Στατική Σημασιολογία του Συστήματος Τύπων της C
(Μια Τυπική Μαθηματική Περιγραφή)

Νικόλαος Σ. Παπασπύρου
(nickie@softlab.ntua.gr)

Εθνικό Μετσόβιο Πολυτεχνείο
Τμήμα Ηλεκτρολόγων Μηχ. και Μηχ. Υπολογιστών
Τομέας Πληροφορικής, Εργαστήριο Λογισμικού
Πολυτεχνειούπολη, 15780 Ζωγράφου, Αθήνα

Περίληψη

Μια γλώσσα προγραμματισμού χαρακτηρίζεται από τη σύνταξη και τη σημασιολογία των προγραμμάτων. Η τυπική περιγραφή της σημασιολογίας, σε αντίθεση με αυτή της σύνταξης, είναι ένα πρόβλημα που έχει απασχολήσει τους ερευνητές για πολλά χρόνια χωρίς να έχει ακόμα βρεθεί ικανοποιητική και γενικά αποδεκτή λύση. Η εφαρμογή των θεωριών και των τεχνικών που έχουν αναπτυχθεί συναντά σοβαρότατα προβλήματα όταν επιχειρείται η περιγραφή πραγματικών γλωσσών προγραμματισμού. Στην εργασία αυτή προτείνεται μια περιγραφή της στατικής σημασιολογίας του συστήματος τύπων της δημοφιλούς γλώσσας προγραμματισμού C. Η στατική σημασιολογία αποβλέπει αφενός στην περιγραφή της στατικής ερμηνείας των δηλώσεων, αφετέρου στο συσχετισμό των εκφράσεων με τύπους και τον έλεγχο τύπων των προγραμμάτων. Για την τυπική περιγραφή χρησιμοποιείται η μαθηματική προσέγγιση (denotational semantics) και ένα σύνολο κανόνων συναγωγής.

1 Εισαγωγή

Η γλώσσα C είναι μια ιδιαίτερα δημοφιλής γλώσσα προγραμματισμού γενικού σκοπού. Σχεδιάστηκε αρχικά από τον Dennis Ritchie το 1972 [1, 2], από τον οποίο έγινε και η πρώτη υλοποίηση. Έκτοτε, η C έχει χρησιμοποιηθεί για τον προγραμματισμό μιας ευρείας περιοχής εφαρμογών και μεταγλωττιστές της είναι διαθέσιμοι σχεδόν για κάθε υπολογιστικό σύστημα. Παρότι η γλώσσα επιτρέπει την ανάπτυξη μη μεταφέρσιμων εφαρμογών, τα προγράμματα σε C είναι συνήθως μεταφέρσιμα, ενίοτε με μικρές τροποποιήσεις. Η C δεν είναι μια γλώσσα προγραμματισμού ιδιαίτερα υψηλού επιπέδου. Τα κύρια χαρακτηριστικά της είναι η οικονομία στην έκφραση, που πραγματοποιείται μέσω της λακωνικής σύνταξής της, ένα μεγάλο σύνολο από τελεστές και τύπους δεδομένων, καθώς και το γεγονός ότι παρέχει άμεση πρόσβαση στο εσωτερικό του υπολογιστή, με αποτέλεσμα τα προγράμματα σε C να έχουν συνήθως πολύ καλές επιδόσεις.

Το 1990 ολοκληρώθηκε η προσπάθεια προτυποποίησης της C η οποία κατέληξε στην έκδοση του προτύπου ANSI/ISO–C [3], διαφόρων συνοδευτικών εγγράφων [4] καθώς και μεταγενέστερων διορθώσεων [5]. Τα έγγραφα αυτά αποτελούν σήμερα το κοινώς αποδεκτό πρότυπο για τη C και σημείο αναφοράς για κατασκευαστές και χρήστες μεταγλωττιστών και άλλων εργαλείων. Το μεγαλύτερο μέρος του προτύπου είναι γραμμένο σε φυσική γλώσσα,[1] με αποτέλεσμα αρκετές φορές να δημιουργούνται αμφιβολίες και προβλήματα παρερμηνείας ως προς τη σημασιολογία της γλώσσας C. Τέτοιες παρερμηνείες είναι αρκετά συχνές, κυρίως λόγω της πολυπλοκότητας της γλώσσας. Αρκετές φορές είναι δύσκολο να πει κανείς με βεβαιότητα αν ένα πρόγραμμα C είναι σύμφωνο με το πρότυπο (και κατά συνέπεια μεταφέρσιμο) και ποια θα είναι τα αποτελέσματα της εκτέλεσής του.

Τα τελευταία χρόνια ιδιαίτερη έμφαση δίνεται στη χρήση *τυπικών μεθόδων* (formal methods) στην επιστήμη υπολογιστών, προκειμένου να αποφεύγονται διφορούμενα και παρερμηνείες από τη χρήση

[1] Μόνο η περιγραφή της σύνταξης της C είναι γραμμένη σε τυπική μορφή.

της φυσικής γλώσσας ή άλλων άτυπων μεθόδων. Για την τυπική περιγραφή της σημασιολογίας γλωσσών προγραμματισμού έχουν κατά καιρούς προταθεί τρεις κύριες προσεγγίσεις: η *λειτουργική* (operational semantics), η *μαθηματική* (denotational or mathematical semantics) και η *αξιωματική* (axiomatic semantics).

Η μαθηματική προσέγγιση, η οποία θα χρησιμοποιηθεί σ' αυτή την εργασία, είναι ένας φορμαλισμός που εισήγαγαν οι Dana Scott και Christopher Strachey, στο τέλος της δεκαετίας του 1960. Από τότε έχει εκτενώς μελετηθεί από διακεκριμένους ερευνητές και έχει χρησιμοποιηθεί ως μέθοδος σημασιολογικής ανάλυσης, περιγραφής, αξιολόγησης αλλά και υλοποίησης διαφόρων γλωσσών προγραμματισμού. Εισαγωγές στη μαθηματική προσέγγιση καθώς και χρήσιμη βιβλιογραφία μπορούν να βρεθούν στα [6] και [7]. Για εκτενέστερη περιγραφή της θεωρίας και των τεχνικών που έχουν αναπτυχθεί, ο αναγνώστης παραπέμπεται στα [8], [9] και [10]. Μια πληρέστερη και πιο πρόσφατη εισαγωγή στη σημασιολογία γλώσσών προγραμματισμού, με ιδιαίτερη έμφαση στη μαθηματική προσέγγιση και τους τύπους δεδομένων, δίνεται στο [11].

