A feature extra implementada foi a adição da possibilidade de se fazer import de funções de outros ficheiros. O primeiro passo foi adicionar uma regra na gramática para se poder fazer parse dos imports em programas plush e criar um Node específico para esta nova regra. A estratégia tomada foi tratar destas adições de novos nós à AST no parser, de forma a preservar-se o estado do type_checker e do compiler. Então, com este objetivo em mente, na função parse_plush, em plush_parser.py, depois de se fazer parse do programa e se obter a AST, é chamada uma função (import_functions), que lê e faz parse do ficheiro do qual se está a fazer o import, coleta as definições das funções importadas e retorna de novo a AST do programa principal, mas com os Nodes de imports substituídos por Nodes com as definições das funções importadas. Ora, aqui surge um desafio pois as funções importadas de um ficheiro podem também usar funções importadas doutros ficheiros, portanto "trazer" essas definições resultaria em erros no type_checker, pois haveria funções que no seu corpo fazem chamadas a outras funções que não estão na AST. Para resolver este problema, faz-se uma chamada recursiva na função import_functions sobre a AST do programa ao qual se estão a importar funções. Desta forma, também se lê e faz-se parse das definições de funções importadas pelos ficheiros que o programa principal está a importar. Com isto surge outro problema, que é "trazer" essas definições de funções de imports com vários níveis de profundidade (função importada de um ficheiro que usa outra função importada de outro ficheiro, etc…). Para este problema, basta acrescentar um parâmetro, na função recursiva, que sirva como acumulador. Este acumulador é passado por todas as chamadas recursivas e vai acumulando todos os Nodes de definições de funções que são necessários. Com isto, no fim acrescenta-se à AST do programa os Nodes guardados no acumulador.

Outra feature feita foi mostrar mensagens de erro com mais detalhes sobre o problema, indicando a linha e onde exatamente na linha está o problema (com um sublinhado). Para fazer isto, foi necessário ler todos os tokens das regras capturadas, de forma a saber as suas posições, e fazer com que os nós guardados na AST, tenham também estes valores (linha, coluna, linha final e coluna final). Também foi acrescentado a estes nós um atributo extra, que guarda a sua representação textual como está no programa lido. Para fazer isto, foi necessário criar um algoritmo de "unparsing" que recebe os tokens das regras e à custa dos valores das posições dos tokens, é reconstruída uma string que corresponde à parte do código que foi capturada por uma dada regra da gramática. Com estes valores das posições dos nós no ficheiro do programa, e da sua representação tal e qual como no programa, no type_checker sempre que se encontra um erro, é mostrada a linha do erro, a representação textual do nó onde se encontrou o erro e um sublinhado, que é obtido com cálculos com os valores das posições inicias e finais dos diferentes tokens que constituem um nó.

Alexandre Figueiredo fc57099