Scala Interface Design for Apron

Tianhan

Goal & Contribution

- ~~Concrete goal: implement an interval analysis with Apron Library and an abstract interpreter. Concretely: I wish to plug in Apron Library into jsy-AI~~
- The desired outcome would be some work that is **extensible** in the future, rather than something that is "complete" and has full functionalities (of Apron).
- Design a general interface that hides all Apron APIs inside. The goal is that, if there're other libraries, the interface would be **general enough** to cover their functionalities as well.

- **Contribution**: an attempt to use Apron in an abstract interpreter and build Scala interface for Apron library

Overview

- Basic Concepts
- Apron Interface
- Scala Interface
  - Abstract classes
  - Implementation classes
- Example usage (Testing)

Overview

- Basic Concepts
- Apron Interface
- Scala Interface
  - Abstract classes
  - Implementation classes
- Example usage (Testing)

Basic Concepts

- $Term ::= const \mid const * variable \mid Interval * variable$
- $Expr ::= Term\ BinaryOp\ Term \mid Expr\ BinaryOp\ Term$
- $Constraint ::= Expr\ Comparator\ 0$

Overview

- Basic Concepts
- Apron Interface
- Scala Interface
  - Abstract classes
  - Implementation classes
- Example usage (Testing)

Apron Interface

- $TreeNode ::= constNode \mid variableNode \mid TreeNode_i \; BOP \; TreeNode_j$
  - $(BOP \in \{+, -, \times, \div, mod, pow\})$
- $NonlinearTerm ::= TreeNode$

- $LinearTerm ::= const * x_i \mid Interval * x_i$
- $Interval ::= (lower\_bound, upper\_bound)$

- $Term ::= LinearTerm \mid NonlinearTerm$

Apron Interface

- $LinearExpr ::= LinearTerm_i + LinearTerm_j \mid LinearExpr + LinearTerm_i \mid LinearExpr + const$
- $NonLinearExpr ::= LinearExpr \mid NonLinearTerm$
- $Constraint ::= LinearExpr\ OP\ 0 \mid NonlinearExpr\ OP\ 0$
  - $(OP \in \{>, \geq, <, \leq, \neq, =\})$

- $Domain ::= \{d_i \mid d_i = x_i \in Interval_i\}$
  - def getbound(expr): Interval
  - def getbound(variable): Interval
  - def satisfy(cons): Boolean

Overview

- Basic Concepts
- Apron Interface
- Scala Interface
  - Abstract classes
  - Implementation classes
- Example usage (Testing)

Scala Interface (abstract classes)

```scala
abstract class AbsTerm {
  def term: Any
}
```

```scala
abstract class AbsExpr {
  def expr: Any
}
```

```scala
abstract class AbsCons(val expr: AbsExpr, val op: Cop) {
  def cons: Any
}
```

```scala
abstract class AbsDom(val dim: (Int, Int), val dom: Array[AbsInterval]) {
  def getBound(expr: AbsExpr): AbsInterval
  def getBound(dim: Int): AbsInterval
  def satisfy(cons: AbsCons): Boolean
}
```

```scala
abstract class AbsInterval(val lb: Double, val ub: Double) {
  def interval: Any
}
```

Apron Interface

- $TreeNode ::= constNode \mid variableNode \mid TreeNode_i\ BOP\ TreeNode_j$
  - $(BOP \in \{+, -, \times, \div, mod, pow\})$
- $NonlinearTerm ::= TreeNode$
- $LinearTerm ::= const * xi \mid Interval$
- $Interval ::= (lower\_bound, upper\_bound)$
- $Term ::= LinearTerm \mid NonlinearTerm$