Σύμφωνα με τη μαθηματική προσέγγιση, η σημασιολογία των γλωσσών προγραμματισμού περιγράφεται με την απόδοση μαθηματικών *ερμηνειών* (denotations) στα προγράμματα και τα τμήματα αυτών. Οι ερμηνείες αυτές είναι συνήθως συναρτήσεις ανώτερου βαθμού πάνω σε *πεδία Scott* (Scott domains) και περιγράφονται με τη βοήθεια κατάλληλων τυπικών μετα−γλωσσών, που συνήθως είναι παραλλαγές του *λογισμού*−λ (λ−calculus).

Εξέχουσα θέση στην περιγραφή μιας σύγχρονης γλώσσας προγραμματισμού κατέχει συνήθως το σύστημα τύπων της. Η εργασία αυτή ασχολείται με την τυπική περιγραφή της στατικής σημασιολογίας του συστήματος τύπων της γλώσσας C, χρησιμοποιώντας τη μαθηματική προσέγγιση. Λαμβανομένων υπόψη της πολυπλοκότητας του συστήματος τύπων της C και του περιορισμένου χώρου, σκοπός της εργασίας δεν είναι να φέρει στο φως όλες τις λεπτομέρειες της τυπικής περιγραφής αλλά να την σκιαγραφήσει προκειμένου να αναδείξει την εφικτότητα της προσέγγισης. Για την πλήρη τυπική περιγραφή της σημασιολογίας του συστήματος τύπων της C, ο αναγνώστης παραπέμπεται στο [12].

Η προσπάθεια αυτή αποτελεί τη βάση μιας γενικότερης προσπάθειας για την πλήρη τυπική περιγραφή της σημασιολογίας της C. Ανάλογη προσπάθεια περιγράφεται στο [13], βασίζεται όμως στο ανεπίσημο πρότυπο της γλώσσας C που βρισκόταν εν ισχύ εκείνη την εποχή και παρουσιάζει διαφορές ως προς το σημερινό επίσημο πρότυπο. Η παρούσα εργασία ασχολείται πληρέστερα με τη στατική σημασιολογία του συστήματος τύπων της C, αποδίδοντας ερμηνεία σε όλους τους τύπους δεδομένων της C, συμπεριλαμβανομένων των τύπων συναρτήσεων και των ημιτελών τύπων. Επίσης, ορίζει ένα τυπικό σύστημα κανόνων συναγωγής για τον έλεγχο τύπων των προγραμμάτων σε C.

Η εργασία αυτή είναι δομημένη ως εξής. Στην ενότητα 2 γίνεται μια συνοπτική εισαγωγή στο σύστημα τύπων της C και περιγράφεται τυπικά η σύνταξη των δηλώσεων. Η ενότητα 3 αποτελεί τον πυρήνα της εργασίας. Εκεί παρουσιάζεται η στατική σημασιολογία του συστήματος τύπων, που διακρίνεται στην απόδοση στατικής ερμηνείας και στον έλεγχο τύπων. Τέλος, η ενότητα 4 περιέχει τη σύνοψη της εργασίας και τις κατευθύνσεις για μελλοντική έρευνα.

2 Το σύστημα τύπων της C

Το *σύστημα τύπων* (type system) μιας γλώσσας προγραμματισμού αναλαμβάνει τη διαχείριση και τον έλεγχο των τύπων των μεταβλητών και γενικά των εκφράσεων ενός προγράμματος [14, 15]. Αποσκοπεί κυρίως στην ελαχιστοποίηση των σφαλμάτων που μπορούν να προκύψουν κατά την εκτέλεση των προγραμμάτων. Η τυπική περιγραφή του συστήματος τύπων μιας γλώσσας προγραμματισμού επιτρέπει τη διατύπωση και την απόδειξη θεωρημάτων, σχετικών με τη συμπεριφορά των προγραμμάτων, και είναι πολύτιμο εφόδιο για τους μελετητές της γλώσσας και τους κατασκευαστές υλοποιήσεων.

Πίνακας 1: Σύνταξη των δηλώσεων της ANSI–C.

declaration–list ::= ϵ | *declaration declaration–list*
declaration ::= *declaration–specifiers init–declarator–list* `;`
declaration–specifiers ::= *storage–class–specifier type–qualifier type–specifier*
storage–class–specifier ::= ϵ | `typedef` | ...
type–qualifier ::= ϵ | `const` | `volatile` | `const volatile`
type–specifier ::= `void` | `char` | `signed char` | `unsigned char` | `short int` | `unsigned short int`
| `int` | `signed int` | `unsigned int` | `long int` | `unsigned long int` | `float`
| `double` | `long double` | *struct–specifier* | *union–specifier* | *enum–specifier* | *typedef–name*
typedef–name ::= *I*
init–declarator–list ::= ϵ | *init–declarator init–declarator–list*
init–declarator ::= *declarator* | ...
struct–specifier ::= `struct` *I* `{` *struct–declaration–list* `}` | `struct` `{` *struct–declaration–list* `}` | `struct` *I*
union–specifier ::= `union` *I* `{` *struct–declaration–list* `}` | `union` `{` *struct–declaration–list* `}` | `union` *I*
struct–declaration–list ::= *struct–declaration* | *struct–declaration struct–declaration–list*
struct–declaration ::= *struct–specifiers struct–declarator–list* `;`
struct–specifiers ::= *type–qualifier type–specifier*
struct–declarator–list ::= *struct–declarator* | *struct–declarator struct–declarator–list*
struct–declarator ::= *declarator* | *declarator* `:` *n*
enum–specifier ::= `enum` *I* `{` *enumerator–list* `}` | `enum` `{` *enumerator–list* `}` | `enum` *I*
enumerator–list ::= *enumerator* | *enumerator enumerator–list*
enumerator ::= *I* | ...
declarator ::= *I* | *declarator* `[` *n* `]` | *declarator* `[ ]`
| *declarator* `(` *parameter–type–list* `)` | `*` *type–qualifier declarator*
parameter–type–list ::= ϵ | *parameter–type–list* `...` | *parameter–declaration parameter–type–list*
parameter–declaration ::= *declaration–specifiers declarator* | *declaration–specifiers abstract–declarator*
abstract–declarator ::= ϵ | *abstract–declarator* `[` *n* `]` | *abstract–declarator* `[ ]`
| *abstract–declarator* `(` *parameter–type–list* `)` | `*` *type–qualifier abstract–declarator*
type–name ::= *type–qualifier type–specifier abstract–declarator*

Το σύστημα τύπων της C περιγράφεται σε φυσική γλώσσα στο πρότυπο. Πρώτα δίνεται μια άτυπη περιγραφή των τύπων δεδομένων [3, §§6.1.2.6, 6.1.2.6] και στη συνέχεια περιγράφονται οι *δηλώσεις* (declarations) [3, §6.5]. Η περιγραφή της διαδικασίας ελέγχου τύπων βρίσκεται επιμερισμένη στην περιγραφή των εκφράσεων και των εντολών [3, §§6.3, 6.6]. Στη συνέχεια της εργασίας γίνεται η υπόθεση ότι ο αναγνώστης είναι εξοικειωμένος με τη γλώσσα προγραμματισμού C και αποφεύγεται η αναλυτική παρουσίαση των δηλώσεων και των λοιπών δομών της γλώσσας.

