Αρχές Γλωσσών Προγραμματισμού

Εργασία 3

Sudoku Solver

Σκοπός της εργασίας αυτής είναι η υλοποίηση προγράμματος Haskell με τη δυνατότητα να συμπληρώνει πλέγματα Sudoku. Ένα πλέγμα Sudoku έχει διαστάσεις 9×9 και είναι χωρισμένο σε 9 μικρότερα υποπλέγματα με διαστάσεις 3×3. Κάθε πλέγμα αποτελείται από 81 κελιά συνολικά, ενώ οι γραμμές, οι στήλες και τα υποπλέγματα του ονομάζονται **μονάδες**. Σε καθένα από τα 81 κελιά του πλέγματος τοποθετείται ένας αριθμός από το 1 έως το 9. Σε ένα μη-συμπληρωμένο πλέγμα, κάποια κελιά έχουν αριθμούς ενώ κάποια άλλα όχι. Για να συμπληρωθεί ένα πλέγμα πρέπει να τοποθετηθούν οι σωστοί αριθμοί στα άδεια κελιά έτσι ώστε οι 9 αριθμοί που βρίσκονται σε κάθε μονάδα (γραμμή, στήλη και πλέγμα 3×3) να είναι διαφορετικοί μεταξύ τους. Ένα μη-συμπληρωμένο πλέγμα Sudoku φαίνεται δεξιά.

	8					2		
				8	4		9	
		6	3	2			1	
	9	7					8	
8			9		3			2
	1					9	5	
	7			4	5	8		
	3		7	1				
		8					4	

Για τους σκοπούς της εργασίας θα μοντελοποιήσουμε τα πλέγματα Sudoku ως Προβλήματα Ικανοποίησης Περιορισμών (**Constraint Satisfaction Problem - CSP**). Ένα CSP περιέχει τα εξής δομικά στοιχεία:

- **Μεταβλητές με πεδία τιμών**
- **Περιορισμούς μεταξύ των μεταβλητών**

Συγκεκριμένα, οι περιορισμοί είναι συναρτήσεις που επιστρέφουν True ή False αναλόγως αν η εκάστοτε ανάθεση τιμών στις μεταβλητές επιτρέπεται ή όχι. Επομένως, ένας περιορισμός καθορίζει τους επιτρεπτούς συνδυασμούς τιμών για ένα υποσύνολο του συνόλου των μεταβλητών. Στην εργασία αυτή, θα θεωρούμε πως οι περιορισμοί είναι πάντα ανάμεσα σε *δύο* μεταβλητές (δηλαδή ότι είναι *δυαδικοί*).

Για την επίλυση ενός CSP πρέπει να βρεθεί μία **ανάθεση τιμών** σε όλες τις μεταβλητές η οποία ικανοποιεί όλους τους περιορισμούς του προβλήματος. Το μεγαλύτερο μέρος της εργασίας θα περιλαμβάνει την υλοποίηση μεθόδων επίλυσης CSP προβλημάτων με σκοπό την συμπλήρωση πλεγμάτων Sudoku.

Η επίλυση ενός CSP περιλαμβάνει την περιήγηση στον χώρο καταστάσεών του. Μια κατάσταση (state) ενός CSP αποτελείται από ένα σύνολο μεταβλητών που έχουν λάβει τιμές (assigned), από ένα σύνολο μεταβλητών που δεν έχουν λάβει τιμές (unassigned) και από ένα πεδίο τιμών για καθεμία από αυτές τις μεταβλητές.

Μία κατάσταση θεωρείται *Κατάσταση Αποτυχίας* αν οποιοδήποτε από τα πεδία τιμών των μεταβλητών που δεν έχουν λάβει τιμή είναι κενό.

Μια κατάσταση θεωρείται *consistent* αν για οποιοδήποτε περιορισμό (x_i, x_j), όπου x_i, x_j μεταβλητές του προβλήματος, για κάθε τιμή v στο πεδίο τιμών της x_i υπάρχει κάποια τιμή w στο πεδίο τιμών της x_j έτσι ώστε αν η x_i λάβει την τιμή v και η x_j λάβει την τιμή w, ο περιορισμός (x_i, x_j) ικανοποιείται.

