Γλώσσες Προγραμματισμού-Μεταγλωττιστές

Eργασία 1η

Φαίδρα Θεοχαρίδη ics22069

Πανεπιστήμιο Μακεδονίας 2023

## 1η Άσκηση

Η αρχική Γραμματική:

GAgent= {S,P,N,T} όπου:

S={Agent}

P=

**Agent** ::= “name” “[“ GuardedActions “]”

**GuardedActions** ::= GuardedActions GuardedAct

| “mem-clear” “;”

**GuardedAct** ::= “detector” “Rel” “T\_NUM” “=>” Action “;”

| “true” “=>” Action “;”

**Action** ::= “forward” “T\_NUM”

| “turn” “left” “Τ\_NUM”

| “turn” “right” “Τ\_NUM”

N={Agent, GuardedActions, GuardedAct, Action}

T={name,[,],mem-clear,;,detector,Rel,T\_NUM,=>,true,forward,turn,left,right}

Παρατηρούμε αριστερή αναδρομή στον κανόνα GuardedActions (**GuardedActions** ::= GuardedActions GuardedAct), για να το μετατρέψουμε σε LL γραμματική προσθέτουμε έναν νέο κανόνα GuardedActionsNew ο οποίος θα είναι της μορφής: **GuardedActionsNew** ::=GuardedAct GuardedActions

|ε

Και θα μετατρέψουμε τον αρχικό κανόνα σε:

**GuardedActions** ::=“mem-clear” “;” GuardedActionsNew

Έπειτα, παρατηρούμε ότι ο κανόνας Action έχει δύο εναλλακτικές παραγωγές που ξεκινάνε με το ίδιο σύμβολο(Action ::= “turn” “left” “Τ\_NUM” | “turn” “right” “Τ\_NUM”). Πραγματοποιούμε αριστερή παραγοντοποίηση προσθέτοντας έναν κανόνα Direction:

**Direction** ::= “left” “Τ\_NUM”

|“right” “Τ\_NUM”

Και μετατρέποντας το Αction σε:

**Action** ::= “forward” “T\_NUM”

| “turn” Direction

Επομένως η νέα γραμματική που προκύπτει είναι:

GAgent= {S,P,N,T} όπου:

S={Agent}

P=

**Agent** ::= “name” “[“ GuardedActions “]”

**GuardedActions** ::=“mem-clear” “;” GuardedActionsNew

**GuardedActionsNew** ::= GuardedAct GuardedActions

|ε

**GuardedAct** ::= “detector” “Rel” “T\_NUM” “=>” Action “;”

| “true” “=>” Action “;”

**Action** ::= “forward” “T\_NUM”

| “turn” Direction

**Direction** ::= “left” “Τ\_NUM”

|“right” “Τ\_NUM”

N={Agent, GuardedActions, GuardedActionsNew, GuardedAct, Action, Direction}

T={name,[,],mem-clear,;,detector,Rel,T\_NUM,=>,true,forward,turn,left,right}

*\*τα υπογραμμισμένα είναι οι νέοι/αλλαγμένοι κανόνες*

**FIRST/FOLLOW**

* FIRST(Agent) = FIRST(“name”) = {“name”}
* FIRST(GuardedActions) = FIRST(“mem-clear”) = {“mem-clear”}
* FIRST(GuardedActionsNew) = FIRST(GuardedAct) U {ε} = FIRST(“detector”) U FIRST(“true”) U {ε}={“detector”,”true”,ε}
* FIRST(GuardedAct) = FIRST(“detector”) U FIRST(“true”) = {“detector”,”true”}
* FIRST(Actions) = FIRST(“forward”) U FIRST(“turn”) = {“forward”,”turn”}
* FIRST(Direction) = FIRST(“left”) U FIRST(“right”) ={“left”,”right”}
* FOLLOW(Agent) = {EOF}
* FOLLOW(GuardedActions) = FOLLOW(GuardedActions) U FIRST(“]”) = {“]”}
* FOLLOW(GuardedActionsNew) = FOLLOW(GuardedActionsNew) U FOLLOW(GuardedActions)={“]”}
* FOLLOW(GuardedAct) = FOLLOW(GuardedAct) U (FIRST(GuardedActionsNew)- {ε}) = {“detector”, “true”}
* FOLLOW(Action) = FOLLOW(Action) U (FIRST(“;”)-{ε}) = {“;”}
* FOLLOW(Direction) = FOLLOW(Direction) U (FOLLOW(Action) - {ε}) = {“;”}

