Εργασία στους Μεταγλωττιστές

Δεύτερο μέρος

Οικονομικό Πανεπιστήμιο Αθηνών

Τμήμα Πληροφορικής

Διδάσκων Βάνα Καλογεράκη

Κατωπόδης Αντώνιος | p3140076
Κουτσοπούλου Αθανασία Μαρία | p3140092
Χαϊκάλης Γεώργιος | p3140218
Χασακής Διονύσιος | p3140219

Εισαγωγή

Το report αυτό, ουσιαστικά αφορά το τέταρτο και τελευταίο μέρος της εργασίας, τη σημασιολογική ανάλυση. Συγκεκριμένα, θα μιλήσουμε για τους διάφορους ελέγχους που υλοποιήσαμε, και θα επεξηγήσουμε την λογική του κώδικα για τον εκάστοτε έλεγχο.

Αρχικά, να αναφέρουμε εδώ, ότι δημιουργήσαμε δύο visitors, κυρίως λόγω της αδυναμίας να χειριστούμε τα function calls μονάχα με ένα visitor, και αυτό λόγω του ότι στην minipython, όπως και στην python άλλωστε, είναι δυνατόν να καλείτε μία συνάρτηση, πριν την γραμμή όπου βρίσκεται η δήλωση και η υλοποίησή της.

Έτσι λοιπόν σε γενικές γραμμές, **κατά τον πρώτο visitor**, αποθηκεύουμε όλα τα σύμβολα που συναντάμε στο symbol table, μαζί με όλες τις πληροφορίες τους, πχ τρέχον τύπος μεταβλητής, προκειμένου να κάνουμε διάφορους ελέγχους ορθότητας που αφορούν τους τύπους. Αν συναντήσουμε κάποιο σύμβολο για το οποίο δεν μπορούμε να αποφανθούμε τη συγκεκριμένη στιγμή τον τύπο του, το αποθηκεύουμε με τύπο «undef», και στην συνέχεια όταν τελικά είμαστε σε θέση να προσδιορίσουμε τον τύπο τον ανανεώνουμε. Πχ κατά τον πρώτο visitor, είναι πολύ πιθανόν, να έχουμε μία μεταβλητή που αρχικοποιείται από το return μιας συνάρτησης, και το οποίο δεν είμαστε σε θέση να προσδιορίσουμε άμεσα, επομένως το σύμβολο αποθηκεύεται με τύπο «undef», μέχρι να βρεθούμε στον second visitor, όπου πλέον ανανεώνουμε το type με το πραγματικό return type της προκειμένης περίπτωσης. **Στον δεύτερο visitor**, έχουμε τον έλεγχο του function call, δηλαδή αν καλείται συνάρτηση που δεν έχει οριστεί. Ακόμη, ανατρέχουμε σε όλο το file, τσεκάροντας εκ νέου αν υπάρχει κάποιο type conflict ή invalid type use, μιας που νωρίτερα είχαμε ελλιπής πληροφορία όπου υπήρχε function call.

Επιπλέον, εκτός από τους δύο visitors, δημιουργήσαμε και την βοηθητική κλάση Utils, η οποία περιέχει μεθόδους που καλούνται συχνά και από τους δύο visitors, έτσι ώστε να κάνουμε πιο ευανάγνωστο τον κώδικα και να μειώσουμε τα duplications κώδικα, που υπήρχαν διαφορετικά.

Ακόμη έχουμε δημιουργήσει την κλάση RootSymbolTable που αντιπροσωπεύει το SymbolTable μας, και αποτελείται από ένα HashMap, όπου αποθηκεύονται οι μεταβλητές, και ένα ArrayList, που αποθηκεύονται οι μέθοδοι.

Τέλος, έχουμε δημιουργήσει και 2 βοηθητικά μοντέλα τα Function, Variable ώστε να διευκολύνουμε τους ελέγχους στους visitors.

