Εθνικό Μετσόβιο Πολυτεχνείο

Σχολη Ηλεκτρολογων Μηχανικων
Και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης &
Υπολογιστων

Μετασχηματισμοί Προγραμμάτων στις Επαγωγικές Βάσεις Δεδομένων με χρήση του Νοηματικού Λογικού Προγραμματισμού

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

του

ΠΕΤΡΟΥ Μ. ΠΟΤΙΚΑ

Πτυχιούχου Τμήματος Πληροφορικής &
Τηλεπικοινωνιών Ε.Κ.Π.Α. (2000)

Αθήνα, Ιούλιος 2006

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων &
Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης &
Υπολογιστων

Μετασχηματισμοί Προγραμμάτων στις Επαγωγικές Βάσεις Δεδομένων με χρήση του Νοηματικού Λογικού Προγραμματισμού

Διδακτορική Διατριβή

του

ΠΕΤΡΟΥ Μ. ΠΟΤΙΚΑ

Πτυχιούχου Τμήματος Πληροφορικής &
Τηλεπικοινωνιών Ε.Κ.Π.Α. (2000)

Τριμελής Συμβουλευτική Επιτροπή: Ευστάθιος Ζάχος, επιβλέπων
Τιμολέων Σελλής
Φώτω Αφράτη

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την 20η Ιουλίου 2006.

. . .	. . .	. . .
Ε. Ζάχος	Τ. Σελλής	Φ. Αφράτη
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγήτρια Ε.Μ.Π.
. . .	. . .	. . .
Π. Ροντογιάννης	Μ. Γεργατσούλης	Γ. Κολέτσος
Επ. Καθ. Παν. Αθηνών	Επ. Καθ. Ιόνιου Παν.	Αν. Καθηγητής Ε.Μ.Π.
	. . .	
	Ν. Παπασπύρου	
	Λέκτορας Ε.Μ.Π.	

Αθήνα, Ιούλιος 2006

. . .

ΠΕΤΡΟΣ Μ. ΠΟΤΙΚΑΣ
Διδάκτωρ Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Στους γονείς μου Μιχάλη και Ελένη.

Πέτρος Ποτίκας
Αθήνα, Ιούλιος 2006

ΠΡΟΛΟΓΟΣ

Η συγγραφή αυτής της διδακτορικής διατριβής ξεκίνησε ουσιαστικά το καλοκαίρι του 2000, όταν συμμετείχα στο ερευνητικό πρόγραμμα νέων ερευνητών ΠΕΝΕΔ '99 (Αρ. Συμβ. 99ΕΔ265): "Νοηματικές Εκτελέσιμες Γλώσσες και Εφαρμογές Πολυμέσων, Υπερμέσων και Εικονικής Πραγματικότητας" που είχε αναλάβει το Πανεπιστημίο Πειραιά, το Ε.Κ.Ε.Φ.Ε. "Δημόκριτος" και το Πανεπιστημίο Ιωαννίνων. Κατά τη διάρκεια αυτού του προγράμματος, είχα τη χαρά και την τύχη να ανακαλύψω μια πολύ ενδιάφερουσα ερευνητική περιοχή που συνδύαζε τη θεωρία με την πράξη. Επιπλέον, είχα την τύχη να συνεργαστώ με δύο εξαίρετους επιστήμονες και δασκάλους, που με έμαθαν να αγαπώ την έρευνα. Πάνω από όλα όμως, τους αισθανόμουν σαν δικούς μου ανθρώπους που κάθε στιγμή ήταν δίπλα μου έτοιμοι να με στηρίξουν. Θα ήθελα, λοιπόν, να ευχαριστήσω τον Επίκουρο Καθηγητή του Τμήματος Πληροφορικής και Τηλεπικοινωνιών του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών Παναγιώτη Ροντογιάννη και τον Επίκουρο Καθηγητή του Τμήματος Αρχειονομίας και Βιβλιοθηκονομίας του Ιόνιου Πανεπιστήμιου Μανόλη Γεργατσούλη για όλα όσα με έχουν διδάξει.

Επιπλέον, θα ήθελα να ευχαριστήσω όλα τα μέλη του Εργαστηρίου Λογικής και Υπολογισμών (CoReLab) του Εθνικού Μετσόβιου Πολυτεχνείου που είναι αυτή την εποχή στο εργαστήριο αλλά και όσους συνεργάζονται ή έχουν συνεργαστεί με αυτό, για τη συνεργασία και τις πολύτιμες ανταλλαγές απόψεων που είχαμε όλα αυτά τα χρόνια. Θα ήθελα λοιπόν να ευχαριστήσω το Λέκτορα της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών Άρη Παγουρτζή που πάντα με συμβούλευε κατά τη διάρκεια των μεταπτυχιαών μου σπουδών, το Λέκτορα του Πανεπιστημίου Ιωαννίνων Χρήστο Νομικό, τον Επίκουρο καθηγητή του Πανεπιστημίου Πελοπονήσσου Κώστα Κούτρα, τους Ευριπίδη Μάρκου, Χριστόδουλο Φραγκουδάκη, Πέτρο Πετρόπουλο, Κατερίνα Ποτίκα, Αντώνη Καβαρνό, Όλγα Τσιαντούλη, Πάνο Χείλαρη, Νίκο Λεονάρδο, Βαγγέλη Μπαμπά, Γεωργία Καούρη, Μιχάλη Λαμπή. Εκτός από τα κοινά επιστημονικά ενδιαφέροντα που είχαμε, μοιραστήκαμε και πολλές προσωπικές στιγμές.

Τέλος, θα ήθελα να ευχαριστήσω ιδιαιτέρως τον επιβλέποντα καθηγητή μου Καθηγητή της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου Ευστάθιο Ζάχο, ο οποίος όλα αυτά τα χρόνια με επέβλεψε και με στήριξε στο διδακτορικό μου και αποτέλεσε έναν μεγάλο δάσκαλο για μένα με την ευρύτητα του πνεύματος του. Επίσης, θα ήθελα να ευχαριστήσω ιδιαίτερα τα άλλα δύο μέλη της Τριμελούς Συμβουλευτικής Επιτροπής, Καθηγητή της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών Τίμο Σελλή και την Καθηγήτρια της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών Φώτω Αφράτη για την καθοδήγηση και συμπαράσταση τους στην περάτωση αυτού του επίπονου και δύσκολου εγχειρήματος. Ακόμα ευχαριστώ τα υπόλοιπα μέλη της Επταμελούς Εξεταστικής Επιτροπής, τον Αναπληρωτή Καθηγητή της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών Γιώργο Κολέτσο και το Λέκτορα της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών Νικόλαο Παπασπύρου.

Κλείνοντας, θα ήθελα να εκφράσω τις ευχαριστίες μου στους γονείς μου Μιχάλη

και Ελένη που με στήριξαν όλα αυτά τα χρόνια των σπουδών μου καθώς επίσης και την αδελφή μου Κατερίνα που πάντα με συμβούλευε και με στήριζε κατά τη διάρκεια των μαθητικών και φοιτητικών μου χρόνων.

Αθήνα, Ιούλιος 2006

ΠΕΡΙΛΗΨΗ

Η εργασία αυτή μελετάει τη σχέση ανάμεσα σε προγράμματα Datalog και προγράμματα του νοηματικού λογικού προγραμματισμού (intensional logic programming). Πιο συγκεκριμένα, παρουσιάζονται μετασχηματισμοί κλάσεων προγραμμάτων Datalog σε προγράμματα Datalog$_{nS}$ και Choice Datalog$_{nS}$, που είναι γλώσσες χρονικού λογικού προγραμματισμού. Οι νέοι μετασχηματισμοί ανήκουν στην κατηγορία βελτιστοποίησης προγραμμάτων σε επαγωγικές βάσεις δεδομένων *μέσω της διάδοσης των τιμών*, στην οποία οι τιμές εισόδου του στόχου του αρχικού προγράμματος διαδίδονται έτσι ώστε να περιορίζονται τα παραγόμενα άτομα κατά την από-κάτω-προς-τα-πάνω (bottom-up) εκτέλεση του νέου προγράμματος. Η από-κάτω-προς-τα-πάνω (bottom-up) αποτίμηση του τελικού προγράμματος μιμείται την από-πάνω-προς-τα-κάτω (top-down) αποτίμηση του αρχικού προγράμματος, συνδυάζοντας τα πλεονεκτήματα των δύο αυτών τρόπων αποτίμησης. Παράλληλα, το αρχικό και το τελικό πρόγραμμα είναι σημασιολογικά ισοδύναμα.