Για να μοντελοποιήσουμε ένα πλέγμα Sudoku ως CSP εργαζόμαστε ως εξής:

- Θεωρούμε πως κάθε κελί είναι μία μεταβλητή. Το πεδίο τιμών της είναι οι αριθμοί 1 έως 9 αν το κελί είναι άδειο και η τιμή του κελιού αν είναι συμπληρωμένο.
- Ορίζουμε τους περιορισμούς του Sudoku, δηλαδή ότι πρέπει οι 9 αριθμοί σε κάθε μονάδα να είναι διαφορετικοί μεταξύ τους.

Παράδειγμα: Ονομάζουμε x_1 την μεταβλητή που αντιπροσωπεύει το κελί που βρίσκεται στην πρώτη στήλη και την πρώτη γραμμή του πλέγματος της παραπάνω εικόνας Sudoku. Το κελί αυτό είναι άδειο, άρα το πεδίο τιμών της x_1 είναι το σύνολο $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Οι περιορισμοί που υπάρχουν για την x_1 είναι ότι πρέπει να έχει διαφορετική τιμή από τις μεταβλητές που βρίσκονται σε ίδιες μονάδες με αυτήν. Για παράδειγμα, αν x_2 είναι η μεταβλητή που αντιπροσωπεύει το κελί ακριβώς κάτω από την x_1, τότε ένας περιορισμός είναι ότι η τιμή που θα ανατεθεί στην x_1 (έστω v_1) πρέπει να είναι διαφορετική από την τιμή που θα ανατεθεί στην x_2 (έστω v_2), δηλαδή $v_1 \neq v_2$ Αυτό ισχύει καθώς οι μεταβλητές x_1 και x_2 βρίσκονται στην ίδια στήλη. Μία πιθανή μοντελοποίηση θα αντιμετώπιζε ακόμα και τα συμπληρωμένα κελιά ως μεταβλητές, με μόνη διαφορά ότι το πεδίο τιμών τους περιέχει μόνο ένα στοιχείο. Για παράδειγμα, το πεδίο τιμών της μεταβλητής του κελιού δεξιά της x_1 θα ήταν το μονοσύνολο $\{8\}$.

Αξίζει να σημειωθεί ότι οι μέθοδοι που θα χρησιμοποιηθούν στην εργασία είναι γενικοί, και μπορούν να εφαρμοστούν για την λύση οποιουδήποτε CSP. Με άλλα λόγια, ο κώδικας που θα γράψετε θα έπρεπε, με ελάχιστες αλλαγές, να μπορεί να λύσει οποιοδήποτε πρόβλημα αν αυτό μοντελοποιηθεί σωστά ως CSP. Βέβαια, για τους σκοπούς της εργασίας, αρκεί να μπορεί να λύσει πλέγματα Sudoku.

Ενδεικτικά, μπορείτε να χρησιμοποιήσετε τα εξής data structures προκειμένου να μοντελοποιήσετε σωστά ένα CSP (προσέξτε ότι οι περιορισμοί που χρησιμοποιούνται είναι πάντα ανάμεσα σε δύο μεταβλητές)[1].

```
data Variable = Var Int [Int] [Constraint]
data Constraint = Cons Int (Int -> Int -> Bool)
data SudokuState = SudokuState [Variable] [Variable]
```

Παρακάτω εξηγούμε τα χαρακτηριστικά των data structures με την σειρά που φαίνονται:

