Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

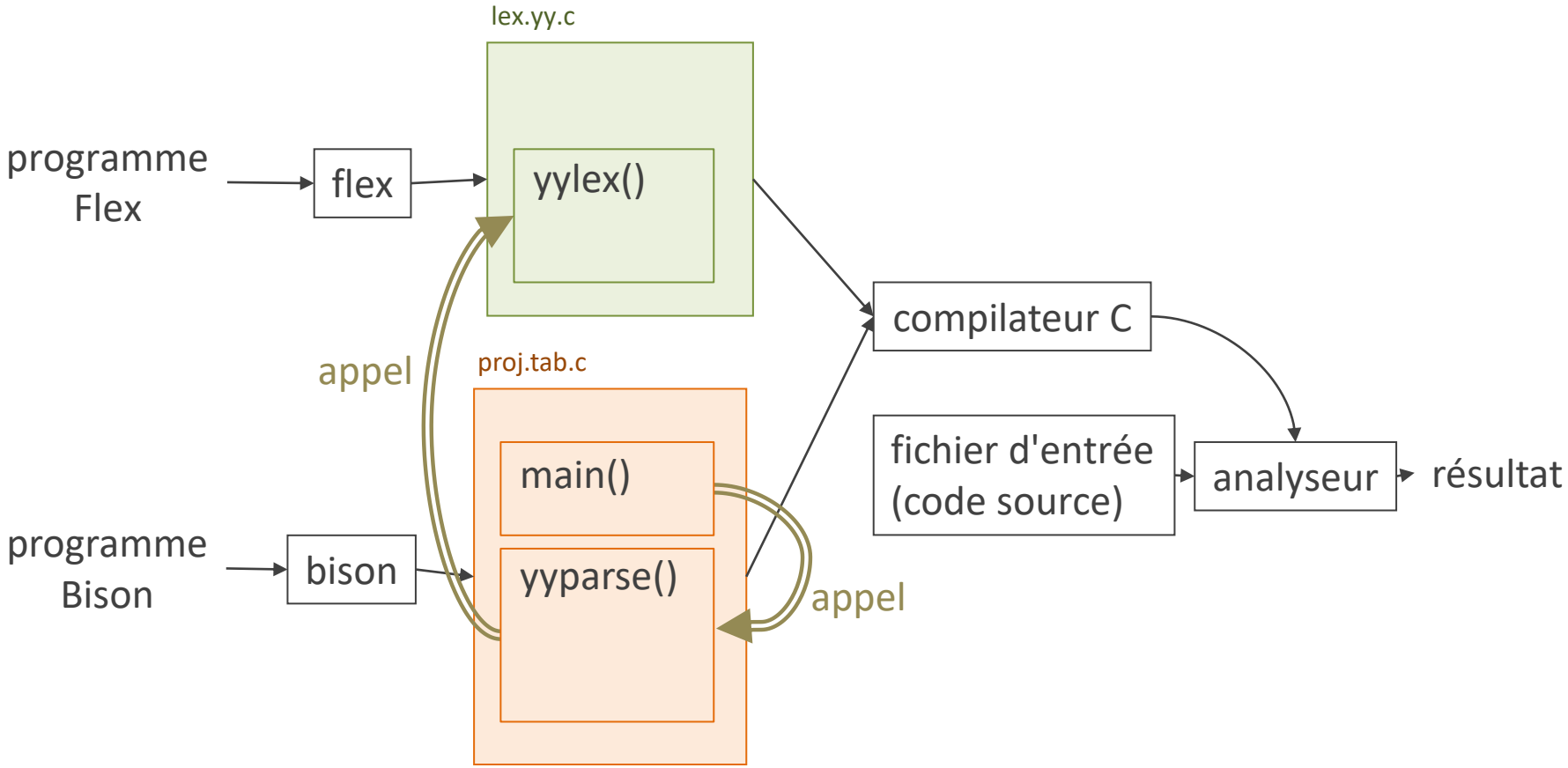
Interface entre Flex et Bison
Traitement des erreurs

