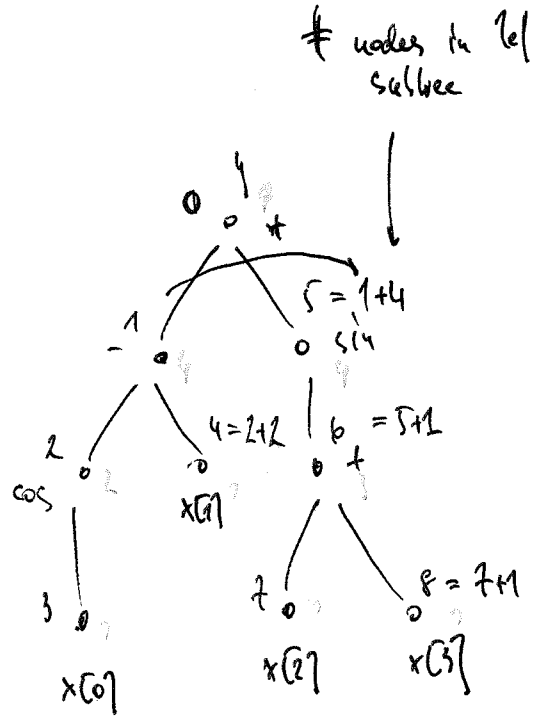
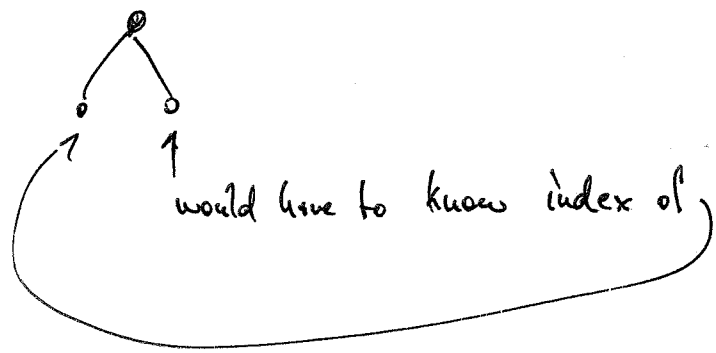
Assignment-level SAC



S-threaded enumeration impossible



Implementation: pick AST of this in terms of unique aux vs names

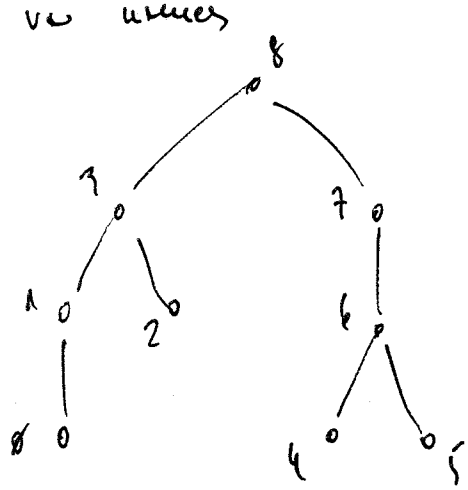


Table 4.4. *Syntax-directed assignment-level SAC for* $y = \sin(x * 2)$.

i	PARSED	ACTION	\$\$.i	\$\$.c	Comment
0	V	S			
	...				
11	$V = F(V$	S			
7		R($P7$)	2	$\boxed{v_2 = x;}$	
	...				
13	V=F(eNC	S			
8		R($P8$)	3	$\boxed{v_3 = 2;}$	
13	V=F(eNe	S			
14		R($P5$)		$v_2 = x;$	$< \ldots = e^{r1}.c$
				$v_3 = 2;$	$< \ldots = e^{r2}.c$
			1	$\boxed{v_1 = v_2 * v_3;}$	$N.c =$ " $*$ "
					$e^{r1}.i = 2$, $e^{r2}.i = 3$
11	V=F(e	S			
15	V=F(e)	S			
18		R($P6$)		$v_2 = x;$	$<$
				$v_3 = 2;$	$<$
				$v_1 = v_2 * v_3;$	$< \ldots = e^{r}.c$
			0	$\boxed{v_0 = \sin(v_1);}$	$F.c =$ "sin", $e^{r}.i = 1$
4	V=e	S			
10	V=e;	S			
14		R($P3$)		$v_2 = x;$	$<$
				$v_3 = 2;$	$<$
				$v_1 = v_2 * v_3;$	$<$
				$v_0 = \sin(v_1);$	$< \ldots = e.c$
				$\boxed{y = v_0;}$	$V.c =$ "y", $e.i = 0$
	...				
0	\$accept	ACCEPT			

```
#define BUFFER_SIZE 100000

typedef struct {
  int i;
  char* c;
} ptNodeType;

#define YYSTYPE ptNodeType
```

