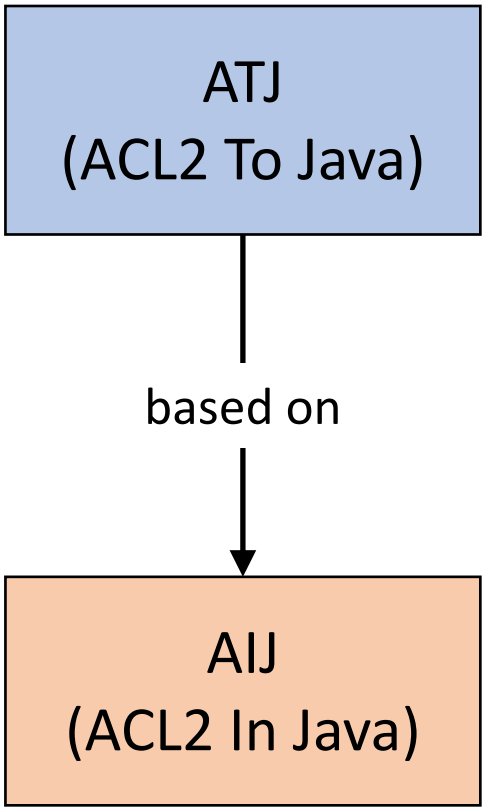
A Simple Java Code Generator for ACL2 Based on a Deep Embedding of ACL2 in Java

Alessandro Coglio

Kestrel Institute

Workshop 2018



Java code generator for ACL2

deep embedding of ACL2 in Java

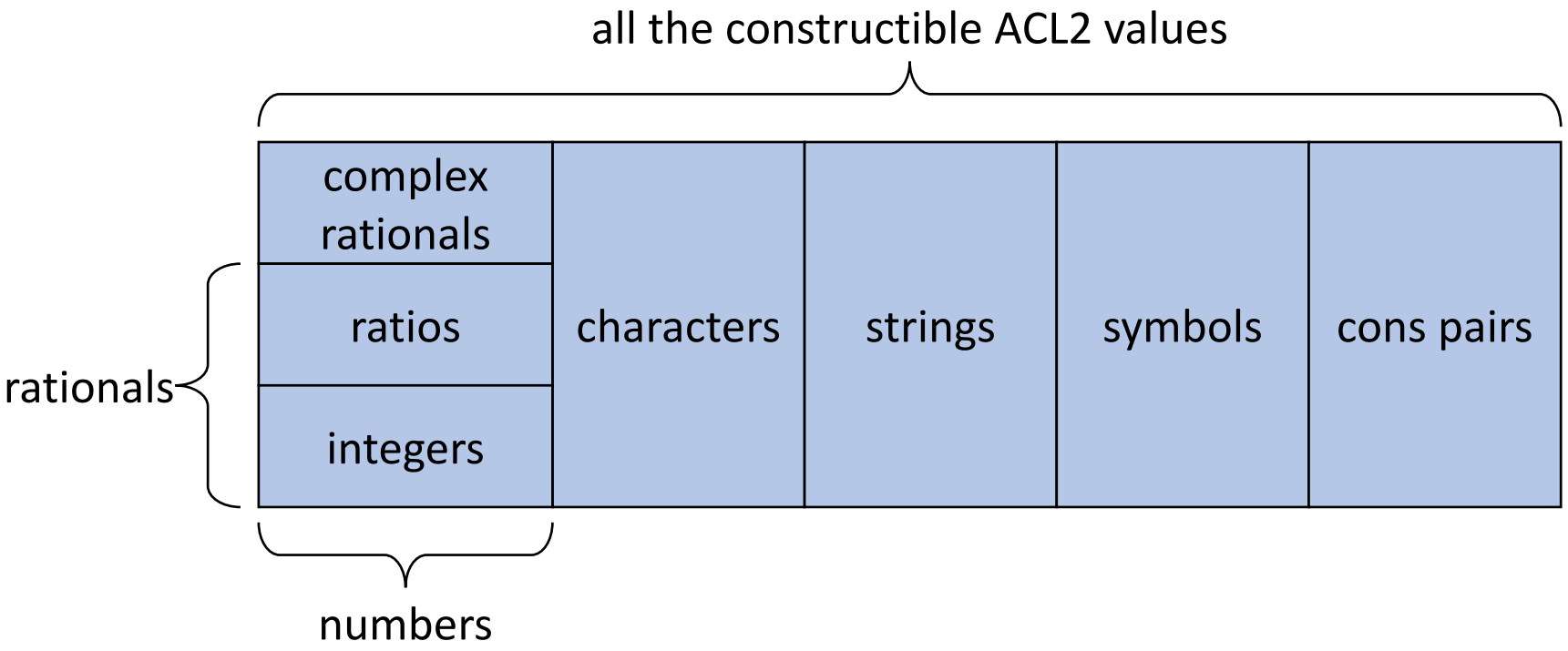
AIJ is a deep embedding in Java of an executable, side-effect-free, non-stobj-accessing subset of the ACL2 language without guards.

AIJ is a deep embedding in Java of an executable, side-effect-free, non-stobj-accessing subset of the ACL2 language without guards.

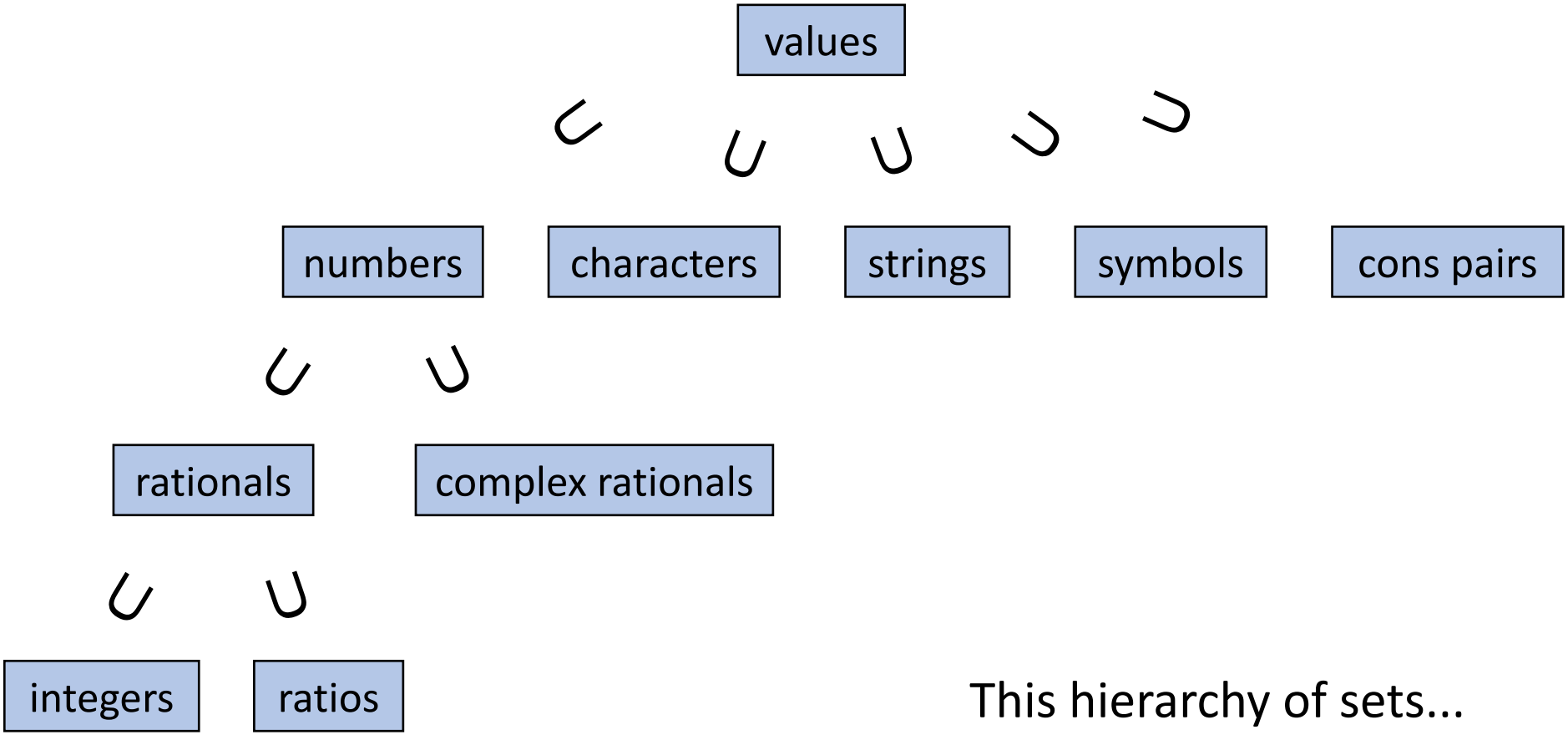
AIJ includes:

- A Java representation of the ACL2 values.

AIJ includes a Java representation of all the constructible ACL2 values.

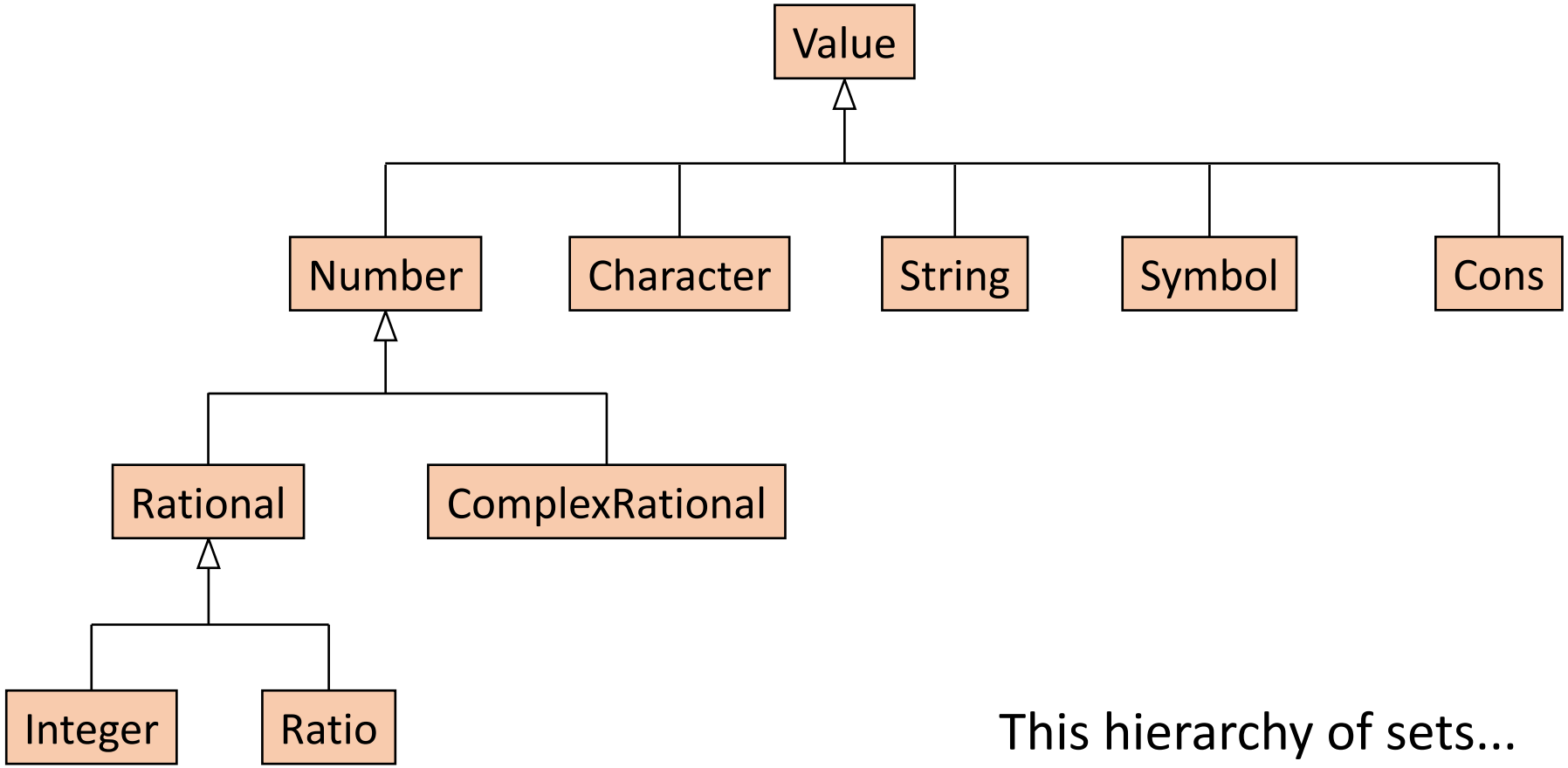


AIJ includes a Java representation of all the constructible ACL2 values.



This hierarchy of sets...

AIJ includes a Java representation of all the constructible ACL2 values.



