**COMPI HW4**

GUR TELEM (206631848) AND AMIT INBAR (315836569)

## 1. Backpatching

```
superLoop:
        if B = 0 goto labelL2
        ; B is true; start checking L1's items
        if L1[0] = 0 goto next
        if L1[1] = 0 goto next
        ...
        if L1[n] = 0 goto next
        goto doAction
labelL2:
        if L2[0] = 0 goto next
        if L2[1] = 0 goto next
        ...
        if L2[n] = 0 goto next
doAction:
        S1
        goto superLoop
next:
....
```

1.2. **Suggest IR except that the B after the super-loop will only be calculated once.** Note that we were not required to use backpatching so we can fairly easily duplicate the code of S1 to avoid re-calculation of where to jump at the end of S1.

Also note that it wasn't required to give an optimized solution so there's no problem duplicating the code of S1.

```
superLoop:
        if B = 0 goto labelL2
labelL1:
        if L1[0] = 0 goto next
        if L1[1] = 0 goto next
        ...
        if L1[n] = 0 goto next
        S1
        goto labelL1
labelL2:
        if L2[0] = 0 goto next
        if L2[1] = 0 goto next
        ...
        if L2[n] = 0 goto next
        S1
        goto labelL2
next:
....
```

1.3. **Write translation scheme using the backpatching method for the IR you suggested from section 1.1.** Note we used two different markers $ML$ for label and $MG$ for goto.

| production | semantic action |
|---|---|
| `S -> super-loop ML1 B @ L1 MG1 @ ML2 L2 ML3 S1 MG2` | `S.next = freshLabel();`<br>`backpatch(L1.falseList, S.next);`<br>`backpatch(L2.falseList, S.next);`<br>`backpatch(B.falseList, ML2.label);`<br>`backpatch(MG1.gotoList, ML3.label);`<br>`backpatch(MG2.gotoList, ML1.label);`<br>`emit(S.next ':');` |
| `L -> L1 B` | `L.falseList = L1.falseList ++ B.falseList;` |
| `L -> B` | `L.falseList = B.falseList;` |
| `B -> id` | `B.falseList = [nextInstr];emit('if'id.name '=''0''goto _');` |
| `B -> 0` | `B.falseList = [nextInstr];emit('goto _');` |
| `B -> 1` | `B.falseList = []; //We expect later that B will have falseList// Const` |
| $MG \to \varepsilon$ | `MG.gotoList = [nextInstr];`<br>`emit('goto _');` |
| $ML \to \varepsilon$ | `ML.label = freshLabel();`<br>`emit(ML.label ':');` |

## 2. Parsing

**2.1. Is the grammar $LR(0)$?** No. Because when we have $L$ in the stack, and the next token is $[$, we get shift-reduce conflict (shift $[$ to progress towards $L[num]$ or reduce $L$ to $E$).

**2.2. Is the grammar $LR(1)$?** Yes. The table is

| | Action | | | | | Go To | | | |
|---|---|---|---|---|---|---|---|---|---|
| | [ | ] | num | , | $ | S | E | L | EL |
| 0 | s3 | | | | | | 1 | 2 | |
| 1 | | | | | acc | | | | |
| 2 | s4 | | | | r1 | | | | |
| 3 | | r4 | s6 | | | | | | 5 |
| 4 | | | s7 | | | | | | |
| 5 | | s8 | | | | | | | |
| 6 | | | | s9 | | | | | |
| 7 | | s10 | | | | | | | |
| 8 | r3 | | | | r3 | | | | |
| 9 | | r4 | s6 | | | | | | 11 |
| 10 | | | | | r2 | | | | |
| 11 | | r5 | | | | | | | |

2.3. **Parse** `[][num]` **and present the stack states in the process.**

| # | Stack State |
|---|---|
| 1 | 0 |
| 2 | 0 [ 3 |
| 3 | 0 [ 3 EL |
| 4 | 0 [ 3 EL 5 |
| 5 | 0 [ 3 EL 5 ] 8 |
| 6 | 0 L |
| 7 | 0 L 2 |
| 8 | 0 L 2 [ 4 |
| 9 | 0 L 2 [ 4 num 7 |
| 10 | 0 L 2 [ 4 num 7 ] 10 |
| 11 | 0 E |
| 12 | 0 E 1 |
| 13 | accepted |

2.4. **Parse** `[num,num][num]` **and present the stack states in the process.**

| # | Stack State |
|---|---|
| 1 | 0 |
| 2 | 0 [ 3 |
| 3 | 0 [ 3 num 6 |
| 4 | 0 [ 3 num 6 , 9 |
| 5 | 0 [ 3 num 6 , 9 num 6 |
| 6 | error |

## 3. DFA

**3.1. Define the lattice for the domain Seth suggests, what are the elements and what is the relation $\sqsubseteq$ and what is $\sqcup$.** Each node in the lattice will be a binary string in the length of the amount of ascii characters.
In that case, the relation $x \sqsubseteq y \equiv ((x \& y) = x)$ and $x \sqcup y \equiv x \mid y$.
In if you need $x \sqcap y \equiv x \& y$.

**3.2. Define the abstract semantic of the syntax.** Let $y, x, z$ strings.
Let $n_y, n_x, n_z$ the current node (domain) which represents $y, x, z$ respectively.
Let $nn$ the node (domain) of the result.

$$T_{const}\left[?\right] \equiv \left[nn := \{x \mid x \text{ is a letter in } const\}\right]$$
$$T_{y+x}\left[n_y, n_x\right] \equiv \left[nn := n_y \sqcup n_x\right]$$

Regarding $*$:
for $n \leq 0$: $T_{y*0}\left[?\right] \equiv \left[nn := 00\ldots00\right]$
For $n > 0$: $T_{y*0}\left[n_y\right] \equiv \left[nn := n_y\right]$
Regarding $replace$:
For $len\left(x\right) < 1$: $T_{y.replace(x,z)}\left[n_y, n_x, n_z\right] \equiv \left[nn := n_y \sqcup n_z\right]$
For $len\left(x\right) < 1$
For $len\left(x\right) < 1$
For $n_x$
Assume $t$ and $T$ are the matching lower/upper case of each other.

$$T_{y.upper()}\left[n_y\right] \equiv \left[nn := \{T \mid t \in n_y\}\right]$$
$$T_{y.lower()}\left[n_y\right] \equiv \left[nn := \{t \mid T \in n_y\}\right]$$

$$T_{y=x}\left[n_x\right] \equiv \left[n_y := n_x\right]$$

**3.3. Present the analysis on `foo`. Draw the $CFG$.**

**3.4. Is using Seth's idea we can prove the assert at line $7$.** No. Since the domain only track which letters **may** appear in the string but it doesn't mean that the letter **must** appear in the string. Meaning that we know that the assert **might** be true but it isn't promised.

**3.5. Mat's idea is to track the letters that must appear in the string.** The relation $x \sqsubseteq y \equiv ((x|y) = x)$ and $x \sqcup y \equiv x \& y$.
In if you need $x \sqcap y \equiv x \mid y$.

**3.6. Ohh well.** Why isn't the lecture/tutorial show any example of doing something like that :-(

3.7. **Not so well.** Here's a meme