The `bison` preprocessor macro YYSTYPE is set to ptNodeType for this purpose. A sufficiently large buffer of characters is required to store the SAC. For simplicity, we define its size statically in `ast.h`.

The `flex` input file includes `ast.h` prior to `parser.tab.h`. Otherwise, it is similar to Listings 4.9 and 4.10.

Syntax-orientierte SAC auf Zuweisungsebene

Syntax-directed assignment-level SAC

Qn no. 4

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "ast.h"

static int sacvc; // sac variable counter

void get_memory(YYSTYPE* v) {
  v->c=malloc(BUFFER_SIZE*sizeof(char)); }
void free_memory(YYSTYPE* v) { if (v->c) free(v->c); }
%}

%token V C F L N

%left L
%left N

%%

sl2_program : s {
    printf("%s",$1.c);
    free_memory(&$1);
  };
s : a
  | a s {
    get_memory(&$$);
    sprintf($$.c,"%s%s",$1.c,$2.c);
    free_memory(&$1); free_memory(&$2);
  };
a : V '=' { sacvc=0; } e ';' {
    get_memory(&$$);
    sprintf($$.c,"%s%s=v%d;\n",$4.c,$1.c,$4.j);
    free_memory(&$1); free_memory(&$4);
  };
e : e L e {
    $$.j=sacvc++;
    get_memory(&$$);
    sprintf($$.c,"%s%sv%d=v%d%sv%d;\n",
                  $1.c,$3.c,$$.j,$1.j,$2.c,$3.j);
    free_memory(&$1);
  }
  | e N e {
    $$.j=sacvc++;
    get_memory(&$$);
    sprintf($$.c,"%s%sv%d=v%d%sv%d;\n",
                  $1.c,$3.c,$$.j,$1.j,$2.c,$3.j);
    free_memory(&$1);
  }
  | F '(' e ')' {
    $$.j=sacvc++;
    get_memory(&$$);
    sprintf($$.c,"%sv%d=sin(v%d);\n",
                  $3.c,$$.j,$3.j);
    free_memory(&$3);
  }
  | V {
    $$.j=sacvc++;
```

```
    get_memory(&$$);
    sprintf($$.c,"v%d=%s;\n",$$.j,$1.c);
    free_memory(&$1);
  }
  | C {
    $$.j=sacvc++;
    get_memory(&$$);
    sprintf($$.c,"v%d=%s;\n",$$.j,$1.c);
    free_memory(&$1);
  };

%%

int yyerror(char *msg) { printf("ERROR: %s \n",msg); return -
1; }

int main(int argc,char** argv)
{
  FILE *source_file=fopen(argv[1],"r");
  lexinit(source_file); yyparse(); fclose(source_file);
  return 0;
}
```

```
%{
#include "ast.h"
#include "parser.tab.h"

#include<stdlib.h> // malloc
#include<string.h> // strcpy

void to_parser() {
  yylval.c=(char*)malloc(BUFFER_SIZE*sizeof(char));
  strcpy(yylval.c,yytext);
}
%}

whitespace      [ \t\n]+
variable   [a-z]
constant   [0-9]

%%

{whitespace}     { }
"sin"            { to_parser(); return F; }
"+"              { to_parser(); return L; }
"*"              { to_parser(); return N; }
{variable}       { to_parser(); return V; }
{constant}       { to_parser(); return C; }
.          { return yytext[0]; }

%%

void lexinit(FILE *source) { yyin=source; }
```

Scanner.l

```
#define BUFFER_SIZE 100000

typedef struct {
  int i;
  char * c;
} astNodeType;