- Κάθε `Variable` έχει έναν ακέραιο αριθμό που είναι το αναγνωριστικό της μεταβλητής, μία λίστα από ακεραίους που αποτελούν το πεδίο τιμών της και μία λίστα με τους περιορισμούς στους οποίους συμμετέχει.
- Κάθε `Constraint` έχει έναν ακέραιο που αντιστοιχεί στο αναγνωριστικό της δεύτερης μεταβλητής που συμμετέχει στον περιορισμό και μία συνάρτηση η οποία δέχεται τις τιμές των δύο μεταβλητών και επιστρέφει True ή False ανάλογα με το αν οι τιμές ικανοποιούν τον περιορισμό ή όχι.
- Κάθε `State` περιέχει δύο λίστες από μεταβλητές. Στην πρώτη αποθηκεύονται οι μεταβλητές στις οποίες έχει ανατεθεί τιμή (assigned) και στη δεύτερη οι μεταβλητές που δεν έχουν ακόμα τιμή (unassigned). Να σημειωθεί ότι μία assigned μεταβλητή έχει μία μόνο τιμή στο πεδίο τιμών της.

Naive Backtracking και AllDifferent-2 (35%)

Για το πρώτο μέρος της άσκησης καλείστε να υλοποιήσετε έναν solver ο οποίος θα λειτουργεί με *Naive Backtracking*. Το *Naive Backtracking* είναι μια μορφή αναζήτησης κατά βάθος των αναθέσεων των μεταβλητών. Μόλις έχουν ανατεθεί όλες οι μεταβλητές με κάποια τιμή γίνεται ο έλεγχος ικανοποίησης των περιορισμών. Αν οι περιορισμοί ικανοποιούνται επιστρέφεται η τρέχουσα ανάθεση, αλλιώς εξετάζονται οι αναθέσεις σε σειρά κατά βάθος αναζήτησης μέχρι να βρεθεί κάποια που να ικανοποιεί τους περιορισμούς.

Στην περίπτωση του Sudoku, κάθε μεταβλητή θα πρέπει να λάβει διαφορετική τιμή από τις μεταβλητές που βρίσκονται σε ίδια μονάδα με αυτή. Αυτοί οι περιορισμοί θα πρέπει να μοντελοποιηθούν ως μια δυαδική σχέση (AllDifferent-2). Δηλαδή, θα πρέπει για δύο οποιεσδήποτε μεταβλητές που βρίσκονται στην ίδια μονάδα να υπάρχει ένας περιορισμός AllDifferent-2 ο οποίος θα επιβάλει να έχουν διαφορετικές τιμές.

[1] Δεν είστε αναγκασμένοι να ακολουθήσετε την συγκεκριμένη μοντελοποίηση. Αυτή δίνεται για δική σας ευχέρεια.

Ο Solver σας θα πρέπει να διαβάζει το όνομα ενός αρχείου Sudoku από την γραμμή εντολών και να επιστρέφει την λύση του. Ένα αρχείο Sudoku είναι ένα text file (.txt) που έχει την μορφή που φαίνεται παρακάτω.

```
...63....
2.6..7.9.
54.9.....
....5....
..3.9...6
9621.....
31.7..2..
...3..4..
.7..62.5.
```

Κάθε γραμμή του αρχείου αντιστοιχεί σε μία γραμμή του πλέγματος Sudoku και οι τελείες (.) συμβολίζουν άδεια κελιά ενώ οι αριθμοί συμβολίζουν συμπληρωμένα κελιά.

Για αυτό το κομμάτι της εργασίας προτείνουμε να δοκιμάσετε την υλοποίησή σας με πλέγματα Sudoku που έχουν **λίγα** (1-7) κενά κελιά.

Forward Checking και MRV (45%)

Αφου υλοποιήσετε τον αλγόριθμο του *Naive Backtracking* θα παρατηρήσετε ότι ο Solver σας μπορεί να βρει λύσεις για πλέγματα Sudoku που έχουν πολύ λίγα κενά κελιά. Προκειμένου να βελτιωθεί η απόδοση του Solver σας και να μπορεί να λύσει ακόμα και τα πιο δύσκολα Sudoku καλείστε να υλοποιήσετε τον αλγόριθμο *Forward Checking* σε συνδυασμό με την μέθοδο επιλογής μεταβλητής για ανάθεση *MRV*.

