Compiler
Assignment 6
Attribute Grammars and Top-Down Translator

403410033 資工三 曾俊宏

May 24, 2017

1 Question 1

a) S-attributed attribute grammar

production	semantic rules
$S \to L\ R$	S.val = L.val + R.val / R.base;
$R \to .\ L$	R.val = L.val R.base = L.base;
$R \to \epsilon$	R.val = 0; R.base = 0;
$L \to B\ L_s$	L.base = Ls.base; L.val = B.val * L.base + Ls.val; L.base *= 2;
$L_s \to B\ L_{s1}$	Ls.base = Ls1.base; Ls.val = B.val * Ls.base + Ls1.val; Ls.base *= 2;
$L_s \to \epsilon$	Ls.base = 1; Ls.val = 0;
$B \to 0$	B.val = 0;
$B \to 1$	B.val = 1;

b) S-attributed attribute grammar → top-down translator

. and $ are surrounded by single quotation mark due to latex rendering issue.

Structures and function definitions

```
typedef struct node {
    double val;
    int base;
} Node;

Node mknode(int base, double val)
{
    Node *res = (Node *)malloc(sizeof(Node));
    res.base = base;
    res.val = val;

    return res;
}
```

Code

```
double *S()
{
    Node *Lnptr, *Rnptr;
    Node *res;
    switch (token) {
    case 0:
    case 1:
        Lnptr = L();
        Rnptr = R();

        res = Rnptr;
        res.val = Lnptr.val + Rnptr.val / Rnptr.base;
        break;
    case '$':
        res = mknode(1, 0);
        break;
    default:
        error();
    }

    return res.val;
}
```

```
Node *R()
{
    Node *Lnptr;
    Node *res;
    switch (token) {
    case '.':
        match('.');

        Lnptr = L();

        res = Lnptr;
        break;
    case '$':
        res = mknode(1, 0);
        break;
    default:
        error();
    }

    return res;
}
```

```
Node *L()
{
    Node *Bnptr, Lsnptr;
    Node *res;

    switch (token) {
    case 0:
    case 1:
        Bnptr = B();
        Lsnptr = Ls();

        res = Lsnptr;
        res.val = Bnptr.val * Lsnptr.base + Lsnptr.val;
        res.base *= 2;
        break;
    case '$':
        res = mknode(1, 0);
        break;
    default:
        error();
    }

    return res;
}
```

```
Node *Ls()
{
    Node *Bnptr, Lsnptr;
    Node *res;

    switch (token) {
    case 0:
    case 1:
        Bnptr = B();
        Lsnptr = Ls();

        res = Lsnptr;
        res.val = Bnptr.val * Lsnptr.base + Lsnptr.val;
        res.base *= 2;
        break;
    case '$':
        res = mknode(1, 0);
        break;
    default:
        error();
    }

    return res;
}
```

```
Node *B()
{
    Node *res;

    switch (token) {
    default:
    case 0:
        match(0);
        res = mknode(1, 0);
        break;
    case 1:
        match(1);
        res = mknode(1, 1);
        break;
    default:
        error();
    }

    return res;
}
```

c) L-attributed attribute grammar

Assume `side = 1` means left-hand side, and `side = 0` means right-hand side

Assume `2^x` in the code means 2 to the power of x

production	semantic rules
$S \to L\ R$	L.side = 1; R.side = 0; S.val = L.val + R.val;
$R \to .\ L$	R.val = L.val L.side = R.side;
$R \to \epsilon$	R.val = 0;
$L \to B\ L_s$	L.len = 1 + Ls.len; Ls.side = L.side; L.val = (L.side == 1) ? B.val * (2^(L.len - 1)) + Ls.val : B.val / 2 + Ls.val / 2;}
$L_s \to B\ L_{s1}$	Ls.len = 1 + Ls1.len; Ls1.side = Ls.side; Ls.val = (Ls.side == 1) ? B.val * (2^(Ls.len - 1)) + Ls1.val : B.val / 2 + Ls1.val / 2;}
$L_s \to \epsilon$	Ls.len = 0; Ls.val = 0;
$B \to 0$	B.val = 0;
$B \to 1$	B.val = 1;

d) L-attributed attribute grammar → top-down translator

. and $ are surrounded by single quotation mark due to latex rendering issue.

Structures and function definitions

```
typedef struct node {
    double val;
    int base;
} Node;

Node mknode(int base, double val)
{
    Node *res = (Node *)malloc(sizeof(Node));
    res.base = base;
    res.val = val;

    return res;
}

int getPower(int power)
{
    int res = 1;
    for (int i = 1; i <= power; i++)
        res *= 2;
    return res;
}
```

Code

```
double *S()
{
    Node *Lnptr, *Rnptr;
    Node *res;

    switch (token) {
    case 0:
    case 1:
        Lnptr = L(1);
        Rnptr = R(0);

        res = Rnptr;
        res.val = Lnptr.val + Rnptr.val;
        break;
    case '$':
        res = mknode(1, 0);
        break;
    default:
        error();
    }

    return res.val;
}
```

```
Node *R(int side)
{
    Node *Lnptr;
    Node *res;

    switch (token) {
    case '.':
        match('.');
        Lnptr = L(side);

        res = Lnptr;
        break;
    case '$':
        res = mknode(1, 0);
        break;
    default:
        error();
    }

    return res;
}
```

```
Node *L(int side)
{
    Node *Bnptr, *Lsnptr;
    Node *res;

    switch (token) {
    case 0:
    case 1:
        Bnptr = B();
        Lsnptr = Ls(side);

        res = Lsnptr;
        res.len = 1 + Lsnptr.len;
        res.val = (side == 1) ? B.val * get_power(res.len - 1) + Lsnptr.val
                  : B.val / 2 + Lsnptr.val / 2;
        break;
    case '$':
        res = mknode(1, 0);
        break;
    default:
        error();
    }

    return res;
}
```

```
Node *Ls(int side)
{
    Node *Bnptr, *Lsnptr;
    Node *res;

    switch (token) {
    case 0:
    case 1:
        Bnptr = B();
        Lsnptr = Ls(side);

        res = Lsnptr;
        res.len = 1 + Lsnptr.len;
        res.val = (side == 1) ? B.val * get_power(res.len - 1) + Lsnptr.val
                  : B.val / 2 + Lsnptr.val / 2;
        break;
    case '$':
        res = mknode(1, 0);
        break;
    default:
        error();
    }

    return res;
}
```

```
Node *B()
{
    Node *res;

    switch (token) {
    default:
    case 0:
        match(0);
        res = mknode(1, 0);
        break;
    case 1:
        match(1);
        res = mknode(1, 1);
        break;
    default:
        error();
    }

    return res;
}
```