Your Connection to ICT Research

Local Search with OscaR.cbls for the Nerds who Want to Contribute

OscaR v4.0 – Spring2018

Renaud De Landtsheer, Thomas Fayolle,
Fabian Germeau, Gustavo Ospina,
Christophe Ponsard

OscaR
OPERATIONAL RESEARCH IN SCALA

FEDER
UNION EUROPEENNE
Wallonie
LE FONDS EUROPEEN DE DEVELOPPEMENT REGIONAL
ET LA WALLONIE INVESTISSENT DANS VOTRE AVENIR.

OscaR

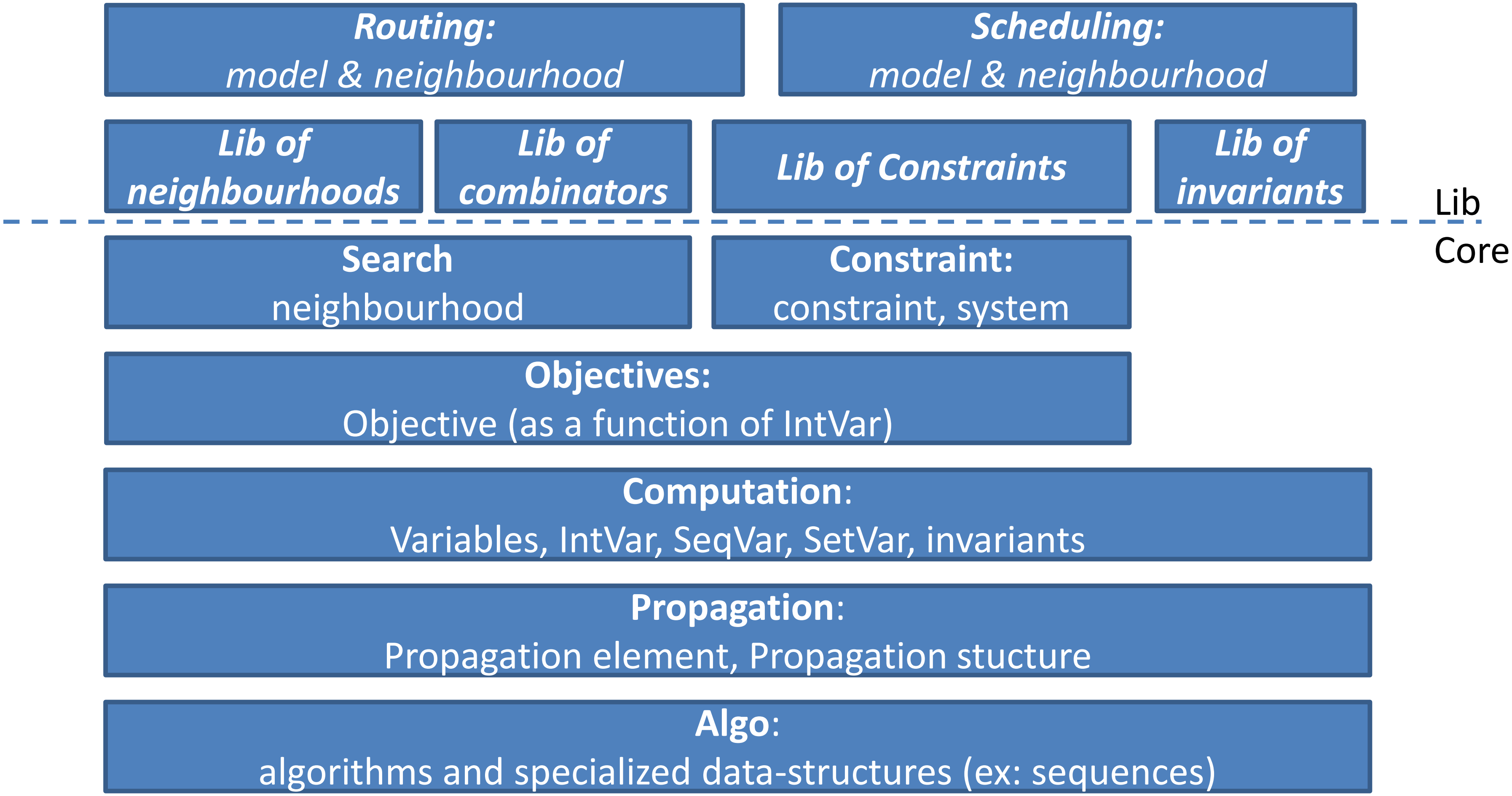
OPERATIONAL RESEARCH IN SCALA

- Oscar
  - Open source framework for combinatorial optimization
  - CP, CBLS
  - Started in 2011