Η σύνταξη των δηλώσεων της γλώσσας C φαίνεται στον Πίνακα 1, με μικρές παραλλαγές από το πρότυπο οι οποίες συζητούνται στη συνέχεια. Ιδιαίτερα χαρακτηριστικά του συστήματος τύπων της γλώσσας, που αποτελούν πηγές δυσκολιών για την τυπική περιγραφή του, είναι η χρήση *ποιοτικών παραλλαγών* (type qualifiers), η ύπαρξη *ημιτελών τύπων* (incomplete types) προκειμένου να ξεπερασθεί το πρόβλημα των *αναδρομικά ορισμένων τύπων* (recursively defined types), η ύπαρξη *πεδίων περιορισμένου μήκους* (bitfields) και κυρίως η εξαιρετικά λακωνική σύνταξη που χρησιμοποιείται.

Ορισμένα στοιχεία του συστήματος τύπων της C έχουν εξαιρεθεί από την τυπική περιγραφή που θα επιχειρηθεί στη συνέχεια. Τα περισσότερα από αυτά χαρακτηρίζονται στο πρότυπο ως *πεπαλαιωμένα* (obsolescent), και σύντομα θα καταργηθούν σε νέα αναθεώρηση της γλώσσας. Τα χαρακτηριστικά αυτά εκφράζουν τον παλαιό τρόπο προγραμματισμού σε C και η προσπάθεια να συμπεριληφθούν θα

αύξανε κατά πολύ την πολυπλοκότητα της τυπικής περιγραφής. Σε σχέση με το πρότυπο, η τυπική περιγραφή που ακολουθεί παρουσιάζει τις εξής διαφορές, κατά σειρά σπουδαιότητας:

(Π1) Για τη δήλωση και τον ορισμό των συναρτήσεων είναι απαραίτητη η χρήση *πρωτοτύπων* (function prototypes).[2] Επίσης για να κληθεί μια συνάρτηση θα πρέπει να είναι ορατό το πρωτότυπό της. Οι παρεκκλίσεις αυτές έχουν υιοθετηθεί στη γλώσσα C++ και το πρότυπο αφήνει να εννοηθεί ότι θα εφαρμοσθούν στην επόμενη αναθεώρηση.

(Π2) Η σειρά των λέξεων–κλειδιών που περιγράφουν τύπους στη σύνταξη των δηλώσεων είναι συγκεκριμένη. Απαγορεύονται δηλώσεις της μορφής `char unsigned typedef NewType;` που ήδη χαρακτηρίζονται πεπαλαιωμένες στο πρότυπο.

(Π3) Δεν επιτρέπεται η χρήση *προσδιορισμών αποθήκευσης* (storage specifiers) και σύνδεσης (linkage) για τα αναγνωριστικά, πλην του `typedef`. Οι προσδιορισμοί αυτοί μπορεί να θεωρηθεί ότι δεν ανήκουν στο σύστημα τύπων.

(Π4) Σε αντίθεση με το πρότυπο, οι τύποι `int` και `signed int` είναι ισοδύναμοι ακόμα και στην περίπτωση των πεδίων περιορισμένου μήκους.

(Π5) Δεν επιτρέπονται ανώνυμα πεδία περιορισμένου μήκους, ούτε τέτοια πεδία με μηδενικό μήκος. Οι παρεκκλίσεις αυτές μπορούν εύκολα να διορθωθούν.

Η τυπική περιγραφή της σημασιολογίας του συστήματος τύπων της C διακρίνεται σε δυο σχεδόν ανεξάρτητες φάσεις. Η περιγραφή της *στατικής σημασιολογίας* (static semantics) ασχολείται με τον έλεγχο τύπων των προγραμμάτων, που πραγματοποιείται συνήθως στο χρόνο μεταγλώττισης. Στην επόμενη ενότητα επιχειρείται η τυπική περιγραφή αυτής της φάσης. Αντίθετα, η περιγραφή της *δυναμικής σημασιολογίας* (dynamic semantics) αποβλέπει στην απόδοση ερμηνείας στους τύπους κατά την εκτέλεση των προγραμμάτων. Προσπάθειες περιγραφής της δυναμικής σημασιολογίας του συστήματος τύπων της C γίνονται στα [13] και [12].

3 Στατική σημασιολογία

Η περιγραφή της στατικής σημασιολογίας του συστήματος τύπων αποσκοπεί αφενός στην απόδοση στατικής ερμηνείας στις δηλώσεις, αφετέρου στη συσχέτιση τύπων με τις εκφράσεις της γλώσσας.

Η στατική ερμηνεία ενός συνόλου δηλώσεων είναι κατ' ουσία μια μερικώς ορισμένη συνάρτηση που σε κάθε αναγνωριστικό αντιστοιχεί τον τύπο που του αποδίδεται μέσω των δηλώσεων. Τα σημασιολογικά πεδία που χρησιμοποιούνται για την περιγραφή της στατικής σημασιολογίας δίνονται στην παράγραφο 3.1, ενώ ο τρόπος κατασκευής της συνάρτησης σημασιολογίας περιγράφεται στην παράγραφο 3.2. Στην παράγραφο 3.3 εξετάζεται ο αναδρομικός ορισμός τύπων δεδομένων.

Η συσχέτιση τύπων με εκφράσεις πραγματοποιείται μέσω ενός συνόλου *κανόνων συναγωγής* (inference rules) που περιέχουν *κρίσεις τύπων* (typing judgements). Στη βάση αυτού του συστήματος συναγωγής βρίσκεται η στατική ερμηνεία των δηλώσεων, που αποδίδει τύπους σε απλά αναγνωριστικά. Η τυπική περιγραφή της συσχέτισης τύπων με εκφράσεις σκιαγραφείται στην παράγραφο 3.4.

3.1 Σημασιολογικά πεδία

Η πολυπλοκότητα του συστήματος τύπων της C αντανακλάται άμεσα στον ορισμό των σημασιολογικών πεδίων για την περιγραφή των τύπων, που δίνεται στον Πίνακα 2. Όλα τα πεδία θεωρούνται *πλήρη δικτυώματα* (complete lattices). Το ελάχιστο στοιχείο συμβολίζεται ως $\bot$ και παριστάνει την απροσδιόριστη τιμή (χρησιμοποιείται για παράδειγμα στους ημιτελείς τύπους). Το μέγιστο στοιχείο συμβολίζεται ως $\top$ και παριστάνει τιμή σφάλματος.

[2] Στην παρούσα εργασία, η δήλωση `int f ();` περιγράφει μια συνάρτηση χωρίς παραμέτρους, αντί μιας συνάρτησης για την οποία δε γίνεται περιγραφή των παραμέτρων. Η μορφή `int f (void);` δε χρησιμοποιείται.