Σημείωση: Και στους 2 visitors, όπου μέσα στην in μέθοδο επισκεπτόμαστε εμείς άλλους κόμβους, έχουμε κάνει override και την case ώστε να αποφύγουμε διπλούς ελέγχους. Όπου δεν έχουμε κάνει apply σε άλλο κόμβο, δεν την έχουμε κάνει override

Οι υλοποιημένοι έλεγχοι

1. Χρήση μη δηλωμένης μεταβλητής

Ο έλεγχος αυτός γίνεται ουσιαστικά εξ ολοκλήρου μέσα στον **first visitor**. Ελέγχουμε απλά εάν το νέο σύμβολο – μεταβλητή, το έχουμε ξανασυναντήσει, ελέγχοντας εάν υπάρχει αποθηκευμένο μέσα στο *symbol table*. Έχουμε υλοποιήσει κα την δυνατότητα έχουμε και τοπικές μεταβλητές σε μια μέθοδο, το οποίο όμως τελικά δεν χρειαζόταν λόγω της μορφής του BNF, αφού σε μια συνάρτηση μπορούμε να έχουμε ως *body* μόνο ένα *statement*.

Ο έλεγχος ουσιαστικά γίνεται σε επίπεδο *identifier*, άρα το κομμάτι του κώδικα που αφορά αυτόν τον έλεγχο είναι όλες οι μέθοδοι όπου υπάρχει *identifier*, μέσα στο *node* που δέχονται ως είσοδο. Δηλαδή οι **inAForStatement**, **inAEqualsStatement**, **inAMinusEqualsStatement**, **inADivEqualsStatement**, **inAArrayStatement**, **inAIdentifierExpression** και **inAExpInBracketsExpression**, όπου κάνουμε *override*. Μέσα σε αυτές κάνουμε ότι ακριβώς αναφέραμε άνω. Προκειμένου να κάνουμε τον έλεγχο, εάν έχει αποθηκευτεί το σύμβολο στο πίνακα συμβόλων, και άρα έχει οριστεί, χρησιμοποιούμε την βοηθητική μέθοδο **getVariableFromId** της *Utils* κλάσης.

Συνοπτικά εκτός από το αν έχουν οριστεί ξανά η μεταβλητές κάνουμε επιπλέον τους παρακάτω ελέγχους στις προαναφερόμενες μεθόδους.

Στην **inAForStatement** (for id1 in id2: stm) γίνεται επιπλέον έλεγχος εάν η μεταβλητή του *"in"* συμβολίζει πίνακα, διότι διαφορετικά θα πρέπει να εμφανίσουμε το αντίστοιχο *error*. Επιπλέον, το id1 μπορεί να ορίζεται τώρα πρώτη φορά ή να έχει οριστεί νωρίτερα, αλλά το id2 πρέπει οπωσδήποτε να έχει οριστεί νωρίτερα.

Στις **inAMinusEqualsStatement** και **inADivEqualsStatement** πρέπει να χρησιμοποιούμαι υποχρεωτικά int values.

Όπως επίσης στις **inAArrayStatement** και **inAExpInBracketsExpression** ελέγχουμε εάν το *expression* που δίνεται ως *index* είναι *string*, ή αν η μεταβλητή που χρησιμοποιείτε δεν είναι *array*, και εμφανίζουμε το αντίστοιχο *error*.

2. Κλήση μη δηλωμένης συνάρτησης

Επειδή λοιπόν, όπως αναφέραμε και άνω, είναι δυνατόν να έχουμε κλήση μιας συνάρτησης που ακόμη δεν έχουμε συναντήσει, ο έλεγχος κλήσης δηλωμένης ή μη δηλωμένης συνάρτησης γίνεται στον **second visitor**. Διαφορετικά, όταν θα υπήρχε μια τέτοια κλήση, ο **first visitor** θα πέταγε *error* ότι η συνάρτηση δεν έχει δηλωθεί, αφού δεν θα είχε κάνει ακόμη visit τον κόμβο της δήλωσης της συνάρτησης και έτσι η συνάρτηση αυτή δεν θα είχε ακόμη αποθηκευτεί στον πίνακα συμβόλων μας. Κάτι τέτοιο όμως σαφώς θα ήταν λάθος (να χτύπαγε compilation error). Για αυτό λοιπόν ο έλεγχος γίνεται στο **second visitor**, που από την στιγμή που καλείται μετά τον first, όλες

οι συναρτήσεις που δηλώνονται – υλοποιούνται μέσα στο file που θέλουμε να κάνουμε compile, έχουν επισκεφτεί από τον first, και επομένως έχουν αποθηκευτεί στον πίνακα συμβόλων, εφόσον πρώτα όμως ελεγχθεί η εγκυρότητα της δήλωσης της συνάρτησης κάθε φορά. Μια δήλωση δεν πρέπει να γίνει αποδεχτή αν δεν αποτελεί πχ σωστό τρόπο overload, αν δηλαδή έχουμε f(x, y) και μετά προσπαθήσουμε να δηλώσουμε f(x, y, z = 5), τότε ο compiler μας θα πρέπει να εμφανίσει error διότι οι υπογραφές των function δεν είναι επιτρεπτό overload, και έτσι και κάνει.

