Bureau d'études
Automates et Théorie des Langages
Documents autorisés – 1h30

1 Prélude

- Télécharger depuis moodle l'archive `be2021.tgz`
- Désarchiver son contenu avec la commande : `tar xzvf be2021.tgz`
- Vous obtenez un répertoire nommé `be2021`
- Renommer ce répertoire sous la forme `be2021_source_Nom1_Nom2` (en remplaçant `Nom1` et `Nom2` par vos noms dans l'ordre alphabétique. Par exemple, si vous êtes le binôme Jacques Requin et Pénélope Pieuvre, vous utiliserez la commande : `mv be2021 be2021_Pieuvre_Requin`

2 Postlude

Lorsque la séance se termine au bout d'1h30 (2h00 pour les étudiants bénéficiant d'un tiers-temps), vous devrez :

- Vérifier que les résultats de vos travaux sont bien compilables (gardez toujours une archive compilable pour ne pas rendre une version qui ne compile pas)
- Créer une archive avec la commande : `tar czvf be2021_Nom1_Nom2.tgz be2021_Nom1_Nom2`
- Déposer cette archive sur moodle

3 Un langage de description de Systèmes

L'objectif du bureau d'étude est de construire deux analyseurs pour une version simplifiée d'un langage de description de systèmes à flots de données (modèles utilisés en automatique, Simulink, SciCos/XCos ou SCADE par exemple). Ceux-ci seront composés d'un analyseur lexical construit avec l'outil `ocamlex` (sujet de TP 1) et d'un analyseur syntaxique construit d'une part, en exploitant l'outil `menhir` pour générer l'analyseur syntaxique (sujet de TP 2), et d'autre part la technique d'analyse descendante récursive programmée en `ocaml` en utilisant la structure de monade (sujet de TP 3).

Voici un exemple de système :

```
model S {
flow f from Ba.ra to HS.pa, HS.pb;
block Ba(pa : in float, pb : in float, ra : out float [ 2 ] );
system HS(pa : in float [ 2 ], ra : out float, pb : in float [ 2 ] , rb : out float [ 2 ] ) {
       block Bb(ra : out float, pa : in float);
       flow fa from pa to Bb.pa;
       flow fb from pb to rb;
}
flow fb from HS.ra to Ba.pa, Ba.pb;
flow fc from HS.rb to;
}
```

Cette syntaxe respecte les contraintes suivantes :

- les terminaux sont les noms du modèles, des systèmes hiérarchiques, des blocs atomiques, des ports de systèmes et de blocks, et des flots de données, les entiers, les mots clés `model`, `system`, `block`, `flow`, `from`, `to`, `in`, `out`, `int`, `float boolean`, les accolades ouvrante `{` et fermante `}`, les parenthèses ouvrante `(` et fermante `)`, les crochets ouvrant `[` et fermant `]`, le point virgule `;` la virgule `,` le deux-point `:` et le point `.` ;

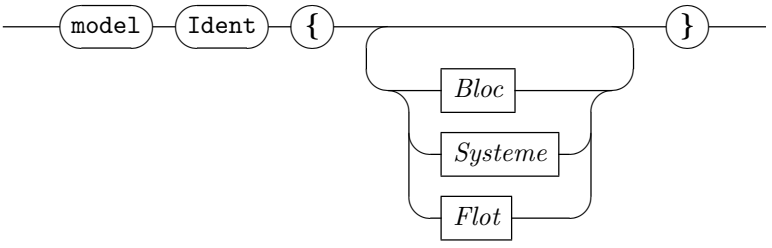
- la définition d'un modèle débute par le mot clé `model` suivi d'un nom de modèle, puis d'une suite d'éléments éventuellement vide comprise entre accolades ouvrante `{` et fermante `}` ;
- un élément est soit un bloc, soit un flot, soit un sous-système ;
- un bloc débute par le mot clé `block` suivi du nom du bloc puis d'une suite non vide de déclarations de ports séparés par des virgules `,` , et comprise entre parenthèse ouvrante `(` et fermante `)` suivie d'un point virgule `;` ;
- un système débute par le mot clé `system` suivi du nom du système puis d'une suite non vide de déclarations de ports séparés par des virgules `,` , et comprise entre parenthèses ouvrante `(` et fermante `)` suivie d'une suite d'éléments éventuellement vide comprise entre accolades ouvrante `{` et fermante `}`, cette suite est suivie d'un point virgule `;` ;
- un flot débute par le mot clé `flow` suivi du nom du flot, du mot clé `from` suivi d'un nom de port éventuellement préfixé par le nom du bloc ou du système auquel il est associé et par un point `.` suivi du mot clé `to` suivie d'une liste éventuellement vide de noms de port éventuellement préfixés par le nom du bloc ou du système auquel il est associé et par un point `.` ;
- une déclaration de port débute par le nom du port puis par un deux-points `:` puis par la modalité `in` ou `out` puis par un type `int`, `float` ou `boolean` et par une déclaration de dimension de tableau optionnelle ;
- une déclaration de dimension de tableau débute par un crochet ouvrant `[` suivi d'une suite non vide d'entiers strictement positifs séparés par des virgules `,` et se termine par un crochet fermant `]`.