Αρχικά, πρέπει να προεπεξεργάζεστε κάθε πλέγμα Sudoku ώστε να αφαιρείτε τις μη-έγκυρες τιμές από τα πεδία τιμών των μεταβλητών. Συγκεκριμένα, για κάθε κελί το οποίο είναι ήδη συμπληρωμένο πρέπει να αφαιρείτε τον αριθμό του κελιού από τα πεδία τιμών όλων των μεταβλητών που βρίσκονται σε ίδια μονάδα με το συμπληρωμένο κελί. Στην συνέχεια, πρέπει να καλείτε τον αλγόριθμο *Forward Checking*.

Ο αλγόριθμος *Forward Checking* με είσοδο μια κατάσταση και μια μέθοδο επιλογής μεταβλητής επιστρέφει είτε μια καινούργια κατάσταση στην οποία όλες οι μεταβλητές έχουν λάβει κάποια τιμή ή αποτυχία. Αρχικά **επιλέγεται κάποια μεταβλητή** v που δεν έχει ανατεθεί ακόμα και γίνεται προσπάθεια ανάθεσης αυτής της μεταβλητής με κάποια τιμή k_i από το πεδίο τιμών της. Για όλες τις μεταβλητές w_i που δεν έχουν λάβει τιμή ακόμα και συμμετέχουν σε κάποιο περιορισμό με την v αφαιρούνται οι τιμές από το πεδίο τιμών τους τις οποίες δεν μπορούν να λάβουν αν η v λάβει την τιμή k_i. Οι τιμές αυτές βρίσκονται ελέγχοντας τους περιορισμούς που υπάρχουν ανάμεσα στην v και την εκάστοτε w_i. Η διαδικασία αυτή ονομάζεται *pruning* του πεδίου τιμών. Σε περίπτωση που μετά την διαγραφή το πεδίο τιμών κάποιας w_i είναι κενό η κατάσταση επαναφέρεται στην κατάσταση εισόδου και η διαδικασία επαναλαμβάνεται για την επόμενη τιμή k_{i+1} στο πεδίο τιμών της v. Σε περίπτωση που δοκιμαστούν όλες οι τιμές k_i και έχουν βρεθεί μόνο καταστάσεις αποτυχίας, τότε επιστρέφεται αποτυχία. Αν βρεθεί τιμή k_i για την οποία καμία unassigned μεταβλητή δεν έχει κενό πεδίο τιμών, τότε **καλείται αναδρομικά η *Forward Checking*** αυτή τη φορά με είσοδο την νέα κατάσταση που περιλαμβάνει την v ως μεταβλητή που έχει λάβει την τιμή k_i και τα ανανεωμένα πεδία τιμών. Σε περίπτωση αποτυχίας, δοκιμάζεται η επόμενη τιμή k_{i+1} στο πεδίο τιμών της v, αλλιώς επιστρέφεται η ολοκληρωμένη ανάθεση που προέκυψε από τις αναδρομικές κλήσεις του *Forward Checking*. Αν εξαντληθούν τα στοιχεία του πεδίου τιμών της v τότε επιστρέφεται αποτυχία.

Κάθε κλήση του *Forward Checking* χρησιμοποιεί μια **μέθοδο επιλογής μεταβλητής**. Προκειμένου να ελαττωθεί ο παράγοντας διακλάδωσης και να καταλήξουμε το γρηγορότερο δυνατό σε μια σωστή ανάθεση των μεταβλητών η μέθοδος επιλογής που θα χρησιμοποιηθεί βασίζεται στο πλήθος των εναπομεινάντων στοιχείων

στο πεδίο τιμών μιας μεταβλητής. Ειδικότερα, κάθε φορά θα πρέπει να διαλέγεται για ανάθεση η μεταβλητή που έχει τις λιγότερες διαθέσιμες τιμές. Αυτή η μέθοδος επιλογής ονομάζεται Minimum Remaining Values (MRV).