Université Gustave Eiffel

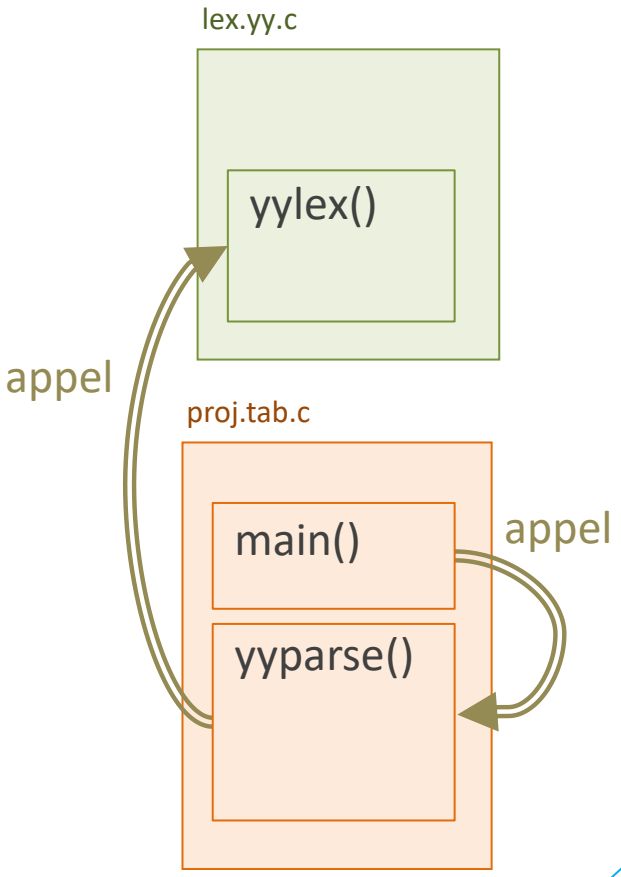
INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Interface Flex-Bison



Interface Flex-Bison




```
[0-9]+   { yylval=atoi(yytext) ;
           return NUMBER; }
.          return yytext[0];
```

Les lexèmes sont représentés par la valeur de retour de yylex()

Type de retour de yylex() : int

- soit un caractère du code source : '(', '+'...
- soit une constante : NUMBER, ELSE...
- signal de fin de fichier : 0

Pas une chaine de caractères

Interface Flex-Bison

```
%{
#include <ctype.h>
int yylex();
int yyerror(char *);
%}
%token NUMBER IDENT
%%
ligne     : expr '\n'
          ;
expr      : expr '+' terme
          | terme
          ;
terme     : terme '*' fact
          | fact
          ;
fact      : '(' expr ')'
          | NUMBER
          | IDENT
          ;
```

Dans le programme Bison

Déclarer les constantes qui représentent des lexèmes : NUMBER, IDENT, ELSE...

Bison les déclare comme constantes entières

Représentation des lexèmes

```
%{
#include <ctype.h>
int yylex();
int yyerror(char *);
%}
%token NUMBER IDENT
%%
ligne     : expr '\n'
          ;
expr      : expr '+' terme
          | terme
          ;
terme     : terme '*' fact
          | fact
          ;
fact      : '(' expr ')'
          | NUMBER
          | IDENT
          ;
```

```
[a-zA-Z_][a-zA-Z_0-9]* {
          strcpy(yylval, yytext);
          return IDENT; }
.         return yytext[0];
```

Valeur de retour de yylex()