Ο κώδικας που αφορά αυτό το κομμάτι που ελέγχουμε αν η γίνεται κλήση μη δηλωμένης συνάρτησης βρίσκεται στην **inAFunctionExpression** και στην **inAFunctionStatement** μέθοδο που κάνουμε override. Και στις δύο ελέγχουμε κατά πόσο έχει δηλωθεί στο αρχείο μας, η αντίστοιχη function που καλείται. Ελέγχουμε αρχικά αν υπάρχει κάποια ορισμένη μέθοδο με το ίδιο όνομα, διαφορετικά εμφανίζουμε το error. Αν υπάρχει μία ή περισσότερες μέθοδοι αποθηκευμένες με το ίδιο όνομα, τότε για κάθε μία από αυτές ελέγχουμε για κάθε δυνατό τρόπο κλήσης της, δηλαδή αν έχει παραμέτρους με default values ελέγχουμε κάθε εναλλακτικό αποδεκτό τρόπο κλήσης, αν αντιστοιχεί σε αυτόν που συναντήσαμε. Και με το αντιστοιχεί εννοούμε αν έχει ίδιο αριθμό παραμέτρων. Διαφορετικά, αν δεν βρεθεί καμία αντιστοίχιση η κλήση δεν είναι αποδεκτή και εμφανίζουμε το αντίστοιχο λάθος. Στο επόμενο έλεγχο, αναλύουμε αναλυτικότερα τον έλεγχο σχετικά με τα ορίσματα.

3. Λάθος ορισμός ορισμάτων σε κλήση συνάρτησης

Εφόσον έχουμε διασφαλίσει πως το όνομα της συνάρτησης της οποίας καλούμε υπάρχει, θα πρέπει να ελέγξουμε αν περνάμε και τα κατάλληλα ορίσματα τα οποία απαιτούνται. Αρχικά ας παρατηρήσουμε το γεγονός πως πρέπει να πάρουμε μία λίστα με όλες τις συναρτήσεις με το ίδιο όνομα. Για κάθε μία από αυτές, θα πρέπει να πάρουμε το εύρος ορισμάτων κλήσης(π.χ. η def f(x,y=2,z=3) έχει εύρος [1,3]) και να ελέγξουμε αν ο αριθμός των παραμέτρων που περνάμε υπάρχει στο εύρος ορισμάτων της συνάρτησης. Με αυτό το τρόπο για κάθε μια συνάρτηση παίρνουμε όλες τις δυνατές κλήσεις της(με τον αριθμό παραμέτρων) και ελέγχουμε τον αριθμό των παραμέτρων, μέχρι να βρούμε αυτή που καλείται. Η κύρια συνάρτηση που το διαχειρίζεται αυτό είναι η **checkLegitFunction** στην κλάση *Utils*. Οι έλεγχοι για τα function calls γίνονται στη περίπτωση του FunctionStatement και του FunctionExpression, άρα στις μεθόδους **inAFunctionStatement**, **inAFunctionExpression** αντίστοιχα.

4. Χρήση string μεταβλητής ως ακέραιο

Εδώ διακρίνουμε 2 περιπτώσεις. Πρώτον, στην πρόσθεση, επιτρέπουμε να γίνονται πράξεις μεταξύ *int* και *int* είτε string και string. Στη περίπτωση αυτή λοιπόν θα πρέπει

να ελέγξουμε αν 2 τύποι που πάνε να προστεθούν είναι διαφορετικοί και όχι *"UNDEF"*(καθώς αυτό θα σημαίνει πως δεν έχουμε ακόμα τις κατάλληλες πληροφορίες για να κρίνουμε, π.χ. σε μια κλήση μιας συνάρτησης, θα αποφανθούμε στο 2ο επισκέπτη). Αρχικά παίρνουμε τον τύπο των 2 expressions τα οποία προστίθενται. Για να το κάνουμε αυτό, έχουμε δημιουργήσει μία μέθοδο στην κλάση *Utils*, την **getExpressionsType** η οποία θα γύρισει:

- Αν το expression είναι value, τον τύπο του
- Αν είναι identifier τότε να επιστρέψει τον τύπο του, αλλιώς *"UNDEF"* αν δεν βρέθηκε
- Αν είναι πρόσθεση τότε να καλέσει την ίδια συνάρτηση αναδρομικά μέχρι να αποφανθεί για το αποτέλεσμα με την ίδια λογική που περιγράψαμε πιο πάνω και αν βρει conflict γυρνάει *"TYPECONFLICT"*
- Αν είναι αφαίρεση, πολλαπλασιασμός ή διαίρεση να καλεί αναδρομικά την ίδια συνάρτηση μέχρι να καταλήξει σε ένα σημείο ώστε να μπορεί να εξάγει κάποιο συμπέρασμα. Τότε αν τα στοιχεία που προστίθενται δεν είναι και τα 2 *int* τότε γυρνάει *"TYPECONFLICT"*
- Αν είναι συνάρτηση να καλέσει την **checkLegitFunction** της κλάσης *Utils* για να δει αν μπορεί να αποφανθεί για τον τύπο που γυρνάει η συνάρτηση. Αυτό ουσιαστικά έχει νόημα μόνο στο 2ο επισκέπτη που θα έχουμε τη λίστα με τις συναρτήσεις και τον τύπο που γυρίζουν αυτές.
- Σε περίπτωση που μιλάμε για ένα array, τότε επιστρέφει *"ARR"*

Έτσι έχοντας πάρει τον τύπο του κάθε στοιχείου που προστίθεται, ελέγχουμε αν κάποιο είναι *"UNDEF"* ή *"ARR"*. Στη περίπτωση αυτή, όμως, δε μπορούμε να αποφανθούμε. Αν περάσει από αυτό τον έλεγχο ελέγχουμε αν είναι και τα 2 *int* ή *string* και αν δεν είναι εμφανίζουμε το αντίστοιχο μήνυμα λάθους. Η υλοποίηση του ελέγχου αυτού γίνεται στην συνάρτηση **inAAdditionExpression**. Η δεύτερη περίπτωση είναι να έχουμε κάποια από τις υπόλοιπες αριθμητικές πράξεις (αφαίρεση, πολλαπλασιασμός, διαίρεση, -=, /=), όπου εδώ, και τα 2 μέρη θα πρέπει απαραίτητα να είναι int. Για την αφαίρεση, το πολλαπλασιασμό και τη διαίρεση, ελέγχουμε στην **inAStringValue**, αν ο πατέρας είναι value και εν συνεχεία αν ο πατέρας του value ήταν κάτι από αυτά τα 3 expressions. Στη περίπτωση αυτή, δε μπορούμε να έχουμε κάποιο string σε κανένα από τα 2 μέρη και έτσι τυπώνουμε το αντίστοιχο σφάλμα. Για τους operators -=, /=, επισκεπτόμαστε το δεξί μέλος και κοιτάμε αν εντοπίσαμε σε αυτό σφάλματα. Αν δεν εντοπίσαμε, τότε παίρνουμε τον τύπο του expression στα δεξιά μέσω της **getExpressionsType** και ελέγχουμε αν ο τύπος αυτός, όσο και ο τύπος της μεταβλητής αριστερά (εφόσον αυτή έχει οριστεί, αλλιώς τυπώνεται σφάλμα) είναι *int*.

5. Λάθος τρόπος χρήσης συνάρτησης

Όπως αναφέρθηκε και προηγουμένως, στον 1ο επισκέπτη για κάθε μια συνάρτηση που διαβάζουμε εντοπίζουμε αν αυτή έχει return, μέσω της **inAReturnStatement**, και ορίζουμε στο function instance το return type της ως το αποτέλεσμα της **getExpressionsType** με είσοδο το expression που χρησιμοποιείται στο return statement. Έτσι στον 2ο επισκέπτη θα μπορέσουμε να ελέγξουμε αν μια συνάρτηση χρησιμοποιείται σωστά ή λάθος. Στις 2 μεθόδους-κόμβους για τη κλήση συναρτήσεων, δηλαδή την **inAFunctionStatement** και **inAFunctionExpression** βρίσκουμε το function instance μέσω της **checkLegitFunction**, η οποία επιστρέφει και το instance της συνάρτησης που χρησιμοποιείται. Στη συνέχεια ελέγχουμε το return type της συνάρτησης, εφόσον αυτό υπάρχει καθώς μια συνάρτηση μπορεί να μην έχει return, και σε περίπτωση που ο πατέρας του κόμβου στον οποίο βρισκόμαστε (της κλήσης της συνάρτησης) είναι αριθμητική πράξη (εκτός της πρόσθεσης, είπαμε επιτρέπεται και η πρόσθεση μεταξύ *int*, για το σκοπό αυτό κάνουμε πάλι override την **inAAdditionExpression** καθώς θέλουμε να ξανά ελέγξουμε ότι και τα 2 μέρη είναι *int* ή string) ή operator -=, /= ή κελί πίνακα, τότε αν το return type της συνάρτησης είναι *int*, τυπώνουμε το αντίστοιχο μήνυμα σφάλματος.