Voici les expressions régulières pour les terminaux complexes :

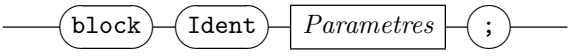
- nom de modèles, de blocs et de systèmes (noté `Ident`) : $[A-Z][a-zA-Z]^\star$
- nom de ports (noté `ident`) : $[a-z][a-zA-Z]^\star$
- valeur entière strictement positive (notée `entier`) : $[1-9][0-9]^\star$

Voici la grammaire au format graphique de Conway :

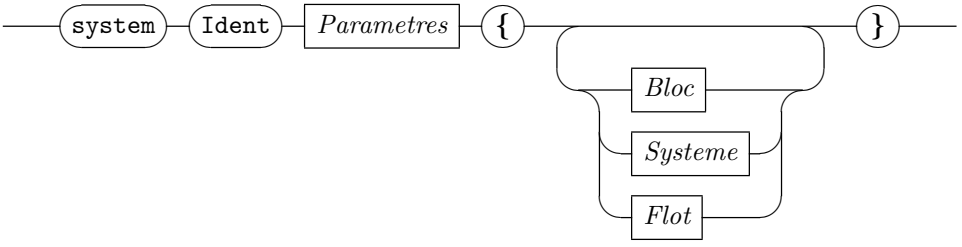
Modele



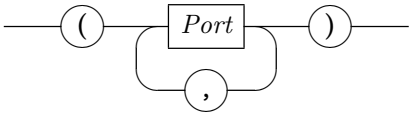
Bloc



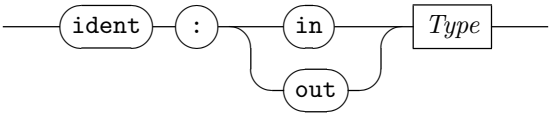
Systeme



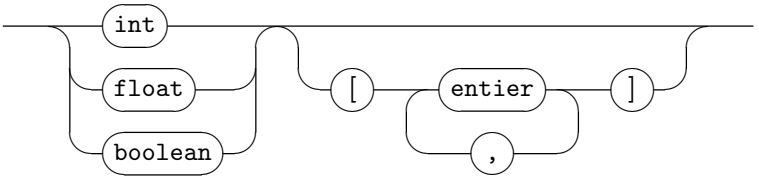
Parametres



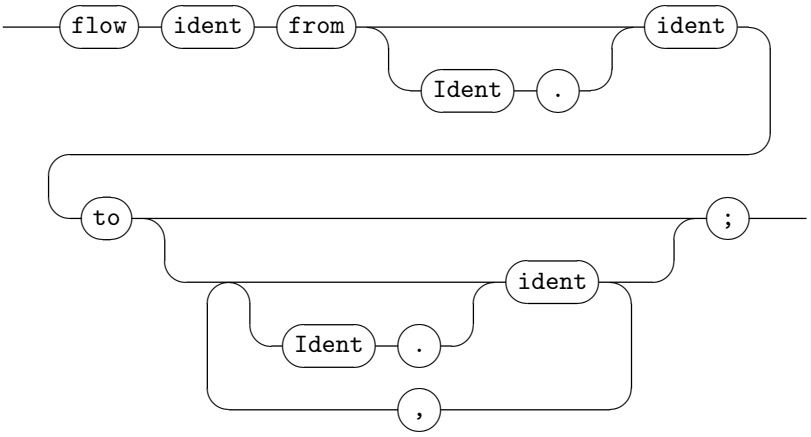
Port



Type



Flot



Voici une grammaire LL(1) sous la forme de règles de production et les symboles directeurs de chaque règle de production :

