Reversible Domains

First we start with a data structure you might not have heard of …

sparse sets

Consider this:

- A variable has a domain of values
- This is fixed at the top of search
- Going down a branch of search the domain may decrease
- Going down a branch in search a domain ***never*** increases
- On a backtrack (up a branch) deleted values might be returned to the domain
- The sparse set goes some way to allowing this

We represent a (unordered) set using two arrays and one pointer

We represent a (unordered) set using two arrays and one pointer

value[i] = x : the ith element of the set is x

We represent a (unordered) set using two arrays and one pointer

value[i] = x : the ith element of the set is x

location[x] = i: x is the ith value in the set

We represent a (unordered) set using two arrays and one pointer

value[i] = x : the ith element of the set is x

location[x] = i: x is the ith value in the set

last: is the position of the last value in the set

We represent a (unordered) set using two arrays and one pointer

value[i] = x : the ith element of the set is x

location[x] = i: x is the ith value in the set

last: is the position of the last value in the set

Values are removed from the set by swapping, and returned by resetting a pointer

last = 4

new Domain(pb,0,4)

location	0	1	2	3	4
value	0	1	2	3	4

last = 4

new Domain(pb,0,4)

location	0	1	2	3	4
value	0	1	2	3	4

last = 3

remove(3)

location	0	1	2	4	3
value	0	1	2	4	*3*

last = 4

new Domain(pb,0,4)

location	0	1	2	3	4
value	0	1	2	3	4

last = 3

remove(3)

location	0	1	2	4	3
value	0	1	2	4	*3*

We have swapped the values 3 and 4, i.e. the value 3 with the last value in the set

- value: 4 goes where 3 was
- location: the value 4 is now in the 3d position of the set
- location: the value 3 is now in the 4th position of the set
- last position in the set is position 3
- ***The value 3 is not in the set!***

last = 4

new Domain(pb,0,4)

location	0	1	2	3	4
value	0	1	2	3	4

last = 3

remove(3)

location	0	1	2	4	3
value	0	1	2	4	*3*

last = 2

remove(1)

location	0	3	2	4	1
value	0	4	2	*1*	*3*

last = 4

new Domain(pb,0,4)

location	0	1	2	3	4
value	0	1	2	3	4

last = 3

remove(3)

location	0	1	2	4	3
value	0	1	2	4	*3*

last = 2

remove(1)

location	0	3	2	4	1
value	0	4	2	*1*	*3*

location of value 0 is 0
location of value 1 is 3
location of value 2 is 2
location of value 3 is 4
location of value 4 is 1

last = 4

new Domain(pb,0,4)

location	0	1	2	3	4
value	0	1	2	3	4

last = 3

remove(3)

location	0	1	2	4	3
value	0	1	2	4	*3*

last = 2

remove(1)

location	0	3	2	4	1
value	0	4	2	*1*	*3*

location of value 0 is 0
location of value 1 is 3
location of value 2 is 2
location of value 3 is 4
location of value 4 is 1

last position is 2

last = 4

new Domain(pb,0,4)

location	0	1	2	3	4
value	0	1	2	3	4

last = 3

remove(3)

location	0	1	2	4	3
value	0	1	2	4	*3*

last = 2

remove(1)

location	0	3	2	4	1
value	0	4	2	*1*	*3*

location of value 0 is 0
location of value 1 is 3
location of value 2 is 2
location of value 3 is 4
location of value 4 is 1

last position is 2

last = 4

new Domain(pb,0,4)

location	0	1	2	3	4
value	0	1	2	3	4

last = 3

remove(3)

location	0	1	2	4	3
value	0	1	2	4	*3*

last = 2

remove(1)

location	0	3	2	4	1
value	0	4	2	*1*	*3*

location of value 0 is 0
location of value 1 is 3
location of value 2 is 2
location of value 3 is 4
location of value 4 is 1

last position is 2

Any thing beyond position 2 is not in the set

last = 4

new Domain(pb,0,4)

location	0	1	2	3	4
value	0	1	2	3	4

last = 3

remove(3)

location	0	1	2	4	3
value	0	1	2	4	*3*

last = 2

remove(1)

location	0	3	2	4	1
value	0	4	2	*1*	*3*

last = 1

remove(0)

location	2	3	0	4	1
value	2	4	*0*	*1*	*3*

Complexity:

- remove(x) is O(1)
  - a swap operation
- contains(x) is O(1)
  - is location[x] <= last?
- removeAllBut(x) is O(1)
  - swap between x and what is in location 0
  - set last to be 0
  - used for instantiation of a variable
- removeBelow(x) is O(n) (amortised O(1))
- removeAbove(x) is O(n) (amortised O(1))
- min and max as above …

How could I go back to the way things were … for example back to where we were before we did all those removals?

last = 1

location	2	3	0	4	1
value	2	4	*0*	*1*	*3*

How could I go back to the way things were … for example back to where we were before we did all those removals?

That's how!

last = 4

location	2	3	0	4	1
value	2	4	*0*	*1*	*3*

How could I go back to the way things were … for example back to where we were before we did all those removals?

O(1)

last = 4

location	2	3	0	4	1
value	2	4	*0*	*1*	*3*

reversible variables

reversible variables

It’s all done with stacks

A ***reversible variable*** & a reversible integer

```
public class ReversibleVar {

    Problem pb;
    int value;
    Stack<Integer> whenChanged;
    Stack<Integer> priorValue;

    public void restoreValue(){
        whenChanged.pop();
        value = priorValue.pop();
    }

}
```

A ***reversible variable*** & a reversible integer

```
public class ReversibleVar {

    Problem pb;
    int value;
    Stack<Integer> whenChanged;
    Stack<Integer> priorValue;

    public void restoreValue(){
        whenChanged.pop();
        value = priorValue.pop();
    }

}
```

A stack of “worlds”

A ***reversible variable*** & a reversible integer

```
public class ReversibleVar {

    Problem pb;
    int value;
    Stack<Integer> whenChanged;
    Stack<Integer> priorValue;

    public void restoreValue(){
        whenChanged.pop();
        value = priorValue.pop();
    }

}
```

A trail (history) of values

A ***reversible variable*** & a reversible integer

```
public class ReversibleVar {

    Problem pb;
    int value;
    Stack<Integer> whenChanged;
    Stack<Integer> priorValue;

    public void restoreValue(){
        whenChanged.pop();
        value = priorValue.pop();
    }

}
```

A ***reversible variable*** & a reversible integer

```
public class ReversibleVar {

    Problem pb;
    int value;
    Stack<Integer> whenChanged;
    Stack<Integer> priorValue;

    public void restoreValue(){
        whenChanged.pop();
        value = priorValue.pop();
    }

}
```

Backtrack ☺

A reversible variable & ***a reversible integer***

```
public class ReversibleInt extends ReversibleVar {

    public ReversibleInt(Problem pb,int initialValue){
        this.pb      = pb;
        whenChanged = new Stack<Integer>();
        priorValue  = new Stack<Integer>();
        value       = initialValue;
        whenChanged.push(pb.world);
    }

    public int getValue(){return value;}

    public void setValue(int x){
        if (value != x && whenChanged.peek() != pb.world){
            pb.trail.peek().add(this);
            whenChanged.push(pb.world);
            priorValue.push(value);
        }
        value = x;
    }
}
```

A reversible variable & ***a reversible integer***

```
public class ReversibleInt extends ReversibleVar {

    public ReversibleInt(Problem pb,int initialValue){
        this.pb      = pb;
        whenChanged = new Stack<Integer>();
        priorValue  = new Stack<Integer>();
        value       = initialValue;
        whenChanged.push(pb.world);
    }

    public int getValue(){return value;}

    public void setValue(int x){
        if (value != x && whenChanged.peek() != pb.world){
            pb.trail.peek().add(this);
            whenChanged.push(pb.world);
            priorValue.push(value);
        }
        value = x;
    }
}
```

A reversible variable & ***a reversible integer***

```
whenChanged.push(pb.world);
```

A reversible variable & ***a reversible integer***

```
whenChanged.push(pb.world);
```

pb is a Problem and world is just an integer, nothing fancy.
We might think of "world" as depth in search.
Top of search is world zero.

