Spécial K

Travail d'étude et de recherche année 2004

Spécial K

Plan

- Introduction et motivations
- Contraintes
- Présentation détaillée du langage
- Organisation du projet (planning)
- Méthodes et techniques utilisées
- Contraintes non fonctionnelles

Spécial K

Introduction et motivations

Avant propos...

Motivations

Objectifs

Gestion du risque et moyens de contrôle

Avant Propos

- SpécialK est un langage basé sur les règles et techniques utilisées par Emmanuel Kounalis dans ses cours d'Algorithmique.
- Inspiré d'un langage abstrait, SpécialK est multiparadigmes (langage logico-fonctionnel)
- Utilisation des "flots" de données (les données circulent en fonction des gardes et des unifications)

Spécial R

Motivations (1)

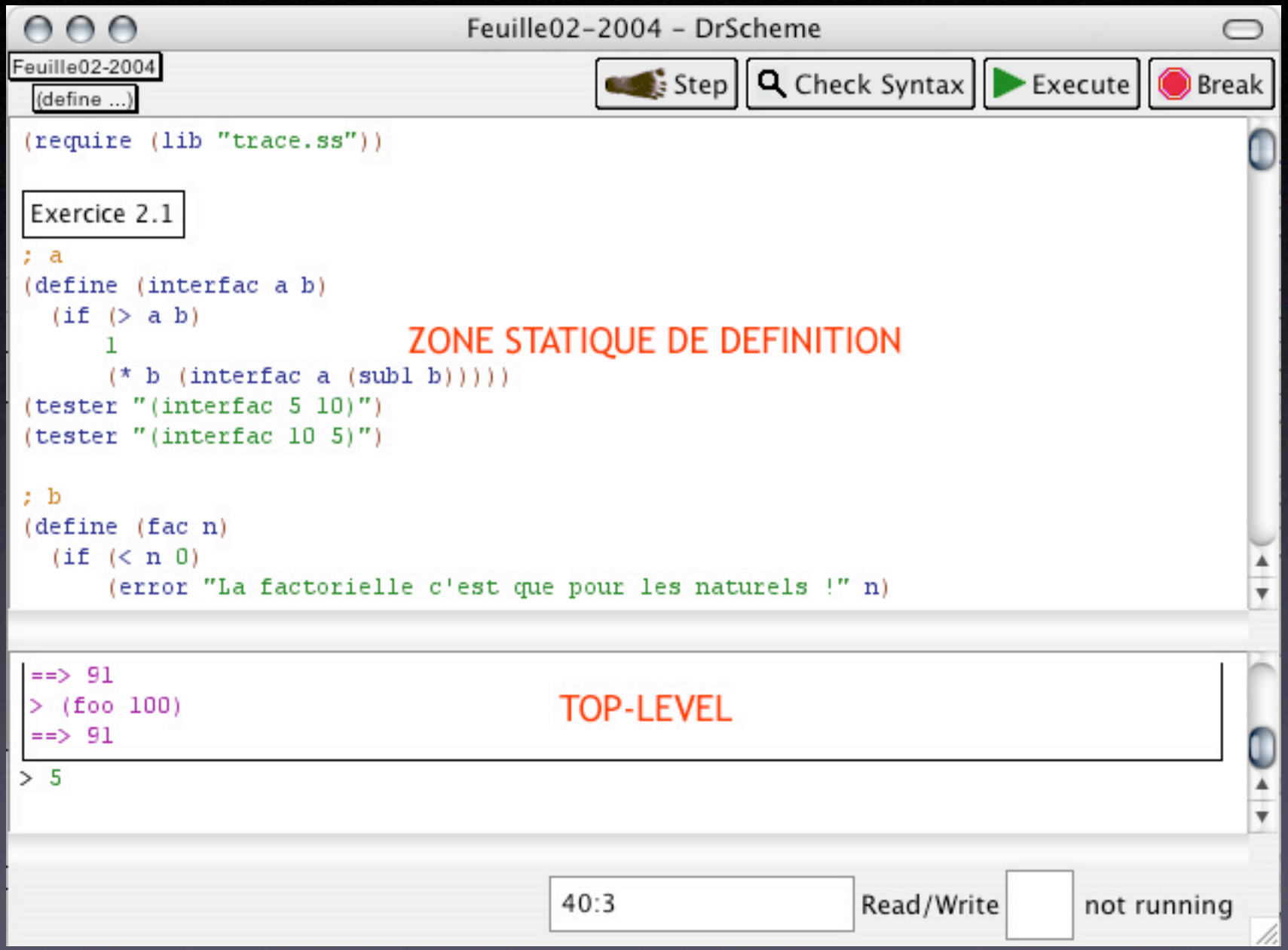
- Permettre le développement d'algorithmes simples vues en cours sans avoir à les convertir dans un autre langage.
- Actuellement on utilise DrScheme et le langage Scheme pour les TPs d'algorithmiques des premier cycles (MI2).
- Il s'agit de développer un "language pack" à *but pédagogique* pour ce logiciel.
- On conserve ainsi les habitudes d'utilisation de chacun

Spécial K

Motivations (2)

- En se basant sur Scheme comme langage hôte on bénéficiera d'un grand nombre de facilitées lors du développement.
- Base d'utilisateurs grandissante dans l'enseignement.
- *Interface graphique éprouvée* de DrScheme.