Apron Interface

- $LinearExpr ::= LinearTerm_i + LinearTermj \mid LinearExpr + LinearTermi \mid LinearExpr + const$
- $NonLinearExpr ::= LinearExpr \mid NonLinearTerm$
- $Constraint ::= LinearExpr\ OP\ 0 \mid NonlinearExpr\ OP\ 0$
  - $(OP \in \{>, \geq, <, \leq, \neq, =\})$
- $Domain ::= \{d_i \mid d_i = x_i \in Intervali\}$
  - def getbound(expr): Interval
  - def getbound(variable): Interval
  - def satisfy(cons): Boolean

Scala Interface (implemented classes)

```
class ApronLinTerm extends AbsTerm {
  private var Term: Linterm0 = _

  def term: Linterm0 = Term
```

```
class ApronNonLinTerm extends AbsTerm {
  private var Term: Texpr0Node = _

  def term: Texpr0Node = Term
```

```
abstract class AbsTerm {
  def term: Any
}
```

```
class ApronLinExpr extends AbsExpr {
  private var Expr: Linexpr0 = _

  def expr = Expr
```

```
class ApronNonLinExpr extends AbsExpr {
  private var Expr: Texpr0Intern = _

  def expr = Expr
```

```
abstract class AbsExpr {
  def expr: Any
}
```

Scala Interface (implementation classes)

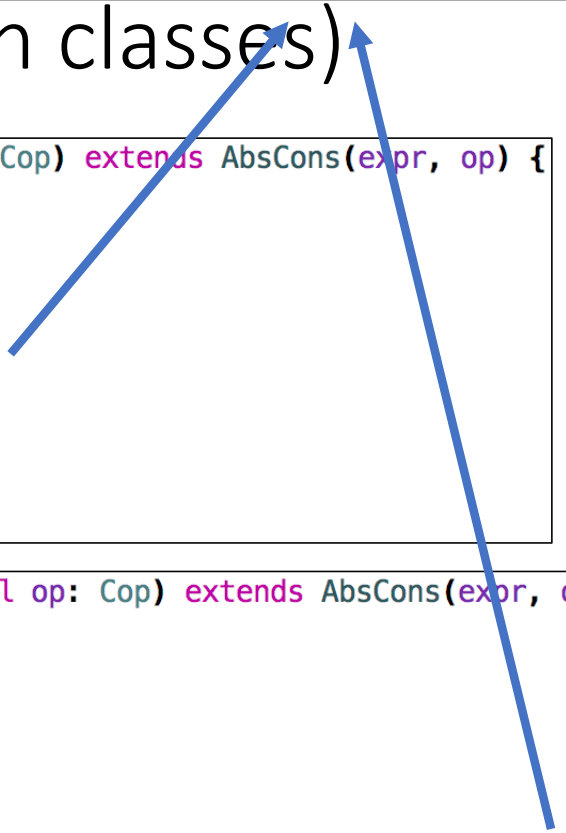
```scala
abstract class AbsCons(val expr: AbsExpr, val op: Cop) {
  def cons: Any
}
```

```scala
class ApronLinCons(override val expr: ApronLinExpr, override val op: Cop) extends AbsCons(expr, op) {
  def cons: Lincons0 = Cons

  private val Cons: Lincons0 = op match {
    case LE => new Lincons0(Tcons0.SUPEQ, neg(expr.expr))
    case LT => new Lincons0(Tcons0.SUP, neg(expr.expr))
    case GE => new Lincons0(Tcons0.SUPEQ, expr.expr)
    case GT => new Lincons0(Tcons0.SUP, expr.expr)
    case NE => new Lincons0(Tcons0.DISEQ, expr.expr)
    case EQ => new Lincons0(Tcons0.EQ, expr.expr)
  }
```

```scala
class ApronNonLinCons(override val expr: ApronNonLinExpr, override val op: Cop) extends AbsCons(expr, op) {
  def cons: Tcons0 = Cons

  private val Cons: Tcons0 = op match {
    case LE => new Tcons0(Tcons0.SUPEQ, neg(expr))
    case LT => new Tcons0(Tcons0.SUP, neg(expr))
    case GE => new Tcons0(Tcons0.SUPEQ, expr.expr)
    case GT => new Tcons0(Tcons0.SUP, expr.expr)
    case NE => new Tcons0(Tcons0.DISEQ, expr.expr)
    case EQ => new Tcons0(Tcons0.EQ, expr.expr)
  }
```