Ne suffit pas toujours à représenter un lexème

Elle correspond à un **terminal** de la grammaire

IDENT : la même valeur pour tous les identificateurs

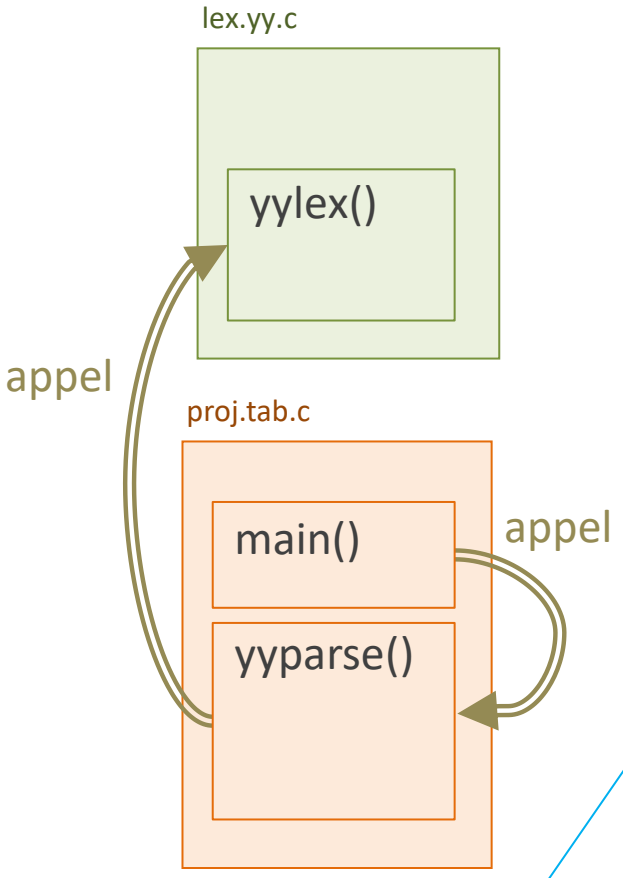
Il faut une autre valeur, différente pour chaque identificateur :

« Attribut » du lexème

Pour un IDENT : la chaine de caractères de l'identificateur

Pour un NUM : la valeur de la constante numérique

Représentation des lexèmes




```
[a-zA-Z_][a-zA-Z_0-9]* {
        strcpy(yylval, yytext);
        return IDENT; }
.       return yytext[0];
```

Flex et Bison n'ont pas de classe lexème
Les lexèmes sont représentés en 2 valeurs

1. Un terminal de la grammaire

Valeur de retour de yylex()
Type de retour de yylex() : int
- soit un caractère du code source : '(', '+'...
- soit une constante : NUMBER, IDENT, ELSE...

Pas une chaine de caractères

2. L'attribut du lexème

`yylval`, variable globale
Type par défaut : int

Attributs des lexèmes

```
%token <ident> IDENT
%token <num> NUM
%%
e   : e '+' t      {printf("+");}
    | t
    ;
t   : t '*' f      {printf("*");}
    | f
    ;
f   : '(' e ')'
    | IDENT        {printf("%s", $1 );}
    | NUM          {printf("%d", $1 );}
    ;
```

Accès à `yylval`

$1, $2…

La numérotation commence à 1 et tient compte des terminaux et non-terminaux dans la règle

Attributs des lexèmes

Terminal	Type de l'attribut	Nom du champ
IDENT	char[64]	ident
NUM	int	num

Type de `yylval`

`IDENT` : la chaine de caractères de l'identificateur

`NUM` : la valeur de la constante numérique

```
[a-zA-Z_][a-zA-Z0-9_]*   { strcpy(yylval.ident, yytext) ;
                           return IDENT; }
[0-9]+                   { sscanf(yytext, "%d", &(yylval.num) );
                           return NUM; }
```

On veut que **`yylval`** n'ait pas le même type pour tous les lexèmes

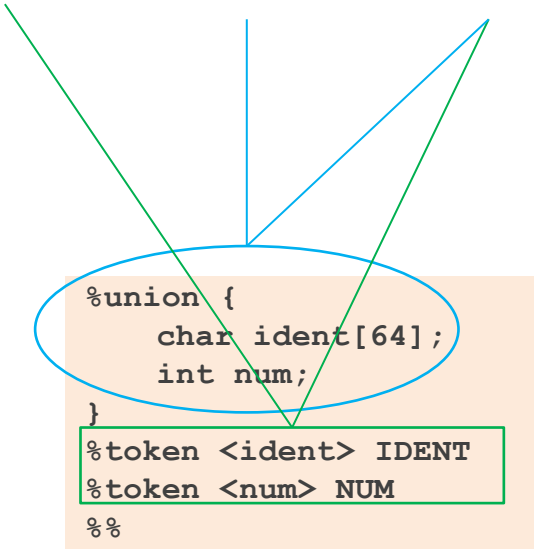
Dans la spécification Bison

- mettre une déclaration **`%union`** avec les types de tous les attributs
- dans la déclaration **`%token`** des lexèmes qui ont un **`yylval`**, préciser le champ de l'union

```
%union {
    char ident[64];
    int num;
}
%token <ident> IDENT
%token <num> NUM
%%
```