Spécial


Feuille02-2004 - DrScheme
Feuille02-2004
(define ...)
Step
Check Syntax
Execute
Break
(require (lib "trace.ss"))
Exercice 2.1
; a
(define (interfac a b)
(if (> a b)
1
(* b (interfac a (sub1 b)))))
(tester "(interfac 5 10)")
(tester "(interfac 10 5)")
; b
(define (fac n)
(if (< n 0)
(error "La factorielle c'est que pour les naturels !" n)
ZONE STATIQUE DE DEFINITION
==> 91
> (foo 100)
==> 91
TOP-LEVEL
> 5
40:3
Read/Write
not running

Spécial K

Objectifs (1)

- L'objectif principal est une intégration correcte à DrScheme, le laboratoire à langages de l'équipe PLT. Il faut mettre au point le coeur du projet (grammaire, compilation, gestion des erreurs)
- Obtenir une réalisation *la plus stable et la plus intuitive possible* car elle sera utilisée par des étudiants.
- Fournir l'ensemble des fonctionnalités nécessaires sans pour autant nuire à la compréhension et à la simplicité.

Spécial R

Objectifs (2)

- Documentation (en ligne) la plus complète possible. Sur l'application et sur le langage lui-même
- *Moindre importance*:
  - Disponibilité d'un ensemble d'algorithmes de base afin de permettre (tris sur les structures de données)
  - Facilitées graphiques de représentations des donnés

Spécial K

Gestion du risque

- Risque principal: la grammaire ne fonctionne pas correctement.
- Gestion des erreurs
- Phase de test avec les étudiants
- Nous sommes entrés en contact avec Emmanuel Kounalis afin de respecter au mieux la sémantique et la syntaxe du langage

Spécial K

Contraintes

Spécial K

- La structure de langage est imposée
- La lisibilité du code produit n’est pas une priorité: on cherche un compromis entre optimisation et lisibilité.
- On devra fournir un “stepper” et un “chronomètre d'exécution”

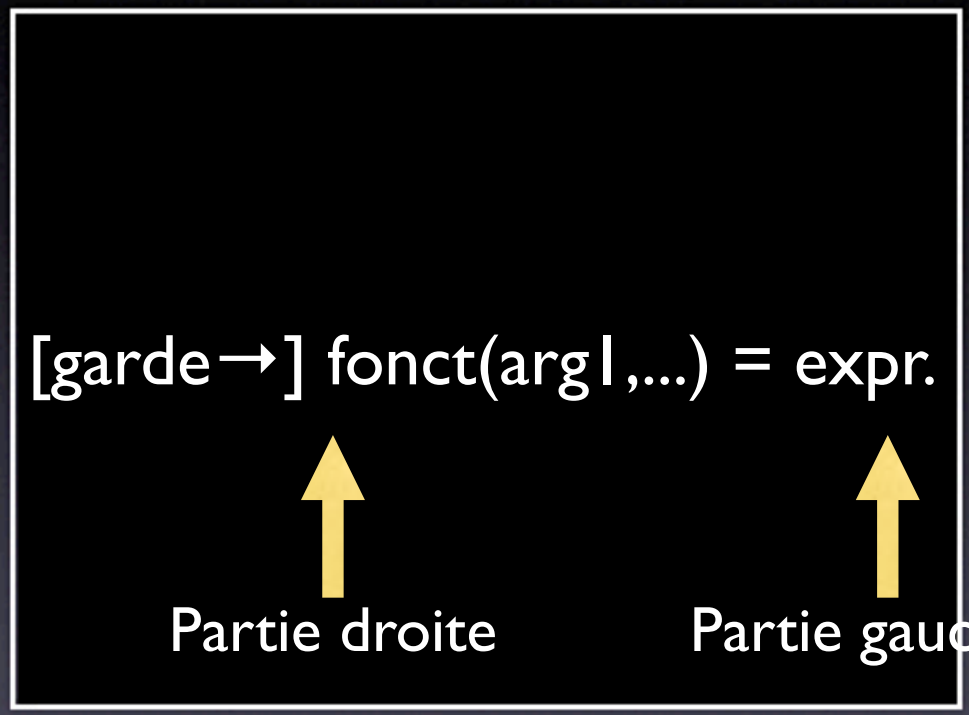
Spécial R

Présentation détaillée du langage

Spécial K

- Langage fonctionnel et logique
- Typage dynamique
- Manipulation des listes et des tableaux
- Le plus proche possible de la syntaxe d'Emmanuel Kounalis

Spécial K



Un programme SpécialK est une suite de clauses de cette forme.