- Open source LGPL license
  - https://bitbucket.org/oscarlib/oscar
  - Implemented in Scala

- Consortium
  - CETIC, UCL, N-Side Belgium
  - Contributions from UPPSALA Sweden

«Use» *Architecture of OscaR.cbls*



Routing:
model & neighbourhood

Scheduling:
model & neighbourhood

Lib of neighbourhoods

Lib of combinators

Lib of Constraints

Lib of invariants

Lib

Core

Search
neighbourhood

Constraint:
constraint, system

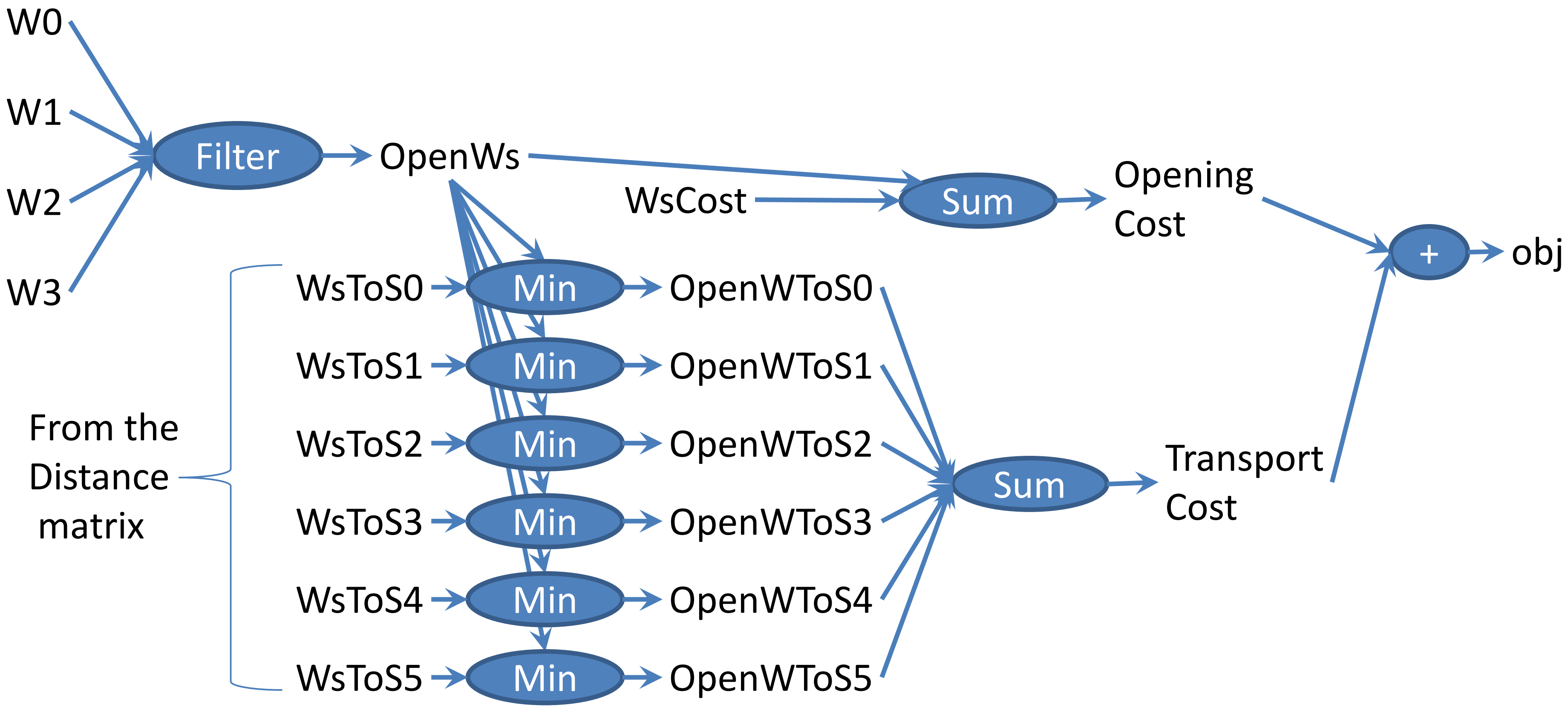
Objectives:
Objective (as a function of IntVar)