Η αφετηρία των μετασχηματισμών που παρουσιάζονται είναι ο μετασχηματισμός διακλαδιζόμενου χρόνου (branching-time transformation) [37, 36]. Η κλάση των Datalog προγραμμάτων στην οποία μπορεί να εφαρμοστεί ο μετασχηματισμός διακλαδιζόμενου χρόνου, είναι τα Chain Datalog προγράμματα, μια κλάση που έχει ιδιαίτερο ενδιαφέρον [55, 19] και αντιστοιχεί στις γραμματικές χωρίς συμφραζόμενα (context free grammars). Ο μετασχηματισμός επεκτείνεται ώστε να εφαρμόζεται σε μια ευρύτερη κλάση προγραμμάτων Datalog. Αυτή η κλάση προγραμμάτων, που αποτελεί ένα γνήσιο υπερσύνολο των Chain Datalog προγραμμάτων, είναι αυτή στην οποία επιτρέπονται πολλαπλές καταναλώσεις μεταβλητών. Για το χειρισμό αυτών των προγραμμάτων [31, 30, 32], χρειάζεται να χρησιμοποιηθούν γλώσσες που υποστηρίζουν μη-ντετερμινιστικές κατασκευές, ειδικότερα τα *κατηγορήματα επιλογής (choice predicates)* [27]. Κύριο χαρακτηριστικό τους είναι ότι σε ένα συγκεκριμένο περιβάλλον (context), παίρνουν μοναδική τιμή.

Καταρχήν, ορίζονται οι αρχικές και οι τελικές γλώσσες των μετασχηματισμών καθώς και η σημασιολογία τους. Κατόπιν παρουσιάζονται οι αλγόριθμοι των νέων μετασχηματισμών και αποδεικνύεται η ορθότητά τους. Στη συνέχεια προτείνονται εκλεπτύνσεις των αλγορίθμων που αυξάνουν την απόδοση της bottom-up εκτέλεσης των τελικών προγραμμάτων. Επιπλέον, παρουσιάζεται μια βελτιστοποιημένη αποδεικτική διαδικασία που εφαρμόζεται στα τελικά προγράμματα, η οποία πάντοτε τερματίζει. Τέλος, συγκρίνονται οι νέοι μετασχηματισμοί με τους ήδη υπάρχοντες και τα αποτελέσματα αυτής της σύγκρισης δείχνουν πως αυτοί έχουν πρακτική σπουδαιότητα, εκτός από την θεωρητική τους σημασία.

Κλείνοντας, παραθέτονται τα αποτελέσματα της διδακτορικής διατριβής και προτείνονται μελλοντικές ερευνητικές κατευθύνσεις. Ενδιαφέρον θα είχε να μελετηθούν προγράμματα στα οποία έχουμε διαφορετικούς τρόπους καταναλώσεων μεταβλητών (π.χ. περισσότερα ορίσματα εξόδου) ώστε να εφαρμόζονται σε όλη τη Datalog και προγράμματα με συναρτησιακά σύμβολα (γενικά λογικά προγράμματα). Ακόμα, η υλοποίηση του bottom-up υπολογισμού των τελικών προγραμμάτων θα μας επέτρεπε να πειραματιστούμε με ένα μεγάλο αριθμό προγραμμάτων και βάσεων δεδομένων και να συγκρίνουμε τους προτεινόμενους μετασχηματισμούς με άλλες γνωστές τεχνικές της περιοχής, αναδεικνύοντας τα πλεονεκτήματα και τα μειονεκτήματα της κάθε προσέγγισης. Τέλος, θα πρέπει να εξεταστεί η δυνατότητα εφαρμογής αυτών των μετασχηματισμών σε νέες περιοχές που σχετίζονται με την εξαγωγή γνώσης με χρήση της μαθηματικής λογικής, όπως είναι ο Σημασιολογικός Ιστός (Semantic Web).

ABSTRACT

In this thesis the relationship between intensional logic programming and deductive databases is studied. More particular, a wide class of Datalog programs is transformed into equivalent Datalog$_{nS}$ or Choice Datalog$_{nS}$ programs. These two languages are temporal logic languages. The new transformation technique belongs to a well known category of optimizations in the area of deductive databases, the *value propagating transformations*, where the input values of the goal are propagated in order to restrict the generation of atoms in the bottom-up computation of the target program. Actually, the bottom-up evaluation of the rewritten program mimics the top-down evaluation of the initial program. In this way, the advantages of both evaluation methods are combined, while the equivalence between the initial and the rewritten program is retained.

The proposed transformation is an extension of the branching-time transformation [37, 36] that was proposed to apply to Chain Datalog programs. This is a very interesting class [55, 19] that corresponds to the context free grammars. The branching-time transformation is extended to apply to a proper superset of Chain Datalog. The new class of Datalog programs are moded Datalog programs, where each variable can be consumed in more than one following atoms. In order to handle this class of programs, a new technique was proposed [31, 30, 32], that uses non-deterministic constructs from the area of intensional logic programming, particularly *choice predicates* [27]. The main feature of choice predicates is the preservation of a unique value under a given context.

First the source and the target languages of the transformation are introduced and their semantics are given. The transformation is defined in an algorithmic way and its correctness is proven. An optimized proof procedure is proposed, which terminates for all target programs. Some refinements of the target program are proposed in order to speedup their execution. Finally, the proposed method is compared with its competitors and the results indicate that it has not only theoretical value, but also practical significance.

Finally, the conclusions of the thesis and some possible future directions are given. There could be extensions of the technique so that it can be applied to capture all Datalog programs and programs with function symbols. Another issue that arises, is the implementation of all methods, so that a detailed comparison could be performed. Moreover, the applicability of this transformation to new areas of knowledge representation using logic, like the Semantic Web, has to be considered.

Περιεχόμενα

Κατάλογος σχημάτων

Κατάλογος Συντμήσεων

ΒΔ	:	Βάση Δεδομένων
ΕΜΠ	:	Εθνικό Μετσόβιο Πολυτεχνείο
Λ.Π.	:	Λογικός Προγραμματισμός

Κεφάλαιο 1

Εισαγωγή

1.1 Λογικός Προγραμματισμός

1.1.1 Εισαγωγικά

Αρχικά, ο Λογικός Προγραμματισμός (Λ.Π.) χρησιμοποιήθηκε για την απόδειξη θεωρημάτων (theorem proving) και την αναπαράσταση γνώσης (knowledge representation) στην Τεχνητή Νοημοσύνη (Artificial Intelligence). Ωστόσο, από τις ιδέες των Gödel το '30 και Hilbert στις αρχές του περασμένου αιώνα, ο Kowalski [22] οδηγήθηκε στην σκέψη, να χρησιμοποιήσει τον δηλωτικό (declarative) αυτό τρόπο για να παράγει και αποτελέσματα (δηλ. να κάνει υπολογισμούς) εκτός από το να αναπαριστά γνώση. Ο Λ.Π. αναπτύχθηκε στις αρχές του '70 κυρίως από τους Kowalski, Colmerauer (δημιουργός της Prolog) και βασίστηκε στη μέθοδο της ανάλυσης (resolution principle) του Robinson [35]. Η σύνταξή του βασίζεται στην λογική πρώτης τάξης ή πρωτοβάθμια ή κατηγορηματική λογική (first order logic) που αρχικά παρουσιάστηκε από τον Frege και αργότερα συμπληρώθηκε από τους Peano και Russel. Η ιδέα ήταν να χρησιμοποιηθεί η μαθηματική λογική εκτός από το σχεδιασμό ολοκληρωμένων κυκλωμάτων για υπολογιστές και για σχεδιασμό και επαλήθευση (reasoning) προγραμμάτων και συστημάτων [6], για να προγραμματίζουμε. Γενικά, μπορούμε να σκεφτούμε ότι το πρόγραμμα είναι το σύνολο των αξιωμάτων (δηλ. αυτά που εμείς θεωρούμε ότι ισχύουν) και ο υπολογισμός είναι η κατασκευαστική απόδειξη μιας ερώτησης μέσα από το πρόγραμμα.