A reversible variable & ***a reversible integer***

```
public class ReversibleInt extends ReversibleVar {

    public ReversibleInt(Problem pb,int initialValue){
        this.pb      = pb;
        whenChanged = new Stack<Integer>();
        priorValue  = new Stack<Integer>();
        value       = initialValue;
        whenChanged.push(pb.world);
    }

    public int getValue(){return value;}

    public void setValue(int x){
        if (value != x && whenChanged.peek() != pb.world){
            pb.trail.peek().add(this);
            whenChanged.push(pb.world);
            priorValue.push(value);
        }
        value = x;
    }
}
```

A reversible variable & ***a reversible integer***

```
public class ReversibleInt extends ReversibleVar {

    public ReversibleInt(Problem pb,int initialValue){
        this.pb      = pb;
        whenChanged = new Stack<Integer>();
        priorValue  = new Stack<Integer>();
        value       = initialValue;
        whenChanged.push(pb.world);
    }

    public int getValue(){return value;}

    public void setValue(int x){
        if (value != x && whenChanged.peek() != pb.world){
            pb.trail.peek().add(this);
            whenChanged.push(pb.world);
            priorValue.push(value);
        }
        value = x;
    }
}
```

A reversible variable & ***a reversible integer***

Is this a new (different) value and is it the first time the value has changed in this "world"?

```
public void setValue(int x){
    if (value != x && whenChanged.peek() != pb.world){
        pb.trail.peek().add(this);
        whenChanged.push(pb.world);
        priorValue.push(value);
    }
    value = x;
}
```

A reversible variable & ***a reversible integer***

If "yes" … the problem (pb) has a trail (and that's a stack) push this reversible integer onto the list that is at the top of the trail (this is the list of reversible that have changed in this world).

```
public void setValue(int x){
    if (value != x && whenChanged.peek() != pb.world){
        pb.trail.peek().add(this);
        whenChanged.push(pb.world);
        priorValue.push(value);
    }
    value = x;
}
```

A reversible variable & ***a reversible integer***

If "yes" … on this reversible integer record when (what world) it changed and save of f the value it had before making the change (yet another stack)

```
public void setValue(int x){
    if (value != x && whenChanged.peek() != pb.world){
        pb.trail.peek().add(this);
        whenChanged.push(pb.world);
        priorValue.push(value);
    }
    value = x;
}
```

A reversible variable & ***a reversible integer***

Change that value (off coarse)!

```
public void setValue(int x){
    if (value != x && whenChanged.peek() != pb.world){
        pb.trail.peek().add(this);
        whenChanged.push(pb.world);
        priorValue.push(value);
    }
    value = x;
}
```

A problem is a problem is a problem

A problem (from the point of view of reversibles & worlds)

```
public class Problem {

    String name;
    public ArrayList<IntVar> variables;
    ArrayList<Constraint> constraints;
    public LinkedList<Constraint> revisionQueue;
    public Stack<ArrayList<ReversibleVar>> trail;
    public VarOrdHeur voh;
    public boolean trace;
    public int world, solutions, nodes, fails;
    public long cpuTime;
    IntVar[] var;
    int[] val;
    public boolean firstProbe, propagationOn;
```

A problem (from the point of view of reversibles & worlds)

```
public class Problem {

    String name;
    public ArrayList<IntVar> variables;
    ArrayList<Constraint> constraints;
    public LinkedList<Constraint> revisionQueue;
    public Stack<ArrayList<ReversibleVar>> trail;
    public VarOrdHeur voh;
    public boolean trace;
    public int world, solutions, nodes, fails;
    public long cpuTime;
    IntVar[] var;
    int[] val;
    public boolean firstProbe, propagationOn;
```

