FORWARD-CHECKING

→ Var[x].val initialisés à -1 ;
→ On affecte les valeurs de leur domaine aux variables dans l'ordre où elle sont stockées dans dom[].*
→ On désactive une valeur d'un domaine en la remplaçant par -1 ;

Variables globales, constantes et structures :

```
Constante : tailleDomaine = 9 ;

Type Contraintes : structure
                int somme
                int varTab[]            //variable concernées par la contrainte
                int indexMaxVarTab
                Finstructure

Type  Variables : structures
                int val ;
//On stock dans domaine le nombre de contraintes actives annulant une valeur du domaine
                int domaine[9][2]
                int cteTab[]            //Contraintes portant sur la variable
                int indexMaxCteTab
        Finstructure
```

Variables var[] = tableau de référence des variables (i) |var|=V
Contraintes cte[] = tableau de référence des contraintes
int sauveMaj[][] //Utilisé pour annuler les mises à jour de domaine
int sauveMajIndex ;

Algorithmes :

```
TantQue numVar<=n Faire
        i=j=k=0
        valide=0
        //Si on est ds le cas d'un retour en arrière on recherche la prochaine valeur du  domaine à affecter à la variable et on annule les maj de domaine liées à l'ancienne valeur
        Si (var[numVar].val!= 0) faire
                int valVar=var[numVar].val
                TantQue i<= var[numVar].indexMaxCteTab Faire
                        Si var.cteTab[i] == difConstraint Faire
                                majDomDifContrainte(var.cteTab[i], numVar, valVar , « retour») ;
                        Sinon Faire
                                majDomSomContrainte(var.cteTab[i],numVar, valVar , « retour ») ;
                        Finsi
                        i++
                FintTantQue
                j = valVar
        FinSi
```

```
        //Boucle sur le domaine des variables
        TantQue (j < tailleDomaine&& valide==0) Faire
                var[numVar].val = var[numVar].dom[j]
                int valAffect=var[numVar].val
                j++ ;
                //On vérifie si la valeur est valide dans le domaine
                Si(Var[numVar].dom[j-1]==-1) Continue FinSi
                //Boucle sur les contraintes des variables
                int affValide;
                TantQue k<= var[numVar].indexMaxCteTab Faire
                        Si var.cteTab[k] = difConstraint Faire
                                affValide=majDomDifContrainte(var.cteTab[k],
                                numVar,valAffect,«affectation»)
                                        Si(!affValide) AnnulMaj() ;Break
                                         FinSi
                        Sinon Faire
                                affValide =majDomSomContrainte(var.cteTab[k],
                                numVar,valAffect,«affectation»)
                                Si(!affValide)  AnnulMaj() ;Break     FinSi
                                Finsi
                                Si(k==indexMaxCteTab) valide=1  FinSi
                        FinSi
                        k++
                FinTantQue
        FinTantQue
        //Si cul de sac on repasse la variable actuelle à l'état non affectée et on recule d'une
variable.
        Si (j==tailleDomaine-1&&valide==0) Faire
                Var[i].val=-1
                numVar - -
        //Sinon affectation variable suivante
        Sinon numVar++
        Finsi
FinTantQue
```

Fonctions auxiliaires :

Les fonctions majDom renvoient un entier : 1 si un domaine est vide et 0 dans le cas contraire.
Si 1 → l'affectation n'est pas valide, on recule d'un variable.
On les utilise pour mettre à jour les domaines dans deux cas :
Lors d'une affectation, on rajoute alors des valeurs interdites.
Lors d'un retour de l'affectation sur une variable préalablement affectée : on supprime alors des valeurs interdites.
Ces deux cas seront différenciés par le paramètre mode qui prendra dans les appels les valeurs « affectation » ou « retour » .

Mise à jour de domaine

```
Fonction majDomDifContrainte(int numCte, int numVarAffectee,int valDom,String mode) : int
valide
Pour i allant de 0 à cte[numCte].indexMaxVarTab  Faire
        int numVar =  cte.varTab[i] ;
        Si(numVar!=numVarAffectee) Faire
                Si (mode==« affectation »)Faire
                //On mémorise la maj dans la variable globale sauvMaj[][]
                        sauveMaj(numVar, valDom);
                        var[numVar].domaine[valDom-1][1]=var[numVar].domaine[valDom-1][1]+1
                        var[numVar].domaine[valDom-1][0]= -1
                / /mode retour
                Sinon Faire
                  //On décrémente le compteur indiquant le nb de contraintes interdisant cette valeur
                  variable[numVar].domaine[valDom-1][1]=var.domaine[valDom-1][1]-1
                  Si  (variable[numVar].domaine[valDom-1][1] == 0) Faire
                      var[numVar].domaine[valDom-1][0]= valDom
                 FinSi
                FinSi
        FinSi
        retourner !domaineVide(numCte)
Finpour
Fin
```