Ένα πρόγραμμα σε καθαρό λογικό προγραμματισμό αποτελείται από ένα σύνολο από κανόνες (rules), που επίσης ονομάζονται οριστικές (definite) Horn προτάσεις (clauses). Κάθε κανόνας έχει τη μορφή $head \leftarrow body$, όπου το άτομο *head* είναι η κεφαλή του κανόνα και *body* είναι το σώμα του κανόνα, που αποτελείται από ένα σύνολο από άτομα που βρίσκονται σε σύζευξη και διαισθητικά λέει πως η κεφαλή ισχύει αν ισχύει το σώμα, δηλ. αν ισχύει κάθε άτομο στο σώμα. Ο μηχανισμός που μας επιτρέπει να κάνουμε ερωτήσεις και να παίρνουμε απαντήσεις, δηλ. να κάνουμε υπολογισμούς είναι απλός, αλλά ταυτόχρονα πολύ ισχυρός, και ονομάζεται ενοποίηση (unification). Το κύριο χαρακτηριστικό, που τον κάνει και τόσο ελκυστικό, είναι ο διαχωρισμός της λογικής ("τι" κομμάτι) από τον έλεγχο ("πως" κομμάτι). Κατά τον Kowalski: "Αλγόριθμος = Λογική + Έλεγχος". Αυτό επιτρέπει την ευκολότερη ανάπτυξη κώδικα, προσφέρει ευκολίες διαίρεσης του προγράμματος σε υπομονάδες (modular) καθώς επίσης και την ευκολότερη επαλήθευση προγραμμάτων (program verification). Επόμενο ήταν, ότι με την υλοποίηση ενός συστήματος που θα μπορούσες να προγραμματίσεις σε λογικό προγραμματισμό, πολλοί θα ήταν εκείνοι που θα ανέπτυσσαν διάφορες εφαρμογές βασισμένες σε αυτή τη γλώσσα προγραμματισμού. Η δημιουργία της γλώσσας λογικού προγραμματισμού Prolog (PROgramming in LOGic) από τον Colmerauer και την ομάδα του [1], έφερε επανάσταση στο λογικό προγραμματισμό και γενικότερα στον προγραμματισμό. Αργότερα με την ανάπτυξη της μηχανής του Warren [53], της επονομαζόμενης WAM, ο Λ.Π. μετατράπηκε σε μια γλώσσα προγραμματισμού ευρείας χρήσης και γενικής χρησιμότητας.

1.1.2 Συντακτικό

Το συντακτικό του Λ.Π. περιλαμβάνει έννοιες από την πρωτοβάθμια λογική (first order logic)[14] και για μια πιο λεπτομερή παρουσίαση μπορεί κανείς να ανατρέξει στα [23, 46, 10].

Όπως και στη λογική, το αλφάβητο μας αποτελείται από *σταθερές, συναρτησιακά* σύμβολα, που καθένα έχει έναν βαθμό (arity) $n > 0$ και *κατηγορήματα*, που καθένα έχει κάποιο βαθμό $m \geq 0$, και αυτά είναι που καθορίζουν την γλώσσα μας L. Αν η γλώσσα δεν περιέχει συναρτησιακά σύμβολα, τότε λέγεται *ελεύθερη συναρτησιακών συμβόλων (function-free language).* Ένα παράδειγμα τέτοιας γλώσσας είναι η **Datalog**. Στη συνέχεια της εργασίας υιοθετούμε τον ακόλουθο συμβολισμό: οι *σταθερές* συμβολίζονται με αριθμούς ή πεζά λατινικά γράμματα, όπως a, b, c και διανύσματα σταθερών με $\vec{e}$. Οι *μεταβλητές* με κεφαλαία λατινικά γράμματα, όπως X, Y, Z και διανύσματα μεταβλητών με $\vec{v}$, τα κατηγορήματα με πεζά λατινικά γράμματα, όπως p, q, r, s, ενώ θα χρησιμοποιηθούν και εκδόσεις τους με δείκτες ή με εκθέτες. Τέλος, πεζά ελληνικά γράμματα, όπως θ, θα χρησιμοποιηθούν για να συμβολίζουμε αντικαταστάσεις. Όταν χρησιμοποιούμε έντονα γράμματα δηλώνουμε ένα διάνυσμα συμβόλων από τον αντίστοιχο τύπο.

Ένας *όρος (term)* είναι είτε μία σταθερά, είτε μία μεταβλητή είτε αν f είναι ένα συναρτησιακό σύμβολο με βαθμό n, και $t_1, t_2, ..., t_n$ είναι όροι, τότε και ο $f(t_1, t_2, ..., t_n)$ είναι όρος. Ένας όρος λέγεται *αποτιμημένος (ground)*, αν δεν περιέχει μεταβλητές. Ένα *άτομο (atom)* είναι μία φόρμουλα $p(t_1, ..., t_n)$, όπου p είναι ένα κατηγορηματικό σύμβολο με βαθμό n και $t_1, ..., t_n$ είναι όροι. Ένα *λεκτικό ή λέκτημα (literal)* είναι ένα άτομο ή η άρνηση ενός ατόμου. Ένα άτομο λέγεται *αποτιμημένο (ground atom)*, αν δεν περιέχει μεταβλητές. Όπως θα δούμε, μετά το μετασχηματισμό στην τελική μορφή τους, οι φόρμουλες στον Λ.Π. είναι ειδικής μορφής και επιτρέπονται μόνο οι σύνδεσμοι $\leftarrow$ και ο $\wedge$. Οι προτάσεις είναι της μορφής:

$$A_1 \vee ... \vee A_m \vee \neg B_1 \vee \neg B_2 ... \vee \neg B_n.$$

όπου $n, m \geq 0$ και τα $A_1, ..., A_m, B_1, ..., B_n$ είναι άτομα.
Στην ειδική περίπτωση που είναι της μορφής:

$$A \vee \neg B_1 \vee \neg B_2 ... \vee \neg B_n.$$

ισοδύναμα γράφεται στη παρακάτω μορφή:

$$A \leftarrow B_1, B_2, ..., B_n.$$

Οι προτάσεις της παραπάνω μορφής λέγονται *προτάσεις Horn (Horn clauses)* *ή οριστικές προτάσεις (definite clauses)*, όταν δηλ. το $m = 1$. Ειδικότερα, στην περίπτωση των οριστικών προτάσεων, το A λέγεται *κεφαλή* της πρότασης και τα $B_1, ..., B_n$, που είναι σε σύζευξη ($B_1 \wedge ... \wedge B_n$) αποτελούν το *σώμα (body)* της πρότασης. Διαισθητικά κάθε πρόταση μας λέει ότι για να ισχύει το A, πρέπει να ισχύει και το $B_1, ...$ και το B_n. Ένας κανόνας της μορφής $A \leftarrow$, δηλ. το σώμα είναι κενό, λέγεται *γεγονός (fact) ή μοναδιαία πρόταση (unit clause)* και αν το A δεν περιέχει μεταβλητές, τότε λέγεται *αποτιμημένο γεγονός (ground fact) ή αποτιμημένη μοναδιαία πρόταση (unit ground fact)*. Μία οριστική πρόταση της μορφής $\leftarrow B_1, ..., B_n$, δηλ. με άδεια κεφαλή, λέγεται *στόχος (goal)*. Ένα *οριστικό λογικό πρόγραμμα (definite logic program)* είναι ένα πεπερασμένο σύνολο από Horn προτάσεις (οριστικές προτάσεις).

1.1.3 Σημασιολογία

Στη σημασιολογία υπάρχουν τρεις προσεγγίσεις, οι οποίες αναφέρονται στο νόημα του προγράμματος, αλλά καθεμιά από διαφορετική σκοπιά. Στην λογική, ένα σύνολο προτάσεων μπορεί να έχει πολλά μοντέλα (άπειρα). Το ζήτημα που τίθεται στον λογικό προγραμματισμό είναι ποιο από τα μοντέλα αυτά είναι το νόημα του προγράμματος, ποια είναι η απάντηση που θέλει να πάρει ο προγραμματιστής από το πρόγραμμα που γράφει. Όπως έχει καθιερωθεί στο λογικό προγραμματισμό, το νόημα ενός προγράμματος είναι το *ελάχιστο μοντέλο Herbrand (minimum Herbrand model)*. Όπως απέδειξαν οι Kowalski και VanEmden το 1976 ([50]), και οι τρεις σημασιολογίες αναφέρονται σε αυτό το μοντέλο. Το συμπέρασμα τελικά είναι, πως οι προσεγγίσεις αυτές είναι ισοδύναμες, δηλ. δίνουν το νόημα του προγράμματος. Παρακάτω θα δούμε αναλυτικότερα ποιο είναι το ελάχιστο μοντέλο Herbrand ενός λογικού προγράμματος.