Spécial K

Exemple de programme

- A chaque appel de fonction on choisi une clause.
- 2 phases pour choisir:
  - Garde
  - Unification

```
fac(0) = 1.          #1

fac(1) = 1.          #2

n > 1 → fac(n) =     #3
          n * fac(n-1).
```

Garde

- La garde est une expression booléenne constituant la condition pour choisir la clause
- Exemple:

```
n>1 → fac(n) = n*fac(n-1)      #3
```

Spécial K

Unification

- Mécanisme proche de celui de Prolog.
- L'unification à 2 règles:
  - Condition supplémentaire:
    Si l'argument est une valeur, l'argument dans l'appel de la fonction doit avoir la même valeur.
    Exemple:
    On souhaite exécuter: fac(x)
    Si x=0 ou x=1 on choisirais les clauses 1 et 2.
    Autrement dit:
    fac(0) = 1. #1
    fac(1) = 1. #2
    est équivalent à:
    n == 0→fac(n) = 1
    n == 1→fac(n) = 1

Unification

- L'unification à 2 régles:
  - Association variables-valeurs (environnement)
    On crée un environnement pour la partie droite
    Exemple:
    On souhaite exécuter: fac(x). avec x>1
    On utilise alors la clause #3
    La partie droite est évaluée dans l'environnement {*n* = *x*}

Unification sur les listes

- Dans une clause de la forme:
  f(c1:c2...:ci:L) = expr
- Que l'on unifie avec f(a1:a2:...:ai...aj). avec i<j
  la partie droite sera exécutée dans l'environnement {c1=a1,...,ci=ai,L=ai+1:...aj}.
- Exemple
  f(c:L) unifié avec f(1:2:3:nil).
  donne {c=1,L=2:3:nil}.

Unification sur les listes

- L'unification ne peut être utilisée qu'avec l'opérateur de liste (:). On ne peut donc pas, par exemple, écrire:

  f(a+1) = ...

Exemples

Tri d'une liste d'entiers

```
sort(nil)     =nil.
sort(c:L)     =insert(c,sort(L)).

insert(e,nil)=e:nil.
e>c →insert(e,c:L)=c:insert(L).
e<=c→insert(e:c:L)=e:c:L.
```

Exemples

Maximum d'un tableau

```
i > length(T)                    → max(T,i,m) = m.
i <= length(T) && T[i] > m       → max(T,i,m) =
                                       max(T,i+1,T[i]).
i <= length(T) && T[i] <= m      → max(T,i,m)  =
                                       max(T,i+1,m).
```

- L’ordre des clauses n’a pas d’importance
- Si à un moment de l'exécution on a le choix entre plusieurs clauses, on provoquera une erreur.

Vérification sémantique

- Le langage SpécialK étant à typage dynamique il est difficile de faire des vérifications de type. Cependant on essaiera de faire des vérification dés que possible.
- Exemple: `(a + b) & c`

Spécial K

Organisation du projet (Planning)

Spécial K

Dépendances des parties

- Le projet est articulé autour de 2 parties indépendantes.
  - Le compilateur (le 'moteur' de SpécialK)
  - Et le reste : la documentation, l'interface utilitaires et les bibliothèques d'exemples d'algorithmes

Dépendances: Le compilateur

- Le compilateur comporte 3 éléments interdépendants:
  - La grammaire
  - La traduction
  - La gestion des erreurs