Computation:
Variables, IntVar, SeqVar, SetVar, invariants

Propagation:
Propagation element, Propagation stucture

Algo:
algorithms and specialized data-structures (ex: sequences)

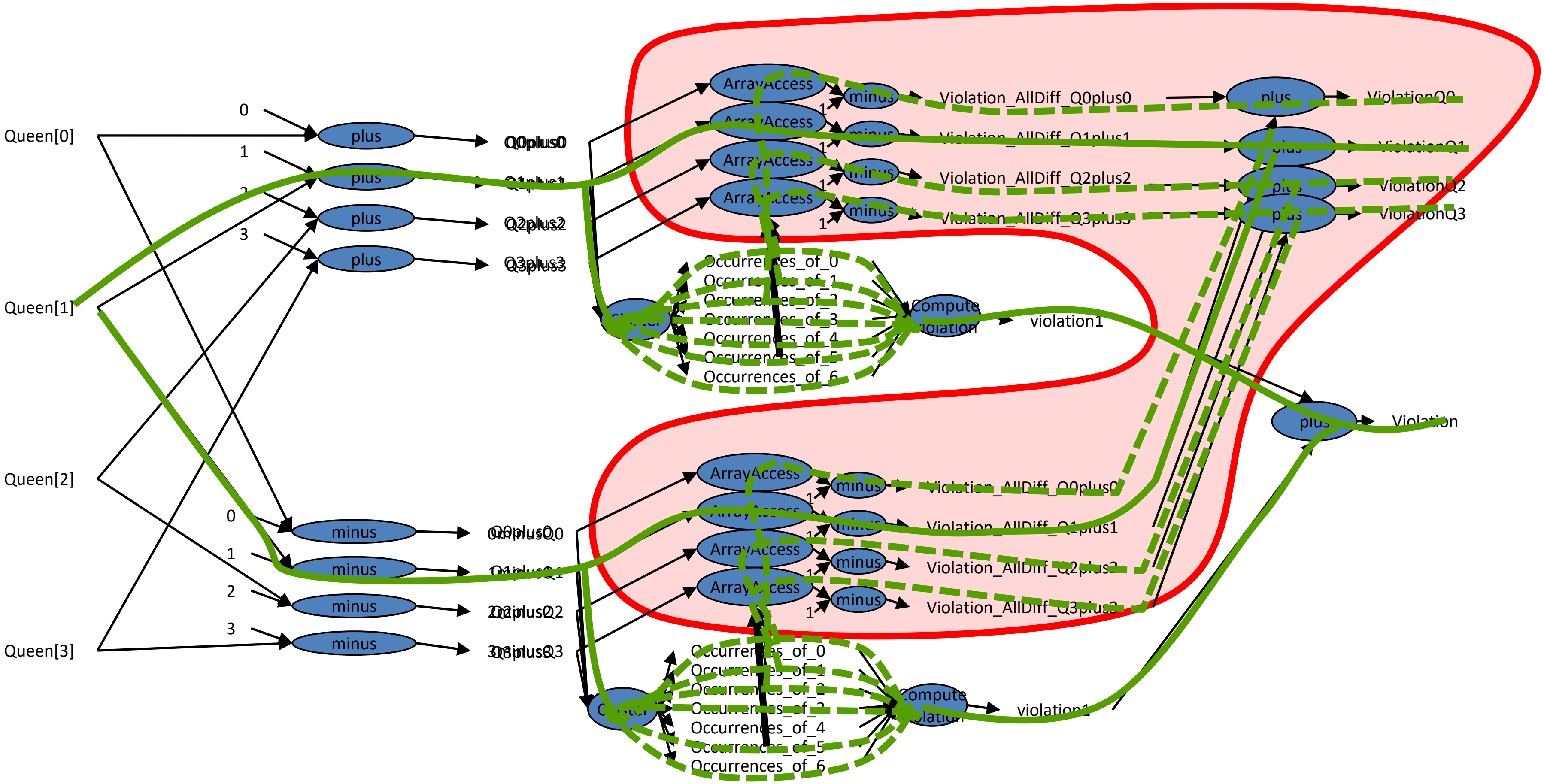
A quick look under the hood: Propagation graph for the WLP(4,6)



Propagation: update the output(s) to reflect a change on the inputs

- **Single wave**: elements are touched at most once
- **Incremental**: all invariants update their outputs incrementally
- **Selective**: only things that need to be updated wrt. changes are updated
- **Partial**: only things contributing to the needed output are updated

A quick look under the hood: Selective + partial propagation



How propagation is coordinated?