1.1.4 Μοντελοθεωρητική σημασιολογία (Modeltheoretic semantics)

Το *σύμπαν Herbrand (Herbrand universe)* U_L του προγράμματος P σε μια πρωτοβάθμια γλώσσα L, είναι το σύνολο των αποτιμημένων όρων που μπορούν να σχηματιστούν από τις σταθερές και τα συναρτησιακά σύμβολα. Αν υπάρχουν συναρτησιακά σύμβολα, τότε το Herbrand universe είναι άπειρο. Αν δεν υπάρχει σταθερά, τότε εισάγουμε εμείς μία. Η *βάση Herbrand (Herbrand base)* B_L είναι το σύνολο των αποτιμημένων ατόμων που σχηματίζονται από τα κατηγορηματικά σύμβολα με στοιχεία του σύμπαντος Herbrand. Αν υπάρχουν συναρτησιακά σύμβολα, τότε η βάση Herbrand είναι άπειρη. Μια *Herbrand ερμηνεία (Herbrand interpretation)* είναι οποιοδήποτε υποσύνολο της βάσης Herbrand. Ένα *μοντέλο Herbrand (Herbrand model)* είναι μία Herbrand ερμηνεία του P, έτσι ώστε για κάθε κανόνα $A \leftarrow B_1, ..., B_n$ η ερμηνεία να ικανοποιεί το κανόνα.

Αν S είναι ένα σύνολο από κλειστούς τύπους (δηλ. χωρίς ελεύθερες μεταβλητές) και F είναι ένας κλειστός τύπος, τότε το F είναι *λογικό επακόλουθο (logical consequence)* του S και συμβολίζεται με $S \models F$, αν κάθε μοντέλο του S είναι και μοντέλο του F. Τότε αποδεικνύεται ότι το $S \cup \{\neg F\}$ δεν έχει μοντέλο. Αν P οριστικό πρόγραμμα σε μια γλώσσα L, αποδεικνύεται ότι:

$$M_P = \{A \in B_L \mid P \models A\}$$

όπου M_P είναι το ελάχιστο μοντέλο του P. Δηλ. το ελάχιστο μοντέλο ενός οριστικού προγράμματος αποτελείται από όλα τα λογικά επακόλουθα του προγράμματος.

Η απόφαση να παίρνουμε το ελάχιστο μοντέλο, αντί κάποιου τυχαίου μοντέλου, είναι απόρροια της *υπόθεσης του κλειστού κόσμου (Closed World Assumption-CWA)*. Με βάση αυτή την υπόθεση, μόνο τα γεγονότα που έχουν καταγραφεί στο πρόγραμμα ισχύουν, ενώ οτιδήποτε άλλο δεν ισχύει. Πιο συγκεκριμένα για τις βάσεις δεδομένων, θεωρούμε πως η βάση έχει καταγράψει όλα τα γεγονότα που ισχύουν, ενώ όσα δεν είναι καταχωρημένα στη βάση δεν ισχύουν. Από την αντίστοιχη πλευρά της λειτουργικής σημασιολογίας (δηλ. της θεωρίας αποδείξεων), τα μόνα γεγονότα που ισχύουν είναι αυτά που μπορούν να αποδειχθούν. Κατά συνέπεια, για να θεωρήσουμε πως η άρνηση ενός ατόμου $r(\vec{a})$ ισχύει, τότε πρέπει να δείξουμε ότι $\Sigma \not\models r(\vec{a})$. Τέλος, μπορούμε να αναφέρουμε πως η υπόθεση κλειστού κόσμου είναι παρεμφερής με την φιλοσοφία της *άρνησης ως αποτυχία (negation as failure)*.

Παράδειγμα 1.1.1 *Έστω το ακόλουθο πρόγραμμα P:*

```
p(X) ← q(X).
q(a).
r(d).
```

Ένα μοντέλο αυτού του προγράμματος είναι: $M = \{\mathtt{p(a)}, \mathtt{q(a)}, \mathtt{r(d)}, \mathtt{r(a)}\}$

Ωστόσο αυτό δεν είναι το ελάχιστο μοντέλο του προγράμματος P. Το ελάχιστο μοντέλο Herbrand είναι: $M_P = \{\mathtt{p(a)}, \mathtt{q(a)}, \mathtt{r(d)}\}$.

Ένα λογικό πρόγραμμα που όλα τα κατηγορηματικά του σύμβολα έχουν βαθμό 0 λέγεται *προτασιακό λογικό πρόγραμμα*. Όπως και στη πρωτοβάθμια λογική, ένας κλειστός τύπος είναι αληθής σε μια ερμηνεία I αν:

- είναι ένα πλήρως αποτιμημένο άτομο A και $A \in I$,
- είναι της μορφής $\neg F$ και ο τύπος F δεν είναι αληθής στην I,
- είναι της μορφής $F \wedge G$ και οι τύποι F και G είναι αληθείς στην I,
- είναι της μορφής $F \vee G$ και κάποιος από τους τύπους F ή G είναι αληθής στην I,
- είναι της μορφής $F \rightarrow G$ (ή $G \leftarrow F$) και είτε ο τύπος F είναι ψευδής στην I είτε ο G είναι αληθής στην I,

- είναι της μορφής $\forall x\ F$ και για όλα τα $d \in U_L$ αν αντικατασταθεί η μεταβλητή x στον τύπο F με d, οι τύποι που προκύπτουν να είναι αληθείς στην I,
- είναι της μορφής $\exists x\ F$ και υπάρχει $d \in U_L$ ώστε αν αντικατασταθεί η μεταβλητή x στον τύπο F με d, ο τύπος που προκύπτει να είναι αληθής στην I,

Ένας κλειστός τύπος είναι ψευδής σε μία ερμηνεία I αν δεν είναι αληθής στην I.

Ακόμα, ορίζουμε την *αποτίμηση (grounding)* μιας πρότασης r σε μία γλώσσα L ως το σύνολο όλων των προτάσεων που προκύπτουν από την r με όλες τις πιθανές αντικαταστάσεις στοιχείων του Herbrand universe U_L για τις μεταβλητές της r. Η αποτίμηση ενός προγράμματος προκύπτει από την ένωση όλων των αποτιμήσεων των προτάσεων του.

1.1.5 Σημασιολογία σταθερού σημείου (Fixpoint semantics)

Έστω P ένα οριστικό λογικό πρόγραμμα. Ορίζεται ένας τελεστής $T_P : 2^{B_P} \rightarrow 2^{B_P}$, όπου 2^{B_P} είναι το σύνολο όλων των Herbrand ερμηνειών του προγράμματος P, ως εξής:

$T_P(I) = \{A_0 \in B_P \mid P$ περιλαμβάνει έναν αποτιμημένο κανόνα $A_0 \leftarrow A_1, ..., A_m$ τέτοιο ώστε $\{A_1, ..., A_m\} \subseteq I\}$

Ο τελεστής αυτός ονομάζεται *τελεστής άμεσου επακόλουθου (immediate consequence operator)* και διαισθητικά σε κάθε εφαρμογή του παράγει μία νέα ερμηνεία η οποία αποτελείται από τα άτομα που ήδη υπήρχαν συν τα άτομα που παράγονται από τους κανόνες που έχουν στο σώμα τους αυτά τα άτομα. Επομένως, αυτός ο τελεστής είναι μονότονος και από το θεώρημα Tarski-Knaster [57] έχει ένα ελάχιστο σταθερό σημείο (least fixpoint). Επιπλέον, ο τελεστής αυτός είναι συνεχής και από το θεώρημα του Tarski [57] είναι το όριο της ακολουθίας:

$$\begin{cases} T_P \uparrow 0 = \emptyset, \\ T_P \uparrow i+1 = T_P(T_P \uparrow i),\ i \geq 0 \end{cases}$$

Σύμφωνα με τη σημασιολογία του σταθερού σημείου έχουμε $M_P = T_P \uparrow \omega$, όπου ω είναι ο πληθικός αριθμός του συνόλου $\mathcal{N}$ των φυσικών αριθμών και M_P το ελάχιστο μοντέλο του P.

Σημείωση: Με τον τελεστή T_P έχουμε ένα κατασκευαστικό τρόπο (αλγόριθμο) για να βρίσκουμε το ελάχιστο μοντέλο ενός λογικού προγράμματος δηλ. ένα τρόπο από-κάτω-προς-τα-πάνω (bottom-up) για να υπολογίζουμε το ελάχιστο Herbrand μοντέλο ενός προγράμματος.