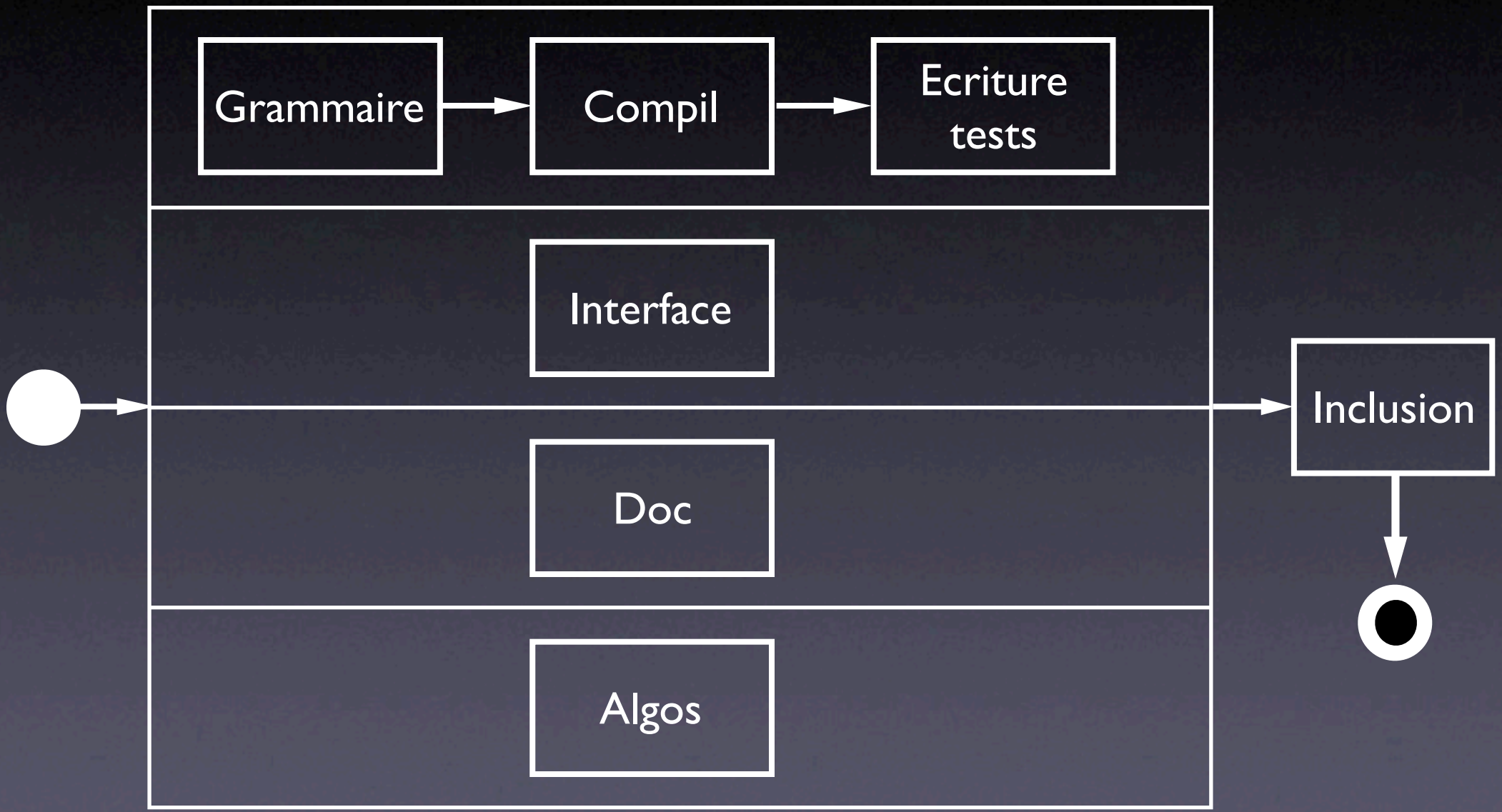
La gestion des erreurs est indissociable de la grammaire et la traduction

Dépendances: le reste

- La documentation: même si elle dépend quelque peu du reste on peut la faire indépendamment
- L'interface graphique: on fixera une interface simple avec le compilateur
- Les exemples d'algorithmes: leur implémentation dépend uniquement de la définition du langage

Spécial K

Planning (1)



Spécial K

Planning (2)

Etape	Temps	Réalisation
Grammaire	Effectué	J. Charles et C. Rodas
Traduction	1 semaine	J. Charles et P. Châtel
Exécution pas à pas	1 semaine	J. Charles, P. Châtel, C.Rodas
Gestion des Erreurs	Base +	J. Charles et P. Châtel
Interface graphique	1 semaine	J. Charles et P. Châtel
Algorithmes	Base +	S. Beucler
Documentation	Base +	S. Beucler

Spécial K

Méthodes et techniques

Spécial K

Méthodes et outils utilisés

- DrScheme
- Parser tools
- Bibliothèques de portabilité
- Archive .plt
- CVS

Spécial K

Documentation

- Intégration avec DrScheme
- LaTeX + tex2page
- Documentation en ligne et sur papier
- Wiki pour discussion internes
- Fichier de type HACKING

Spécial K

Contraintes non-fonctionnelles

Spécial K

- Pas de contraintes lourdes au niveau de la configuration matérielle
- Possibilité d'environnement léger
- Temps de traitement
- LGPL