Projet Concept de Programmation

Université Paul Sabatier Toulouse 3

plan du projet de l'année 2004-2005

Plan de la présentation

- Interface avec Caml et Java
- Calcul de l'abscisse
- Calcul de l'ordonnée
- Traçage effectif du graphe

Les constructeurs

- Il faut définir un constructeur par type de sommet
- C'est dans la partie Caml ou Java qu'il faut gérer la traduction du programme en un graphe répondant aux contraintes suivantes...
- Il manque le constructeur pour le **goto_simple** !

Assignation, return, ...

Liste complète :

assignation, fin_programme (return) , instr_skip

- Ces instructions n'ont pas de continuation
- Elles doivent etre composées avec des séquences, des if, etc.
- Leur label est la valeur de l'instruction :
  - Assignation (i=3), label : i=3
  - fin_programme (return 0), label : " 0"
  - Skip, pas de label (NULL)

La séquence et le goto

La séquence

- Le constructeur de séquence permet de composer deux instructions ou séquences d'instruction
- Après la dernière instruction du premier argument, on passe à la première instruction du second argument

Le goto_simple

- Cette instruction n'apparait pas dans un programme de départ
- Elle a un fils (le sommet pointé) et pas de label
- Il faut l'ajouter à la fin d'un corps de boucle (elle pointe alors vers le sommet while correspondant) au à la fin d'une partie then d'une condition (elle pointe alors vers la continuation)

La condition

- Ce constructeur permet de coder un if ... then ... else ...fi
- Pour le constructeur if, il faut ajouter une séquence entre la fin de la partie else et la continuation du if (ce qui suit le fi)
- Avant de faire un appel aux constructeurs C, il faut aussi ajouter un goto_simple à la fin de la partie else et qui pointe vers la continuation
- Pour faire cet ajout (en Caml et en Java), il faut faire un pré-traitement qui va ajouter aussi une opération de séquence.

La boucle

- Il s'agit du constructeur while qui a un fils, le corps de boucle
- L'insertion du goto_simple doit etre fait dans la partie Caml ou Java
- La valeur du successeur de ce goto_simple doit etre mise à jour lors de la construction du sommet while
- Il est conseillé d'utiliser un parcours récursif simple du corps de boucle