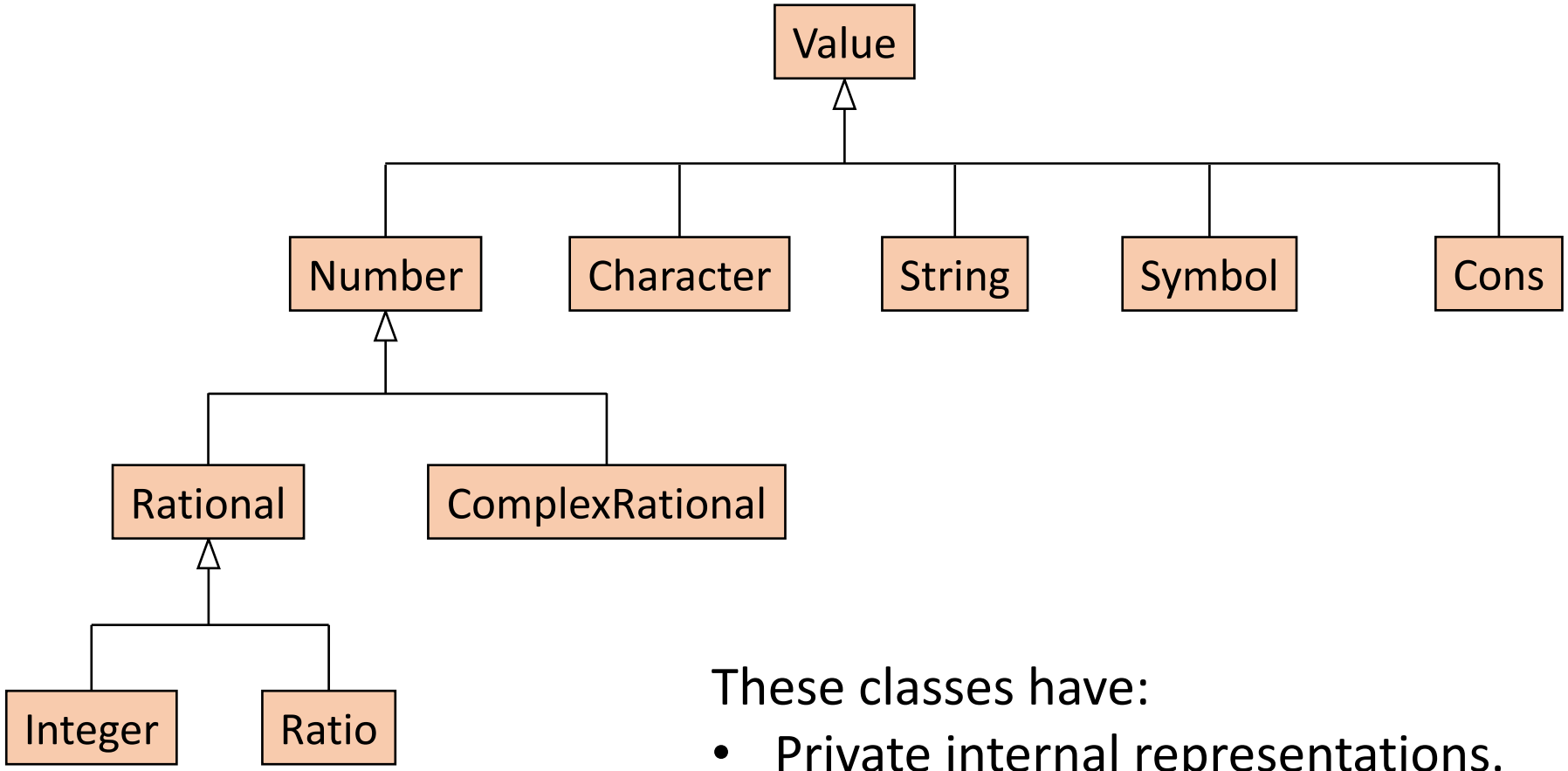
This hierarchy of sets...

... is represented as a hierarchy of Java classes.

AIJ includes a Java representation of all the constructible ACL2 values.

```
public final class Integer extends Rational {

    // internal representation:
    private final java.math.BigInteger bigInt;

    // factory methods (to build values):
    public static Integer make(int ...) {...}
    public static Integer make(long ...) {...}
    public static Integer make(java.math.BigInteger ...) {...}

    // getter methods (to unbuild values):
    public int getJavaInt() {...}
    public long getJavaLong() {...}
    public java.math.BigInteger getJavaBigInteger() {...}

    ...
}
```

AIJ includes a Java representation of all the constructible ACL2 values.



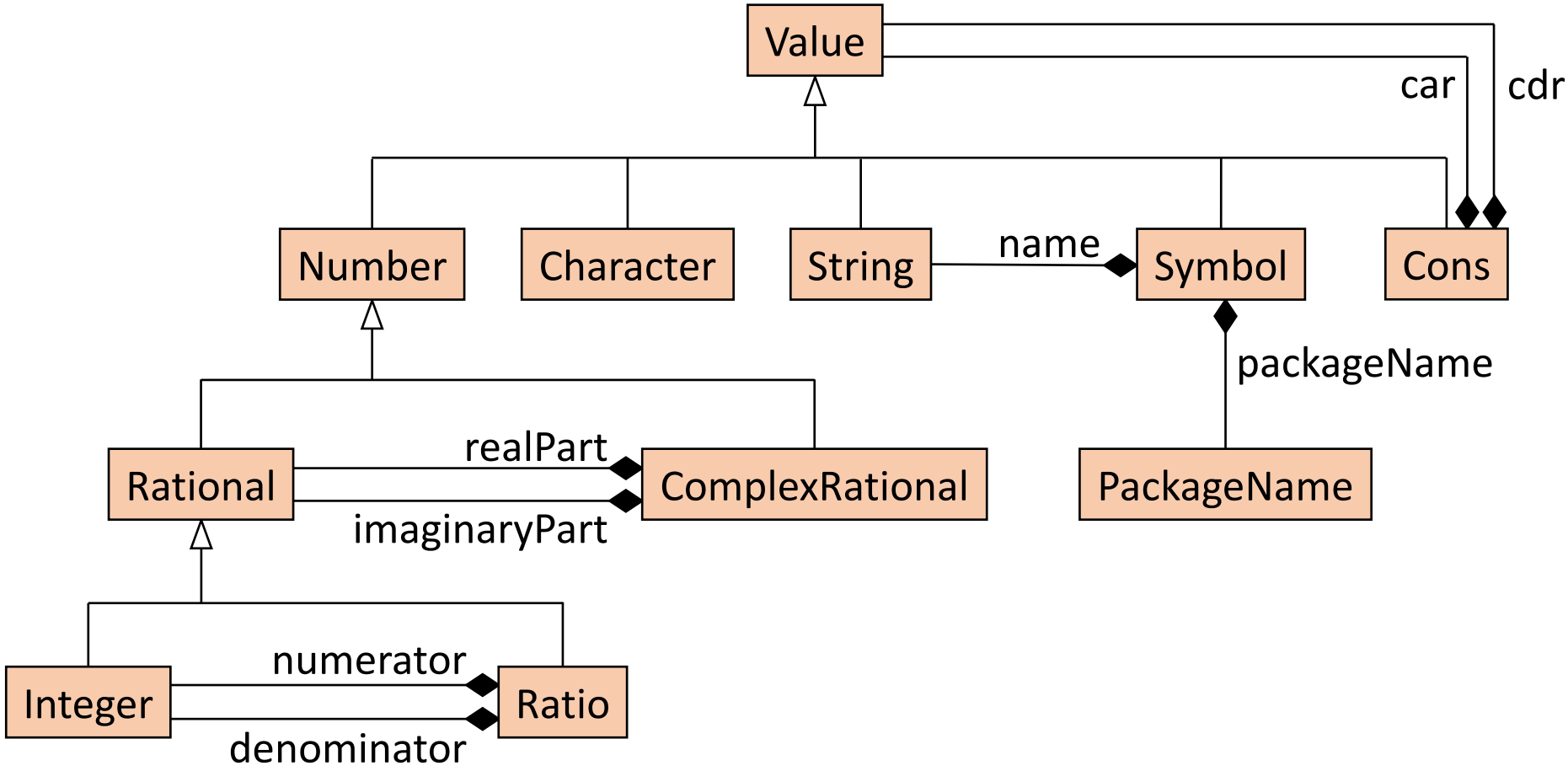
These classes have:

- Private internal representations.
- Public factory methods for building.
- Public getter method for unbuilding.