1.1.6 Λειτουργική σημασιολογία (Operational semantics)

Η τελευταία σημασιολογία βασίζεται στην μέθοδο της ανάλυσης (resolution) [35] για την απόδοση νοήματος σε ένα λογικό πρόγραμμα και στηρίζεται στην παραγωγή γνώσης από ήδη υπάρχουσα με την εφαρμογή ενός συμπερασματικού κανόνα (inference rule) ένα πεπερασμένο αριθμό φορών. Με την ανάλυση (resolution) μπορούμε να ελέγχουμε αν ένα άτομο (αποτιμημένο ή μη) είναι λογικό επακόλουθο ενός προγράμματος χωρίς να υπολογίζουμε το ελάχιστο μοντέλο Herbrand. Στη συνέχεια ορίζουμε μερικές έννοιες που είναι βασικές στην λειτουργική σημασιολογία.

- *Αντικατάσταση (Substitution)*: Μία αντικατάσταση θ είναι ένα πεπερασμένο σύνολο της μορφής $\{V_1/t_1, ..., V_n/t_n\}$, όπου τα V_i είναι μεταβλητές και τα t_i είναι όροι (κάθε t_i είναι διαφορετικό από το αντίστοιχο V_i και όλα τα V_i είναι διαφορετικά μεταξύ τους). Δηλ. μια αντικατάσταση ορίζει τα V_i να έχουν πάρει την τιμή t_i. Η κενή αντικατάσταση ορίζεται από το κενό σύνολο.

- *Στιγμιότυπο (Instance)*: Το E είναι ένας όρος, λεκτικό, σύζευξη ή διάζευξη λεκτικών, το $E\theta$ λέγεται στιγμιότυπο του E μέσω της αντικατάστασης $\theta = \{V_1/t_1, ..., V_n/t_n\}$ αν στο E αντικατασταθούν *ταυτόχρονα οι εμφανίσεις των μεταβλητών* V_i *με τα* t_i.

  π.χ. $E = f(x, y)$, $\theta = \{x/g(y), y/5\}$, *τότε* $E\theta = f(g(y), 5)$

- Παραλλαγή (Variant)*: Αν E και F είναι όροι, λεκτικά, συζεύξεις ή διαζεύξεις λεκτικών, το E λέγεται παραλλαγή του F (ή το F παραλλαγή του E) αν υπάρχουν αντικαταστάσεις θ και σ τέτοιες ώστε $E = F\theta$ και $F = E\sigma$. Δηλ. ουσιαστικά όταν δύο όροι είναι ίδιοι απλά έχουν διαφορετικά ονόματα μεταβλητών.*

- Σύνθεση (Composition) αντικαταστάσεων*: Αν $\theta = \{u_1/s_1, ..., u_m/s_m\}$ και $\sigma = \{v_1/t_1, ..., v_n/t_n\}$ είναι δύο αντικαταστάσεις, η σύνθεσή τους $\theta\sigma$ είναι μία νέα αντικατάσταση $\{u_1/s_1\sigma, ..., u_m/s_m\sigma, v_1/t_1, ..., v_n/t_n\}$, διαγράφοντας από αυτήν κάθε $u_i/s_i\sigma$ για το οποίο $u_i = s_i\sigma$ και κάθε v_j/t_j για το οποίο $v_i \in \{u_1, ..., u_m\}$.*

1.1.7 Ενοποίηση (Unification)

Οι παραπάνω έννοιες χρειάζονται για να ορίσουμε μια από τις βασικότερες έννοιες του λογικού προγραμματισμού, την *ενοποίηση (unification)*. Η ενοποίηση αποτελεί το κυριότερο μέρος μιας υλοποίησης ενός συστήματος λογικού προγραμματισμού ή αυτομάτων αποδείξεων. Το γενικότερο πρόβλημα είναι η επίλυση εξισώσεων μεταξύ όρων.

Τα άτομα ή οι όροι s και t λέγονται ενοποιήσιμα αν υπάρχει αντικατάσταση θ που να κάνει τα s και t ίσα, δηλ. οι όροι $s\theta$ και $t\theta$ να είναι ίδιοι. Ειδικότερα, αν $S = \{E_1, ..., E_n\}$ είναι ένα σύνολο όρων ή ατόμων, μία αντικατάσταση θ λέγεται ενοποιητής (unifier) για το S αν το $S\theta = \{E_1\theta, ..., E_n\theta\}$ είναι μονοσύνολο. Ένας ενοποιητής θ για το S είναι ο γενικότερος ενοποιητής (*most general unifier-mgu*) για το S, αν για κάθε ενοποιητή σ του S, υπάρχει αντικατάσταση γ τέτοια ώστε $\sigma = \theta\gamma$.

Παράδειγμα 1.1.2 *Έστω τα άτομα* $q(1, X_1, X_2)$ *και* $q(X_3, X_4, X_5)$. *Ένας ενοποιητής των δύο αυτών ατόμων είναι ο* θ, *όπου* $\theta(X_3) = 1, \theta(X_1) = \theta(X_4) = \theta(X_2) =$

$\theta(X_5)$. Ωστόσο, αυτός δεν είναι ο γενικότερος ενοποιητής, καθώς δεσμεύει τις τιμές των X_1, X_2, X_4, X_5 περισσότερο από ότι πρέπει.

Η διαδικασία υπολογισμού του mgu λέγεται ενοποίηση (unification). Το πρόβλημα ενοποίησης είναι το ακόλουθο πρόβλημα απόφασης: δίνονται όροι s και t, είναι ενοποιήσιμοι; Ο Robinson το 1965 ([35]) περιέγραψε έναν αλγόριθμο που λύνει αυτό το πρόβλημα και αν η απάντηση είναι καταφατική, υπολογίζει ένα γενικότερο ενοποιητή των δύο όρων. Ο αλγόριθμος αυτός έχει εκθετικό χρόνο και χώρο και αυτό κυρίως λόγω του τρόπου αναπαράστασης των όρων ως strings από σύμβολα. Με καλύτερη αναπαράσταση (π.χ. με κατευθυνόμενους ακυκλικούς γράφους, DAG), αυτός ο αλγόριθμος μπορεί να βελτιωθεί και να φτάσει σε γραμμικό χρόνο. Ένας καλύτερος αλγόριθμος είναι των Martelli και Montanari [24]. Ο αλγόριθμος είναι ο ακόλουθος:

Μη ντετερμινιστικά επίλεξε από το σύνολο των εξισώσεων μία εξίσωση της μορφής που είναι παρακάτω και εκτέλεσε την αντίστοιχη ενέργεια.

(1) $f(s_1, ..., s_n) = f(t_1, ..., t_n)$	αντικατάστησε με $s_1 = t_1, ..., s_n = t_n$
(2) $f(s_1, ..., s_n) = g(t_1, ..., t_m)$	σταμάτα με αποτυχία,
(3) $x = x$	διέγραψε την εξίσωση,
(4) $t = x$ όπου t δεν είναι μεταβλητή	αντικατάστησε με την εξίσωση $x = t$
(5) $x = t$ όπου x εμφανίζεται στον t και x εμφανίζεται οπουδήποτε αλλού	εφάρμοσε την αντικατάσταση x/t σε όλες τις άλλες εξισώσεις
(6) $x = t$ όπου x εμφανίζεται στον t και x διαφέρει από το t	σταμάτα με αποτυχία

Ο αλγόριθμος αυτός έχει πολυπλοκότητα χρόνου $O(na(n))$, όπου a είναι η αντίστροφη συνάρτηση του Ackermann και για αυτό η a αυξάνει πολύ αργά. Για την ενοποίηση έχουμε το ακόλουθο θεώρημα:

Θεώρημα 1.1.1 *Το πρόβλημα της ενοποίησης είναι P-complete.*

Αυτό σημαίνει ότι είναι σχεδόν απίθανο να είναι στη κλάση ιεραρχίας NC, άρα είναι από τη φύση του σειριακό (μη-παραλληλοποιήσιμο)[28].