Plan de la présentation

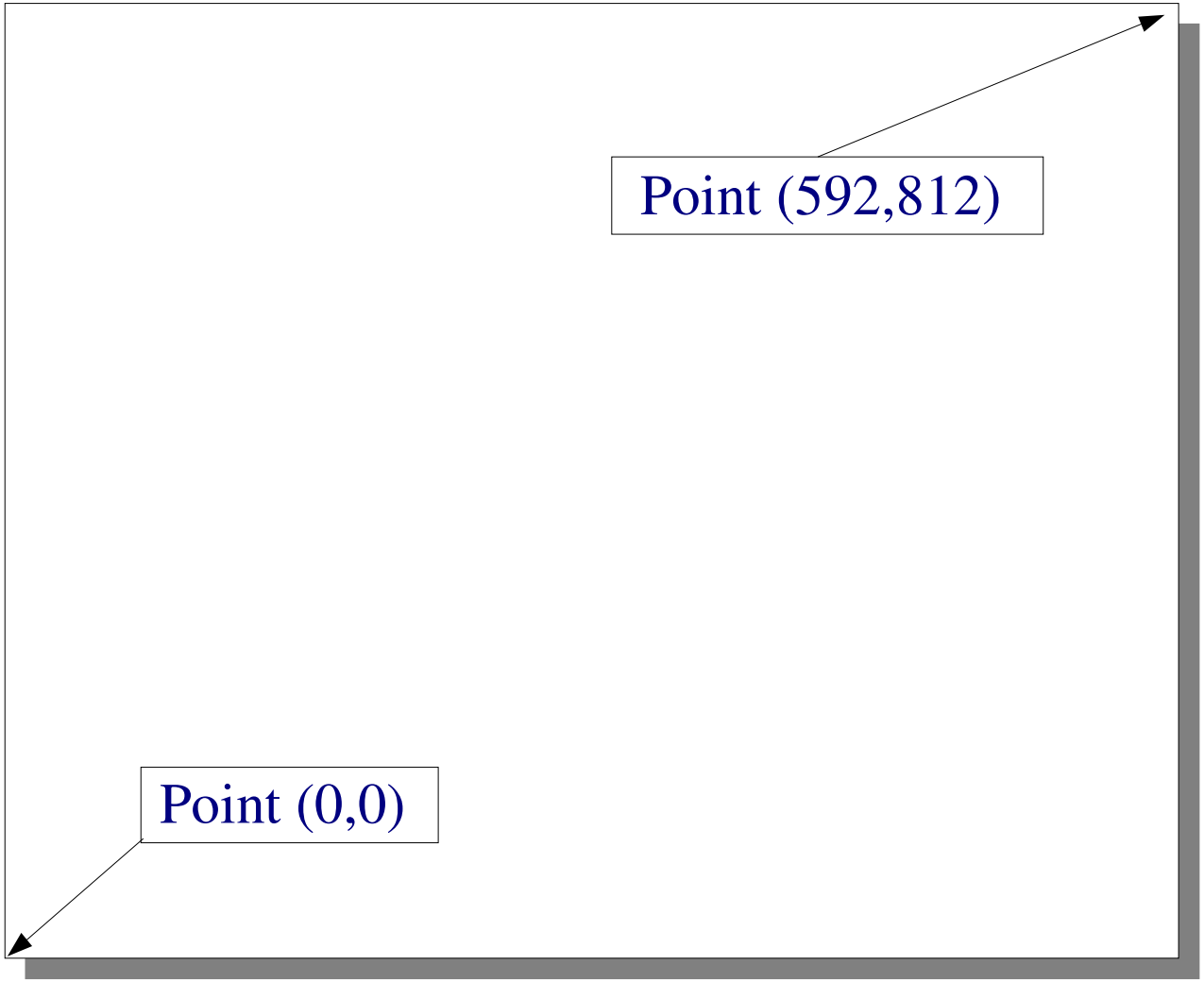
- Interface avec Caml et Java
- Calcul de l'abscisse
- Calcul de l'ordonnée
- Traçage effectif du graphe

Interface du module

Ce module doit définir deux fonctions :

- *Place_abscisse*
  - Effectue un parcours du graphe pour calculer l'abscisse de chaque sommet
  - Met à jour un tableau associant à l'Id d'un sommet l'abscisse de ce sommet
- *get_abscisse*
  - Lit le tableau calculé par *place_abscisse*
  - Rend l'abscisse du sommet dont l'Id est passée en paramètre