AIJ includes a Java representation of all the constructible ACL2 values.

```
public final class Ratio extends Rational {

    // internal representation:
    private final Integer numerator;
    private final Integer denominator;

    ...
}
```

AIJ includes a Java representation of all the constructible ACL2 values.

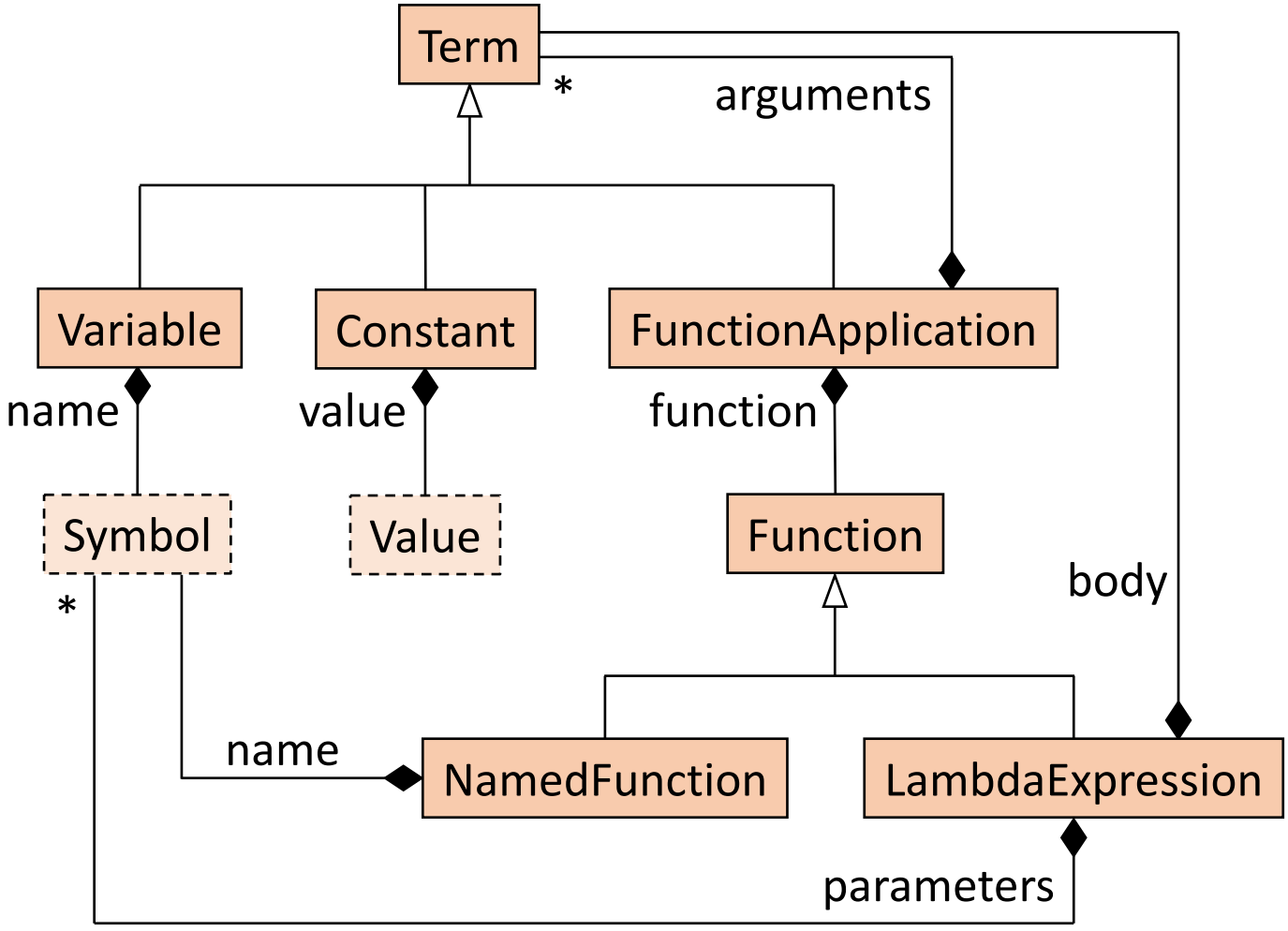


AIJ is a deep embedding in Java of an executable, side-effect-free, non-stobj-accessing subset of the ACL2 language without guards.

AIJ includes:

- A Java representation of the ACL2 values.
- A Java representation of the ACL2 terms.

AIJ includes a Java representation of the ACL2 (translated) terms.



These classes are structured analogously to the classes for values.

AIJ is a deep embedding in Java of an executable, side-effect-free, non-stobj-accessing subset of the ACL2 language without guards.

AIJ includes:

- A Java representation of the ACL2 values.
- A Java representation of the ACL2 terms.
- A Java representation of the ACL2 environment.

AIJ includes a Java representation of (part of) the ACL2 environment.

```
public final class Environment {

    // package definitions:
    private static final
        java.util.Map<PackageName, Symbol[]> packageDefs;
    // function definitions:
    private static final
        java.util.Map<Symbol, LambdaExpression> functionDefs;

    // methods to build the environment:
    public static void addPackageDef(...) {...}
    public static void addFunctionDef(...) {...}

    ...
}
```

AIJ is a deep embedding in Java of an executable, side-effect-free, non-stobj-accessing subset of the ACL2 language without guards.

AIJ includes:

- A Java representation of the ACL2 values.
- A Java representation of the ACL2 terms.
- A Java representation of the ACL2 environment.
- A Java implementation of the ACL2 primitives.

AIJ includes a Java implementation of all the ACL2 primitive functions.

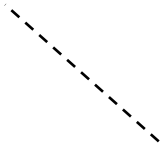
```
ACL2 !>(strip-cars *primitive-formals-and-guards*)
(ACL2-NUMBERP BAD-ATOM<=
              BINARY-* BINARY-+ UNARY-- UNARY-/
              < CAR CDR CHAR-CODE CHARACTERP CODE-CHAR
              COMPLEX COMPLEX-RATIONALP COERCE
              CONS CONSP DENOMINATOR EQUAL IF IMAGPART
              INTEGERP INTERN-IN-PACKAGE-OF-SYMBOL
              NUMERATOR PKG-IMPORTS
              PKG-WITNESS RATIONALP REALPART STRINGP
              SYMBOL-NAME SYMBOL-PACKAGE-NAME SYMBOLP)
ACL2 !>
```

these functions have
no ACL2 definitions

primitive ⇒ built-in
⇏

AIJ includes a Java implementation of all the ACL2 primitive functions.

```
class Primitive {
    private static Value execBinaryPlus(Value x, Value y) {...}
    private static Value execUnaryMinus(Value x) {...}
    private static Value execCar(Value x) {...}
    private static Value execComplex(Value x, Value y) {...}
    private static Value execConsp(Value x) {...}
    private static Value execEqual(Value x) {...}
    private static Value execPkgImports(Value x) {...}
    ...
}
```



no `execIf` method
(`if` is treated specially)

AIJ includes a Java implementation of all the ACL2 primitive functions.

```
class Primitive {

    private static Value execBinaryPlus(Value x, Value y) {...}
    private static Value execUnaryMinus(Value x) {...}
    private static Value execCar(Value x) {...}
    private static Value execComplex(Value x, Value y) {...}
    private static Value execConsp(Value x) {...}
    private static Value execEqual(Value x) {...}
    private static Value execPkgImports(Value x) {...}
    ...

    static Value call(Symbol function, Value[] values) {
        if (function.equals(Symbol.BINARY_PLUS)) {
            // call execBinaryPlus
        } else if (function.equals(Symbol.UNARY_MINUS)) {
            // call execUnaryMinus
        } ...
    }
}
```

AIJ includes a Java implementation of all the ACL2 primitive functions.