- When model is closed
  - Static propagation graph are sorted
    by distance to decision (aka input) variables
    - So each element belongs to a certain layer
    - There are sot so many layers, but they are very wide
  - Each propagation element is tagged by this distance

- Upon propagation

```
toPropagateHeap.insert(changedInputVariables)
while(toPropagateHeap.nonEmpty){
        toPropagateHeap.popFirst.propagate()
        toPropagateHeap.insert(newNodesToPropagate)
}
```

  - toPropagateHeap sorts by distance (stored as a tag)
  - toPropagateHeap aggregates on same layer
    - Insert is
      - O(log(nbLayers)) -time for the first element in the layer,
      - O(1) -time for other ones
    - Pop is
      - O(log(nbLayers)) -time for the last element in the layer,
      - O(1) -time for other ones

Propagation & Notification

- Propagation
  - is the global process of updating the model
  - Managed by the store
  - It calls *propagate()*
    - On the relevant propagation elements
    - In the right order
- Propagation elements can be
  - Variables:
    - When a variable is propagated, it notifies its value change to the invariants listening to it
  - Invariants:
    - They can update their output in code that implement this method
    - That they can also perform the updates when they are notified

Adding new invariants to OscaR.cbls

- Propagation revisited
- A simple invariant

  sum
- Dynamic invariants

  SumElement
- Variable and their notifications
  - Int
  - Set
    - Value-wise notification
  - Seq
    - Checkpoints
- Generic approach for global routing constraints

Helpers available for the simplest one

- Helpers are abstract invariants parameterized by
  - Some IntValue
  - Some function
    - Int → Int
    - Int × Int → Int
  - A domain; the rage of possible value for the output of the function, given the range of the IntValue
- Example:

```
case class Abs(v: IntValue)
  extends Int2Int(
    v,   // the IntValue
    (x: Int) => x.abs, // the transformation function
    if (v.min < 0 && 0 < v.max) 0 to (-v.min max v.max)
    else{
      val a = v.min.abs
      val b = v.max.abs
      if(a<b) a to b else b to a
     })
```

A simple invariant: sum

```
class Sum(vars: Iterable[IntValue])
  extends IntInvariant(
        vars.map(_.value).sum, // initial value
        vars.map(_.min).sum to vars.map(_.max).sum) // range of output
  with IntNotificationTarget{

  for (v <- vars) registerStaticAndDynamicDependency(v)
  finishInitialization()

  override def notifyIntChanged(v: ChangingIntValue, id:Int,
                                oldVal: Int, newVal: Int) {
    this :+= newVal - oldVal
  }

  override def checkInternals(c: Checker) {
    c.check(this.value == vars.map(_.value).sum)
  }
}
```

A helper for invariants whose single output is an IntValue

Which variable you want to be notified about when they change value

When vars change value they call this method from IntNotificationTarget trait id is a value transmitted through register… method

A debug procedure that you should implement and that is called by the store when instantiated with flag debug=true

Notification

- When an invariant is notified, the following method is called by the variable:

```
def notifyIntChanged(v: ChangingIntValue, id:Int,
                                         oldVal: Int, newVal: Int)
```

Where:

- V is a reference to the variable
- Id is an integer value that is optionally passed as a parameter to the method for registering dependencies
- oldVal is the value before the change
- newVal is the value after the change

Static and dynamic graph

- Invariants declare their dependencies:
  - What variable they listen to
  - What variable they control (set the value of)

- Static propagation graph
  - Declared at startup
  - Used to coordinate propagation wave

- Dynamic propagation graph
  - Edges are a subset of the static propagation graph
  - Can be changed by invariants
  - Used by variable to notify listening invariants