A problem (from the point of view of reversibles & worlds)

```
public class Problem {

    String name;
    public ArrayList<IntVar> variables;
    ArrayList<Constraint> constraints;
    public LinkedList<Constraint> revisionQueue;
    public Stack<ArrayList<ReversibleVar>> trail;
    public VarOrdHeur voh;
    public boolean trace;
    public int world, solutions, nodes, fails;
    public long cpuTime;
    IntVar[] var;
    int[] val;
    public boolean firstProbe, propagationOn;
```

A problem (from the point of view of reversibles & worlds)

```
public void pushWorld(){
    trail.push(new ArrayList<ReversibleVar>());
    world++;
}

public void popWorld(){
    ArrayList<ReversibleVar> current = trail.pop();
    for (ReversibleVar v : current) v.restoreValue();
    world--;
}
```

A problem (from the point of view of reversibles & worlds)

When we are going to try something … that might not work

```
public void pushWorld(){
    trail.push(new ArrayList<ReversibleVar>());
    world++;
}
```

A problem (from the point of view of reversibles & worlds)

Undoing all changes! Typical use is a backtrack.

```
public void popWorld(){
    ArrayList<ReversibleVar> current = trail.pop();
    for (ReversibleVar v : current) v.restoreValue();
    world--;
}
```

A constrained integer variable, IntVar, is then …

A constrained integer variable, IntVar, is then …

```
public class IntVar extends Var implements Iterable<Integer> {

    public String name;
    public ArrayList<Constraint> constraints;
    public LinkedList<Integer> deletions;
    Problem pb;
    public Domain domain;

    public IntVar(String name,int lwb, int upb,Problem pb){
        this.name   = name;
        this.pb     = pb;
        domain      = new Domain(pb,lwb,upb);
        constraints = new ArrayList<Constraint>();
        deletions   = new LinkedList<Integer>();
        pb.variables.add(this);
    }
```

A constrained integer variable, IntVar, is then …

```
public class IntVar extends Var implements It

    public String name;
    public ArrayList<Constraint> c
    public LinkedList<Integer>
    Problem pb;
    public Domain dom

    public Int                    c lwb, int upb,Problem pb){
        th
                                ;
                    = new Domain(pb,lwb,upb);
                nts = new ArrayList<Constraint>();
            cions   = new LinkedList<Integer>();
        pb.variables.add(this);
    }
```

But more of that later

Implementation of Domain

d.remove(x)

```
public boolean remove(int x){
    if (x < lwb || x > upb || !contains(x)) return false;
    int v = x - lwb;
    int w = value[last.getValue()];
    swap(value,last.getValue(),location[v]);
    location[w] = location[v];
    location[v] = last.getValue();
    last.setValue(last.getValue() - 1);
    if (isEmpty()) throw new CPException("remove: "+ x +" The domain is now empty.");
    if (x == min.getValue()) min.setValue(getNewMin());
    if (x == max.getValue()) max.setValue(getNewMax());
    return true;
}
```

d.remove(x)

```
public boolean remove(int x){
    if (x < lwb || x > upb || !contains(x)) return false;
    int v = x - lwb;
    int w = value[last.getValue()];
    swap(value,last.getValue(),location[v]);
    location[w] = location[v];
    location[v] = last.getValue();
    last.setValue(last.getValue() - 1);
    if (isEmpty()) throw new CPException("remove: "+ x +" The domain is now empty.");
    if (x == min.getValue()) min.setValue(getNewMin());
    if (x == max.getValue()) max.setValue(getNewMax());
    return true;
}
```

d.remove(x)

```
public boolean remove(int x){
    if (x < lwb || x > upb || !contains(x)) return false;
    int v = x - lwb;
    int w = value[last.getValue()];
    swap(value,last.getValue(),location[v]);
    location[w] = location[v];
    location[v] = last.getValue();
    last.setValue(last.getValue() - 1);
    if (isEmpty()) throw new CPException("remove: " + x + " The domain is now empty.");
    if (x == min.getValue()) min.setValue(getNewMin());
    if (x == max.getValue()) max.setValue(getNewMax());
    return true;
}
```

```
int getNewMin(){
    if (isEmpty()) throw new CPException("Domain is empty.");
    for (int i=min.getValue()-lwb;i<n;i++)
        if (location[i] <= last.getValue()) return i + lwb;
    return Integer.MAX_VALUE;
}
```

d.remove(x)