Ας δούμε όμως πώς λειτουργεί η ενοποίηση. Αν έχουμε ένα στόχο $G :\leftarrow A_1, ..., A_{k-1}, A_k, A_{k+1}, ..., A_n$, επιλέγουμε ένα άτομο A_k έτσι ώστε να ενοποιείται με μέγιστο κοινό ενοποιητή θ με την κεφαλή ενός κανόνα $C : A \leftarrow B_1, ..., B_m$, και από αυτή την ενοποίηση προκύπτει ένας νέος κανόνας $G' :\leftarrow A'_1, ..., A'_{k-1}, B'_1, ..., B'_m, A'_{k+1}, ..., A'_n$, όπου τα $A'_1, ...A'_{k-1}, B'_1, ..., B'_m, A'_{k+1}, ..., A'_n$ προκύπτουν από τα αντίστοιχα μη τονούμενα με εφαρμογή σε αυτά του mgu θ. Η γενική ιδέα εδώ είναι να εφαρμόσουμε αυτόν τον κανόνα ανάλυσης έναν αριθμό φορών (πεπερασμένο), ώστε να καταλήξουμε τελικά σε ένα κενό κανόνα, δηλ. σε κανόνα της μορφής $\leftarrow$ ή $\Box$, άρα σε αντίφαση, που ονομάζεται *SLD-απόρριψη (SLD-refutation)*, όποτε θα έχουμε αποδείξει αυτό που θέλουμε, με απαγωγή σε άτοπο. Με άλλα λόγια, αν έχεις μια *επερώτηση (query)*, $\leftarrow p(\vec{X})$ και θέλεις να βρεις τις απαντήσεις για αυτή την επερώτηση ή αν είναι αποτιμημένη, να βρεις αν ισχύει ή όχι, δεν έχεις παρά να πάρεις την άρνηση της $\neg p(\vec{X})$ και να καταλήξεις σε άτοπο. Ειδικά στην Prolog, ο κανόνας που εφαρμόζεται είναι να παίρνουμε το αριστερότερο άτομο από τον κανόνα C και λέγεται *SLD-resolution* από τις λέξεις Selecting (ένα literal χρησιμοποιώντας) Linear (στρατηγική, σε) Definite clauses. Με βάση τη λειτουργική σημασιολογία το σύνολο των $A \in B_L$ για τα οποία το $P \cup \{\leftarrow A\}$ έχει κάποια *SLD-απόρριψη* ονομάζεται *σύνολο επιτυχίας (success set)* του P, συμβολικά γράφεται $SS(P)$.

Με βάση τη λειτουργική σημασιολογία, το νόημα ενός Λ.Π. είναι το σύνολο επιτυχίας του που ταυτίζεται με το ελάχιστο μοντέλο Herbrand του.

Το συμπέρασμα, σύμφωνα με τους Kowalski ,VanEmden [50], είναι ότι οι *τρεις αυτές σημασιολογίες είναι ισοδύναμες*, δηλ. και οι τρεις συμφωνούν και μας δίνουν το ελάχιστο μοντέλο Herbrand ενός προγράμματος.

1.1.8 Datalog

Με την ανάπτυξη των βάσεων δεδομένων (databases), έγινε φανερή η σχέση που υπάρχει ανάμεσα στη λογική και στις βάσεις δεδομένων ([25]). Η λογική όχι μόνο μπορούσε να αναπαραστήσει γνώση, αλλά επιπλέον μπορούσε να χρησιμοποιηθεί σαν μια πολύ εκφραστική γλώσσα επερωτήσεων (query language). Έτσι άρχισαν να αναπτύσσονται οι *επαγωγικές βάσεις δεδομένων (deductive databases)*, γύρω στο '80, που εκτός από θεωρητικό ενδιαφέρον για την ανάπτυξη των κλασικών βάσεων δεδομένων (σχεσιακές βάσεις δεδομένων, relational databases), άρχισαν να αναπτύσσονται και πραγματικά συστήματα που χρησιμοποιήθηκαν σε εμπορικές εφαρμογές (Π.χ. Coral [33], XSB [44]). Η επιπλέον δύναμη που προσέφεραν, ήταν η δυνατότητα να εκφράζονται και αναδρομικά προγράμματα (π.χ. το μεταβατικό κλείσιμο (transitive closure) μιας σχέσης) σε αντίθεση με τις σχεσιακές βάσεις που δεν την υποστήριζαν (το πρότυπο SQL-99 υποστηρίζει ένα είδος αναδρομής). Η πιο διαδεδομένη γλώσσα επαγωγικών βάσεων δεδομένων είναι η γλώσσα Datalog ([11, 49, 2, 34, 3, 31, 25, 4, 5, 30]), στην οποία οι όροι και τα άτομα δεν έχουν συναρτησιακά σύμβολα, αλλά μόνο σταθερές και μεταβλητές, οι οποίες χρησιμοποιούνται για να σχηματιστεί το (πεπερασμένο) σύμπαν Herband ενός προγράμματος. Επιπλέον, δεν επιτρέπεται ισότητα και άρνηση στο σώμα των κανόνων. Επίσης, αναπτύχθηκαν και διάφορες παραλλαγές της Datalog, όπως Datalog$^{\neg}$, Datalog$_{1s}$, Datalog$_{nS}$ (Datalog με άρνηση, Datalog με successor και Datalog με ένα ακριβώς συναρτησιακό σύμβολο), βλέπε [12, 18].

Στον Λ.Π. οι σχέσεις των σχεσιακών βάσεων δεδομένων παριστάνονται με σύνολα από αποτιμημένα άτομα $p(a_1, ..., a_n)$. Όλες οι πλειάδες (tuples) με το ίδιο κατηγορηματικό σύμβολο p αποτελούν τον ορισμό μιας σχέσης. Το σύνολο όλων των κατηγορηματικών συμβόλων μαζί με μία πιθανώς άπειρη *περιοχή* ορίζουν το *σχήμα* της βάσης δεδομένων. Με κάθε βάση δεδομένων D αντιστοιχούμε ένα σύμπαν U_D που περιλαμβάνει όλες τις σταθερές που εμφανίζονται στην D, αλλά μπορεί και περισσότερες.

Τα κατηγορήματα που εμφανίζονται στη κεφαλή κάποιου κανόνα ενός προγράμματος P τα ονομάζουμε *IDB (intensional database predicates)*, ενώ τα κατηγορήματα που εμφανίζονται μόνο στο σώμα κάποιου κανόνα τα ονομάζουμε *EDB (extensional database predicates)*. Επιπλέον, θεωρούμε πως κατηγορήματα που εμφανίζονται μόνο στο σώμα κανόνων είναι EDB κατηγορήματα. Τα EDB κατηγορήματα απαρτίζουν τη βάση δεδομένων D. Ένα Datalog πρόγραμμα P μαζί με μία ΒΔ συμβολίζεται με P_D.

Ένα παράδειγμα ενός Datalog προγράμματος είναι το ακόλουθο:

Παράδειγμα 1.1.3 *Η ΒΔ D είναι η ακόλουθη:*

```
father(john, mary).
father(joe, kurt).
mother(mary, joe).
mother(tina, kurt).
```

Το σχήμα αυτής της βάσης δεδομένων D είναι $\{\texttt{father}, \texttt{mother}\}$ *με* $U_D = \{\texttt{john}, \texttt{joe}, \texttt{tina}, \texttt{mary}, \texttt{kurt}, \texttt{susan}\}$. *Παρατηρήστε ότι η susan δεν εμφανίζεται στο D.*

Θεωρούμε το datalog πρόγραμμα P που υπολογίζει όλους τους προγόνους της βάσης δεδομένων:

```
parent(X,Y) ← father(X,Y).
parent(X,Y) ← mother(X,Y).
ancestor(X,Y) ← parent(X,Y).
parent(X,Y) ← parent(X,Z),ancestor(Z,Y).
person(X).
```

Στο παράδειγμα τα κατηγορήματα `father`, `mother` *είναι κατηγορήματα* εισόδου *ή* EDB (extensional predicates) ή database predicates, *ενώ τα* `ancestor`, `person`, `parent` *είναι κατηγορήματα* εξόδου *ή* IDB (intensional predicates).

Το Datalog πρόγραμμα αυτό μαζί με την βάση δεδομένων D υπολογίζει την σχέση R με σχήμα {`ancestor`, `person`, `parent`} που περιλαμβάνει τις ακόλουθες πλειάδες:

```
parent(mary, joe).
parent(john, joe).
ancestor(mary, joe).
ancestor(john, joe).
person(john).
person(susan).
```

Γενικά, οι διαφορές που έχει η Datalog (επαγωγικές ΒΔ) σε σχέση με το Λ.Π. είναι οι παρακάτω:

- Η Datalog δεν έχει συναρτησιακά σύμβολα (άρα το Herbrand Universe και η Herbrand Base είναι πεπερασμένα).
- Ο υπολογισμός γίνεται από-κάτω-προς-τα-πάνω (bottom-up, set oriented) στις επαγωγικές ΒΔ, ενώ στο Λ.Π. γίνεται από πάνω-προς-τα-κάτω (top-down, tuple at a time).