Πίνακας 2: Πεδία για τη στατική σημασιολογία.

Βοηθητικά πεδία.

I	:	$\mathbf{Ide}$		αναγνωριστικά (identifiers)
t	:	$\mathbf{Tag}$		ετικέττες (tags)
		$\mathbf{IdeType}$	$=$ *normal* | *typedef* | *tag*	τύποι αναγνωριστικών (identifier types)

Περιγραφή τύπων.

τ	:	$\mathbf{Type}_{dat}$	$=$ *void* | *char* | *signed char* | *unsigned char*	τύποι δεδομένων (data types)
			| *short int* | *unsigned short int*	
			| *int* | *unsigned int* | *long int* | *unsigned long int*	
			| *float* | *double* | *long double* | $\mathrm{ptr}\,[\phi]$	
			| $\mathrm{enum}\,[\epsilon]$ | $\mathrm{struct}\,[t,\pi]$ | $\mathrm{union}\,[t,\pi]$	
q	:	$\mathbf{Qual}$	$=$ *noqual* | *const* | *volatile* | *const volatile*	ποιοτικές παραλλαγές (type qualifiers)
α	:	$\mathbf{Type}_{obj}$	$=$ $\mathrm{obj}\,[\tau,q]$ | $\mathrm{array}\,[\alpha,n]$	τύποι αντικειμένων (object types)
f	:	$\mathbf{Type}_{fun}$	$=$ $\mathrm{func}\,[\tau,p]$	τύποι συναρτήσεων (function types)
ϕ	:	$\mathbf{Type}_{den}$	$=$ α | f	παριστώμενοι τύποι (denotable types)
m	:	$\mathbf{Type}_{mem}$	$=$ α | $\mathrm{bitfield}\,[\tau,q,n]$	τύποι μελών (member types)
v	:	$\mathbf{Type}_{val}$	$=$ τ | f	τύποι τιμών (value types)
θ	:	$\mathbf{Type}_{phr}$	$=$ $\exp\,[v]$ | $\mathrm{lvalue}\,[m]$	τύποι φράσεων (phrase types)

Περιγραφή περιβαλλόντων.

e	:	$\mathbf{Ent}$	περιβάλλοντα τύπων (type environments)
ϵ	:	$\mathbf{Enum}$	περιβάλλοντα απαριθμήσεων (enumeration environments)
π	:	$\mathbf{Memb}$	περιβάλλοντα μελών (member environments)
p	:	$\mathbf{Prot}$	πρωτότυπα συναρτήσεων (function prototypes)

Βοηθητικές συναρτήσεις περιβαλλόντων.

$\cdot[\cdot \mapsto \cdot]$	:	$\mathbf{Ent} \times (\mathbf{Ide} \times \mathbf{IdeType}) \times \mathbf{Type}_{den} \to \mathbf{Type}_{den}$	ενημέρωση
$\cdot[\cdot]$	:	$\mathbf{Ent} \times (\mathbf{Ide} \times \mathbf{IdeType}) \to \mathbf{Type}_{den}$	αναζήτηση
$\cdot[\cdot\ \&]$	:	$\mathbf{Ent} \times (\mathbf{Ide} \times \mathbf{IdeType}) \to \mathbf{Ent} \times \mathbf{Type}_{den}$	αναζήτηση ετικέττας

Τα βοηθητικά πεδία **Ide** και **Tag** περιγράφουν αντίστοιχα το σύνολο των αναγνωριστικών και το σύνολο των ετικεττών, ενώ το πεδίο **IdeType** χρησιμοποιείται για τη διάκριση των ιδιοτήτων των αναγνωριστικών. Το πεδίο $\mathbf{Type}_{dat}$ περιγράφει τους βασικούς τύπους δεδομένων της γλώσσας C και τον κενό τύπο *void*. Οι βασικοί τύποι δεδομένων αντιπροσωπεύουν στοιχεία που μπορούν να αποθηκευθούν στη μνήμη και να αντιμετωπισθούν ως ξεχωριστές οντότητες. Ας σημειωθεί ότι στους βασικούς τύπους δεδομένων συμπεριλαμβάνονται σύνθετοι τύποι, όπως π.χ. ο τύπος εγγραφής $\mathrm{struct}\,[t,\pi]$, γιατί η γλώσσα C επιτρέπει την ανάθεση τέτοιων αντικειμένων και το πέρασμα αυτών ως παραμέτρων. Αντίθετα, δεν συμπεριλαμβάνονται τύποι πινάκων ή συναρτήσεων. Το πεδίο **Qual** περιγράφει τις δυνατές ποιοτικές παραλλαγές των βασικών τύπων δεδομένων. Το πεδίο $\mathbf{Type}_{obj}$ περιγράφει τους τύπους αντικειμένων. Τα αντικείμενα είτε ανήκουν στους βασικούς τύπους δεδομένων και φέρουν κάποια ποιοτική παραλλαγή, είτε είναι πίνακες αντικειμένων. Οι τύποι δεδομένων της C, όπως ορίζονται στο πρότυπο, ολοκληρώνονται με το πεδίο $\mathbf{Type}_{fun}$ που περιγράφει τους τύπους συναρτήσεων.

Για τη σημασιολογική περιγραφή απαιτείται η περαιτέρω διάκριση των τύπων. Το πεδίο $\mathbf{Type}_{den}$ περιγράφει τους τύπους που μπορούν να αποδοθούν σε αναγνωριστικά μεταβλητών ή συνώνυμα τύπων. Το πεδίο $\mathbf{Type}_{mem}$ περιγράφει τους τύπους μελών εγγραφών ή ενώσεων και, εκτός από τύπους αντικειμένων, εμπεριέχει τύπους περιορισμένου μήκους. Το πεδίο $\mathbf{Type}_{val}$ περιγράφει τους τύπους

εκφράσεων που έχουν αποτιμηθεί. Αυτοί μπορούν να είναι είτε βασικοί τύποι δεδομένων, χωρίς ποιοτική παραλλαγή, είτε τύποι συναρτήσεων. Τέλος, το πεδίο $\mathbf{Type}_{phr}$ περιγράφει τους τύπους *φράσεων*, δηλαδή τους τύπους που αποδίδονται σε τμήματα προγραμμάτων. Στο πλαίσιο αυτής της εργασίας, το πεδίο αυτό περιέχει μόνο τους τύπους $\exp[\tau]$ και $\mathrm{lvalue}[m]$, που περιγράφουν αντίστοιχα εκφράσεις που έχουν αποτιμηθεί και l–values.