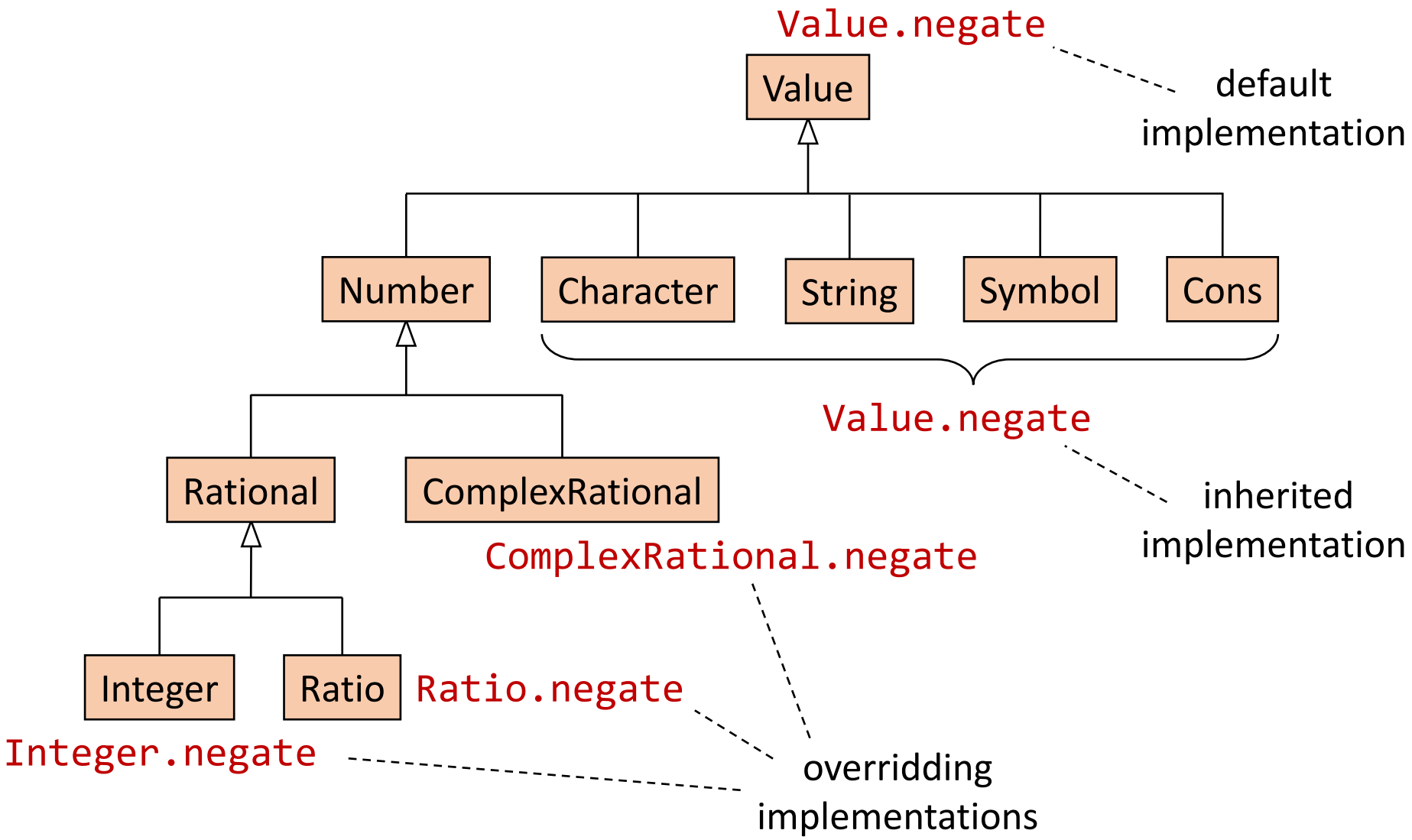
```
class Primitive {

    private static Value execUnaryMinus(Value x) {
        assert x != null;
        return x.negate();
    }

    ...
}
```

AIJ includes a Java implementation of all the ACL2 primitive functions.

```
public abstract class Value {

    Number negate() {
        return Integer.ZERO;
    }

    ...
}
```

default implementation

AIJ includes a Java implementation of all the ACL2 primitive functions.



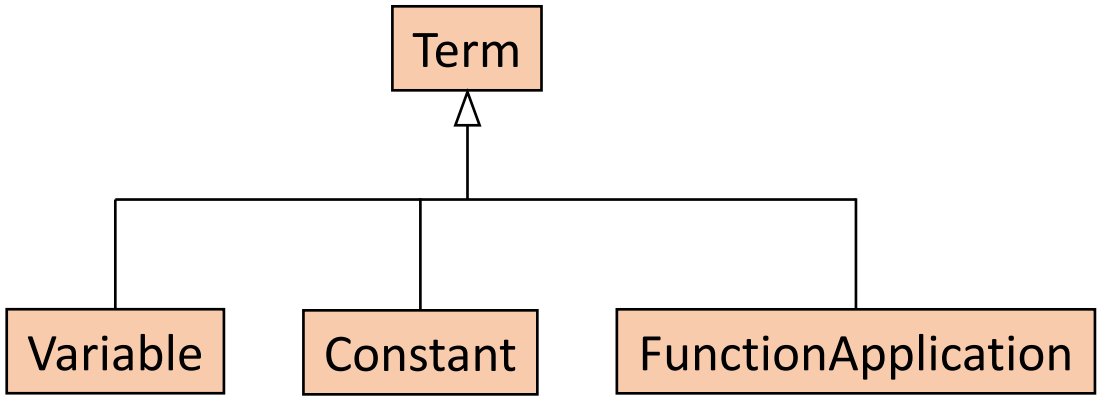
AIJ is a deep embedding in Java of an executable, side-effect-free, non-stobj-accessing subset of the ACL2 language without guards.

AIJ includes:

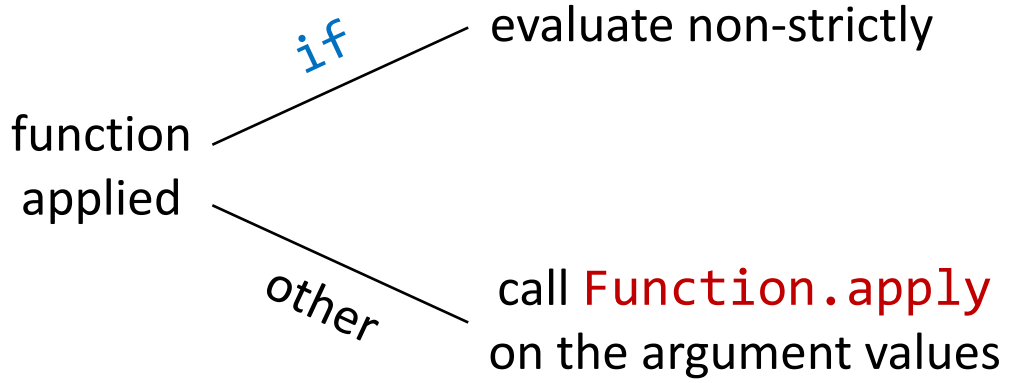
- A Java representation of the ACL2 values.
- A Java representation of the ACL2 terms.
- A Java representation of the ACL2 environment.
- A Java implementation of the ACL2 primitives.
- A Java implementation of the ACL2 evaluator.

