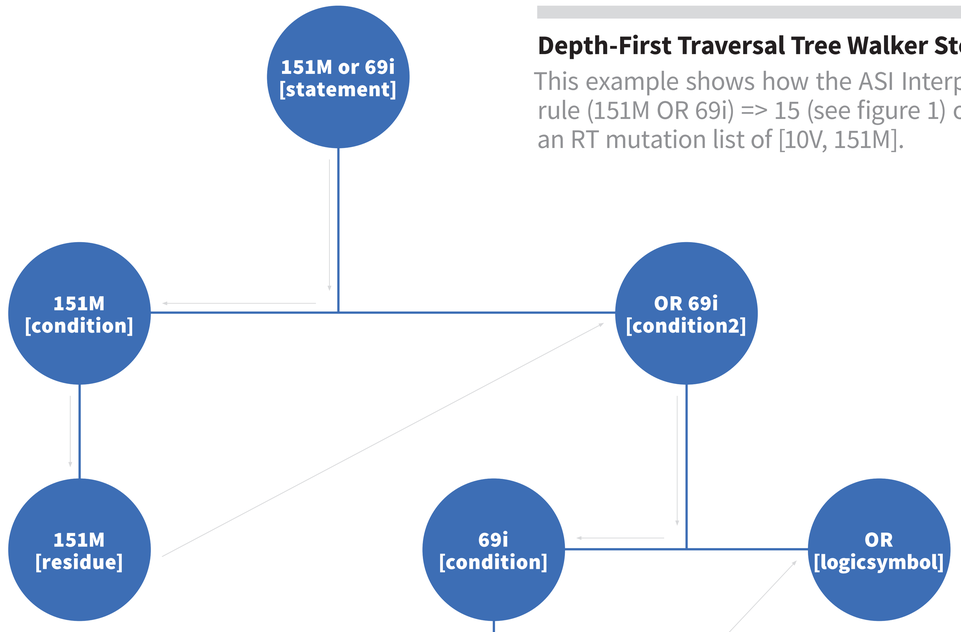
# Calculating Antiretroviral Drug Resistance

## An Innovative Open Source Tool

**Adina Stoica, Frontier Science and Technology Research Foundation, Inc.**
**Robert W. Shafer, Stanford University**
**Stephen Hart, Frontier Science and Technology Research Foundation, Inc.**

The Algorithm Specification Interface (ASI) Interpreter is an open source Java library that makes antiretroviral drug resistance calculations using customizable algorithms.

The ASI interpreter's XML-based approach allows the rules for calculating resistance to be changed without altering program code. The rules used are easily readable by other scientists, enhancing transparency and replicability. The ASI Interpreter's grammar is flexible enough to express an enormous variety of possible rules for calculating resistance or other results that can be calculated on the basis of viral mutations.



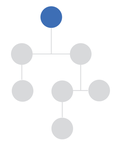

### Depth-First Traversal Tree Walker Steps: An Example

This example shows how the ASI Interpreter executes the rule (151M OR 69i) => 15 (see figure 1) on a sequence with an RT mutation list of [10V, 151M].

**Step 1**
The generated depth-first traversal tree for rule *151M* or *69i*.

No action defined at this point. In general, if the evaluation of the node is true, *1* will be pushed onto the stack, else *0*.

Stack



**Step 2**
The next element evaluated will be *151M [condition]* based on the depth-first tree traversal algorithm.

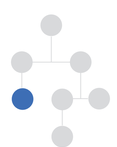
There is no action defined for the condition production rule, so nothing is pushed onto the stack.

Stack

**Step 3**
The next element evaluated will be *151M [residue]* based on the depth-first tree traversal algorithm.

According to the residue production rule definition, since *151M* is in the mutation list, the program will evaluate the node to true and *1* will be pushed onto the stack.

1
Stack



**Step 4**
Next, all children of the *OR 69i* node will be visited.

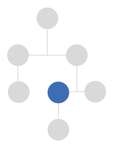
There is no action defined for the condition production rule, so nothing is pushed onto the stack.

1
Stack

**Step 5**
The next element evaluated will be the *69i [condition2]*.

There is no action defined for the condition production rule, so nothing is pushed onto the stack.