Hidden Singles, Naked Pairs και Συνθήκες Αποτυχίας (20%)

Για να επιτευχθούν ακόμα καλύτερα αποτελέσματα και να ελαττωθεί ο χώρος αναζήτησης μπορούν να εφαρμοστούν μερικές τεχνικές Sudoku που αφαιρούν στοιχεία από τα πεδία τιμών των μεταβλητών. Μια πολύ απλή τεχνική είναι αυτή των Hidden Singles όπου όταν για κάποια μονάδα υπάρχει κάποια μεταβλητή c που περιέχει στο πεδίο τιμών της κάποια τιμή v που δεν την περιέχει κανένα άλλο κελί στην ίδια μονάδα, τότε αναγκαστικά θα πρέπει το c να λάβει την τιμή v.

Η τεχνική των Naked Pairs είναι η εξής: Όταν δύο κελιά μιας μονάδας έχουν το ίδιο πεδίο τιμών μεγέθους δύο, τότε οι κοινές τιμές των δύο αυτών κελιών θα πρέπει να αφαιρεθούν από τα πεδία τιμών των υπόλοιπων μεταβλητών της ίδιας μονάδας.

Μέχρι τώρα θεωρούσαμε μια κατάσταση ως *Κατάσταση Αποτυχίας* αν σε αυτή υπάρχει κάποια unassigned μεταβλητή με κενό πεδίο τιμών. Υπάρχουν όμως περιπτώσεις όπου μια κατάσταση μπορεί να θεωρηθεί *Κατάσταση Αποτυχίας* νωρίτερα. Για παράδειγμα, ας υποθέσουμε πως τα πεδία τιμών τριών κελιών μιας μονάδας είναι τα εξής: [1,2], [1,2], [1,2], είναι προφανές ότι δεν μπορεί να υπάρξει ανάθεση των κελιών έτσι ώστε όλα να έχουν διαφορετική τιμή. Αυτή η ιδέα μπορεί να γενικευτεί, έτσι ώστε όταν n μεταβλητές της ίδιας μονάδας έχουν το ίδιο πεδίο τιμών dom, με $|dom| < n$, η κατάσταση να θεωρείται *Κατάσταση Αποτυχίας*.

Οι τεχνικές αυτές εφαρμόζονται στην αρχή κάθε αναδρομικής κλήσης, μειώνοντας έτσι τον χώρο αναζήτησης της μεθόδου. Σε αυτό το κομμάτι της εργασίας, καλείστε να υλοποιήσετε τις 3 παραπάνω τεχνικές.

Maintaining Arc Consistency (Bonus 30%)

Ο αλγόριθμος *Maintaing Arc Consistency (MAC)* αποτελεί φυσική επέκταση του αλγορίθμου *Forward Checking*. Στον αλγόριθμο Forward Checking μας ενδιέφερε για οποιαδήποτε ανάθεση μεταβλητής τα πεδία τιμών των unassigned μεταβλητών να είναι μη-κενά, αφότου αφαιρέθουν από αυτά οι μη-επιτρεπτές τιμές. Στον *MAC* μας ενδιαφέρει το state του CSP να είναι consistent καθ' όλη την διάρκεια εκτέλεσης του αλγορίθμου.

Συγκεκριμένα, ο αλγόριθμος *MAC* δουλεύει ως εξής:

1. Κάθε φορά που γίνεται ανάθεση τιμής σε κάποια μεταβλητή, έστω x_i, ο *MAC* προσθέτει σε μία στοίβα όλα τα constraints στα οποία συμμετέχει η μεταβλητή αυτή σαν δυάδες (x_j, x_i). Αν η x_i πριν γίνει assigned έχει μόνο μία τιμή στο πεδίο τιμών της τότε ο *MAC* τερματίζει αναθέτοντας στην x_i την τιμή αυτή.