Amotrized O(1)

```
public boolean remove(int x){
    if (x < lwb || x > upb || !contains(x)) return false;
    int v = x - lwb;
    int w = value[last.getValue()];
    swap(value,last.getValue(),location[v]);
    location[w] = location[v];
    location[v] = last.getValue();
    last.setValue(last.getValue() - 1);
    if (isEmpty()) throw new CPException("remove: " + x + " The domain is now empty.");
    if (x == min.getValue()) min.setValue(getNewMin());
    if (x == max.getValue()) max.setValue(getNewMax());
    return true;
}
```

```
int getNewMin(){
    if (isEmpty()) throw new CPException("Domain is empty.");
    for (int i=min.getValue()-lwb;i<n;i++)
        if (location[i] <= last.getValue()) return i + lwb;
    return Integer.MAX_VALUE;
}
```

d.removeAllBut(x)

```
public boolean removeAllBut(int x){
    if (x < lwb || x > upb || !contains(x))
        throw new CPException("removeAll: The value "+ x +" is not in the domain.");
    if (size() == 1) return false;
    int v = x - lwb;
    int w = value[0];
    swap(value,0,location[v]);
    location[w] = location[v];
    location[v] = 0;
    last.setValue(0);
    min.setValue(x);
    max.setValue(x);
    if (isEmpty()) throw new CPException("Domain is empty.");
    return true;
}
```

d.removeAllBut(x)

```
public boolean removeAllBut(int x){
    if (x < lwb || x > upb || !contains(x))
        throw new CPException("removeAll: The value "+ x +" is not in the domain.");
    if (size() == 1) return false;
    int v = x - lwb;
    int w = value[0];
    swap(value,0,location[v]);
    location[w] = location[v];
    location[v] = 0;
    last.setValue(0);
    min.setValue(x);
    max.setValue(x);
    if (isEmpty()) throw new CPException("Domain is empty.");
    return true;
}
```

d.removeAllBut(x)

O(1)

```
public boolean removeAllBut(int x){
    if (x < lwb || x > upb || !contains(x))
        throw new CPException("removeAll: The value "+ x +" is not in the domain.");
    if (size() == 1) return false;
    int v = x - lwb;
    int w = value[0];
    swap(value,0,location[v]);
    location[w] = location[v];
    location[v] = 0;
    last.setValue(0);
    min.setValue(x);
    max.setValue(x);
    if (isEmpty()) throw new CPException("Domain is empty.");
    return true;
}
```

d.removeBelow(x)

```
public int min(){return min.getValue();}

public boolean removeBelow(int x){
    boolean changed = false;
    for (int v=min();v<x;v++)
        changed = remove(v) || changed;
    if (isEmpty()) throw new CPException("removeBelow: "+ x +" The domain is empty.");
    return changed;
}
```

d.removeBelow(x)

```
public int min(){return min.getValue();}

public boolean removeBelow(int x){
    boolean changed = false;
    for (int v=min();v<x;v++)
        changed = remove(v) || changed;
    if (isEmpty()) throw new CPException("removeBelow: "+ x +" The domain is empty.");
    return changed;
}
```

d.removeBelow(x)

Amortized O(m)

```
public int min(){return min.getValue();}

public boolean removeBelow(int x){
    boolean changed = false;
    for (int v=min();v<x;v++)
        changed = remove(v) || changed;
    if (isEmpty()) throw new CPException("removeBelow: "+ x +" The domain is empty.");
    return changed;
}
```

d.removeAbove(x)

Amortized O(m)