AIJ includes a recursive Java implementation of the ACL2 evaluator "in the logic", i.e. with `(set-guard-checking :none)`.

```
abstract Value eval(java.util.Map<Symbol, Value> bindings);
```

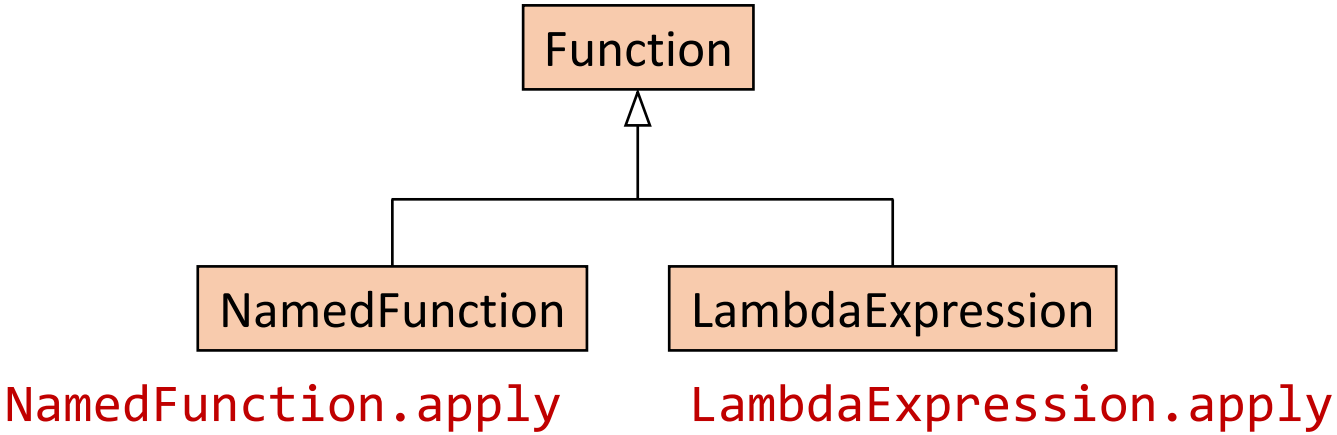


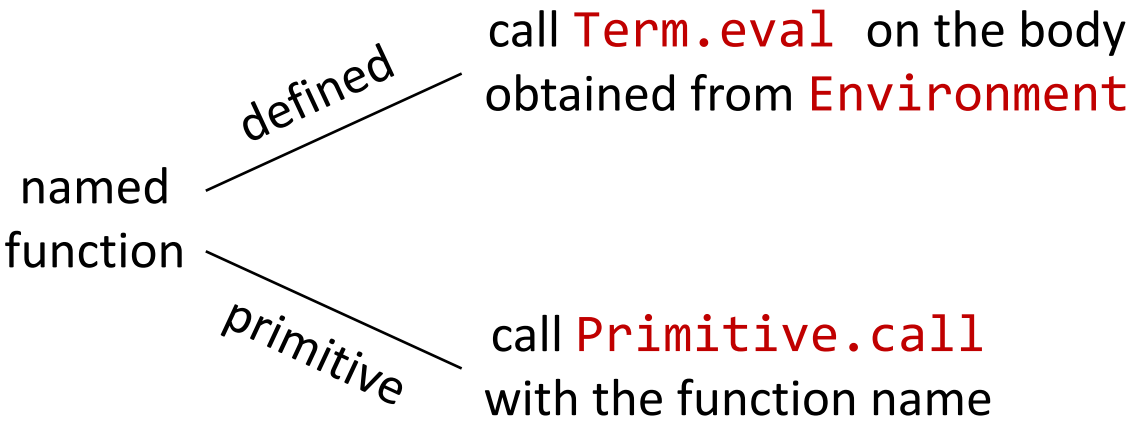
`Variable.eval` `Constant.eval` `FunctionApplication.eval`



AIJ includes a recursive Java implementation of the ACL2 evaluator "in the logic", i.e. with `(set-guard-checking :none)`.

```
abstract Value apply(Value[] values);
```





AIJ is a deep embedding in Java of an executable, side-effect-free, non-stobj-accessing subset of the ACL2 language without guards.

AIJ includes:

- A Java representation of the ACL2 values, terms, and environment.
- A Java implementation of the ACL2 primitives and evaluator.

AIJ is a deep embedding in Java of an executable, side-effect-free, non-stobj-accessing subset of the ACL2 language without guards.

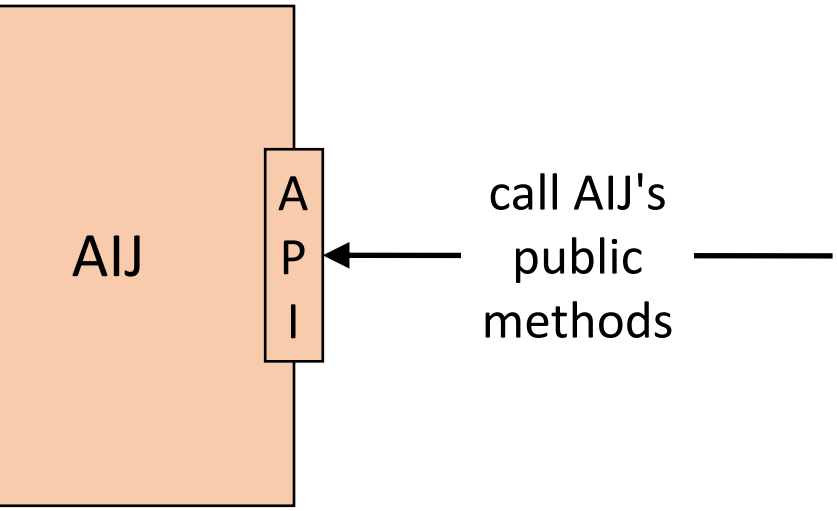
AIJ includes:

- A Java representation of the ACL2 values, terms, and environment.
- A Java implementation of the ACL2 primitives and evaluator.

AIJ provides a Java API to:

- Build the ACL2 environment.

AIJ provides a Java API to build the ACL2 environment: package and function definitions, and their constituents.




```
// build the package definitions:
... PackageName.make(...)
... Symbol.make(...)
Environment.addPackageDef(...);
Environment.addPackageDef(...);
...

// build the function definitions:
... Variable.make(...)
... NamedFunction.make(...)
... FunctionApplication.make(...)
Environment.addFunctionDef(...);
Environment.addFunctionDef(...);
...
```

code external to AIJ

AIJ is a deep embedding in Java of an executable, side-effect-free, non-stobj-accessing subset of the ACL2 language without guards.