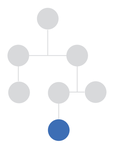
1
Stack



**Step 6**
*69i* is not in the mutation list, so the node will be evaluated to false and *0* will be pushed onto stack.

All nodes on the left side of this branch were visited. Next, it will start depth-first traversal of the right side of this branch.

0
1
Stack



**Step 7**
The *OR* production rule implementation expects two elements in the stack.

*0* and *1* are popped off the stack, and the boolean *OR* evaluation is applied. The result *1* will be pushed onto the stack.

1
Stack

**Step 8**
All nodes have been visited and the final value in the stack is *1*.

*1* is popped off the stack and evaluated to true. *151M or 69i* statement for mutation list *[10V, 151M]* is evaluated to true so the associated score of *15* is pushed onto the stack.

15
Stack



## Background

How mutations affect the efficacy of drugs is complex. Typically more than one mutation is needed to confer resistance, and particular mutation combinations affect each other differently.

Given a sequence, one can predict which drugs are more or less likely to work. This has many practical applications:

- Selecting a starting treatment if a person is infected by someone who had drug resistance
- Determining a new treatment when another fails
- Investigating resistance present in virus transmitted to newly infected patients in public health work

HIV-1 resistance interpretations are most commonly algorithmic, using rules based on mutations, such as those found in the Stanford HIV Drug Resistance Database (HIVDB) or the France Recherche Nord & Sud Sida-HIV Hépatites (ANRS). While most algorithm implementations have rules built into the system as program code, the ASI Interpreter's approach decouples scientific rules from computer code by instantly compiling and executing any properly stated scientific rules.

This approach was pioneered by the HIVDB group. The ASI Interpreter was developed by Frontier Science in collaboration with the HIVDB group. The resulting library was released in December 2017 under an open source software license, and is available to all resistance researchers.

## Approach

The ASI Interpreter's approach has two components: the XML grammar file and the application code.

The XML file is a flexible, structured grammar file that defines scientific rules and elements used in analysis.

The ASI Interpreter receives a list of mutations, parses the XML file, and applies its rules to calculate resistance for each specified drug, and produce resistance interpretations for each drug-sequence combination. The library uses SableCC, an open source grammar compiler library. SableCC is used to create a skeleton of strictly-typed abstract syntax trees and tree walkers.

## Advantages of Approach

**Transparency:** Any researcher can read the XML file and see exactly how the algorithm works.

**Reproducibility and replicability:** Anyone can run the library, using an algorithm file, and the results will be the same.

**Flexibility:** Algorithms can be created or modified without needing to modify and rebuild the application's code. Many different algorithm types are supported, and could be extended to other organisms.

**Supported algorithm types:** ANRS, Rega Institute, IAS-USA guidelines, WHO mutation lists. Users can also create their own algorithms.

**Platform Independent:** The library is written in Java and can run on most computer systems.

**Self-contained:** No database dependencies or external files are needed. Everything, other than input sequence data, is contained in the algorithm XML file.

**Easy Integration:** Requires minimal code to integrate with other software applications.

## ASI-Compliant Algorithms

ASI-compliant algorithms evaluate resistance for any drug based on mutations in the relevant gene, using rules stated in an XML file.

In the Stanford algorithm (figure 1), each rule assigns points if a predicate condition is met. Conditions can be simple (true if a specific mutation if present) or complex (for example, true if three specified mutations are simultaneously present). The sum of all points across the set of rules for a drug produces a raw score. This score is then mapped to a resistance level using an equivalence table embedded in the XML file.

In other algorithms, such as ANRS and Rega (figure 2), a predicate condition directly points to a resistance level, which is assigned when the condition is true. There are usually several rules, each pointing to a different level, for a given drug.

## How It Works

- The ASI Interpreter is a library, not a standalone application. It can be integrated with another program that needs to provide resistance results.
- The ASI Interpreter requires three inputs: an algorithm file, a gene region, and a list of mutations to evaluate.
- The algorithm file specifies the rules that are used to calculate resistance for each drug of interest. Each rule matches a condition (a set of mutation criteria) to an action. An action can be a score and resulting resistance level (figure 1), a resistance level (figure 2), or a comment (figure 3).
- The ASI Interpreter takes the algorithm file that it is given and converts the rules into a logical structure to evaluate the given list of mutations. The logical structure takes the form of a tree in which each step in the detailed logic evaluating a rule is a node. The ASI Interpreter then systematically walks through the tree.
- Based on the type of actions specified in the rules, the interpreter can report a resistance level, a score (translated into a resistance level), and a set of comments.