Attributs des lexèmes

Terminal	Type de l'attribut	Nom du champ
`IDENT`	`char[64]`	`ident`
`NUM`	`int`	`num`



La déclaration `%union` fait le lien entre le type des attributs et les champs de l'union

Les déclarations `%token` font le lien entre le terminal et le champ de l'union

Bison en déduit le type de l'attribut pour chaque terminal et déclare `yylval` dans le fichier .h

Attributs des lexèmes

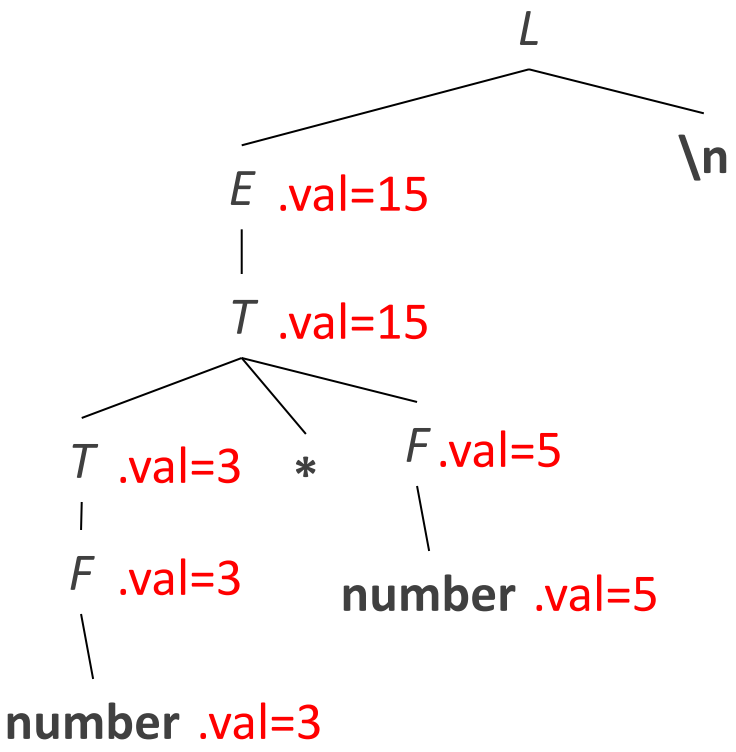
```
%union {
    char ident[64];
    int num;
}
%token <ident> IDENT
%token <num> NUM
%%
e   : e '+' t     {printf("+");}
    | t
    ;
t   : t '*' f     {printf("*");}
    | f
    ;
f   : '(' e ')'
    | IDENT       {printf("%s", $1 );}
    | NUM         {printf("%d", $1 );}
    ;
```

S'il y a une déclaration `%union`

`$1`, `$2`... accèdent automatiquement au champ déclaré

donc écrire `$1` et non `$1.ident`, `$1.num`

Arbre décoré



Les nœuds de l'arbre de dérivation sont occupés par des terminaux et des non-terminaux

Les attributs des lexèmes sont dans les feuilles

Les autres nœuds peuvent avoir des attributs aussi

Exemples d'utilisation

1. Calculer la valeur d'une expression
2. Construire un arbre au fur et à mesure de l'analyse syntaxique

Dans les actions, créer des nœuds d'arbre et sauvegarder leur adresse dans les attributs

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Sommaire

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Traitement des erreurs de syntaxe

Émission des messages (diagnostic)

On choisit une des hypothèses possibles

Exemple : **e = a + b c ;**

- opérateur manquant (**e = a + b * c ;**)
- identificateur en trop (**e = a + b ;**)
- erreur lexicale (**e = a + bc ;**)