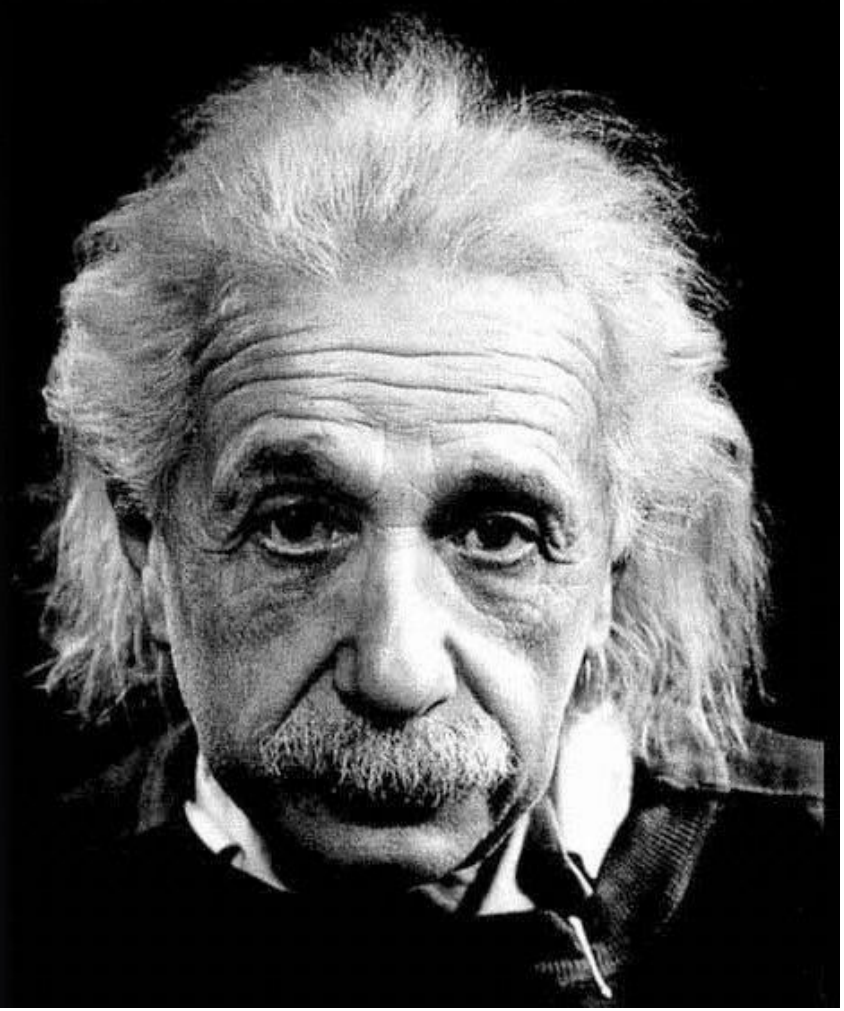
```
public boolean removeAbove(int x){
    boolean changed = false;
    for (int v=max();v>x;v--)
        changed = remove(v) || changed;
    if (isEmpty()) throw new CPException("removeAbove: "+ x +" The domain is empty.")
    return changed;
}
```

There are only 3 places weeSeepy throws exceptions

- In Domain: a domain becomes empty
- In Problem: search finds a solution
- In IntVar: attempt to get the value of an uninstantiated variable

“Everything should be made as simple as possible, but not simpler.”

Albert Einstein



Code , notes, papers

Index of /~pat/weeSeepy/ja | dblp: Pierre Schaus | SolverCheck: Declarative Testing | [Part2] Domains and Sparse

www.dcs.gla.ac.uk/~pat/weeSeepy/java/weeSeepy/domain/

Index of /~pat/weeSeepy/java/weeSeepy/domain

Icon	Name	Last modified	Size	Description
[PARENTDIR]	Parent Directory		-	
[ ]	Domain.java	2019-10-16 18:20	3.5K	
[ ]	DomainIterator.java	2019-09-24 17:02	610	