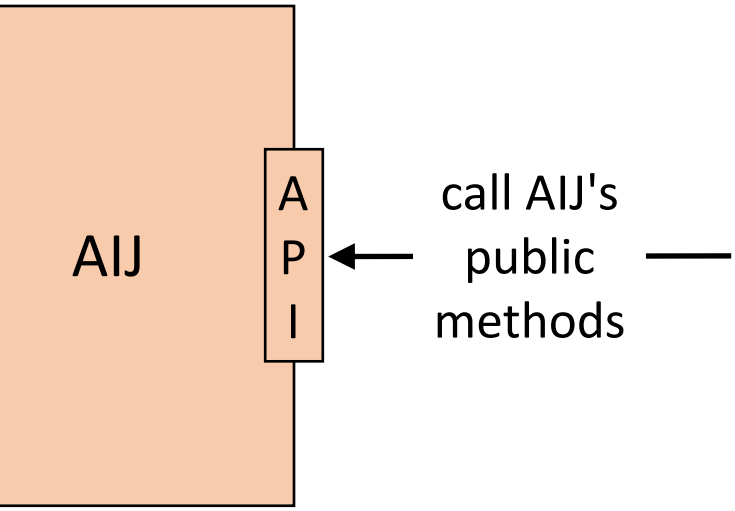
AIJ includes:

- A Java representation of the ACL2 values, terms, and environment.
- A Java implementation of the ACL2 primitives and evaluator.

AIJ provides a Java API to:

- Build the ACL2 environment.
- Evaluate ACL2 function calls:
  - Build the argument values.
  - Call the function on the arguments, obtaining a result.
  - Unbuild the result value.

AIJ provides a Java API to evaluate ACL2 function calls.




```
// build the argument values:
Value arg = Integer.make(100);
Value[] args = new Value[]{arg};

// call the function
// on the arguments,
// obtaining a result:
Symbol fun =
    Symbol.make("ACL2", "FACTORIAL");
Value result =
    Environment.call(fun, args);

// unbuild the result value:
java.math.BigInteger bigInt =
    result.getJavaBigInteger();
```

code external to AIJ

AIJ is a deep embedding in Java of an executable, side-effect-free, non-stobj-accessing subset of the ACL2 language without guards.

AIJ includes:

- A Java representation of the ACL2 values, terms, and environment.
- A Java implementation of the ACL2 primitives and evaluator.

AIJ provides a Java API to:

- Build the ACL2 environment.
- Evaluate ACL2 function calls.

This API enables external Java code to “run” ACL2 code in Java.

AIJ is a deep embedding in Java of an executable, side-effect-free, non-stobj-accessing subset of the ACL2 language without guards.

AIJ includes:

- A Java representation of the ACL2 values, terms, and environment.
- A Java implementation of the ACL2 primitives and evaluator.

AIJ provides a Java API to:

- Build the ACL2 environment.
- Evaluate ACL2 function calls.

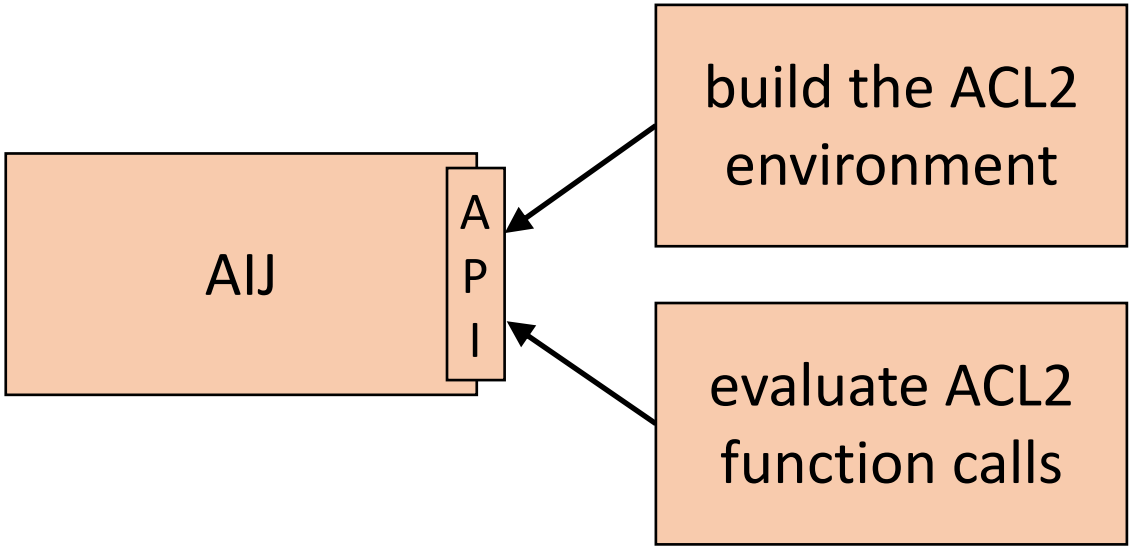
This API enables external Java code to “run” ACL2 code in Java.

By construction, the ACL2 code run via AIJ is:

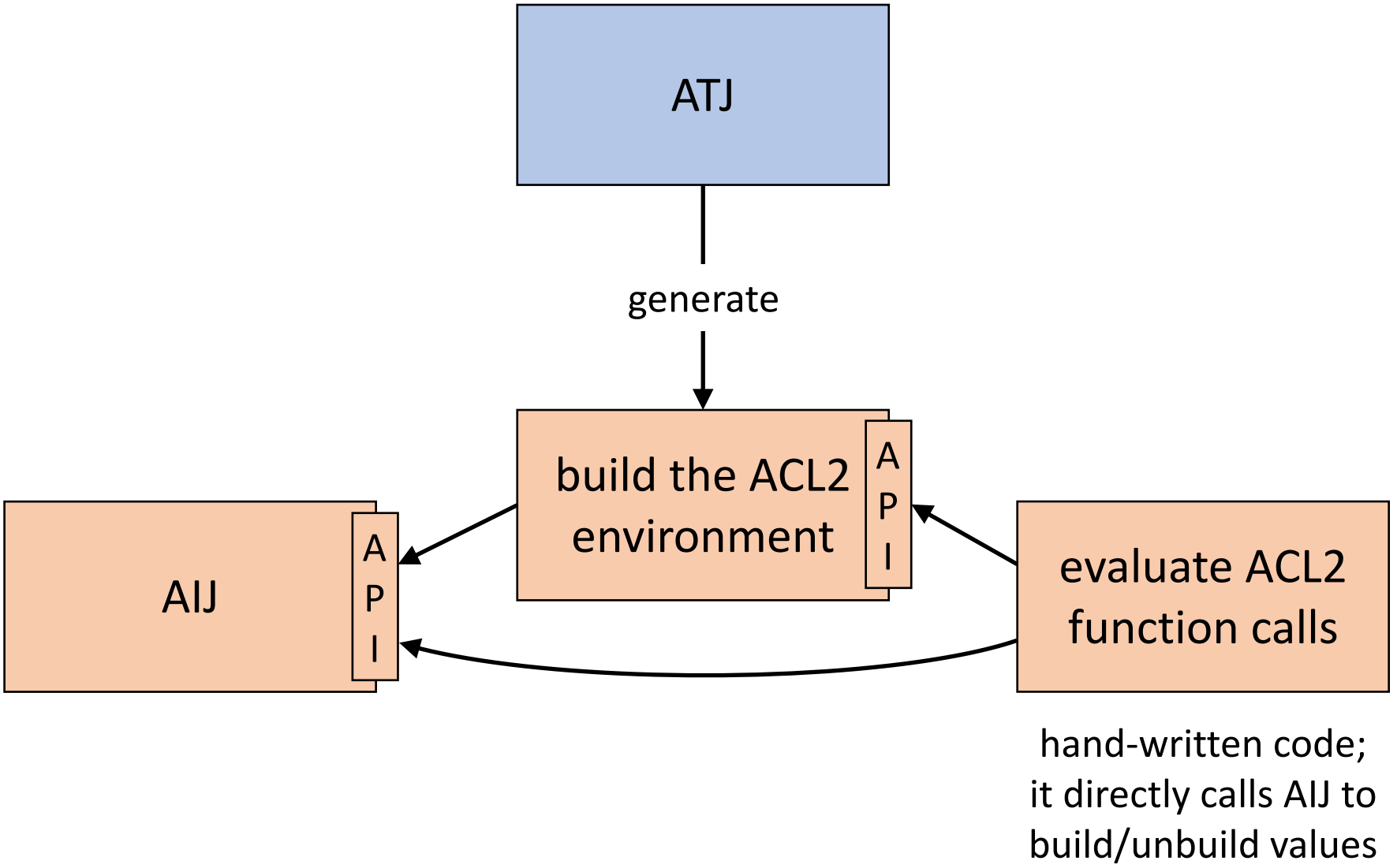
- Executable – only primitive and defined functions.
- Side-effect-free – no special Java code to carry out side effects.
- Non-stobj-accessing – only values are passed to function calls.
- Without guards – evaluation “in the logic”.

