What is a Constraint?

- A constraint is used to restrict the domain value of variables.
- In other words, it will either remove domain values or do nothing.

What does it actually do?

- **Post**: attach the constraint to variables
- **Propagate**: domain reduction
A constraint is implemented by means of two classes in ILOG Solver, i.e.

- A Handle class
  class IlcConstraint
- An Implementation class.
  class IlcConstraintI
class IlcConstraintI : public IlcGoalI {
public:
    IlcConstraintI(IlcManager m);
    IlcConstraintI(IlcManagerI* manager);
    virtual void display(ostream &str) const;
    IlcConstraintI* getCopy(IlcManagerI* = 0);
    virtual IlcConstraintI* makeCopy(IlcManagerI* manager) const;
    virtual IlcConstraintI* makeOpposite() const;
    virtual void metaPost(IlcGoalI* metaconstraint);
    virtual void propagate()=0;
    virtual void post()=0;
    virtual IlcBool isViolated() const;
};
Three Pure Virtual Functions

- **void propagate();**
  - Define the propagation (domain reduction) behaviors.

- **void post();**
  - Define which events `propagate()` executes.

- **IlcBool isViolated();**
  - Return `IlcTrue` if the constraint cannot be satisfied and vice versa.
What to propagate?

- Define the propagation rules of the new constraint before implementing it.

- For example, the propagation rules of \((x \neq y)\) would be,
  - If \(x\) is bound to a value, then the domain of \(y\) cannot contain this value.
  - If \(y\) is bound to a value, then the domain of \(x\) cannot contain this value.
Translation with Modifiers

So, \((x \neq y)\) can be defined by,

```java
If(x.isBound())
    y.removeValue(x.getValue());
If(y.isBound())
    x.removeValue(y.getValue());
```

Various elementary modifiers can be used to define a new constraint,

- `setRange()`, `setValue()`
- `setMin()`, `setMax()`
- `removeValue()`,
- `removeInterval()`,
- ...etc.
Propagation Events

- **Value Event**
  - When a variable has been bounded.

- **Range Event**
  - When the domain boundary of the variable has been changed.

- **Domain Event**
  - When the domain of the variable has been modified.
An Event Example

- Let’s consider a constrained integer variable, \( \text{var}[3..4] \).
- If we post a constraint, \( \text{var} \neq 3 \),
- Which event(s) will be triggered?
There are three kinds of attachment function for integer expression.

- `whenValue(IlcGoal ct)`
  - It attaches the goal `ct` to the value event of the expression under consideration.

- `whenRange(IlcGoal ct)`
  - It attaches the goal `ct` to the range event of the expression under consideration.

- `whenDomain(IlcGoal ct)`
  - It attaches the goal `ct` to the domain event of the expression under consideration.
Whenever any of these event is triggered, the `execute()` function of the corresponding `IlcGoal` will be called.

Since the class `IlcConstraint` is a subclass of the class `IlcGoal`, `IlcConstraint` can be directly attached to integer expression.

And the `execute()` function first calls `post()` (only for 1st time invocation) and then `propagate()`.
For example, attaching a constraint \( ct \) to variable \( x \) and \( y \) in value event can be implemented by:

\[
\begin{align*}
  x & \text{.whenValue}(ct) ; \\
  y & \text{.whenValue}(ct) ;
\end{align*}
\]
In the previous example, the constraint ($x \neq y$) is said to be violated, if and only if variable $x$ and $y$ are bound to the same value. Hence,

```c
IlcBool isViolated() {
    return (x.isBound() && y.isBound() &&
            x.getValue() == y.getValue());
}
```
class MyDiff : public IlcConstraintI {
    IlcIntExp _x, _y;

public:
    MyDiff(IlcManager m, IlcIntExp x, IlcIntExp y);
    ~MyDiff(){}; // empty destructor
    virtual void propagate();
    virtual void post();
    virtual IlcBool isViolated() const;
};

MyDiff::MyDiff(IlcManager m, IlcIntExp x, IlcIntExp y) : IlcConstraintI(m), _x(x), _y(y) {}
void MyDiff::propagate () {
    if (_x.isBound()) _y.removeValue(_x.getValue());
    if (_y.isBound()) _x.removeValue(_y.getValue());
}

void MyDiff::post () {
    x.whenValue(this);
    y.whenValue(this);
}

IlcBool MyDiff::isViolated() const {
    return (_x.isBound() && _y.isBound() &&
            _x.getValue()==_y.getValue());
}
The Handle Class

- The Handle class is used to access the corresponding implementation class.
- It can be implemented as follow,

```
IlcConstraint myDiff(IlcIntExp x, IlcIntExp y){
    IlcManager m = x.getManager();
    return new (m.getHeap()) MyDiff(m, x, y);
}
```
IlcConstraint myDiff(IlciIntExp x, IlcIntExp y) {
    IlcManager m = x.getManager();
    return new (m.getHeap()) MyDiff(m, x, y);
}

int main() {
    IlcManager m(IlcNoEdit);
    IlcIntVar X(m, 0, 10, "X"),
    IlcIntVar Y(m, 0, 10, "Y");

    m.add(myDiff(X, Y));
    cout << X << endl << X << endl;

    m.add(X == 3);
    cout << X << endl << X << endl;

    m.end(); return 0;
}
Output

\[ x[0..10] \]
\[ y[0..10] \]

\[ x[5] \]
\[ y[0..4 6..10] \]
Any Questions...