Placement de l'abscisse



Cible du traçage du graphe

- Hauteur : 812 pts
- Largeur : 592 pts

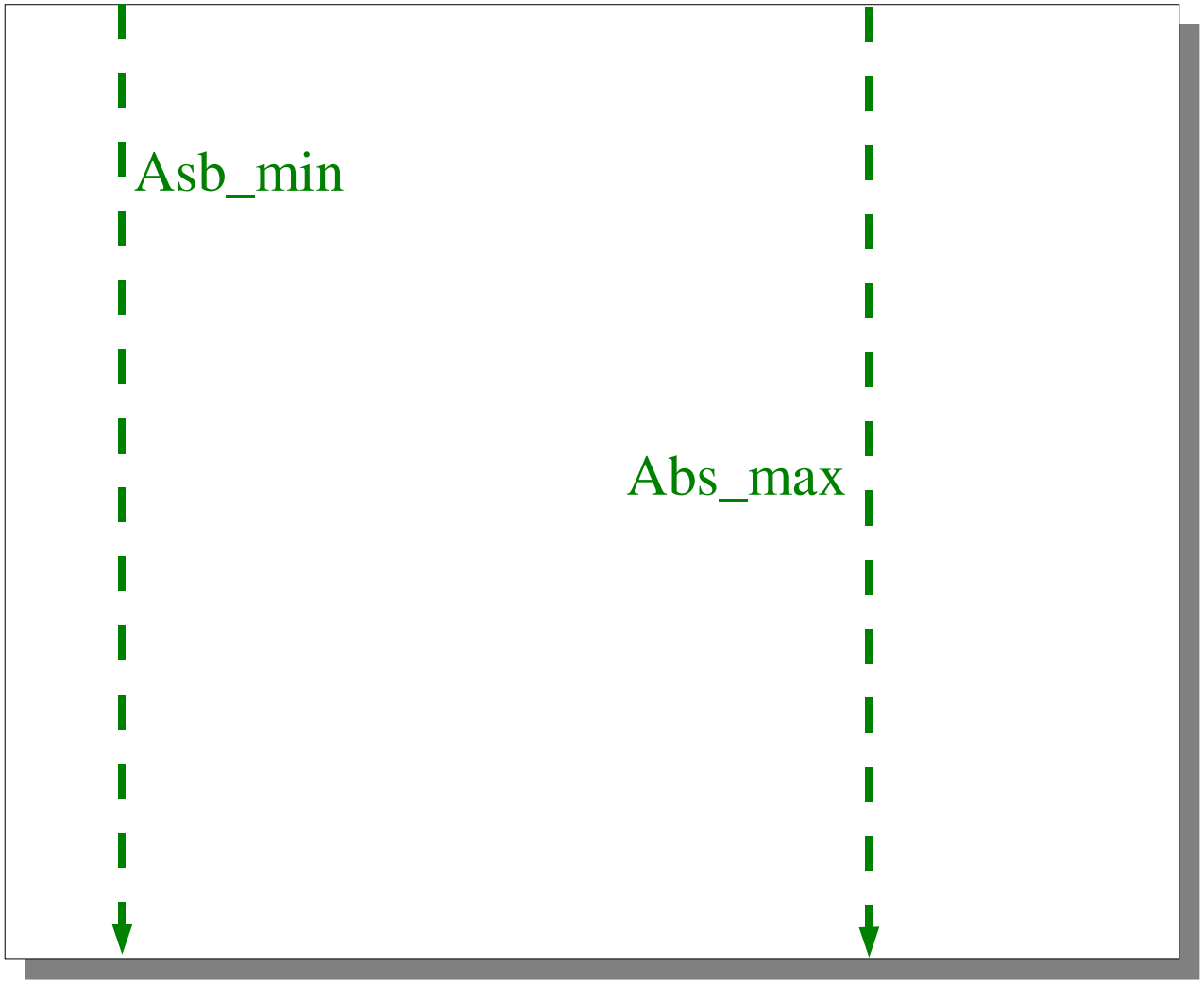
Placement de l'abscisse

Cible du traçage du graphe

Algorithme :