Ο ορισμός των σημασιολογικών πεδίων ολοκληρώνεται με τον ορισμό των πεδίων που περιγράφουν *περιβάλλοντα*. Το πεδίο **Ent** αναλαμβάνει την αντιστοίχιση των αναγνωριστικών σε τύπους. Παρόμοια, τα πεδία **Enum** και **Memb** χρησιμοποιούνται αντίστοιχα για τα μέλη απαριθμήσεων και εγγραφών ή ενώσεων. Τέλος, το πεδίο **Prot** χρησιμοποιείται για την περιγραφή των τυπικών παραμέτρων συναρτήσεων. Τα πεδία που περιγράφουν περιβάλλοντα μπορούν να ορισθούν εύκολα, χρησιμοποιώντας συναρτήσεις από το πεδίο των αναγνωριστικών σε κατάλληλα πεδία τύπων. Στην παρούσα εργασία δε θεωρείται σκόπιμος ο ακριβής ορισμός αυτών των πεδίων, παρά μόνο ο ορισμός κάποιων συναρτήσεων διαχείρισης των περιβαλλόντων. Κάποιες από αυτές φαίνονται στον Πίνακα 2.

3.2 Στατική σημασιολογία δηλώσεων

Η στατική ερμηνεία των δηλώσεων βασίζεται, όπως είναι φυσικό, στη σύνταξη των δηλώσεων. Σε κάθε μη τερματικό σύμβολο *nt* που εμφανίζεται στον Πίνακα 1 αποδίδεται ως ερμηνεία μια σημασιολογική συνάρτηση, που συμβολίζεται ως $[\![nt]\!]$. Μερικές από αυτές τις συναρτήσεις και οι εξισώσεις ορισμού τους δίνονται στον Πίνακα 3. Το σύνολο αυτών των εξισώσεων αποσκοπεί στον ορισμό της σημασιολογικής συνάρτησης:

$$[\![\textit{declaration–list}]\!] \ : \ \mathbf{Ent} \rightarrow \mathbf{Ent}$$

με την οποία ορίζεται η στατική ερμηνεία ενός συνόλου δηλώσεων. Η συνάρτηση αυτή δέχεται ως παράμετρο ένα αρχικό περιβάλλον τύπων και επιστρέφει ένα νέο περιβάλλον τύπων, που προκύπτει από την προσθήκη των νέων δηλώσεων στο αρχικό.

Η περιγραφή της στατικής σημασιολογίας για μια πραγματική γλώσσα προγραμματισμού όπως η C είναι ένα πολύ δύσκολο πρόβλημα. Προκειμένου να περιγραφούν όλες οι πτυχές του συστήματος δηλώσεων της C, είναι απαραίτητο να περιπλακούν οι τύποι των σημασιολογικών συναρτήσεων και οι εξισώσεις ορισμού τους. Στον Πίνακα 4 σκιαγραφείται η στατική ερμηνεία των δηλώσεων εγγραφών. Αξιοπρόσεκτα είναι τα εξής σημεία:

- Στην πρώτη εξίσωση, με την εμφάνιση της ετικέττας ορίζεται ο ημιτελής τύπος $\mathrm{struct}[I, \perp]$, ο οποίος συμπληρώνεται στο τέλος της δήλωσης.
- Στη δεύτερη εξίσωση, οι τύποι εγγραφών που ορίζονται δεν φέρουν ετικέττα αλλά είναι διακεκριμένοι. Αυτό επιτυγχάνεται με τη βοηθητική συνάρτηση fresh–untagged.
- Στην τρίτη εξίσωση, αναζητείται στο περιβάλλον e ο τύπος εγγραφής με ετικέττα I. Αν αυτός δε βρεθεί, τότε πρέπει να θεωρηθεί ημιτελής και να ενημερωθεί κατάλληλα το περιβάλλον.

3.3 Αναδρομικά ορισμένοι τύποι

Η γλώσσα C επιτρέπει την αναδρομική δήλωση τύπων. Αυτό επιτυγχάνεται με τη χρήση ημιτελών εγγραφών ή ενώσεων. Η απλούστερη δήλωση αναδρομικά ορισμένου τύπου είναι η εξής:

```
struct tag { struct tag * p; };
```

Πίνακας 3: Περιγραφή στατικής σημασιολογίας.