**Figure 1: Algorithm with Score and Resistance Level**

```
<ALGORITHM>
    <ALGNAME>HIVDB</ALGNAME>
    <ALGVERSION>8.4</ALGVERSION>
    <ALGDATE>2017-06-16</ALGDATE>
    <GLOBALRANGE>
        (-INF TO 9 => 1,
        10 TO 14  => 2,
        15 TO 29  => 3,
        30 TO 59  => 4,
        60 TO INF => 5)
    </GLOBALRANGE>
    <DRUG>
        <NAME>3TC</NAME>
        <FULLNAME>lamivudine</FULLNAME>
        <RULE>
            <CONDITION>
                SCORE FROM(
                    MAX (65N => 15, 65R => 30),
                    67d => 15,
                    MAX (69i => 30, 69d => 15),
                    77L => 5,
                    116Y => 5,
                    MAX (151L => 10, 151M => 15),
                    (151M OR 69i) => 15)
            </CONDITION>
            <ACTIONS>
                <SCORERANGE>
                    <USE_GLOBALRANGE/>
                </SCORERANGE>
            </ACTIONS>
        </RULE>
    </DRUG>
</ALGORITHM>
```

Translates score to resistance level

Rule used in example on right

## ASI Interpreter Output

**At the level of a gene (such as protease or RT) in a sequence:**

- The mutations that were actively used in the scoring for the sequence
- The comments that were generated at the gene level

**At the level of each drug:**

- (If the algorithm uses points) The point score
- The resistance level
- Comments that were generated for the drug
- (If the algorithm uses points) For each rule that was triggered in scoring the drug: the points conferred and mutation(s) that triggered the rule

## Future features in planning

- Comments based on resistance level results for a given drug.
- Multi-gene rules, such as using both GAG and PR mutations to calculate resistance for protease inhibitors.
- Support for additional viruses.
- Mutation sets that can be scored as a single mutation, for when a mutation definition is not specific to one residue.
- Mutation type: defines classification level of mutation type to be reported. This would increase the transparency of type assignments in algorithm XML (such as PI major and minor).
- Supporting different scoring logic depending on the subtype.
- Expand beyond codon-level resistance calculations.

**Figure 2: Algorithm with Resistance Level**

```
<ALGORITHM>
    <ALGNAME>Rega v9.1.0</ALGNAME>
    <ALGVERSION>9.1.0</ALGVERSION>
    <LEVEL_DEFINITION>
        <ORDER>6</ORDER>
        <ORIGINAL>Resistant GSS 0</ORIGINAL>
        <SIR>R</SIR>
    </LEVEL_DEFINITION>
    <DRUG>
        <NAME>AZT</NAME>
        <FULLNAME>zidovudine</FULLNAME>
        <RULE>
            <CONDITION>
                SELECT ATLEAST 1 FROM
                (151M, 69i)
            </CONDITION>
            <ACTIONS><LEVEL>6</LEVEL></ACTIONS>
        </RULE>
    </DRUG>
</ALGORITHM>
```

**Figure 3: Algorithm with Comments**

```
<ALGORITHM>
    <ALGNAME>HIVDB</ALGNAME>
    <ALGVERSION>8.4</ALGVERSION>
    <ALGDATE>2017-06-16</ALGDATE>
    <MUTATION_COMMENTS>
        <GENE>
            <NAME>RT</NAME>
            <RULE>
                <CONDITION>151M</CONDITION>
                <ACTIONS>
                    <COMMENT ref="RT151M"/>
                </ACTIONS>
            </RULE>
        </GENE>
    </MUTATION_COMMENTS>
</ALGORITHM>
```

**Acknowledgements:**
Eric Buckley, Kyle Lambert, David Goss, and Suzanne Siminski

**The ASI Interpreter is developed and maintained by Frontier Science.**

**View the source code and get involved.**
**www.github.com/FrontierScience**