სიმბოლ თეიბლის დიზაინი:

სიმბოლ თეიბლში გვაქვს რეალურად სქოუფების ლისთი, რომელებსაც დინამიურად ვამატებთ და ვშლით იმის მიხედვით პროგრამის რა ნაწიშლი ვიმყოფებით. თითოეულ სქოუფში შენახული გვაქვს მეფის სახით სიმბოლო და მისი მახასიათებლები, ყველაფერი რაც შემდგომ გვესაჭიროება სხვადასხვა ანალიზისთვის. თვითონ სქოუფების დამატება და ამოვარდნა ხდება “SemanticChecking” ლისენერ კლასში სოუფის შესაბამისის წესის (იქნება ეს ფუნქცია, ლეთ სთეითმენთი თუ პროგრამ) ენთერ და ეგზით მეთოდებში. შევქმენით სქოუფ და სიმბოლ კლასები. სქოუფი არის ინტერფეისი. ორი იმპლემენტაციით ჯენერიქ სქოუფი და ფუნქციის სქოუფი (ფუნქციის სქოუფს დამატებით ვიყენებთ იმის გასაგებად თუ რითარნ სთეითმენთი, რომელსაც მივადექით რომელ ფუნქციას ეკუთვნის). სიმბოლ კლასი უბრალოდ სიმბოლოს ატრიბუტების გამაერთიანებელია.

სემანტიკის შემოწმების დიზაინი:

სემანტიკური ანალიზისთვის ვინახავთ ყველა წესისთვის სხვადასხვა ატრიბუტებს. ნაწილობრივ ატრიბუტების გამოანგარიშება გვიწერია თვითონ ანტლერის ფაილში (კონკრეტულად ის ნაწილი, რომლის გამოთვლასაც არ ესაჭიროებოდა სიმბოლ თეიბლი), ხოლო მეორე ნაწილი გვაქვს აღწერილი “SemanticChecking” ლისენერ კლასში. აქვე გვაქვს სიმბოლ თეიბლის კლასი, რომელიც პარალელურად იცვლება. ატრიბუტების გრამატიკით გამოთვლილი ატრიბუტების დახმარებით ვანხორციელებთ უშუალო სემანტიკურ შემოწმებებს. მაგალითად ret_stat ის წესში ვნახულობთ რომელ ფუნქციასაც ეკუთვნის ეს სთეითმენთი მის რითარნ ტიპს (სიმბოლ თეიბლიდან) და შემდგომ მისი შვილის optreturn-ის ატრიბუტს, ექსფრეშენის ტიპს, ვადარებთ ერთმანეთს და შესაბამის ერორს ვლოგავთ. ერორ ჰენდლინგისთვის გვაქ ერორების ინამი და მეთოდი, ერთგვარი ასერტი, რომელიც იღებს ქონდიშენს (რომლის დაკმაყოფილების შემთხვევაშიც უნდა დალოგოს ერორ) , ასევე ერორის ტიპს და დამატებით მონაცემს ისეთი როგორიც ხაზია, რომელზეც ეს ერორი მოხდა. დამატებით ვიყენებთ თაიფ კლას, სხვადასხვა ბეისიქ და შემდგომ განსაზღვრული ტიპებზე ინფორმაციის შესანახად და ადვილად შესადარებლად.

IR:

სამწუხაროა :(

ბაგები და დაუსრულებელი მახასიათებლები:

1) ერორის ქოლუმნს არ ვითვლით მხოლოდ ხაზს, რომელზეც ეს ერორი მოხდა.
2) IR არ არის ბოლომდე გამართული