- $[\![\, \mathit{declaration} \,]\!] \;:\; \mathbf{Ent} \to \mathbf{Ent}$
  $[\![\, \mathit{declaration\text{–}specifiers}\ \mathit{init\text{–}declarator\text{–}list}\ \texttt{;} \,]\!] \;=\; [\![\, \mathit{init\text{–}declarator\text{–}list} \,]\!] \circ [\![\, \mathit{declaration\text{–}specifiers} \,]\!]$
- $[\![\, \mathit{declaration\text{–}specifiers} \,]\!] \;:\; \mathbf{Ent} \to \mathbf{Ent} \times \mathbf{Type}_{den} \times (\mathbf{Ent} \to \mathbf{Ide} \times \mathbf{Type}_{den} \to \mathbf{Ent})$
  $[\![\, \mathit{storage\text{–}class\text{–}specifier}\ \mathit{type\text{–}qualifier}\ \mathit{type\text{–}specifier} \,]\!] \;=\; \lambda e.$
  $\quad \mathbf{let}\ \langle e', \phi \rangle = [\![\, \mathit{type\text{–}specifier} \,]\!]\, e$
  $\quad \mathbf{in}\ \langle e', [\![\, \mathit{type\text{–}qualifier} \,]\!]\, \phi, [\![\, \mathit{storage\text{–}class\text{–}specifier} \,]\!] \rangle$
- $[\![\, \mathit{init\text{–}declarator\text{–}list} \,]\!] \;:\; \mathbf{Ent} \times \mathbf{Type}_{den} \times (\mathbf{Ent} \to \mathbf{Ide} \times \mathbf{Type}_{den} \to \mathbf{Ent}) \to \mathbf{Ent}$
  $[\![\, \epsilon \,]\!] \;=\; \lambda \langle e, \phi, g \rangle.\ e$
  $[\![\, \mathit{init\text{–}declarator}\ \mathit{init\text{–}declarator\text{–}list} \,]\!] \;=\; \lambda \langle e, \phi, g \rangle.\ [\![\, \mathit{init\text{–}declarator\text{–}list} \,]\!]\, \langle g\, e\, ([\![\, \mathit{init\text{–}declarator} \,]\!]\, \langle e, \phi \rangle), \phi, g \rangle$
- $[\![\, \mathit{init\text{–}declarator} \,]\!] \;:\; \mathbf{Ent} \times \mathbf{Type}_{den} \to \mathbf{Ide} \times \mathbf{Type}_{den}$
  $[\![\, \mathit{declarator} \,]\!] \;=\; \lambda \langle e, \phi \rangle.\ [\![\, \mathit{declarator} \,]\!]\, \langle e, \phi \rangle$
- $[\![\, \mathit{storage\text{–}class\text{–}specifier} \,]\!] \;:\; \mathbf{Ent} \to \mathbf{Ide} \times \mathbf{Type}_{den} \to \mathbf{Ent}$
  $[\![\, \epsilon \,]\!] \;=\; \lambda e.\, \lambda \langle I, \phi \rangle.\ \mathrm{isComplete}(\phi) \to e[\langle I, \mathit{normal} \rangle \mapsto \phi]\,, \top$
  $[\![\, \epsilon \,]\!] \;=\; \lambda e.\, \lambda \langle I, \phi \rangle.\ e[\langle I, \mathit{typedef} \rangle \mapsto \phi]$
- $[\![\, \mathit{declarator} \,]\!] \;:\; \mathbf{Ent} \times \mathbf{Type}_{den} \to \mathbf{Ide} \times \mathbf{Type}_{den}$
  $[\![\, I \,]\!] \;=\; \lambda \langle e, \phi \rangle.\ \langle I, \phi \rangle$
  $[\![\, \mathit{declarator}\ \texttt{[}\ n\ \texttt{]} \,]\!] \;=\; \lambda \langle e, \phi \rangle.\ \mathbf{case}\ \phi\ \mathbf{of}\ o \to [\![\, \mathit{declarator} \,]\!]\, \langle e, \mathrm{array}\,[o, n] \rangle$
  $[\![\, \mathit{declarator}\ \texttt{[ ]} \,]\!] \;=\; \lambda \langle e, \phi \rangle.\ \mathbf{case}\ \phi\ \mathbf{of}\ o \to [\![\, \mathit{declarator} \,]\!]\, \langle e, \mathrm{array}\,[o, \bot] \rangle$
  $[\![\, \texttt{*}\ \mathit{type\text{–}qualifier}\ \mathit{declarator} \,]\!] \;=\; \lambda \langle e, \phi \rangle.\ [\![\, \mathit{declarator} \,]\!]\, \langle e, [\![\, \mathit{type\text{–}qualifier} \,]\!]\, (\mathrm{obj}\,[\mathrm{ptr}\,[\phi], \mathit{noqual}]) \rangle$
  $[\![\, \mathit{declarator}\ \texttt{(}\ \mathit{parameter\text{–}type\text{–}list}\ \texttt{)} \,]\!] \;=\; \lambda \langle e, \phi \rangle.\ \mathbf{case}\ \phi\ \mathbf{of}$
  $\quad \mathrm{obj}\,[\tau, q] \to [\![\, \mathit{declarator} \,]\!]\, \langle e, \mathrm{func}\,[\tau, [\![\, \mathit{parameter\text{–}type\text{–}list} \,]\!]\, e\, p_\circ] \rangle$
- $[\![\, \mathit{type\text{–}qualifier} \,]\!] \;:\; \mathbf{Type}_{den} \to \mathbf{Type}_{den}$
  $[\![\, \epsilon \,]\!] \;=\; \lambda \phi.\ \phi$
  $[\![\, \texttt{const} \,]\!] \;=\; \lambda \phi.\ \mathrm{qualify}\ \mathit{const}\ \phi$ (κ.ο.κ.)
- $[\![\, \mathit{type\text{–}specifier} \,]\!] \;:\; \mathbf{Ent} \to \mathbf{Ent} \times \mathbf{Type}_{den}$
  $[\![\, \texttt{void} \,]\!] \;=\; \lambda e.\ \langle e, \mathrm{obj}\,[\mathit{void}, \mathit{noqual}] \rangle$
  $[\![\, \mathit{typedef\text{–}name} \,]\!] \;=\; \lambda e.\ \langle e, [\![\, \mathit{typedef\text{–}name} \,]\!]\, e \rangle$ (κ.ο.κ.)
- $[\![\, \mathit{typedef\text{–}name} \,]\!] \;:\; \mathbf{Ent} \to \mathbf{Type}_{den}$
  $[\![\, I \,]\!] \;=\; \lambda e.\ e[\langle I, \mathit{typedef} \rangle]$

Πίνακας 4: Περιγραφή στατικής σημασιολογίας (τύποι εγγραφών).

- $[\![\, \mathit{struct\text{–}specifier} \,]\!] \;:\; \mathbf{Ent} \to \mathbf{Ent} \times \mathbf{Type}_{den}$
  $[\![\, \texttt{struct}\ I\ \texttt{\{}\ \mathit{struct\text{–}declaration\text{–}list}\ \texttt{\}} \,]\!] \;=\; \lambda e.$
  $\quad \mathbf{let}\ e_c \;\;\;\; = e[\langle I, \mathit{tag} \rangle \mapsto \mathrm{obj}\,[\mathrm{struct}\,[I, \bot], \mathit{noqual}]]$
  $\qquad \langle e', \pi \rangle = [\![\, \mathit{struct\text{–}declaration\text{–}list} \,]\!]\, \langle e_c, \pi_\circ \rangle$
  $\qquad \phi \;\;\;\; = \mathrm{obj}\,[\mathrm{struct}\,[I, \pi], \mathit{noqual}]$
  $\quad \mathbf{in}\ \langle e'[\langle I, \mathit{tag} \rangle \mapsto \phi], \phi \rangle$
  $[\![\, \texttt{struct}\ \texttt{\{}\ \mathit{struct\text{–}declaration\text{–}list}\ \texttt{\}} \,]\!] \;=\; \lambda e.$
  $\quad \mathbf{let}\ \langle e', \pi \rangle = [\![\, \mathit{struct\text{–}declaration\text{–}list} \,]\!]\, \langle e, \pi_\circ \rangle$
  $\quad \mathbf{in}\ \langle e', \mathrm{obj}\,[\mathrm{struct}\,[\mathrm{fresh\text{–}untagged}, \pi], \mathit{noqual}] \rangle$
  $[\![\, \texttt{struct}\ I \,]\!] \;=\; \lambda e.\ e[\langle I, \mathit{tag} \rangle\ \&]$

Πίνακας 5: Σημασιολογία αναδρομικά ορισμένων τύπων.