- Parcours récursif du graphe

Placement de l'abscisse



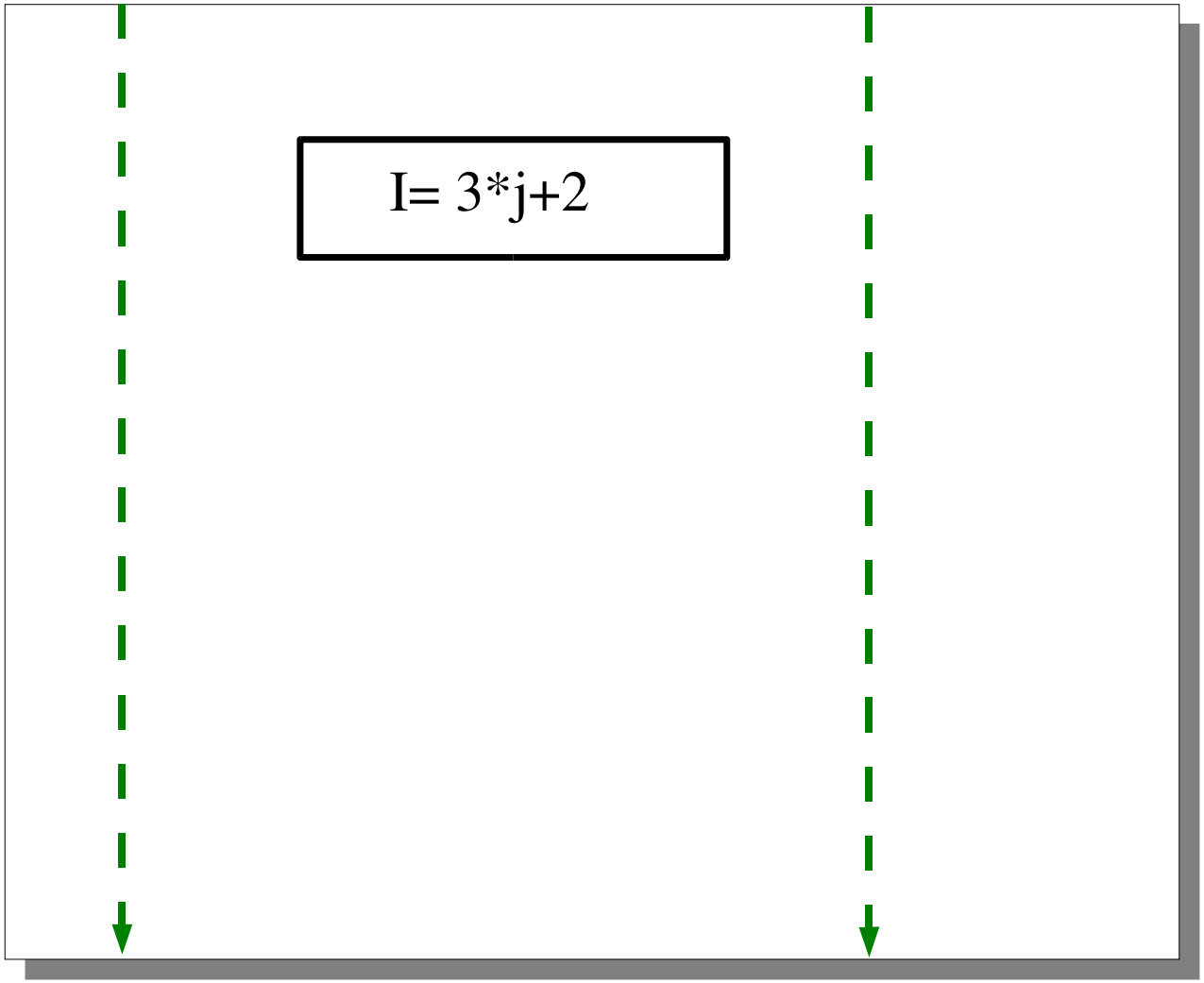
Cible du traçage du graphe

Algorithme :

- Parcours récursif du graphe
- Maintenance de :
  - abs_min
  - abs_max

Ce sont des paramètres d'une fonction appelée récursivement !