- For an invariant to play with dynamic dependencies, it must be defined `with VaryingDependencies`
  - So it gets additional back-end data-structure and the method, `key = registerDynamicDependency(var)`
  - This method returns a « key » to unregister. To unregister, simply call `key.performRemove()`
  - Both methods have O(1)-time complexity

A sum invariant with a subset (1/2)

```
case class SumElements(vars: Array[IntValue], cond: SetValue)
  extends IntInvariant(cond.value.map(vars(_).value).sum)
  with VaryingDependencies
  with IntNotificationTarget with SetNotificationTarget{

  val keyForRemoval =  Array.fill(vars.length) {null}

  registerStaticDependency(cond)
  registerDeterminingDependency(cond)
  registerstaticDependencies(vars)
  for(i <- cond.value){
    keyForRemoval(i) = registerDynamicDependency(vars(i))
  }

  finishInitialization()

  override def notifyIntChanged(v: ChangingIntValue, index: Int,
                                oldVal: Int, newVal: Int) {
    this :+= (newVal - oldVal)
  }
```

A sum invariant with a subset (2/2)

```
  override def notifySetChanges(v: ChangingSetValue, d: Int,
                                addedValues: Iterable[Int],
                                removedValues: Iterable[Int],
                                oldValue: SortedSet[Int],
                                newValue: SortedSet[Int]) {
    for (added <- addedValues){
      keyForRemoval(added) = registerDynamicDependency(
                                     vars(added))
      this :+= vars(added).value
    }
    for(removed <- removedValues) {
      keyForRemoval(removed).performRemove()
      keyForRemoval(removed) = null
      this :-= vars(removed).value
    }
  }

  override def checkInternals(c:Checker) {
    c.check(this.value == cond.value.map(vars(_).value).sum)
  }
}
```

Propagation after notification

- Most invariants can update their update in the notification procedure
- Sometime this is not desirable,
  - because computation takes a significant amount of time,
  - it is better to wait for all notifications to be received, and perform this computation after
- To this end, invariants can also be propagated
  - They have to override the method

```
override def performInvariantPropagation()
```

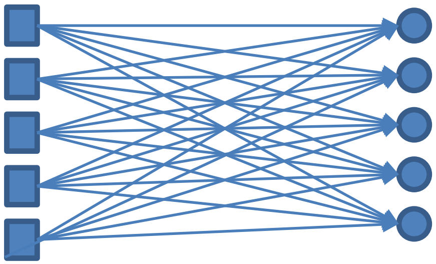
  - Upon notification, the invariant must schedule itself for propagation (set itself into the propagation heap) by calling

```
scheduleForPropagation()
```

A common stereotype found in many CBLS models

- An array of CBLS variable and an array of Invariants

v variables



i invariants listening to the array

v*i dependencies in static propagation graph

```
SumElements(
    Array[IntValue],
    cond: SetValue)
```

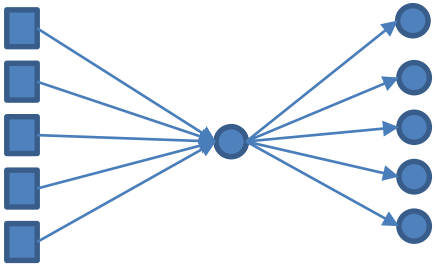
- Many dependencies in the static propagation graph
  - Close requires sorting the propagation graph, slow down!
- Also: the invariants all want to compute the same static result on the array

Ex: $\sum_{v\ in\ the\ array} v.max$

Solution: Bulking

- OscaR.cbls can create an « artificial » bulky node in the middle

v variables



i invariants listening to the array

```
SumElements(
    Array[IntValue],
    cond: SetValue)
```

v+i dependencies in
static propagation graph (instead of v*i)

- This node is called « bulk »
  - Used to symbolize dependencies,
  - Reduces memory consumption
  - Speed up graph algo that run when `store.close()`
  - Can memoïze static results computed by invariants (see API)