- $\mathbf{rec}\,[\![\,\mathit{declaration\text{-}list}\,]\!] \;:\; \mathbf{Ent} \to \mathbf{Ent}$
  $\mathbf{rec}\,[\![\,\mathit{declaration\text{-}list}\,]\!] \;=\; \lambda\, e.\; \mathbf{let}\; f = [\![\,\mathit{declaration\text{-}list}\,]\!]\; \mathbf{in}\; \mathbf{clo}\,(f\; e)\; f$

Πίνακας 6: Αφηρημένη σύνταξη των εκφράσεων της ANSI–C.

expression ::= *I* | *n* | *f* | *c* | *s* | *expression* `[` *expression* `]` | *expression* `(` *arguments* `)` | *expression* `.` *I*
| *expression* `->` *I* | `sizeof` *expression* | `sizeof (` *type–name* `)` | *unary–operator expression*
| `(` *type–name* `)` *expression* | *expression binary–operator expression* | *unary–assignment expression*
| *expression binary–assignment expression* | *expression* `?` *expression* `:` *expression*

arguments ::= ϵ | *expression* `,` *arguments*

unary–operator ::= `&` | `*` | `+` | `-` | $\sim$ | `!`

binary–operator ::= `*` | `/` | `%` | `+` | `-` | `<<` | `>>` | `<` | `>` | `<=` | `>=` | `==` | `!=` | `&` | `^` | `|` | `&&` | `||` | `,`

unary–assignment ::= ++ (prefix) | ++ (postfix) | -- (prefix) | -- (postfix)

binary–assignment ::= `=` | `*=` | `/=` | `%=` | `+=` | `-=` | `<<=` | `>>=` | `&=` | `^=` | `|=`

δηλαδή ένας τύπος εγγραφής που περιέχει ένα δείκτη σε όμοια εγγραφή. Η στατική ερμηνεία της παραπάνω δήλωσης θα πρέπει να αντιστοιχεί στην ετικέττα `tag` τον τύπο t, όπου t είναι η ελάχιστη λύση της εξίσωσης:[3]

$$t \;=\; \mathrm{obj}\,[\mathrm{struct}\,[\mathit{tag}, \{\, p \mapsto \mathrm{ptr}\,[t]\,\}], \mathit{noqual}]$$

Με παρόμοιο τρόπο επιτρέπεται η αμοιβαία αναδρομική δήλωση τύπων που απαιτεί την εύρεση των ελάχιστων λύσεων συστημάτων εξισώσεων.

Προκειμένου να περιγραφεί σωστά η σημασιολογία των αναδρομικά ορισμένων τύπων, απαιτείται το *κλείσιμο* (closure) αυτών των τύπων στο περιβάλλον όπου ορίζονται [13]. Το κλείσιμο αυτό επιτυγχάνεται μέσω της σημασιολογικής συνάρτησης **rec**, που ορίζεται στον Πίνακα 5. Με τη χρήση του τελεστή **clo** κατασκευάζεται μια άπειρη ακολουθία περιβαλλόντων, κάθε ένα από τα οποία είναι μια καλύτερη προσέγγιση του προηγούμενου.[4] Αυτό είναι δυνατό να αποδειχθεί με επαγωγή επάνω στη σύνταξη των δηλώσεων. Το όριο της ακολουθίας είναι το αποτέλεσμα του κλεισίματος.

3.4 Συναγωγή τύπων

Η κύρια χρήση του συστήματος τύπων μιας γλώσσας προγραμματισμού είναι ο έλεγχος τύπων. Για να συμβεί αυτό απαιτείται πρώτα η συσχέτιση τύπων με τις εκφράσεις της γλώσσας. Αυτό συνήθως γίνεται μέσω ενός συστήματος κανόνων συναγωγής, βασισμένων στη σύνταξη των εκφράσεων. Η αφηρημένη σύνταξη (abstract syntax) των εκφράσεων της C δίνεται στον Πίνακα 6.

Το σύστημα συναγωγής τύπων βασίζεται στη χρήση *κρίσεων τύπων*. Οι κρίσεις τύπων είναι συμβολικές εκφράσεις της μορφής $\mathit{Env} \vdash \mathit{Expr} \diamond \mathit{Type}$, όπου το σύμβολο $\diamond$ αντικαθίσταται από σύμβολα συγκεκριμένων σχέσεων. Μια τέτοια συμβολική έκφραση διαβάζεται ως "δεδομένου του περιβάλλοντος *Env*, η έκφραση *Expr* συσχετίζεται με τον τύπο *Type* μέσω της σχέσης $\diamond$". Στην αρχή του Πίνακα 7 δίνονται οι διάφορες μορφές κρίσεων τύπων που χρησιμοποιούνται στο σύστημα

[3] Η λύση αυτή συχνά παριστάνεται στη βιβλιογραφία ως: $\mu\, t.\ \mathrm{obj}\,[\mathrm{struct}\,[\mathit{tag}, \{\, p \mapsto \mathrm{ptr}\,[t]\,\}], \mathit{noqual}]$

[4] Ο τελεστής **clo** είναι παρόμοιος με τον τελεστή ελάχιστου σταθερού σημείου **fix**. Ορίζεται ως: $\mathbf{clo}\; e\; f = \bigsqcup_{n=0}^{\infty}\; f^n\; e$, με την προϋπόθεση ότι η συνάρτηση f είναι συνεχής και $f^n\; e \sqsubseteq f^{n+1}\; e\;,\; \forall n \geq 0$.

Πίνακας 7: Συναγωγή τύπων.

Κρίσεις τύπων.

$e \vdash I \triangleleft \phi$	το αναγνωριστικό I έχει δηλωθεί με τύπο ϕ.
$e \vdash E : \theta$	η έκφραση E έχει τύπο θ.
$e \vdash E :: \theta$	η έκφραση E έχει τύπο θ, μετά από τις υπονοούμενες τροποποιήσεις (implicit conversions).
$e \vdash E \gg \tau$	η έκφραση E μπορεί να ανατεθεί σε τύπο δεδομένων τ.

Κανόνες συναγωγής.

$$\frac{e \vdash I \triangleleft \alpha}{e \vdash I : \mathrm{lvalue}\,[\alpha]} \qquad \frac{e \vdash I \triangleleft f}{e \vdash I : \exp\,[f]} \qquad \frac{}{e \vdash n : \exp\,[\mathit{int}]} \qquad \frac{}{e \vdash f : \exp\,[\mathit{double}]}$$

$$\frac{e \vdash *\,(E_1{+}E_2) \;:\; \theta}{e \vdash E_1\,[E_2\,] \;:\; \theta} \qquad \frac{\begin{array}{c} e \vdash E :: \exp\,[\mathrm{ptr}\,[\mathrm{func}\,[\tau, p]]] \\ p \text{ έχει } n \text{ παραμέτρους } p_i \text{ και όχι αποσιωποιητικά} \\ e \vdash E_i \gg p_i\ ,\ \forall i = 1, \ldots, n \end{array}}{e \vdash E\,(E_1\,, \ldots, E_n) \;:\; \exp\,[\tau]}$$

$$\frac{e \vdash E : \mathrm{lvalue}\,[\mathrm{obj}\,[\mathrm{struct}\,[t, \pi], q]] \quad \pi \vdash I \triangleleft m}{e \vdash E.I : \mathrm{lvalue}\,[m]} \qquad \frac{e \vdash E :: \exp\,[\mathrm{ptr}\,[\mathrm{obj}\,[\mathrm{struct}\,[t, \pi], q]]] \quad \pi \vdash I \triangleleft m}{e \vdash E{-}{>}I : \mathrm{lvalue}\,[m]}$$