Placement de l'abscisse

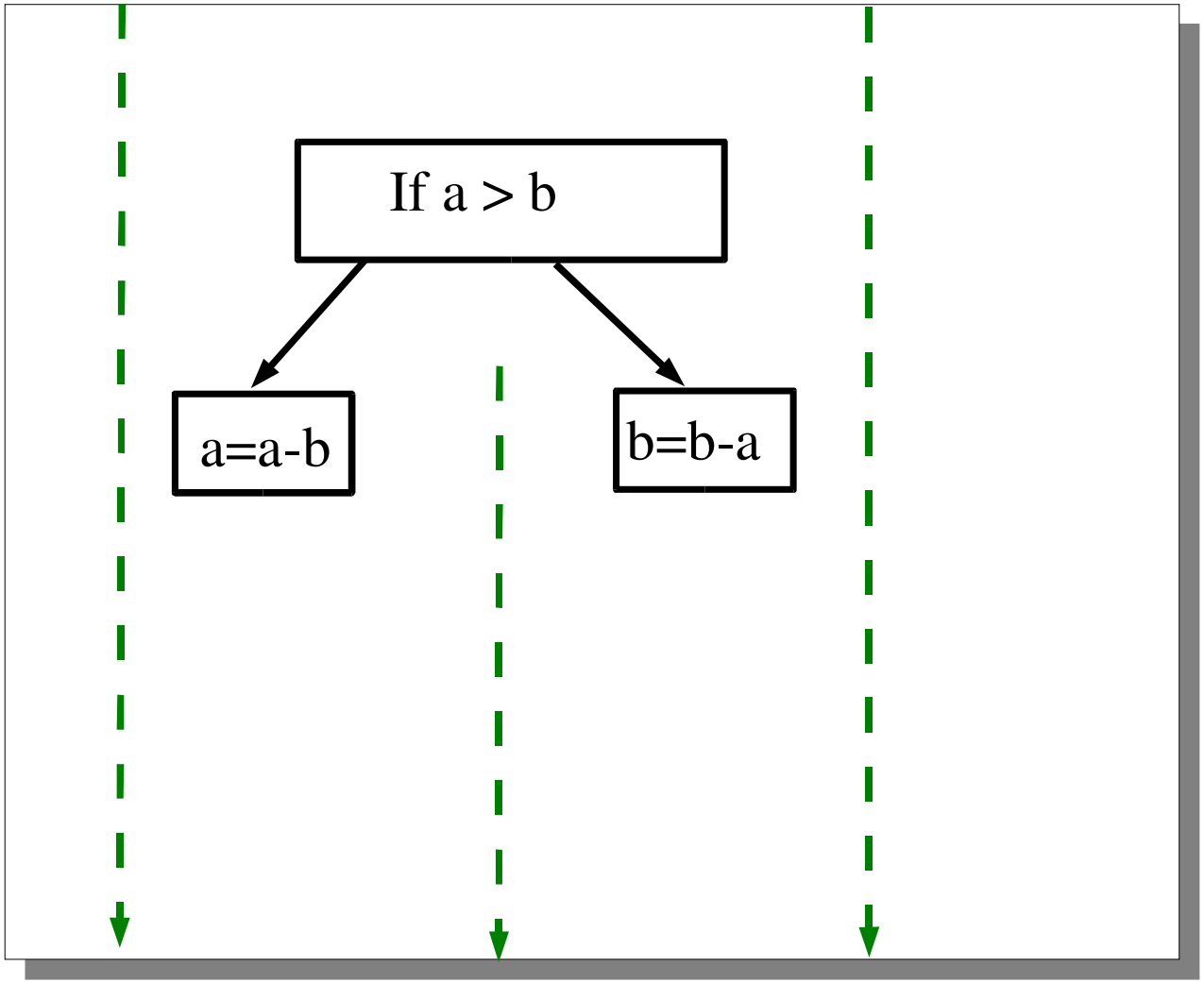


Cible du traçage du graphe

Algorithme :

- Parcours récursif du graphe
- Maintenance de :
  - abs_min
  - abs_max
- Cas terminal : on place le sommet au milieu de l'intervalle
- Cas sequentiel : on appelle la fonction sur les deux arguments avec les memes paramètres