Le cas des contraintes de somme est plus compliqué à gérer.
Dans le cas ou l'on voudrait éliminer le plus de valeur possible des domaines à chaque affectation il faudrait faire des affectations test en dehors de la boucle principale afin d'identifier les valeurs maximales des domaine (en tenant compte des valeurs déjà éliminées par d'autre contraintes!) de ces variables aboutissant à un échec car trop grandes :

j=nombre de variable restantes à affecter (X1,X2,….Xj)
S=somme de la contrainte
a=somme des variables déjà affecté(es)
b=somme des variables non affectées dans la boucle principale

```
Pour k allant de 1 à j
        Xj=max{Xj.dom}
        b=Xj
        Pour h allant de 1 à j Faire
                Si (h==k) continue
```

```
                Xh=min{Xh.dom}
                b=b+Xh
                ->Maj des domaines des autres variables en supprimant min{Xh.dom}, donc une
                autre boucle sur les Xj sans Xk et Xh
        Finpour
        Si a>S-a Faire
                Xj.dom=Xj.dom\max{Xj.dom}
        Finsi
Finpour
```

Cet algorithme n'est pas fonctionnel, il est surtout la pour montrer la complexité de cette façon de faire, avec un fonction qui se met à affecter des variables en dehors de la boucle principale à qui se rôle échoit normalement.
De plus si l'on veut supprimer tous les maximum ne respectant pas l'arc-consistance de chaque domaine des variables restantes à affectées il faut le répéter plusieurs fois.
Cette façon de faire demande donc beaucoup d'opération et risque fort d'être coûteuse en temps de calcul.

Nous allons choisir une solution bien moins coûteuse :
On supprime des domaines des Xj uniquement les valeurs Z telles que :
S-a-Z<0 avec un traitement particulier quand la variables est la dernière non-affectée :
on supprime tout Z tel que S-a-Z!=0 ;

```
Fonction majDomSomContrainte(int numCte,int numVarAffectee, int valeurVar,String mode):int
valide
//calcul de la somme des variables déjà affectées
int sommeVar=0
int max=0
int nbVarNonAffecte=0
int varNonAffecte[]                                     //DYNAMIQUE ??
Pour i allant de 0 à contrainte[numCte].indexMaxVarTab Faire
        int numVar=contrainte[numCte].varTab[i] ;
        Si(var[numVar]!=0) Faire
                sommeVar = sommeVar+var[numVar].val
        Sinon faire
                nbVarNonAffectes = nbVarNonAffectes + 1
                varNonAffecte[nbVarNonAffectes-1]=numVar
        Finsi
Finpour
max = contrainte[numCte].somme – sommeVar
//Toutes var affectées, leurs sommes doit égalisée la contrainte
Si nbVarNonAffecte == 0 && mode== « affectation» Faire
        Retourner (max==0)
Sinon si  nbVarNonAffecte == 1
        Si max>tailleDomaine Faire
                retourner 0
//1 var à affecter, on laisse une seule valeur dans le domaine qui permettra d'égaliser la somme
        Sinon Faire
                Pour i allant de 0 à tailleDomaine-1
                        Si i !=max-1 Faire
                                Si(mode== »affectation ») Faire
```

```
                                        var[varNonAffecte[0]].domaine[i][0]=-1
                                        var[varNonAffecte[0]].domaine[i][1]
                                        =var[varNonAffecte[0]].domaine[i][1]+1
                                        //On mémorise la maj dans la variable globale sauvMaj[][]
                                        sauveMaj(varNonAffecte[0], i+1) ;
                                //mode retour
                                Sinon Faire
                                        var[varNonAffecte[0]].domaine[i][1]=
                                        var[varNonAffecte[0]].domaine[i][1]-1
                                        Si(var[varNonAffecte[0]].domaine[i][1]==0) Faire
                                                var[varNonAffecte[0]].domaine[i][0]==i+1
                                        FinSi
                                FinSi
                        Finsi
                Finpour
        Finsi

//Si nbVarNonAffecte >1
Sinon Faire
        Pour j allant de 0 à  nbVarNonAffecte-1
                Pour i allant de max-1 à tailleDomaine Faire
                        Si(mode== »affectation ») Faire
                                var[varNonAffecte[j]].domaine[i][0]=-1
                                var[varNonAffecte[j]].domaine[i][1]=
                                var[varNonAffecte[j]].domaine[i][1]+1
                                //On mémorise la maj dans la variable globale sauvMaj[][]
                                sauveMaj(varNonAffecte[j], i+1) ;
                        //mode retour
                        Sinon
                                var[varNonAffecte[j]].domaine[i][1]=
                                var[varNonAffecte[j]].domaine[i][1]-1
                                Si(var[varNonAffecte[j]].domaine[i][1]==0) Faire
                                        var[varNonAffecte[0]].domaine[i][0]==i+1
                                FinSi
                        FinSi
                Finpour
        Finpour
Finsi
//On verifie s'il existe des domaines vides
retourner !domaineVide(numCte)
Fin
```

```
Fonction domaineVide(int numCte) : int vide
Pour i allant de 0 à indexMaxVarTab Faire
        int numVar = cte[varTab[i]] ;
        TantQue j <= tailleDomaine-1
                if(var[numVar].domaine[j][0]!=-1) break ;
                j++ ;
        Finpour
        Si j==tailleDomaine retourne 1 ;
Finpour
return 0 ;
Fin
```