#define YYSTYPE astNodeType
```

ast.h

```
sdsac : parser.tab.c lex.yy.c
	gcc -g $^ -lfl -o $@

parser.tab.c : parser.y
	bison -d $<

lex.yy.c : scanner.l
	flex $<

clean :
	rm -f parser.tab.* lex.yy.* sdsac

.PHONY: clean
```

Makefile

```
#include<iostream>
#include<cmath>
using namespace std;

int f(double& x, double& y) {
  double v0,v1,v2,v3,v4,v5;
//#include "test.in"
#include "test.out"
}

int main() {
  double x=1.,y=2.;
  f(x,y);
  cout << x << "\t" << y << endl;
  return 0;
}
```

Treiber für Funktionsauswertung

Table 4.5. *Syntax-directed tangent-linear code for* $y = \sin(x*2)$; *set* $v_i^{(1)} \equiv$ `vi_` *to establish the link with the code that is generated by the syntax-directed tangent-linear code compiler.*

i	PARSED	ACTION	\$\$.i	\$\$.c
0	V	S		
	...			
11	$V = F(V$	S		
7		R($P7$)	1	$v_2^{(1)} = x^{(1)};\ v_2 = x;$
	...			
13	V=F(eNC	S		
8		R($P8$)	2	$v_3^{(1)} = 0;\ v_3 = 2;$
13	V=F(eNe	S		
14		R($P5$)		$v_2^{(1)} = x^{(1)};\ v_2 = x;$ $v_3^{(1)} = 0;\ v_3 = 2;$
			3	$v_1^{(1)} = v_2^{(1)} * v_3 + v_2 * v_3^{(1)};\ v_1 = v_2 * v_3;$
11	V=F(e	S		
15	V=F(e)	S		
18		R($P6$)		$v_2^{(1)} = x^{(1)};\ v_2 = x;$ $v_3^{(1)} = 0;\ v_3 = 2;$ $v_1^{(1)} = v_2^{(1)} * v_3 + v_2 * v_3^{(1)};\ v_1 = v_2 * v_3;$
			4	$v_0^{(1)} = \cos(v_1) * v_1^{(1)};\ v_0 = \sin(v_1);$
4	V=e	S		
10	V=e;	S		
14		R($P3$)		$v_2^{(1)} = x^{(1)};\ v_2 = x;$ $v_3^{(1)} = 0;\ v_3 = 2;$ $v_1^{(1)} = v_2^{(1)} * v_3 + v_2 * v_3^{(1)};\ v_1 = v_2 * v_3;$ $v_0^{(1)} = \cos(v_1) * v_1^{(1)};\ v_0 = \sin(v_1);$ $y^{(1)} = v_0^{(1)};\ y = v_0;$
	...			
0	\$accept	ACCEPT		

```
if(x<y) {
  x=sin(x);
  while (y<x) { x=sin(x*3); }
  y=4*x+y;
}
```

yields

```
if (x<y) {
  v0_=x_; v0=x;
  v1_=cos(v0)*v0_; v1=sin(v0);
  x_=v1_; x=v1;
```

Syntax-orientierte Tangenten-linear Code

Syntax-directed Target-Given Code

plan.y

```
%{

#include <stdio.h>
#include <stdlib.h>
#include "ast.h"

extern int yylex();
extern void lexinit(FILE*);

static int sacvc; // SAC variable counter

void get_memory(YYSTYPE* v) {
  v->c=malloc(BUFFER_SIZE*sizeof(char));
}
void free_memory(YYSTYPE* v) {
  if (v->c) free(v->c);
}

%}

%token V C F L N

%left L
%left N

%%

sl_program : s
  {
    printf("%s",$1.c);
    free_memory(&$1);
  }
  ;
s : a
  | a s
  {
    get_memory(&$$);
    sprintf($$.c,"%s%s",$1.c,$2.c);
    free_memory(&$1); free_memory(&$2);
  }
  ;
a : V '='
  {
    sacvc=0;
  }
  e ';'
  {
    get_memory(&$$);
    sprintf($$.c,"%s%s_=v%d_; %s=v%d; \n",
                 $4.c,$1.c,$4.j,$1.c,$4.j);
    free_memory(&$1); free_memory(&$4);
  }
  ;
e : e L e
  {
    $$.j=sacvc++;
    get_memory(&$$);
```

