Prova de corretude compilador C para pdl

A sintaxe de PDL é baseada em 2 padrões, sendo eles formulas atômicas e programas atômicos.
Programas atômicos são os menores programas possíveis e formulas atomicas são as menores formulas possíveis.
Nesse algoritmo um programa atômico tem no maximo 1 operando, foi escolhido essa forma por convenção de 3 registradores.
Sendo assim, toda declaração de variavel e função de atribuição com apenas 1 operador, um programa atômico,
logo uma lista de declarações é uma lista de programas atomicos. Por exemplo:

int i -> [int i]
int i = 10 -> declaração :[int i] | função: [i = 10]

Um programa pode ser descrito como uma lista de declarações ou uma lista de funções, sendo que pela recursão toda concatenação dos mesmos é tratada sendo que as declarações são tratadas de forma vista anteriormente.

No caso de termos uma função vai recair em casos mais especificos que vão ser tratados isoladamente.
Como por construção a recursão cuida de sendo tratar o caso mais interno para o mais externo recairemos
sempre sobre o caso mais básico. Temos então os casos das estruturas de repetição, condição e escopo.
A condição de escopo é a mais simples, pois como em PDL é ignorado, um escopo é estruturado toda vez que a recursão entra em uma função,
logo esse caso é ignorado pela conversão.

Logo mostraremos que para um caso generico Sigma de um subconjunto de C todas as transformações feitas são válidas ja que temos prova de que as tsanformações de uma linguagem de programação para pdl está correto comprovadamente, logo iremos mostrar que a nossa conversão de algoritmo equivale a conversão que ja está comprovada.

Caso do Assign:

A conversão de termos Assign de um programa para pdl é feita da seguinte forma:

int i = 1 + 2 + 3 -> declarãção: [int i] | função: [i = 1 + 2 + 3]

Como visto anteriormente toda função Assign utiliza apenas 1 operando para o nosso algoritmo,
logo no caso de mais operandos é utilizado identifiers auxiliares para esse quesito, por exemplo:

int i = 1 + 2 + 3 -> declarãção: [int i, int aux] | função: [aux = 2 + 3, i = 1 + aux]

Logo temos que para o caso simples de 1 operando é trivial verificar que as transformações são identicas, pois é fácil ver que é uma adição direta na arvore.
Para n>1 operandos temos que é facil verificar tambem visto que para uma função generica teremos:

int g = n + (n+1) + (n+2) + (n+3) + ... + (n+m)

logo a transformação ficara declaração ficará:

aux n   = (n + (m-1)) + (n+m)
aux n-1 = (n + (m-2)) + aux n
aux n-2 = (n + (m-3)) + aux n-1
...
aux   = (n+1) + aux 2
g   = (n) + aux

logo temos que esta transformação é trivialmente transformada em:

int g = n + (n+1) + (n+2) + (n+3) + ... + (n+m)

e portanto temos que as 2 transformações são equivalentes.

Caso do IF:

A conversão em PDL de um if simples ocorre da seguinte forma:

```
if(alfa){
        beta;
}
```

vira:

```
(alfa?;beta)
```

No caso de um if else, temos a escolha não deterministica:

```
if(alfa){
        beta;
}else{
        gama;
}
```

Logo:

((alfa?beta)U¬alfa?gama)

O algoritmo faz essa conversão e insere na arvore os casos possiveis como beta ou alfa, gerando ramificações. Assim levando ao caso base de cada ramificação e sendo tratada pela recursão.

Neste compilador a conversão do ifs ocorre da seguinte forma:

Caso Call:

Nesse caso o que estamos fazendo na prática é nada mais nada menos que entrando em uma função normalmente,
logo quando temos a chamada de uma função externa é facil ver que é apenas uma função que será inserida na arvore normalmente.

Caso While:

O * no PDL é representado da forma que pode ser repetido zero ou mais vezes,
logo não temos controle por quantas iterações que serão executadas no programa, no while temos essa abstração.

Logo espera-se que o compilador transforme:

```
While(alfa){
        Beta
}
```

em:

(alfa?;beta)*

Logo como a recursão sempre levará ao caso base, está correto.

Caso doWhile:

Em PDL o doWhile é um while com apenas uma execução antes, assim basta colocar uma iteração executada antes.

```
do{
        alfa
}while(beta);
```

alfa;(beta?alfa)*

Caso FOR:

O for também pode ser tratada da mesma forma que o while mas com algumas ressalvas.

Logo espera-se que:

```
for(int i = 0; i < 10;i = i + 1){
        alfa
}
```

```
(int i;i = 0;(i<10)?;alfa;i = i + 1)*
```