$$\frac{e \vdash E : \mathrm{lvalue}\,[\alpha]}{e \vdash \&E : \exp\,[\mathrm{ptr}\,[\alpha]]} \qquad \frac{e \vdash E : \exp\,[f]}{e \vdash \&E : \exp\,[\mathrm{ptr}\,[f]]} \qquad \frac{e \vdash E : \exp\,[\mathrm{ptr}\,[\alpha]]}{e \vdash *E : \mathrm{lvalue}\,[\alpha]} \qquad \frac{e \vdash E : \exp\,[\mathrm{ptr}\,[f]]}{e \vdash *E : \exp\,[f]}$$

$$\frac{\begin{array}{cc} e \vdash E_1 :: \exp\,[\tau_1] & e \vdash E_2 :: \exp\,[\tau_2] \\ \mathrm{isArithmetic}(\tau_1) & \mathrm{isArithmetic}(\tau_2) \\ \multicolumn{2}{c}{\tau' = \mathrm{arithConv}(\tau_1, \tau_2)} \end{array}}{e \vdash E_1{+}E_2 \;:\; \exp\,[\tau']} \qquad \frac{\begin{array}{c} e \vdash E_1 :: \exp\,[\mathrm{ptr}\,[\alpha_1]] \\ e \vdash E_2 :: \exp\,[\tau_2] \\ \mathrm{isIntegral}(\tau_2) \end{array}}{e \vdash E_1{+}E_2 \;:\; \exp\,[\mathrm{ptr}\,[\alpha_1]]}$$

(και συμμετρικά)

συναγωγής τύπων για τη γλώσσα C. Στη συνέχεια του ίδιου πίνακα δίνονται μερικοί από τους *κανόνες συναγωγής* που συσχετίζουν τύπους με τις εκφράσεις της C. Κάθε κανόνας έχει έναν αριθμό από υποθέσεις, που είναι κρίσεις τύπων ή άλλα κατηγορήματα, και ένα συμπέρασμα.

4 Συμπεράσματα

Στην εργασία αυτή δίνεται με συνοπτικό τρόπο μια τυπική περιγραφή της στατικής σημασιολογίας του συστήματος τύπων της γλώσσας προγραμματισμού C. Εκτενής τυπική περιγραφή τόσο της στατικής όσο και της δυναμικής σημασιολογίας του ίδιου συστήματος δίνεται στο [12]. Η περιγραφή της στατικής σημασιολογίας αποσκοπεί αφενός στην απόδοση στατικής ερμηνείας στις δηλώσεις, αφετέρου στην κατασκευή ενός συστήματος κανόνων συναγωγής για τον έλεγχο των τύπων των προγραμμάτων. Είναι πολύτιμο εφόδιο για τους κατασκευαστές υλοποιήσεων της γλώσσας, ως μονοσήμαντη ερμηνεία του προτύπου, αλλά και για τους θεωρητικούς μελετητές της γλώσσας, καθώς επιτρέπει τη διατύπωση και απόδειξη θεωρημάτων σχετικών με τη συμπεριφορά των προγραμμάτων.

Η εργασία αυτή είναι μέρος γενικότερης έρευνας πάνω στη σημασιολογία της γλώσσας C. Σκοπός της έρευνας αυτής είναι η πλήρης τυπική περιγραφή της σημασιολογίας της C με χρήση της μαθηματικής προσέγγισης. Η σημασιολογία του συστήματος τύπων τοποθετείται στη βάση της συνολικής περιγραφής που, στο πλαίσιο μελλοντικής έρευνας, πρόκειται να συμπληρωθεί με τη σημασιολογία των εκφράσεων, των εντολών και των ορισμών συναρτήσεων. Αξίζει να σημειωθεί ότι οι τύποι των συναρτήσεων ερμηνείας αυτών των δομών συνδέονται άμεσα με τη δυναμική σημασιολογία του συστήματος τύπων, γεγονός που ενισχύει τη σπουδαιότητα του τελευταίου. Στόχο

μελλοντικής έρευνας επίσης θα αποτελέσει η τυπική περιγραφή της σημασιολογίας γλωσσών όπως η C++ και η Java που, αν και βασίζονται εν μέρει στη C, εκφράζουν περισσότερο μοντέρνους τρόπους προγραμματισμού.

Βιβλιογραφία

[1] D. M. Ritchie, S. C. Johnson, M. E. Lesk, and B. W. Kernighan, "The C programming language," *BSTJ*, vol. 57, pp. 1991–2020, July–Aug. 1978.

[2] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*. Englewood Cliffs, NJ: Prentice Hall, 2nd ed., 1988.

[3] American National Standards Institute, New York, NY, *ANSI/ISO 9899–1990, American National Standard for Programming Languages: C*, 1990. Revision and redesignation of ANSI X3.159–1989.

[4] American National Standards Institute, New York, NY, *Rationale for American National Standard for Information Systems: Programming Language C*, 1989. Supplement to ANSI X3.159–1989.

[5] American National Standards Institute, New York, NY, *Technical Corrigendum Number 1 to ANSI/ISO 9899–1990 American National Standard for Programing Languages: C*, 1994.

[6] R. D. Tennent, "The denotational semantics of programming languages," *Communications of the ACM*, vol. 19, pp. 437–453, Aug. 1976.

[7] P. D. Mosses, "Denotational semantics," in *Handbook of Theoretical Computer Science* (J. van Leeuwen, ed.), vol. B, ch. 11, pp. 577–631, Elsevier Science Publishers B.V., 1990.

[8] R. E. Milne and C. Stachey, *A Theory of Programming Language Semantics*. London, UK: Chapman and Hall, 1976.

[9] J. E. Stoy, *Denotational Semantics: The Scott–Strachey Approach to Programming Language Theory*. Cambridge, MA: MIT Press, 1977.

[10] D. A. Schmidt, *Denotational Semantics: A Methodology for Language Development*. Newton, MA: Allyn and Bacon, 1986.

[11] R. D. Tennent, *Semantics of Programming Languages*. Englewood Cliffs, NJ: Prentice Hall, 1991.

[12] N. S. Papaspyrou, "An attempt to formalize the type system of C," Tech. Rep. CSD–SW–TR–3–97, National Technical University of Athens, Software Engineering Laboratory, 1997.

[13] R. Sethi, "A case study in specifying the semantics of a programming language," in *Proceedings of the 7th Annual ACM Symposium on Principles of Programming Languages*, pp. 117–130, Jan. 1980.

[14] J. C. Mitchell, "Type systems for programming languages," in *Handbook of Theoretical Computer Science* (J. van Leeuwen, ed.), vol. B, ch. 8, pp. 365–458, Elsevier Science Publishers B.V., 1990.

[15] L. Cardelli, "Typeful programming," in *Formal Description of Programming Concepts*, State–of–the–art Reports, pp. 431–507, IFIP, 1991.