Your Connection to ICT Research

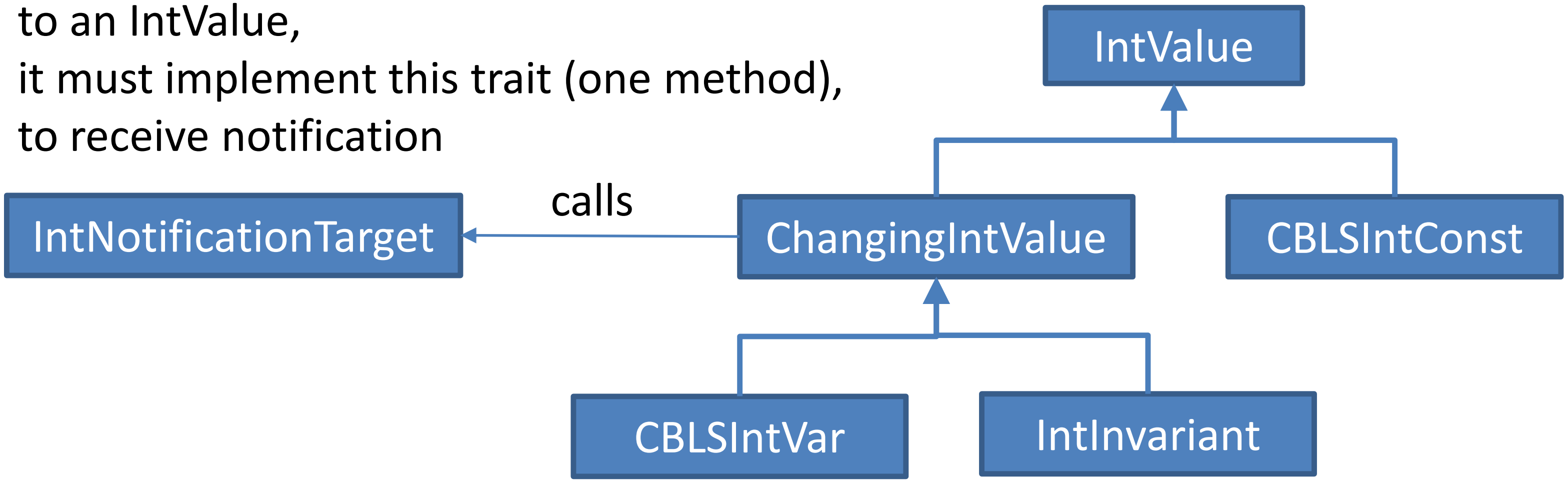
A sum invariant with Bulking

```
case class SumElements(vars: Array[IntValue], cond: SetValue)
  extends IntInvariant(cond.value.map(vars(_).value).sum)
  with Bulked[IntValue, Unit] with VaryingDependencies
  with IntNotificationTarget with SetNotificationTarget{

  val keyForRemoval =  Array.fill(vars.length) {null}

  registerStaticDependency(cond)
  registerDeterminingDependency(cond)

  ~~registerstaticDependencies(vars)~~
  bulkRegister(vars)
```

Object model of Int type (similar for Set and Seq)

Invariant should input IntValue,
so they can be given CBLSIntVar, IntConst
or other invariant as input

If an invariant listens
to an IntValue,
it must implement this trait (one method),
to receive notification



An invariant with a single output,
of type Integer

Value-wise notification for Set

- « Only call me about these values »
  - Invariants specify a set of integer values to the variable
  - It is notified about change only when at least one of these value is added to or removed from the set

- HowTo:
  - Dyamic dependency is declared through:

```
val key:ValueWiseKey =
          registerDynamicValueWiseDependency(cond)
```

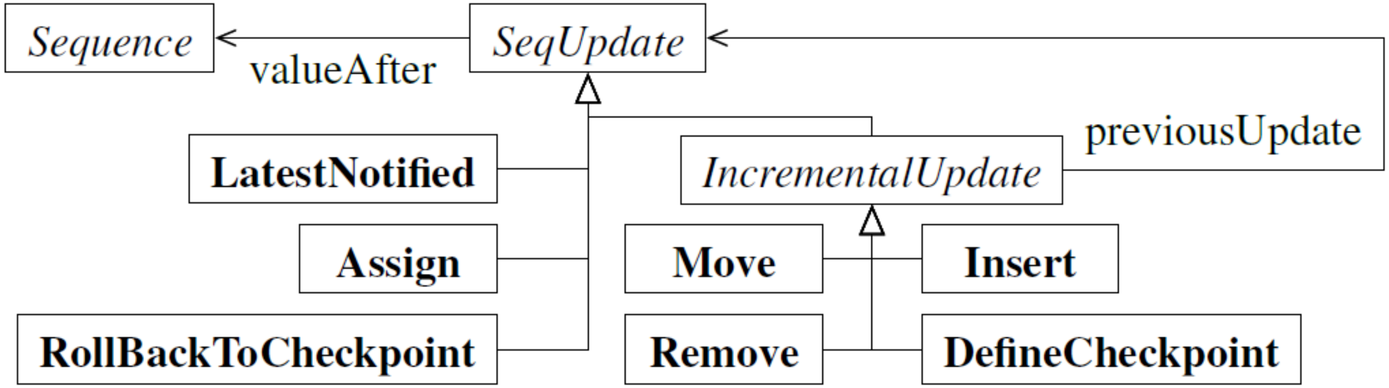
  - The returned key has two additional methods:

```
key.addToKey(newValueToListenAbout)
key.removeFromKey(valueIAmNotInterestedAboutAnymore)
```