- Στις επαγωγικές ΒΔ ξεχωρίζουμε τη ΒΔ από τους κανόνες, μιας και θεωρούμε πως η ΒΔ είναι μεγάλη και πως ένα πρόγραμμα παίρνει σαν είσοδο τη ΒΔ (τα EDBs) και υπολογίζει σαν έξοδο ένα νέο σύνολο από γεγονότα που το κατηγορηματικό τους σύμβολο ανήκει στα IDBs.

Στη συνέχεια θα δείξουμε πως μπορούμε να μετασχηματίσουμε ένα Datalog πρόγραμμα σε ένα σύστημα εξισώσεων της θετικής σχεσιακής άλγεβρας (positive relational algebra RA^+) [11], δηλ. στη σχεσιακή άλγεβρα που περιέχει τις πράξεις της *επιλογής (selection)*, *προβολής (projection)*, *καρτεσιανού γινομένου (cartesian product)* και της *ένωσης (union)* (δεν περιλαμβάνεται η *διαφορά (difference)*). Ακόμα θα δείξουμε πως μπορούμε να κάνουμε και τον αντίστροφο μετασχηματισμό, δηλ. κάθε μια από τις παραπάνω πράξεις να τις εκφράσουμε με Datalog προγράμματα. Με αυτό τον τρόπο θα έχουμε δείξει πως η Datalog είναι υπερσύνολο της θετικής σχεσιακής άλγεβρας. Κάθε IDB κατηγόρημα του Datalog προγράμματος αντιστοιχεί σε μια *σχεσιακή μεταβλητή (variable relation)* και κάθε EDB κατηγόρημα σε μια *σχεσιακή σταθερά (constant relation)*. Το να βρούμε τη *λύση* του συστήματος εξισώσεων σημαίνει να βρούμε τη τιμή των σχεσιακών μεταβλητών, ώστε να ικανοποιείται το σύστημα.

Πιο ειδικά, για το πρώτο μέρος, κάθε πρόταση μετασχηματίζεται σε μια σχέση εγκλεισμού της θετικής σχεσιακής άλγεβρας. Όλες οι σχέσεις εγκλεισμού που αναφέρονται στο ίδιο κατηγόρημα (δηλ. η κεφαλή της Datalog πρότασης έχει το ίδιο IDB κατηγόρημα) ενώνονται στη συνέχεια για να μας ορίσουν ολόκληρη τη σχέση (που αντιστοιχεί σε αυτό το IDB). Επιπλέον θα δούμε πως μετασχηματίζεται ο στόχος της επερώτησης στην αντίστοιχη εξίσωση της θετικής σχεσιακής άλγεβρας.

Έστω μια Datalog πρόταση της μορφής:

$$C : p(a_1, \ldots, a_n) \leftarrow q_1(b_1, \ldots, b_k), \ldots, q_m(b_s, \ldots, b_h).$$

Έστω πως οι $P, Q_1, \ldots, Q_m$ δηλώνουν τις intensional και extensional σχέσεις που αντιστοιχούν στα κατηγορήματα $p, q_1, \ldots, q_m$. Παρατηρήστε πως κάποιο από τα q_i μπορεί να είναι ίδιο με το p. Ο μετασχηματισμός θα συσχετίσει με την πρόταση C μια σχέση εγκλεισμού της μορφής:

$$Expr(Q_1, \ldots, Q_m) \subseteq P.$$

Ο αλγόριθμος μετασχηματισμού βασίζεται στο συντακτικό της πρότασης και απαιτεί κάποιες έννοιες και δύο συναρτήσεις μετασχηματισμού όρων:

- $occurs(a_i, RHS)$, επιστρέφει την τιμή αληθές, αν ο όρος a_i εμφανίζεται στο δεξί μέρος της πρότασης, ψευδές διαφορετικά.
- $corr(i)$, επιστρέφει για κάθε a_i του αριστερού μέρους της πρότασης την αντίστοιχη θέση j της πρώτης μεταβλητής b_j του δεξιού μέρους (αν υπάρχει), τέτοια ώστε $b_j = a_i$.
- $const(a_i)$, επιστρέφει αληθές αν το a_i είναι σταθερά (αντίστοιχα για b_j).
- $var(a_i)$ είναι η άρνηση του $const$.

- $newvar(x)$ φτιάχνει μια νέα μεταβλητή x που δεν υπάρχει.
- $E < x\ y >$, όπου E είναι ένα string, x, y, σύμβολα, αντικαθιστά στο string E το x με y.
- $EQ(x, y)$, κάνει το x ίσο με το y.

Ο αλγόριθμος 1 πρώτα μετασχηματίζει το αριστερό μέρος της πρότασης, ώστε να μην έχει σταθερές, στη συνέχεια να μην έχει ίδιες μεταβλητές και στη συνέχεια κάνει το ίδιο και στο δεξιό μέρος. Τέλος, παίρνει το καρτεσιανό γινόμενο των σχέσεων που υπάρχουν στο δεξιό μέρος:

Παράδειγμα 1.1.4 *Άς δούμε πως εφαρμόζεται ο παραπάνω αλγόριθμος. Έστω ο κανόνας:*

$$\texttt{p(X,a,X,Z)} \leftarrow \texttt{s(X,Y),r(Y,b,Z).}$$

με εφαρμογή της απαλοιφής της σταθεράς από το αριστερό μέρος έχουμε:

$$\texttt{T(p(X,N1,X,Z)} \leftarrow \texttt{s(X,Y),r(Y,b,Z),eq(N1,a))}$$

με εφαρμογή της απαλοιφής της ίδιας μεταβλητής στο αριστερό μέρος έχουμε:

$$\texttt{T(p(N2,N1,X,Z)} \leftarrow \texttt{s(X,Y),r(Y,b,Z),eq(N1,a),eq(N2,X))}$$

και με τον τελευταίο κανόνα που αναφέρεται στο αριστερό μέρος:

$$\Pi_{8,6,1,5}\texttt{T'(s(X,Y),r(Y,b,Z),eq(N1,a),eq(N2,X))}$$

ενώ με εφαρμογή του κανόνα απαλοιφής της σταθεράς στο δεξιό μέρος:

$$\Pi_{8,6,1,5}\sigma_{4=b}\texttt{T'(s(X,Y),r(Y,N3,Z),eq(N1,a),eq(N2,X))}$$

και με απαλοιφή των μεταβλητών που εμφανίζονται περισσότερες από μια φορές στο δεξιό μέρος:

$$\Pi_{8,6,1,5}\sigma_{4=b\wedge 1=9\wedge 2=3}\texttt{T'(s(N4,Y),r(N5,N3,Z),eq(N1,a),eq(N2,X))}$$

και τέλος εφαρμόζοντας τον τελευταίο κανόνα για το δεξιό μέρος:

$$\Pi_{8,6,1,5}\sigma_{4=b\wedge 1=9\wedge 2=3}\texttt{T'(S} \times \texttt{R} \times \texttt{EQ} \times \texttt{EQ)}$$

Επομένως αν έχουμε m_p πλήθος κανόνων που ορίζουν ένα IDB κατηγόρημα p και οι αντίστοιχες σχέσεις της θετικής σχεσιακής άλγεβρας είναι οι $Expr_i(Q_1, \ldots, Q_m) \subseteq P$, για $1 \leq i \leq m_p$, τότε παράγουμε την ακόλουθη εξίσωση για την P:

$$P = Expr_1(Q_1, \ldots, Q_m) \cup \cdots \cup Expr_{m_p}(Q_1, \ldots, Q_m).$$

Η ένωση αυτή των ανισοτήτων εκφράζει το ελάχιστο μοντέλο Herbrand για το Datalog πρόγραμμα.