Placement de l'abscisse

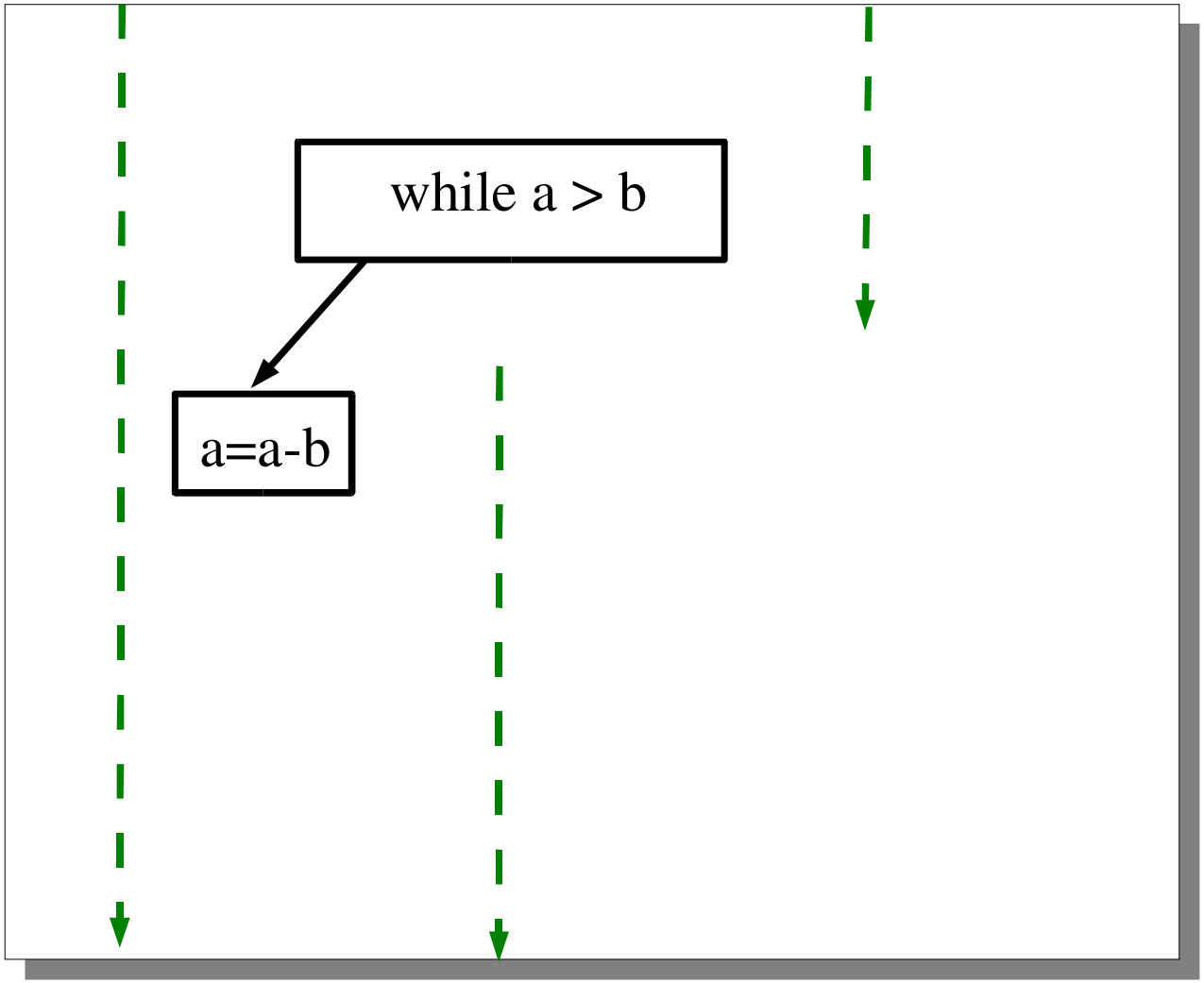


Cible du traçage du graphe

Algorithme :

- Parcours récursif du graphe
- Maintenance de :
  - abs_min
  - abs_max
- Cas If :
  - On divise l'intervalle en deux
  - Le traitement du goto_simple permet de revenir à l'intervalle initial

