
read_ga_input	str	dict	FileNotFoundError
verify_ga_input	dict	None	ValueError

6.4 Semantics

6.4.1 State Variables

ga_input_dict, which is a dictionary that contains:

- num_geoms: $\mathbb{R}_{>0} = \{x \in \mathbb{R} : x > 0\}$
- num_atoms: $\mathbb{R}_{>0} = \{x \in \mathbb{R} : x > 0\}$
- num_slots: $\mathbb{R}_{>0} = \{x \in \mathbb{R} : x > 0\}$
- num_filled: $\mathbb{R}_{>0} = \{x \in \mathbb{R} : x > 0\}$
- num_muts: $\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} : x \geq 0\}$
- num_swaps: $\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} : x \geq 0\}$
- t_size: $\mathbb{R}_{>0} = \{x \in \mathbb{R} : x > 0\}$

6.4.2 Environment Variables

`ga_input_file`: str representing the file that exists in the working directory.

6.4.3 Assumptions

This module is responsible for all type checking and no errors will come from incorrect passing of variables (except input related to the molecule - covered in [13](#)).

6.4.4 Access Routine Semantics

`[accessProg —SS]()`:

- transition: `[if appropriate —SS]`
- output: `[if appropriate —SS]`
- exception: `[if appropriate —SS]`

`[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]`

`[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]`

`read_ga_input(ga_input_file)`:

- transition: open `ga_input_file` and read its contents
- output: dictionary (`ga_input_dict`) that contains the values listed in State Variables
- exception: `FileNotFoundError` := `ga_input_file` \notin current working directory

`verify_ga_input(ga_input_dict)`:

- transition:
- output: `None`
- exception: `ValueError`
 - `num_filled > num_slots`
 - `num_swaps > num_geoms`
 - `t_size > num_filled`
 - not an integer type
 - missing key/unknown key

6.4.5 Local Functions

`None`

7 MIS of GA Control

[Use labels for cross-referencing —SS]

[You can reference SRS labels, such as R1. —SS]

[It is also possible to use L^AT_EX for hyperlinks to external documents. —SS]

7.1 Module

[Short name for the module —SS]

7.2 Uses

7.3 Syntax

7.3.1 Exported Constants

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

7.4 Semantics

7.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

7.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

7.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

7.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]

- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

7.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

8 MIS of Fit_G

[You can reference SRS labels, such as R1. —SS]

[It is also possible to use L^AT_EX for hypperlinks to external documents. —SS]

8.1 Module

[Short name for the module —SS]

8.2 Uses

8.3 Syntax

8.3.1 Exported Constants

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

8.4 Semantics

8.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

8.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

8.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

8.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

8.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

9 MIS of Tournament

[Use labels for cross-referencing —SS]

[You can reference SRS labels, such as R1. —SS]

[It is also possible to use L^AT_EX for hyperlinks to external documents. —SS]

9.1 Module

[Short name for the module —SS]

9.2 Uses

9.3 Syntax

9.3.1 Exported Constants

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

9.4 Semantics

9.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

9.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

9.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

9.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]

- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

9.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

10 MIS of Crossover & Mutation

[You can reference SRS labels, such as R1. —SS]

[It is also possible to use L^AT_EX for hypperlinks to external documents. —SS]

10.1 Module

[Short name for the module —SS]

10.2 Uses

10.3 Syntax

10.3.1 Exported Constants

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

10.4 Semantics

10.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

10.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

10.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

10.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

10.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

11 MIS of Ring

[You can reference SRS labels, such as R1. —SS]

[It is also possible to use L^AT_EX for hypperlinks to external documents. —SS]

11.1 Module

[Short name for the module —SS]

11.2 Uses

11.3 Syntax

11.3.1 Exported Constants

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

11.4 Semantics

11.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

11.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

11.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

11.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

11.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

12 MIS of Output

[You can reference SRS labels, such as R1. —SS]

[It is also possible to use L^AT_EX for hypperlinks to external documents. —SS]

12.1 Module

[Short name for the module —SS]

12.2 Uses

12.3 Syntax

12.3.1 Exported Constants

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

12.4 Semantics

12.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

12.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

12.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

12.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

12.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

13 MIS of Molecule Input

[You can reference SRS labels, such as R1. —SS]

[It is also possible to use L^AT_EX for hypperlinks to external documents. —SS]

13.1 Module

[Short name for the module —SS]

13.2 Uses

13.3 Syntax

13.3.1 Exported Constants

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

13.4 Semantics

13.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

13.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

13.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

13.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

13.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

14 MIS of Energies

[You can reference SRS labels, such as R1. —SS]

[It is also possible to use L^AT_EX for hypperlinks to external documents. —SS]

14.1 Module

[Short name for the module —SS]

14.2 Uses

14.3 Syntax

14.3.1 Exported Constants

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

14.4 Semantics

14.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

14.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

14.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

14.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

14.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

15 MIS of RMSD

[You can reference SRS labels, such as R1. —SS]

[It is also possible to use L^AT_EX for hypperlinks to external documents. —SS]

15.1 Module

[Short name for the module —SS]

15.2 Uses

15.3 Syntax

15.3.1 Exported Constants

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

15.4 Semantics

15.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

15.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

15.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

15.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

15.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

References

- Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.
- Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL <http://citeseer.ist.psu.edu/428727.html>.

16 Appendix

[Extra information if required —SS]