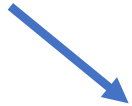


Scala Interface (implementation classes)

```scala
class ApronDom(override val dim: (Int, Int), override val dom: Array[AbsInterval]) extends AbsDom(dim, dom) {
  private val man = new Polka(true)
  private val a0 = new Abstract0(man, dim._1, dim._2, dom.map(int => int.interval.asInstanceOf[Interval]))
```

```scala
abstract class AbsDom(val dim: (Int, Int), val dom: Array[AbsInterval]) {
  def getBound(expr: AbsExpr): AbsInterval
  def getBound(dim: Int): AbsInterval
  def satisfy(cons: AbsCons): Boolean
}
```

```scala
class ApronInterval(lb: Double, ub: Double) extends AbsInterval(lb, ub) {
  def interval = new Interval(lb, ub)
```



```scala
abstract class AbsInterval(val lb: Double, val ub: Double) {
  def interval: Any
}
```

Overview

- Basic Concepts
- Apron Interface
- Scala Interface
  - Abstract classes
  - Implementation classes
- Example Usage (Testing)

Example usage

```
val dom = new ApronDom((2, 1), Array(new ApronInterval(1, 2), new ApronInterval(-3, 5), new ApronInterval(0.75, 1.2)))
println(dom + "\n\n\n")

println("Bound of x2: " + dom.getBound(2).interval + "\n")
```

```
Domain:
  [1.0,2.0]
  [-3.0,5.0]
  [0.75,1.2]



Bound of x2: [0.75,1.2]
```

Example usage

```
val ltrms = Array(new ApronLinTerm(-5, 1), new ApronLinTerm(0.1, 0.6, 0), new ApronLinTerm(0.1, 2))
println(ltrms.foldLeft("Linear terms:\n")((acc, t) => acc + "  " + t + "\n"))
val linexpr = new ApronLinExpr(2, ltrms)
println("Linear expression: " + linexpr + "\n")
println("Bound of linear expression: " + dom.getBound(linexpr).interval + "\n")
val lincons = new ApronLinCons(linexpr, LE)
println("Linear constraint: " + lincons + "\n")
println("If the given domain satisfies the linear constraint: " + dom.satisfy(lincons))
```

```
Linear terms:
  -5x1
  [0.1,0.6]x0
  1.0000000000000001e-01x2

Linear expression:  [0.1,0.6]x0 -5x1 +1.0000000000000001e-01x2 +2

Bound of linear expression: [-22.825,18.32]

Linear constraint:  [0.1,0.6]x0 -5x1 +1.0000000000000001e-01x2 +2 <= 0

If the given domain satisfies the linear constraint: false
```

Example usage

```
val txpr = new ApronNonLinTerm(ADD, new ApronNonLinTerm(MUL, 0, 1), new ApronNonLinTerm(DIV, 2, 2.0))
println("Nonlinear term: " + txpr + "\n")
val texpr = new ApronNonLinExpr(txpr)
println("Nonlinear expression: " + texpr + "\n")
println("Bound of nonlinear expression:  " + dom.getBound(texpr).interval + "\n")
val tcons = new ApronNonLinCons(texpr, LE)
println("Linear constraint: " + tcons + "\n")
println("If the given domain satisfies the linear constraint: " + dom.satisfy(tcons))
```

```
Nonlinear term: x0 * x1 + x2 / 2e+00

Nonlinear expression: x0 * x1 + x2 / 2e+00

Bound of nonlinear expression:  [-5.625,10.600000000000001]

Linear constraint: x0 * x1 + x2 / 2e+00 <= 0

If the given domain satisfies the linear constraint: false
```

Thank you