2. Ο *MAC* παίρνει το πρώτο στοιχείο της στοίβας, έστω (x_j, x_i) και επιβάλει για κάθε στοιχείο v του πεδίου τιμών της μεταβλητής x_j να υπάρχει τουλάχιστον μία τιμή της μεταβλητής x_i η οποία να ικανοποιεί τους περιορισμούς που υπάρχουν ανάμεσα στις δύο μεταβλητές. Αν δεν υπάρχει τέτοια τιμή, ο αλγόριθμος αφαιρεί την τιμή v από το πεδίο τιμών της x_j και προχωράει στην επόμενη τιμή. Εφόσον στο τέλος της παραπάνω διαδικασίας το μέγεθος του πεδίου τιμών της x_j έχει ελαττωθεί, τότε ο αλγόριθμος προσθέτει στην στοίβα όλα τα constraints στα οποία συμμετέχει η μεταβλητή x_j σαν δυάδες του τύπου (x_k, x_j).

3. Το βήμα 2 επαναλαμβάνεται έως ότου η στοίβα αδειάσει. Αν σε οποιοδήποτε σημείο εκτέλεσης του αλγορίθμου το πεδίο τιμών κάποιας μεταβλητής γίνει κενό, τότε η αρχική ανάθεση απορρίπτεται.

Σε αυτό το κομμάτι της εργασίας, καλείστε να υλοποιήσετε τον αλγόριθμο *MAC* για την επίλυση πλεγμάτων Sudoku.

Παραδοτέο

Είναι σημαντικό να διαβάσετε *προσεκτικά* την εκφώνηση και να κατανοήσετε τα ζητούμενα της πριν αρχίσετε την υλοποίηση της εργασίας.

Αν χρειάζεστε περαιτέρω πληροφορίες για τα CSPs και τους τρόπους επίλυσής τους μπορείτε να συμβουλευτείτε *τις διαφάνειες του κ. Κουμπαράκη*.

Για κάθε ένα από τα μέρη της εργασίας, υλοποιήστε ένα αντίστοιχο πηγαίο πρόγραμμα Haskell με τις σχετικές συναρτήσεις. Να συμπεριλάβετε και ένα `main.hs` αρχείο το οποίο θα υλοποιεί στοιχειώδεις λειτουργίες εισόδου-εξόδου. Συγκεκριμένα, τρέχοντας:

```
main <option> example.txt
```

πρέπει να εκτελείται ο αντίστοιχος αλγόριθμος πάνω στο Sudoku του αρχείου `example.txt`, και να εκτυπώνεται το συμπληρωμένο πλέγμα στο τερματικό. Το όρισμα `<option>` παίρνει τις τιμές `naive` για τον αλγόριθμο *Naive Backtracking*, `fc` για τον αλγόριθμο *Forward Checking*, `mac` για τον αλγόριθμο *MAC*.

Φροντίστε το πρόγραμμά σας να μεταγλωττίζεται με τον Glasgow Haskell Compiler (ghc) και ότι τρέχει σε περιβάλλον GHCi.

Επιπλέον, περιγράψτε συνοπτικά τις όποιες σχεδιαστικές επιλογές σας σε ενα README.

Το παραδοτέο σας θα πρέπει να είναι ένα αρχείο zip με όλα τα αρχεία που αναφέρονται παραπάνω.

Στην εργασία αυτή απαγορεύεται η χρήση external modules της Haskell.

Η εργασία μπορεί να υλοποιηθεί είτε ατομικά, είτε από ομάδες των δύο ατόμων (πρέπει να περιλάβατε στο README τα ονόματα και των δύο μελών της ομάδας).

Για οποιαδήποτε περαιτέρω διευκρίνηση, μπορείτε να αναρτήσετε τις ερωτήσεις σας στην αντίστοιχη περιοχή συζητήσεων του eclass ή να απευθυνθείτε μέσω email σε κάποιον από τους υπεύθυνους συνεργάτες:

Κωστής Σειράς - sdi1800174 [at] di.uoa.gr
Αντώνης Κορίνθιος - sdi2100067 [at] di.uoa.gr