6. Επανάληψη δήλωσης συνάρτησης με τον ίδιο αριθμό ορισμάτων

Για να αποτρέψουμε τη δήλωση μιας συνάρτησης με ίδιο όνομα και ίδιο αριθμό παραμέτρων, έχουμε ορίσει τη κλάση Function η οποία έχει βοηθητικά πεδία, όπως όνομα, λίστα παραμέτρων, τύπος γυρίσματος, πόσες default και non default παραμέτρους έχει κλπ. Στην αρχή του **inAFunction** αρχικοποιούμε κάποιες βοηθητικές μεταβλητές για τη συνάρτηση και στην **outAFunction** τις κάνουμε πάλι default.

- Αρχικά παίρνουμε το τρέχον function που διαβάζουμε και βρίσκουμε ποιες παραμέτρους έχει(αριθμό defaults και non defaults) καθώς και το όνομα της. Για να το κάνουμε αυτό, επισκεπτόμαστε τον κόμβο παραμέτρων.
- Έτσι, η **inAArgument**, αναλαμβάνει πλέον το καθήκον να αναλύσει και να προσθέσει στη λίστα παραμέτρων της συνάρτησης την συγκεκριμένη παράμετρο.
    - Έτσι, αν το assignValue είναι *null*, τότε δε μπορούμε να ξέρουμε το τύπο της παραμέτρου και απλά το προσθέτουμε ως nondefault παράμετρο με τύπο *“UNDEF”*.
    - Αν το *assignValue* δεν είναι *null*(σημαίνει ότι έχουμε κάποια default τιμή), τότε επισκεπτόμαστε το κόμβο αυτό ώστε να βρούμε το τύπο της και να το ορίσουμε στη παράμετρο.

        - Έτσι στη μέθοδο **inAAssignValue**, επισκεπτόμαστε τον επόμενο κόμβο που είναι είτε *string* value είτε *int* value και ορίζουμε το αντίστοιχο type.

- Επαναλαμβάνουμε την ίδια ακριβώς δουλεία στην **inAArgument** που εκτελέσαμε για τη πρώτη παράμετρο και για τις υπόλοιπες παραμέτρους, που βρίσκονται στην *AParameters*. Τη δουλειά αυτή την αναλαμβάνει η **inAParameters**.

- Έπειτα καλούμε τη συνάρτηση **hasAlreadyBeenDefined** ώστε να εντοπίσουμε αν η συνάρτηση υπάρχει ήδη.

    - Στην **hasAlreadyBeenDefined** παίρνουμε μια λίστα με όλα τα functions που έχουμε ήδη αποθηκεύσει στο *symbolTable* τα οποία έχουν ίδιο όνομα με τη συνάρτηση που ψάχνουμε. Αν δε βρει καμία γυρνάει -1. Στη συνέχεια παίρνουμε το parameter range(δηλ. το σύνολο των αριθμών παραμέτρων με τον οποίο μπορεί να καλεστεί μία συνάρτηση, π.χ. η συνάρτηση f(x,y,a=1,b=2) έχει εύρος [2,4]) της συνάρτησης που αναζητούμε και για κάθε μία από τις συναρτήσεις της λίστας, παίρνουμε το parameters range της και ελέγχουμε αν ο έχουν έστω και ένα κοινό στοιχείο(το range της συνάρτησης που διαβάζουμε με την i).

- Αν η **hasAlreadyBeenDefined** γυρίσει -1, πράγμα που σημαίνει ότι δε βρέθηκε κάποια συνάρτηση με ίδια ορίσματα στο symbol table τότε τη προσθέτουμε σε αυτόν, αλλιώς τυπώνουμε αντίστοιχο σφάλμα.