C'est toujours un peu arbitraire

Redémarrage

Pour que l'analyseur traite la suite après la première erreur

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Méthodes de redémarrage

Mode panique

Sauter un ou plusieurs lexèmes à partir de l'erreur

Règles d'erreur

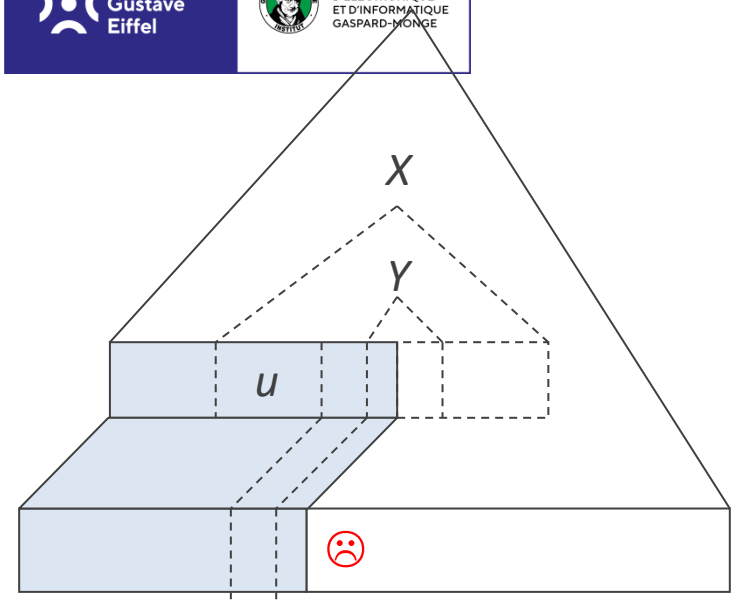
Ajouter à la grammaire des constructions incorrectes avec message d'erreur

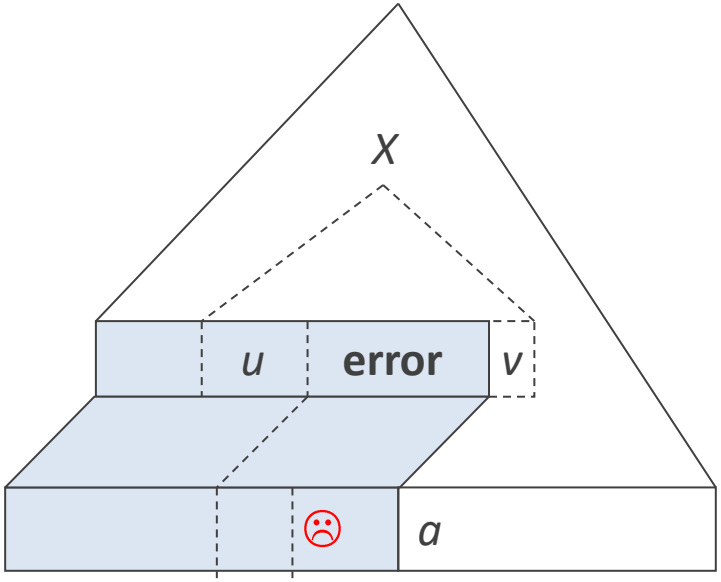
La grammaire utilisée par l'analyseur est distincte de celle décrite dans le manuel d'utilisation

Mode correction ou réparation

Modifier les prochains lexèmes pour reconstituer ce que serait la donnée sans l'erreur détectée

Mode panique avec Bison





On veut qu'en cas d'erreur détectée à l'intérieur d'un non-terminal *X*, l'analyse syntaxique redémarre dans ce *X*