**Υλοποίηση δύο κανόνων της γραμματικής**

void Agent(){ /\*ξεκινάμε από το αρχικό σύμβολο της γραμματικής\*/

if (token==TK\_name) /\*FIRST(Agent)= {“name”}\*/

{match(TK\_name); match (TK\_LB); GuardedActions();match (TK\_RB);}

/\*κάνουμε match το name και έπειτα τη αριστερή αγκύλη, καλούμε τη GuardedActions και τέλος κάνουμε match τη δεξιά αγκύλη\*/

else error\_syntax();} /\*αν υπάρχει λάθος, καλείται η error\_syntax, και εμφανίζεται το σφάλμα\*/

void GuardedActions(){

if (token==TK\_memcl) /\*FIRST(GuardedActions)= {“mem-clear”}\*/

{match(TK\_memcl); match (TK\_SC); GuardedActionsNEW();} /\*κάνουμε match το mem-clear, το ερωτηματικό και καλούμε τη GuardedActionsΝEW\*/

else error\_syntax();}

**Κώδικας**

%{

#include <stdio.h>

#include <string.h>

/\* ADD your token definitions here \*/

#define TK\_name 300

#define TK\_memcl 301

#define TK\_detector 302

#define TK\_Rel 303

#define TK\_NUM 304

#define TK\_implies 305 //=>

#define TK\_LB 306 //left bracket

#define TK\_RB 307 //right bracket

#define TK\_SC 308 //semi-colon

#define TK\_true 309

#define TK\_forward 310

#define TK\_turn 311

#define TK\_right 312

#define TK\_left 313

/\* Needed for syntax analyser\*/

#define LEXEND 0

int line = 0;

%}

name [a-z]+

Rel (">"|"<"|"=")

detector ("left-laser"|"right-laser"|"front-sonar")

nzdigit [1-9]

T\_NUM ({nzdigit}|0)+

newline \n|\x0A|\x0D|\x0A

ws [ \t]+

%%

{Rel} {line++; return TK\_Rel;}

{T\_NUM} {line++; return TK\_NUM;}

"mem-clear" {line++; return TK\_memcl;}

"->" {line++; return TK\_implies;}

"[" {line++; return TK\_LB;}

"]" {line++; return TK\_RB;}

";" {line++; return TK\_SC;}

"true" {line++; return TK\_true;}

"forward" {line++; return TK\_forward;}

"turn" {line++; return TK\_turn;}

"right" {line++; return TK\_right;}

"left" {line++; return TK\_left;}

{detector} {line++; return TK\_detector;}

{name} {line++; return TK\_name;}

{newline} {line++;}

{ws} {/\*Do Nothing\*/}

%%

int token = 0;

/\* Add your LL functions forward here \*/

void Agent(void);

void GuardedActions(void);

void GuardedActionsNEW(void);

void GuardedAct(void);

void Action(void);

void Direction(void);

void error\_syntax(void);

void match(int tk);

/\* Add your function implementations here. \*/

void Agent(){

if (token==TK\_name)

{match(TK\_name); match (TK\_LB); GuardedActions();match (TK\_RB);}

else error\_syntax();}

void GuardedActions(){

if (token==TK\_memcl)

{match(TK\_memcl); match (TK\_SC); GuardedActionsNEW();}

else error\_syntax();}

void GuardedActionsNEW(){

if (token == TK\_detector)

{match(TK\_detector); GuardedAct(); GuardedActionsNEW();}

else if (token == TK\_true)

{match(TK\_true); GuardedAct(); GuardedActionsNEW();}

else if (token!=TK\_RB)

error\_syntax();}

void GuardedAct(){

if (token == TK\_Rel)

{match(TK\_Rel); match(TK\_NUM); match(TK\_implies); Action(); match(TK\_SC);}

else if (token == TK\_implies)

{ match(TK\_implies); Action(); match(TK\_SC);}

else error\_syntax();}

void Action() {

if (token == TK\_forward)

{match(TK\_forward); match(TK\_NUM);}

else if (token == TK\_turn )

{match(TK\_turn); Direction();}

else error\_syntax();

}

void Direction() {

if (token == TK\_left)

{match(TK\_left); match(TK\_NUM);}

else if (token == TK\_right)

{match(TK\_right); match(TK\_NUM);}

else error\_syntax();

}

void error\_syntax(void){

printf("Syntax Error reading %s at line %d of input. \n",yytext, line);

exit(-1);

}

void match(int tk){

if (token == tk) token = yylex();

else error\_syntax();

}

int main(int argc, char \*\*argv ){

++argv, --argc; /\* skip over program name \*/

if ( argc > 0 )

yyin = fopen( argv[0], "r" );

else

yyin = stdin;

token = yylex();

Agent();

if (token == LEXEND) printf("\n Success! \n");

else error\_syntax();

return 0;

}

## 

## 2η Άσκηση

Η γραμματική:

Gagent= {S,P,N,T} όπου:

S={Agent}

P=

**agent** ::= beliefs plans

**beliefs** ::= beliefs belief | ε

**belief** ::= predicate “.”

**predicate** ::= “Atom” “(“ terms “)”

**plans** ::= plans plan | ε

**plan** :: = trig\_event “:” context “<-” body “.”

**trig\_event** ::= “+” predicate | “-” predicate | “+” goal | “-” goal

**context** ::= “true” | cliterals

**cliterals** ::= literal | literal “&” cliterals

**literal** ::= predicate | “not” “(“ predicate “)” | boolExpr

**goal** ::= “!” predicate | “?” predicate

**body** ::= “true” | actions

**actions** ::= action | action “;” actions

**action** ::= predicate | goal | belief\_update

**belief\_update** ::= “+” predicate | “-” predicate

**terms** ::= term | term “,” terms

**term** ::= “Var”|“Atom”|“Number”|“Atom” “(“ terms “)”

**boolExpr** ::= boolE | boolExpr “|” boolE

**boolE** ::= boolarg relOp boolarg

**boolarg** ::= “Number” | “Var”

**relOp** ::= “>” | “<” | “=” | “>=” | “=<”

N={agent, beliefs, belief, predicate, plans, plan, trig\_event, context, cliterals, literal, goal, body, actions, action, belief\_update, terms, term, boolExpr, boolE, boolarg, relOp}

T={., (, ), :, <-, +, -, true, &, not, !, ?, ;, ,, |, Number, Var, <, >, =, >=, =<}

**Κώδικας**

**agentSpeak.l**

%{

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

int line=1;

%}

D [0-9]

REAL {D}+\.{D}+

NUM {D}+|{REAL}

CAPLETTER [A-Z]

LOWLETTER [a-z]

LETTER {CAPLETTER}|{LOWLETTER}

VARIABLE {CAPLETTER}({LETTER}|{D}|"\_")\*

ATOM {LOWLETTER}({LETTER}|{D}|"\_")\*

ws [ \t]+

newline \n|\x0A|\x0D\x0A

%%

"+" {return '+';}

"-" {return '-';}

"." {return '.';}

"," {return ',';}

":" {return ':';}

";" {return ';';}

"(" {return '(';}

")" {return ')';}

"!" {return '!';}

"?" {return '?';}

"&" {return '&';}

"|" {return '|';}

"=" {return '=';}

">" {return '>';}

"<" {return '<';}

"<-" {return T\_ASS;}

">=" {return T\_GE;}

"=<" {return T\_LE;}

"true" {return T\_TRUE;}

"not" {return T\_NOT;}

{VARIABLE} {return T\_VAR;}

{ATOM} {return T\_Atom;}

{NUM} {return T\_NUM;}

{newline} { line++;}

{ws} { /\*Do nothing \*/ }

. { /\*Do nothing \*/ }

%%

**agentSpeak.y**

%{

#include <stdio.h>

#include <stdlib.h>

// To remove persistent warning..

int yylex();

void yyerror (const char \* msg);

%}

%define parse.error verbose

//token declarations

%token '('

%token ')'

%token ','

%token '+'

%token '-'

%token '.'

%token ':'

%token ';'

%token '!'

%token '?'

%token '&'

%token '|'

%token '='

%token '<'

%token '>'

%token T\_ASS "<-"

%token T\_GE ">="

%token T\_LE "=<"

%token T\_TRUE "true"

%token T\_NOT "not"

%token T\_VAR "Var"

%token T\_Atom "Atom"

%token T\_NUM "Number"

%%

agent : beliefs plans

;

beliefs : /\*empty\*/

| beliefs belief

;

belief : predicate '.'

| error '?'

;

predicate : "Atom" '(' terms ')'

;

plans : /\*empty\*/

| plans plan

;

plan : trig\_event ':' context "<-" body '.'

;

trig\_event : '+' predicate | '-' predicate

| '+' goal | '-' goal

;

context : "true"

| cliterals

;

cliterals : literal

| literal '&' cliterals

;

literal : predicate

| "not" '(' predicate ')'

| boolExpr

;

goal : '!' predicate

| '?' predicate

;

body : "true"

| actions

;

actions :action

| action ';' actions

;

action : predicate

| goal

| belief\_update

;

belief\_update : '+' predicate

| '-' predicate

;

terms : term

| term ',' terms

;

term : "Var"

|"Atom"

|"Number"

|"Atom" '(' terms ')'

;

boolExpr : boolE

| boolExpr '|' boolE

;

boolE : boolarg relOp boolarg

;

boolarg : "Number"

| "Var"

;

relOp : '>'

| '<'

| '='

| ">="

| "=<"

;

%%

#include "agentSpeak.lex.c"

void yyerror (const char \* msg)

{

printf("Error(line %d) : %s\n",line, msg);

}

int main(int argc, char \*\*argv ){

++argv, --argc; /\* skip over program name \*/

if ( argc > 0 )

yyin = fopen( argv[0], "r" );

else

yyin = stdin;

int result = yyparse();

if (result == 0 && yynerrs == 0)

printf("Syntax OK!\n");

else

printf("There were %d errors in code. Failure!\n", yynerrs);

return result;

}