Apache/2.4.6 (CentOS) Server at www.dcs.gla.ac.uk Port 80

dcs.gla.ac.uk/~pat/weeSeep | dblp: Pierre Schaus | SolverCheck: Declarative Testing | [Part2] Domains and Sparse

www.dcs.gla.ac.uk/~pat/weeSeepy/java/weeSeepy/domain

```
package weeSeepy.domain;

import java.util.*;
import weeSeepy.problem.*;
import weeSeepy.reversibles.*;
import weeSeepy.exceptions.*;

public class Domain {

    Problem pb;
    int[] value;
    int[] location;
    public int lwb, upb, n;
    ReversibleInt last, min, max;


    public Domain(Problem pb,int lwb, int upb){
        this.pb   = pb;
        this.lwb  = lwb;
        this.upb  = upb;
        n         = upb - lwb + 1;
        value     = new int[n];
        location  = new int[n];
        last      = new ReversibleInt(pb,n-1);
        min       = new ReversibleInt(pb,lwb);
        max       = new ReversibleInt(pb,upb);
        for (int i=0;i<n;i++) value[i] = i;
        for (int i=0;i<n;i++) location[i] = i;

    }

    void swap(int[] array,int i,int j){
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }

    public boolean remove(int x){
        if (x < lwb || x > upb || !contains(x)) return false;
        int v = x - lwb;
        int w = value[last.getValue()];
        swap(value,last.getValue(),location[v]);
        location[w] = location[v];
        location[v] = last.getValue();
        last.setValue(last.getValue() - 1);
        if (isEmpty()) throw new CPException("remove: "+ x +" The domain is now empty.");
        if (x == min.getValue()) min.setValue(getNewMin());
        if (x == max.getValue()) max.setValue(getNewMax());
```

Index of /~pat/weeSeepy/java | dblp: Pierre Schaus | SolverCheck: Declarative Testing c | [Part2] Domains and SparseSe

www.dcs.gla.ac.uk/~pat/weeSeepy/java/weeSeepy/reversibles/

Index of /~pat/weeSeepy/java/weeSeepy/reversibles

Icon	Name	Last modified	Size	Description
[PARENTDIR]	Parent Directory		-	
[ ]	ReversibleBool.java	2019-10-16 18:19	876	
[ ]	ReversibleInt.java	2019-10-16 18:20	618	
[ ]	ReversibleVar.java	2019-10-16 18:19	314	

Apache/2.4.6 (CentOS) Server at www.dcs.gla.ac.uk Port 80

References

[1] P. Briggs and L. Torczon. An efficient representation for sparse sets. *LOPLAS*, 2(1-4):59–69, 1993.

[2] Laurent Michel, Pierre Schaus, Pascal Van Hentenryck. MiniCP: A lightweight solver for constraint programming, 2018. Available from `https://minicp.bitbucket.io`.

[3] V. le Clément, P. Schaus, C. Solnon, and C. Lecoutre. Sparse-sets for domain implementation. In *TRICS - Techniques foR Implementing Constraint programming Systems, CP 2013 Workshop.*, 2013.

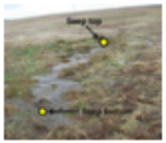
Also see notes 033 and 034

weeSeepy: Note | dblp: Pierre Sch | SolverCheck: Declar | [Part2] Domain

www.dcs.gla.ac.uk/~pat/weeSeepy

Computing Science
University of Glasgow

weeSeepy: a small CP toolkit



- Home Page
- News
- Notes
- Slides
- Code
- Links
- Exercises

Below is a trail of working notes on weeSeepy. These are written in a relatively informal manner, and are based on the APES (Algorithms, Problems and Empirical Studies) footnotes, Blue Book Notes from the dream group, and those by Dijkstra.

- 033-weeSeepy: an introduction
- 034-weeSeepy: Reversible Domains
- 035-weeSeepy: Reification
- 036-weeSeepy: A comparison with choco
- 037-weeSeepy: variables and constraints
- 038-weeSeepy: greater than, sum and the max constraint
- 039-weeSeepy: searching for a next solution