Java code external to AIJ:

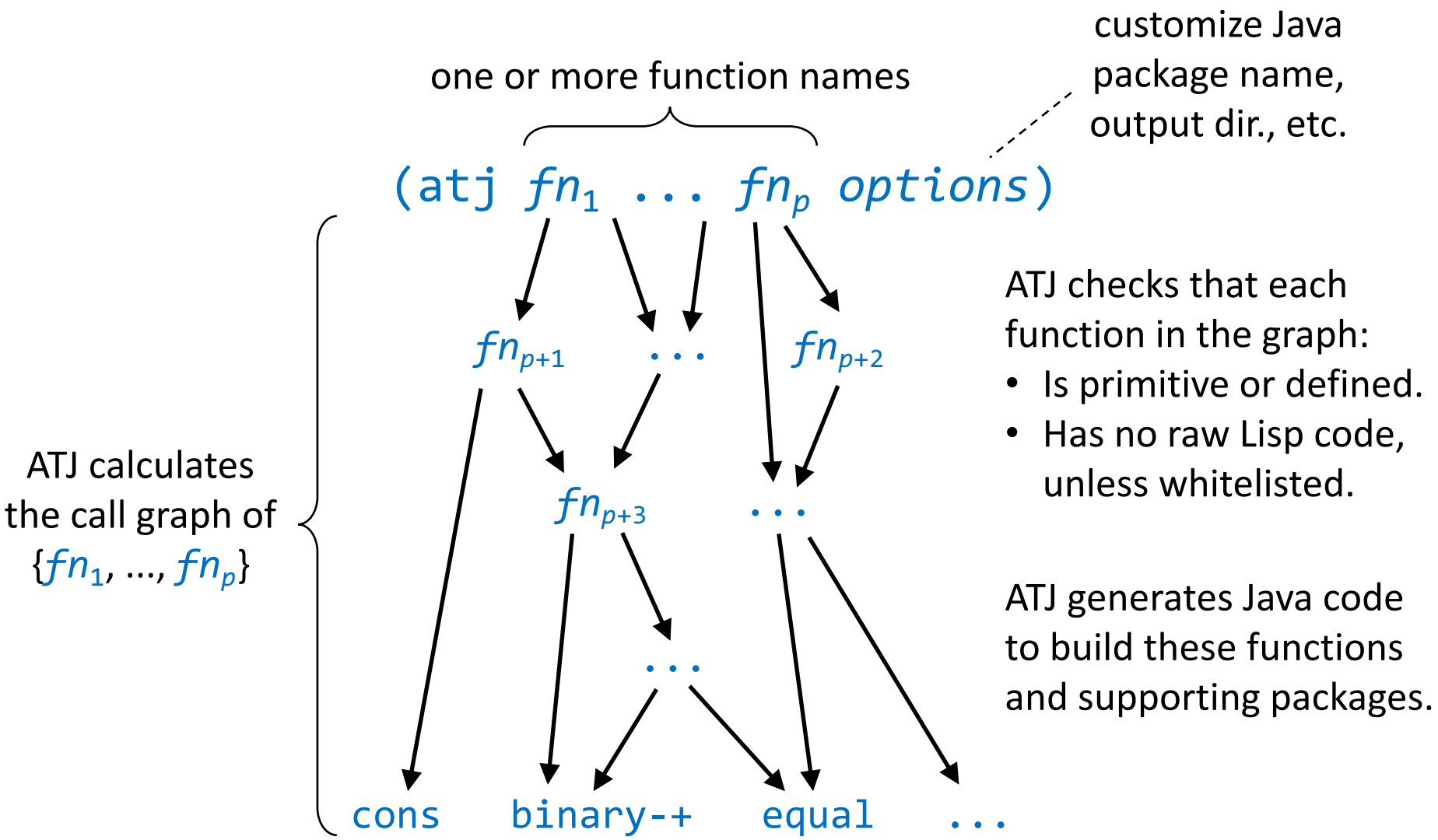
- Builds the ACL2 environment.
- Evaluates ACL2 function calls.



ATJ generates the Java code to build an ACL2 environment.



ATJ is written in ACL2. It provides a macro `atj` to generate Java files.



ATJ checks that each function in the graph:

- Is primitive or defined.
- Has no raw Lisp code, unless whitelisted.

ATJ generates Java code to build these functions and supporting packages.

ATJ generates a single Java class in a Java file.

builds the non-primitive functions in the call graph of {fn_1, ..., fn_p} and the supporting packages

```
public class ... {

    ...

    // API method to build the ACL2 environment:
    public static void initialize() {...}

    // API method to evaluate ACL2 function calls:
    public static Value call(Symbol function,
                             Value[] arguments) {...}
}
```

thin wrapper of `Environment.call`

How does evaluation time in AIJ compare to evaluation time in ACL2?

	ACL2 with guard checking `t`	ACL2 with guard checking `:none`	AIJ
factorial	1	0.8–1x	0.4–3.2x
Fibonacci	1	8–10x	17–30x
ABNF parser	1	16–87x	19–22x
tests	baseline	slowdown compared to previous column	slowdown compared to previous column

The current speed may be adequate to some applications.

There are many opportunities for further optimization.

Future work includes:

- Optimization of AIJ.
- Evaluation with ACL2's other guard checking settings.
- Side effects – native Java.
- Stobjs (user-defined).
- State – side effects.
- Generated code that calls hand-written code.
- Shallowly embedded representations of ACL2 functions in Java.
- Formal verification of the Java code.
- Generation of code in other programming languages.