Sauvegarde et annulation de mises à jours de domaines :

Lors de l'annulation d'une affectation pour cause de domaine vide sur une variable il faut pouvoir revenir en arrière sur les mises à jours des domaines.
On peut faire cela d'une manière similaire à ce qui est fait lors de la mise à jours des domaines lors d'une affectation. On peut même transformer les fonctions **majDomSomContrainte** et **majDomDifContrainte** afin qu'elles puissent effectuer des mise à jours ou les défaire selon le contexte depuis lequel elles sont appelées mais:
→ entraîne un surcoût important en calcul
→ complexifie ces deux fonctions avec le rajout d'un paramètre (String mode « maj » ou « anulMaj » par exemple) et des conditions sur ce paramètre pour les actions à effectuer. Ce problème pourrait être régler en créant de nouvelle fonction auxiliaires.
→ il faudrait indiquer à l'algorithme à quel moment de la boucle sur les constantes un domaine a été trouvé vide afin d'annuler uniquement les mises à jour de domaine ayant été effectuées.

Un solution qui paraît moins coûteuse en calcul mais un peu plus coûteuse en mémoire serait de stocker en mémoire les mises à jour effectuées lors d'une affectation afin de pouvoir les défaire facilement lorsque l'on trouve un domaine vide et qu'on affecte la valeur suivante du domaine de la variable (ou que l'on revient en arrière sur la variable précédente).
Ce stockage est effacé dès qu'une affectation est validée, pour être rempli avec les informations des mise à jours des domaines en relation avec l'affectation de la nouvelle variables active.
Un stockage sous forme de tableau dynamique semble approprié :
int sauveMajDom[numVar][valeurInterdite]

Après réflexion il semble opportun d'utiliser les deux solutions :
->Dans le cas d'une affectation invalidée au cours des vérifications de contrainte, on annule les mises à jour de domaine grâce à sauvMaj[][].
→Dans le cas d'un retour sur une variable déjà affectée, il serait trop coûteux de sauvegarder toutes les mises à jour de domaines entraînées par toutes les variables affectées. On utilise donc les fonctions **majDomSomContrainte** et **majDomDiffContrainte** en mode « retour » afin qu'elles annulent les mise à jour de domaine liée à l'ancienne valeur de la variable que l'on souhaite réaffecter.

```
Fonction  sauveMaj(int numVar, int ValInterdite) : void
        sauveMaj[sauveMajIndex][0]=numVar ;
        sauveMaj[sauveMajIndex][1]=valInterdite ;
        sauveMajIndex++ ;
```

```
Fonction AnnulMaj() : void
        Pour i allant de 0 à sauveMajIndex Faire
                numVar=sauveMaj[i][0] ;
                indexDom=sauvMaj[i][1] ;
                var[numVar].domaine[indexDom][1]=var[numVar].domaine[indexDom][1]-1
                 Si  (variable[numVar].domaine[indexDom][1] == 0) Faire
                       var[numVar].domaine[indexDom][0]=  indexDom+1
                  FinSi
        Finpour
Fin
```