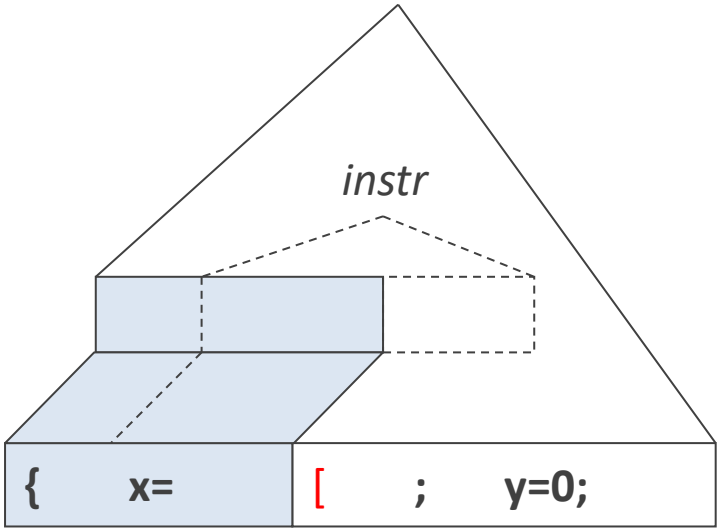
On ajoute à la grammaire des règles du type

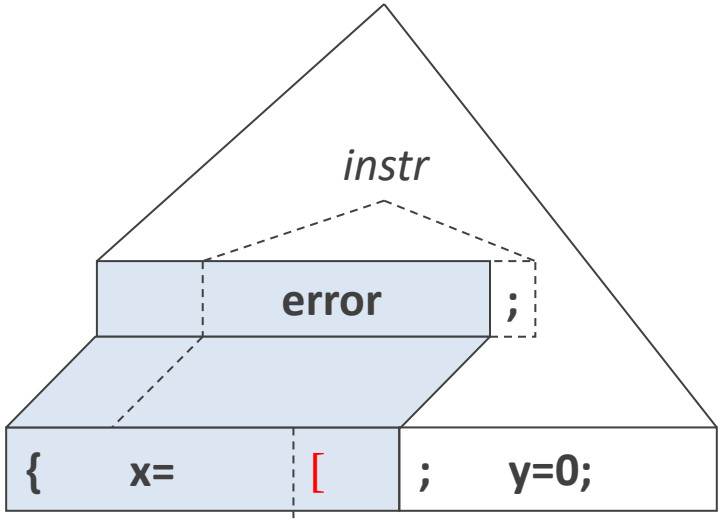
X : *u* **error** *v*

En cas d'erreur, l'analyseur

- dépile jusqu'à *u* et empile **error**
- saute des terminaux de la donnée jusqu'à rencontrer un terminal *a* acceptable selon la table
- s'il trouve, reprend l'analyse normalement

Mode panique avec Bison





{ *x* = [; *y* = 0 ;

instr : **error ';'**

L'analyseur
- dépile jusqu'à un symbole qui a une transition par **error** dans l'automate
- empile **error**
- saute "**[**" et rencontre "**;**" qui est acceptable selon la table
- reprend l'analyse normalement

Mode correction avec l'analyse LR

se1 émettre "opérande manquant" et empiler 3 (**id**)

se2 émettre "parenthèse fermante en trop" et sauter **)** dans la donnée

se3 émettre "opérateur manquant" et empiler 4 (**+**)

se4 émettre "parenthèse fermante manquante" et empiler 9 (**)**)

Ajouter dans les cases vides de la table des appels à des fonctions de traitement d'erreur

Les fonctions émettent un message et effectuent des actions pour redémarrer après l'erreur

Université Gustave Eiffel

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Exemple

	terminaux						n.-t.
état	**id**	**+**	*****	**(**	**)**	**$**	*E*
0 (ε)	e3	**se1**	**se1**	e2	**se2**	**se1**	1
1 (*E*)	**se3**	e4	e5	**se3**	**se2**	acc	**1**
2 (**(**)	e3	**se1**	**se1**	e2	**se2**	**se1**	6
3 (**id**)	**r4**	r4	r4	**r4**	r4	r4	
4 (**+**)	e3	**se1**	**se1**	e2	**se2**	**se1**	7
5 (*****)	e3	**se1**	**se1**	e2	**se2**	**se1**	8
6 (*E*)	**se3**	e4	e5	**se3**	e9	**se4**	**1**
7 (*E*)	**r1**	r1	e5	**r1**	r1	r1	**1**
8 (*E*)	**r2**	r2	r2	**r2**	r2	r2	
9 (**)**)	**r3**	r3	r3	**r3**	r3	r3	