Placement de l'abscisse

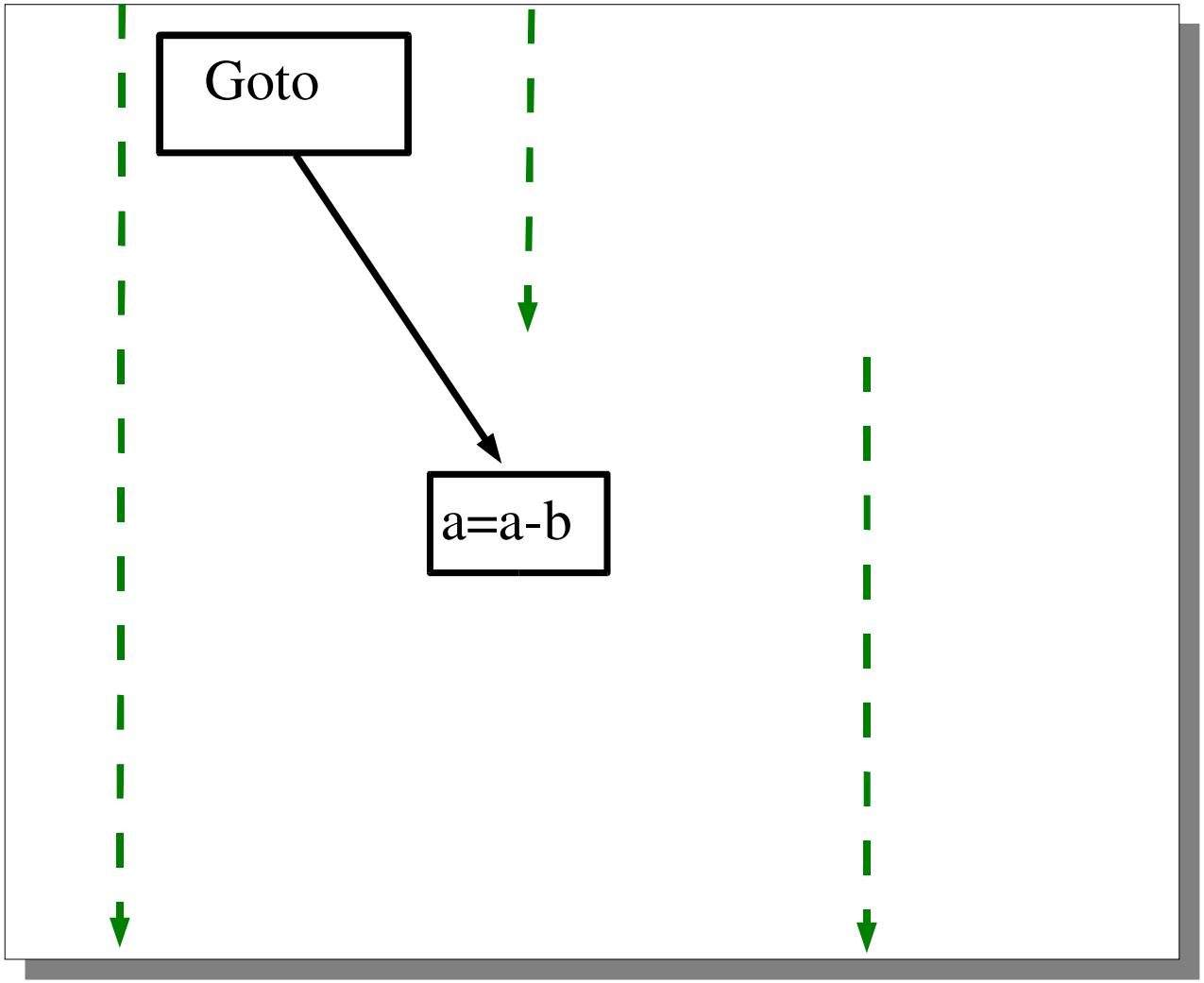


Cible du traçage du graphe

Algorithme :

- Parcours récursif du graphe
- Maintenance de :
  - abs_min
  - abs_max
- Cas while :
  - On divise l'intervalle en deux pour le corps
  - On part dans la partie gauche

Placement de l'abscisse



Cible du traçage du graphe

Algorithme :

- Parcours récursif du graphe
- Maintenance de :
  - abs_min
  - abs_max
- Cas goto_simple :
  - On élargit l'intervalle à droite pour le fils
- Correction : dès qu'on tombe sur un sommet ayant déjà une abscisse, on s'arrete.