```
    sprintf($$.c,"%s%sv%d_=v%d_%sv%d_; v%d=v%d%sv%d; \n",
                 $1.c,$3.c,$$.j,$1.j,$2.c,$3.j,
                 $$.j,$1.j,$2.c,$3.j);
    free_memory(&$1);
  }
  |e N e
  {
    if (!strcmp($2.c,"*")) {
      $$.j=sacvc++;
      get_memory(&$$);
      sprintf($$.c,
        "%s%sv%d_=v%d_*v%d+v%d*v%d_; v%d=v%d%sv%d; \n",
         $1.c,$3.c,$$.j,$1.j,$3.j,$1.j,$3.j,
         $$.j,$1.j,$2.c,$3.j);
      free_memory(&$1);
    }
  }
  | F '(' e ')'
  {
    if (!strcmp($2.c,"sin")) {
      $$.j=sacvc++;
      get_memory(&$$);
      sprintf($$.c,"%sv%d_=cos(v%d)*v%d_; v%d=sin(v%d);\n",
                   $3.c,$$.j,$3.j,$3.j,$$.j,$3.j);
      free_memory(&$3);
    }
  }
  | V
  {
    $$.j=sacvc++;
    get_memory(&$$);
    sprintf($$.c,"v%d_=%s_; v%d=%s;\n",$$.j,$1.c,$$.j,$1.c);
    free_memory(&$1);
  }
  | C
  {
    $$.j=sacvc++;
    get_memory(&$$);
    sprintf($$.c,"v%d_=0; v%d=%s;\n",$$.j,$$.j,$1.c);
    free_memory(&$1);
  }
  ;

%%

int yyerror(char *msg) {
  printf("ERROR: %s \n",msg);
  return -1;
}

int main(int argc,char** argv) {
  FILE *source_file=fopen(argv[1],"r");
  lexinit(source_file);
  yyparse();
  fclose(source_file);
  return 0;
}
```

```
%{
#include "ast.h"
#include "parser.tab.h"

#include<stdlib.h> // malloc
#include<string.h> // strcpy

void to_parser() {
  yylval.c=(char*)malloc(BUFFER_SIZE*sizeof(char));
  strcpy(yylval.c,yytext);
}
%}

whitespace      [ \t\n]+
variable   [a-z]
constant   [0-9]

%%

{whitespace}     { }
"sin"            { to_parser(); return F; }
"+"              { to_parser(); return L; }
"*"              { to_parser(); return N; }
{variable}       { to_parser(); return V; }
{constant}       { to_parser(); return C; }
.          { return yytext[0]; }

%%

void lexinit(FILE *source) { yyin=source; }
```

Scanner.l

```
#define BUFFER_SIZE 100000

[illegible] {
  [illegible];
  [illegible] * c;
} astNodeType;

#define YYSTYPE astNodeType
```

Fct. 6

```
sdtlc : parser.tab.c lex.yy.c
	gcc $^ -lfl -o $@

parser.tab.c : parser.y
	bison -d $<

lex.yy.c : scanner.l
	flex $<

clean :
	rm -f parser.tab.* lex.yy.* sdtlc

.PHONY: clean
```

makefile

```
#include<iostream>
#include<cmath>
using namespace std;

int f_(double& x, double& x_, double y, double y_) {
  double v0,v1,v2,v3,v4,v5;
  double v0_,v1_,v2_,v3_,v4_,v5_;
#include "test.out"
}

int main() {
  double x=1.,y=2.;
  double x_=1.,y_=0.;
  f_(x,x_,y,y_);
  cout << x_ << endl;

  x=1.,y=2.;
  x_=0.,y_=1.;
  f_(x,x_,y,y_);
  cout << x_ << endl;
  return 0;
}
```

Trick für Tapenade-kompatiblen Code

```
#include<iostream>
#include<cmath>
using namespace std;

int f(double& x, double y) {
  double v0,v1,v2,v3,v4,v5;
#include "test.in"
}

int main() {
  const double h=1e-8;
  double x=1.,y=2.;
  f(x,y);

  double x_=1.+h,y_=2.;
  f(x_,y_);
  cout << (x_-x)/h << endl;

  x_=1.,y_=2.+h;
  f(x_,y_);
  cout << (x_-x)/h << endl;

  return 0;
}
```

Treibe für finite Differenzen