- This adds a filter to the notifications

Sequences and their symbolic notifications

- Incremental updates
  - Three incremental operations:
    - Insert
    - Delete
    - Move(from,to,moveAfter,flip)
  - Additional operations
    - rollBack
    - assign
- Symbolic notification messages: *SeqUpdate*



- *Check our paper at CPAIOR'18*

OrElse combinator

- Only one method to implement:

```
class OrElse(a: Neighborhood, b: Neighborhood)
     extends NeighborhoodCombinator(a, b) {

  override def getMove(obj: Objective,
                       initialObj:Int,
                       acceptanceCriterion:(Int, Int)=>Boolean)
               : SearchResult = {

    a.getMove(obj, initialObj, acceptanceCriterion) match {
      case NoMoveFound =>
        a.reset()
        b.getMove(obj, initialObj, acceptanceCriterion)
      case x => x
    }
  }

}
```

References about OscaR.cbls internals

1. Renaud De Landtsheer, Christophe Ponsard, OscaR.cbls : an open source framework for constraint-based local search, 27th ORBEL Annual Meeting, Kortrijk, Belgium, February 7-8 2013.
2. Renaud De Landtsheer, Yoann Guyot, Gustavo Ospina, Christophe Ponsard, Local Search with OscaR.CBLS, Workshop Design and Analysis of Meta-heuristics, Antwerp, 17-18 March 2016.
3. Renaud De Landtsheer, Yoann Guyot, Gustavo Ospina, Christophe Ponsard, Towards the Complexity of Differentiation Through Lazy Updates in Local Search Engines, 30th ORBEL Annual Meeting, Louvain-La-Neuve, Belgium, January 28-29 2016
4. Renaud De Landtsheer, Yoann Guyot, Gustavo Ospina, Christophe Ponsard, Adding a Sequence Variable to the OscaR.CBLS Engine, 31th ORBEL Annual Meeting, Brussels, Belgium, February 2-3, 2017
5. Renaud De Landtsheer, Gustavo Ospina, Yoann Guyot, Fabian Germeau, Christophe Ponsard, Supporting Efficient Global Moves on Sequences in Constraint-based Local Search Engines, Proceedings of the 6th International Conference on Operations Research and Enterprise Systems, 171-180, 2017, Porto, Portugal
6. Renaud De Landtsheer, Yoann Guyot, Gustavo Ospina, and Christophe Ponsard. Recent developments of metaheuristics, chapter Combining Neighborhoods into Local Search Strategies, pages 43–57. Springer, 2018.
7. Generic Support for Global Routing Constraint in Constraint-Based Local Search Frameworks, Quentin Meurisse, Renaud De Landtsheer, 32th ORBEL Annual Meeting, Liege, Belgium, February 1-2 2018
8. Renaud De Landtsheer, Fabian Germeau, Yoann Guyot, Gustavo Ospina, Christophe Ponsard, Easily Building Complex Neighbourhoods With the Cross-Product Combinator, 32th ORBEL Annual Meeting, Liege, Belgium, February 1-2 2018
9. Renaud De Landtsheer, Yoann Guyot, Gustavo Ospina, Fabian Germeau, and Christophe Ponsard, Reasoning on Sequences in Constraint-Based Local Search Frameworks, accepted at CPAIOR2018, 15th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research June 26-29, 2018, Delft, The Netherlands