Plan de la présentation

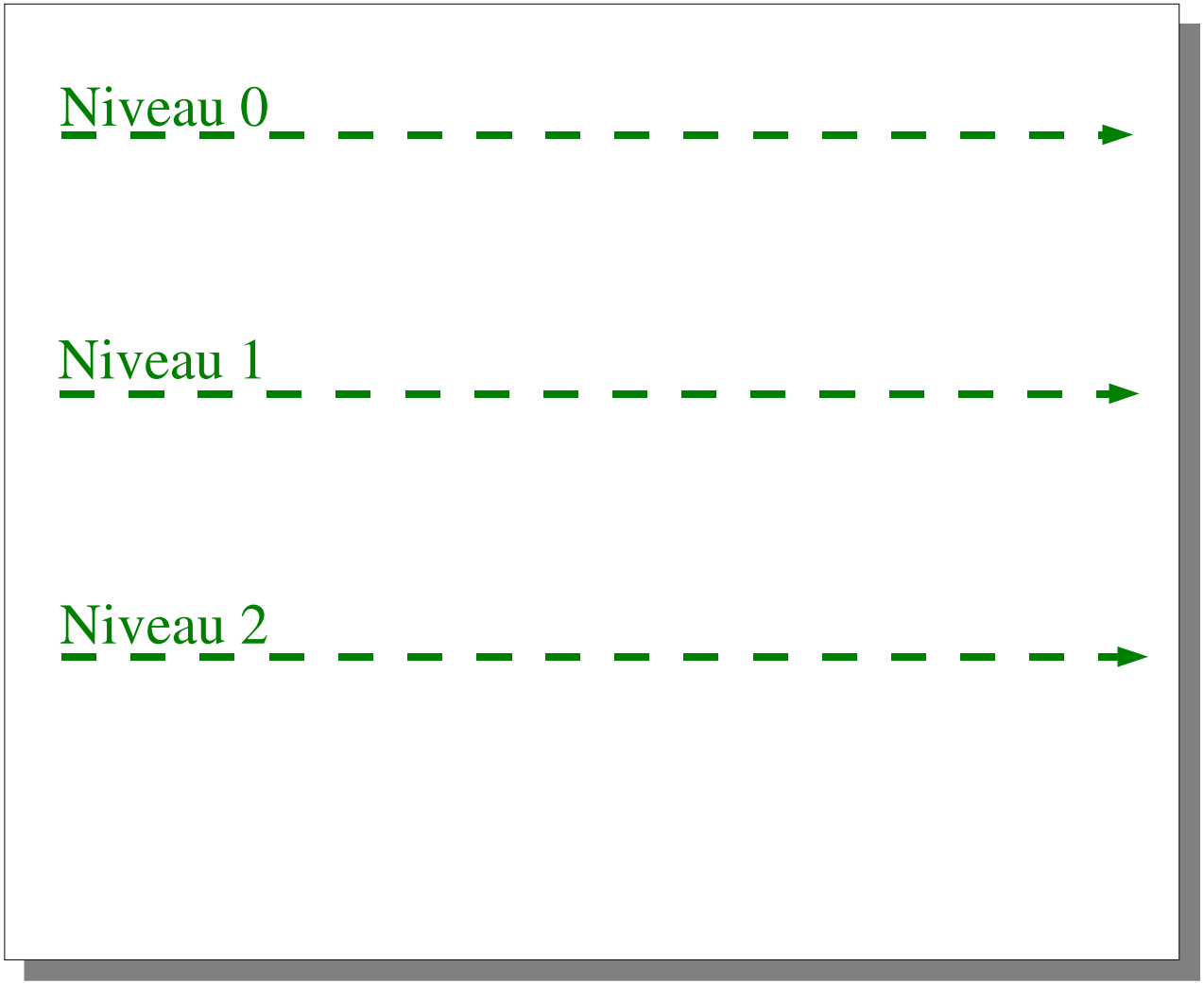
- Interface avec Caml et Java
- Calcul de l'abscisse
- Calcul de l'ordonnée
- Traçage effectif du graphe

Interface du module

Ce module doit définir deux fonctions :

- *Place_ordonnée*
  - Effectue un parcours du graphe pour calculer l'ordonnée de chaque sommet
  - Met à jour un tableau associant à l'Id d'un sommet l'ordonnée de ce sommet
- *get_ordonnée*
  - Lit le tableau calculé par *place_ordonnée*
  - Rend l'ordonnée du sommet dont l'Id est passée en paramètre

Placement de l'ordonnée



Algorithme :

- Parcours récursif du graphe pour calculer le niveau d'un sommet
- Le niveau vaut 0 tout en haut
- Plus le niveau est élevé, plus le sommet est bas
- La fonction rend la hauteur maximale du sous-graphe exploré

Placement de l'ordonnée

Algorithme :

- Appel à un successeur
  - Avec niveau+1
- goto_simple :
  - Jamais d'appel récursif
  - Le niveau du successeur est mis à jour s'il n'est pas déjà défini
- Mise à jour :
  - Choisir le maximum entre niveau courant et le paramètre niveau de la fonction

Plan de la présentation

- Interface avec Caml et Java
- Calcul de l'abscisse
- Calcul de l'ordonnée
- Traçage effectif du graphe

Fichiers fournis

Un certain nombre de fichiers sont fournis pour le traçage du graphe :

- Les fichiers noeud.c et noeud.h du module noeud
- Les fichiers postscript fin.ps et preambule.ps, qui doivent etre trouvés par le module noeud
- Un exécutable permettant d'afficher les fichiers postscript (gv ou ghostview)

Compilation du fichier noeud.c

- La constante symbolique REP_PS doit etre définie et etre le chemin (en chaine de caractères) du répertoire où se trouvent les fichiers postscript

- La constante symbolique CHEMIN_GV doit etre définie et valoir le chemin complet d'un interpreteur Postscript