1.	$R \rightarrow$ `model` ***Ident*** `{` S_E `}`	`model`
2.	$S_E \rightarrow \Lambda$	`}`
3.	$S_E \rightarrow E\ S_E$	`block` `system` `flow`
4.	$E \rightarrow$ `block` ***Ident*** P `;`	`block`
5.	$E \rightarrow$ `system` ***Ident*** P `{` S_E `}`	`system`
6.	$E \rightarrow$ `flow` ***ident*** `from` N_Q `to` L_N `;`	`flow`
7.	$N_Q \rightarrow$ ***ident***	***ident***
8.	$N_Q \rightarrow$ ***Ident*** `.` ***ident***	***Ident***
9.	$L_N \rightarrow \Lambda$	`;`
10.	$L_N \rightarrow N_Q\ S_N$	***Ident ident***
11.	$S_N \rightarrow \Lambda$	`;`
12.	$S_N \rightarrow$ `,` $N_Q\ S_N$	`,`
13.	$P \rightarrow$ `(` L_P `)`	`(`
14.	$L_P \rightarrow D_P\ S_P$	***ident***
15.	$S_P \rightarrow \Lambda$	`)`
16.	$S_P \rightarrow$ `,` $D_P\ S_P$	`,`
17.	$D_P \rightarrow$ ***ident*** `:` $M\ T\ O_T$	***ident***
18.	$M \rightarrow$ `in`	`in`
19.	$M \rightarrow$ `out`	`out`
20.	$T \rightarrow$ `int`	`int`
21.	$T \rightarrow$ `float`	`float`
22.	$T \rightarrow$ `boolean`	`boolean`
23.	$O_T \rightarrow \Lambda$	`,` `)`
24.	$O_T \rightarrow$ `[` ***entier*** S_V `]`	`[`
25.	$S_V \rightarrow \Lambda$	`]`
26.	$S_V \rightarrow$ `,` ***entier*** S_V	`,`

4 Analyseur syntaxique ascendant

Vous devez travailler dans le répertoire `ascendant`.

Vous compilerez régulièrement les modifications réalisées pour détecter les erreurs au plus tôt.

Vous testerez régulièrement votre travail en ajoutant des tests de difficulté croissante dans le répertoire `tests` à la racine de l'archive. Les noms des tests doivent être préfixés par `ok_` pour les tests qui doivent réussir et par `ok_` pour les tests qui doivent échouer. Les tests seront pris en compte dans la notation.

La sémantique de l'analyseur syntaxique consiste à afficher les règles appliquées pour l'analyse.

Complétez les fichiers `Lexer.mll` (analyseur lexical) puis `Parser.mly` (analyseur syntaxique). Le programme principal est contenu dans le fichier `MainSystem.ml`. La commande `dune build MainSystem.exe` produit l'exécutable `_build/default/MainSystem.exe` qui prend comme paramètre le fichier à analyser. L'exemple de ce sujet est disponible dans le répertoire `tests`.

5 Analyseur syntaxique par descente récursive

Vous devez travailler dans le répertoire `descendant`.

Vous compilerez régulièrement les modifications réalisées pour détecter les erreurs au plus tôt.

Vous testerez régulièrement votre travail en ajoutant des tests de difficulté croissante dans le répertoire `tests` à la racine de l'archive. Les noms des tests doivent être préfixés par `ok_` pour les tests qui doivent réussir et par `ok_` pour les tests qui doivent échouer. Les tests seront pris en compte dans la notation.

L'analyseur syntaxique devra afficher les règles appliquées au fur et à mesure de l'analyse. Les éléments nécessaires sont disponibles en commentaires dans le fichier.

Complétez les fichiers `Scanner.mll` (analyseur lexical) puis `Parser.ml` (analyseur syntaxique). Attention, le nom du fichier contenant l'analyseur lexical est différent de celui du premier exercice car les actions lexicales effectuées sont différentes (l'analyseur lexical du premier exercice renvoie

l'unité lexicale reconnue; l'analyseur lexical du second exercice construit la liste de toutes les unités lexicales et renvoie cette liste). Le programme principal est contenu dans le fichier `MainSystem.ml`. La commande `dune build MainSystem.exe` produit l'exécutable `_build/default/MainSystem.exe` qui prend comme paramètre le fichier à analyser. L'exemple de ce sujet est disponible dans le répertoire `tests`.