Algorithm 1 Αλγόριθμος μετατροπής Datalog πρότασης σε σχέση της RA$^+$

Είσοδος: *clause* $C = LHS \leftarrow RHS$.
$(= p(a_1, \ldots, a_n) \leftarrow q_1(b_1, \ldots, b_n), \ldots, q_m(b_s, \ldots, b_h).)$
Έξοδος: μια έκφραση της μορφής $Expr(Q_1, \ldots, Q_m)$ των σχέσεων που αντιστοιχούν στα κατηγορήματα $q_1, \ldots, q_m$.

begin
  if $\exists i$: $const(a_i)$ **then** {υπάρχει σταθερά στο αριστερό μέρος}
  begin
    $newvar(x)$; return $T(LHS < a_i, x > \leftarrow RHS, EQ(x, a_i))$
  end
  else if $\exists i, j$: $a_i = a_j$ **then** {υπάρχουν δύο ίδιες μεταβλητές στο αριστερό μέρος}
    begin
      $newvar(x)$; return $T(LHS < a_i, x > \leftarrow RHS, EQ(x, a_j))$
    end
    else {είναι στη σωστή μορφή}
    begin
      return $\Pi_{corr(1),\ldots,corr(n)} T'(RHS)$
    end
  {**end if**}
  {συνεχίζουμε με το δεξιό μέρος}
  if $\exists j$: $const(b_j)$ **then** {υπάρχει σταθερά στο δεξιό μέρος}
  begin
    $newvar(x)$; return $\sigma_{j=b_j} T'(RHS < b_j, x >)$
  end
  else if $\exists i, j$: $b_i = b_j$ **then** {υπάρχουν δύο ίδιες μεταβλητές στο δεξιό μέρος}
    begin
      $newvar(x)$; return $\sigma_{i=j} T'(RHS < b_i, x >)$
    end
    else {είναι στη σωστή μορφή}
    begin
      return $\times_{i=1}^{m} Q_i$
    end
  {**end if**}
end

Παράδειγμα 1.1.5 *Έστω το ακόλουθο πρόγραμμα Datalog:*

```
p(X) ← p(Y),r(X,Y).
p(X) ← s(Y,X).
q(X,Y) ← p(X),r(X,Z),s(Y,Z).
```

που μεταφράζεται σε:

$$\Pi_2(\mathtt{P} \bowtie_{1=2} \mathtt{R}) \subseteq \mathtt{P}$$
$$\Pi_2\mathtt{S} \subseteq \mathtt{P}$$
$$\Pi_{1,4}((\mathtt{P} \bowtie_{1=1} \mathtt{R}) \bowtie_{3=2} \mathtt{S}) \subseteq \mathtt{Q}$$

Από αυτές τις σχέσεις εγκλεισμού έχουμε τις ακόλουθες ισότητες:

$$\Pi_2(\mathtt{P} \bowtie_{1=2} \mathtt{R}) \cup \Pi_2\mathtt{S} = \mathtt{P}$$
$$\Pi_{1,4}((\mathtt{P} \bowtie_{1=1} \mathtt{R}) \bowtie_{3=2} \mathtt{S}) = \mathtt{Q}$$

Στη συνέχεια θα δούμε πως μεταφράζουμε *λογικούς στόχους* σε *αλγεβρικές επερωτήσεις*. Οι στόχοι της Datalog μεταφράζονται σε προβολές και επιλογές σε μια σχεσιακή μεταβλητή του συστήματος αλγεβρικών εξισώσεων. Για το σκοπό αυτό επεκτείνουμε την πράξη της προβολής στο κενό σύνολο ορισμάτων:

$$\Pi_{\emptyset}E = \begin{cases} \text{"}no\text{"}, \text{αν η E είναι το κενό σύνολο} \\ \text{"}yes\text{"}, \text{διαφορετικά} \end{cases}$$

Χρησιμοποιούμε αυτή την ειδική περίπτωση για να μπορούμε να απαντάμε και στόχους (ερωτήσεις) που περιέχουν μόνο σταθερές, οπότε η απάντηση είναι απλά ένα "yes" ή ένα "no". Έστω p το κατηγορηματικό σύμβολο του στόχου και έστω πως το i όρισμα έχει την σταθερή τιμή c_i στο στόχο και P είναι η σχεσιακή μεταβλητή που αντιστοιχεί στο p. Τότε, η σταθερά c_i μεταφράζεται σε μια επιλογή $i = c_i$.

a) Μια *επιλογή στο στόχο* χρειάζεται όταν τουλάχιστον ένα όρισμα του στόχου έχει σταθερή τιμή. Αν έχουμε περισσότερα από ένα ορίσματα που έχουν σταθερές τιμές, τότε παίρνουμε την τομή όλων των επιλογών.

b) Μια *προβολή στο στόχο* $\Pi_{\emptyset}P$ παράγεται μόνο όταν όλα τα ορίσματα του στόχου έχουν σταθερές τιμές.

Παρακάτω φαίνονται μερικά παραδείγματα περιπτώσεων που μπορούμε να έχουμε για έναν στόχο:

1. $p(X). \Leftrightarrow P$
2. $p(a). \Leftrightarrow \Pi_{\emptyset}(\sigma_{1=a}P)$
3. $q(a,b). \Leftrightarrow \Pi_{\emptyset}(\sigma_{1=a\wedge 2=b}Q)$
4. $q(a,b,X). \Leftrightarrow \sigma_{1=a\wedge 2=b}Q$

Παράδειγμα 1.1.6 *Έστω το προηγούμενο Παράδειγμα 1.1.5 με στόχο* $\leftarrow$ `q(a,Z)`. *Θεωρούμε πως οι σχεσιακές μεταβλητές που αντιστοιχούν στα IDB* p, q *είναι* X_1, X_2 *και οι σχεσιακές σταθερές που αντιστοιχούν στα EDB* r, s *είναι* C_1, C_2. *Τότε το σύστημα εξισώσεων που προκύπτει είναι:*

$$\Pi_2(\mathtt{X}_1 \bowtie_{1=2} \mathtt{C}_1) \cup \Pi_2\mathtt{C}_2 = \mathtt{X}_1$$
$$\Pi_{1,4}(\mathtt{X}_1 \bowtie_{1=1} \mathtt{C}_1) \bowtie_{3=2} \mathtt{C}_2) = \mathtt{X}_2$$

Η επερώτηση (query) $\leftarrow$ `q(a,Z)` *γίνεται* $\sigma_{1=a}\mathtt{X}_2$.

Από την άλλη μεριά, μπορούμε να μεταφράζουμε εξισώσεις της RA^+ σε κανόνες Datalog. Για το λόγο αυτό θα δείξουμε στη συνέχεια πως μπορούμε να μεταφράσουμε τις πράξεις της RA^+ σε κανόνες Datalog.

Προβολή. Έστω η εξίσωση $X = \Pi_{i_1,\ldots,i_h}Y$, με X μια intensional σχέση με πληθικότητα h και Y μια intensional ή extensional σχέση με πληθικότητα k ($h \leq k$). Αυτή η εξίσωση αντιστοιχεί στον ακόλουθο κανόνα Datalog:

$$x(a_{i_1}, a_{i_2}, \ldots, a_{i_h}) \leftarrow y(a_1, a_2, \ldots, a_k).$$

Επιλογή. Έστω η εξίσωση $X = \sigma_{i_1=t_1 \wedge i_2=t_2 \wedge i_n=t_n}Y$, με X, Y να έχουν την ίδια πληθικότητα k, τα i_j είναι θέσεις ορισμάτων και τα t_j είναι είτε σταθερές είτε θέσεις ορισμάτων. Αυτή η εξίσωση αντιστοιχεί στον κανόνα Datalog:

$$x(a_1, \ldots, a_k) \leftarrow y(a_1, \ldots, a_k), eq(a_{i_1}, \tau_1), \ldots, eq(a_{i_n}, \tau_n).$$

όπου eq είναι ένα κατηγόρημα που είναι αληθές ανν το δεύτερο όρισμα είναι ίσο με το πρώτο και

$$\tau_j = \begin{cases} t_j, \text{αν η } t_j \text{ είναι μια σταθερά} \\ a_{t_j}, \text{αν } t_j \text{ είναι μια θέση ορίσματος} \end{cases}$$

Καρτεσιανό γινόμενο. Έστω η εξίσωση $X = Y_1 \bowtie \cdots \bowtie Y_n$, με X να έχει πληθικότητα k, Y_i να έχει πληθικότητα k_i, $1 \leq i \leq n$ και $k = k_1 + \cdots + k_n$. Η εξίσωση αυτή μεταφράζεται στον ακόλουθο κανόνα:

$$x(a_1, \ldots, a_k) \leftarrow y_1(a_1, \ldots, a_{k_1}), \ldots, y_n(a_{k-k_{n-1}+1}, \ldots, a_k).$$

Ένωση. Έστω η εξίσωση $X = Y_1 \cup Y_2$, με X, Y_1 και Y_2 να έχουν πληθικότητα n. Τότε η εξίσωση αυτή μεταφράζεται στους ακόλουθους κανόνες:

$$x(a_1, \ldots, a_n) \leftarrow y_1(a_1, \ldots, a_n).$$

$$x(a_1, \ldots, a_n) \leftarrow y_2(a_1, \ldots, a_n).$$

Μπορούμε να παρατηρήσουμε πως το *join (ζεύξη)* μπορεί να υλοποιηθεί με την πράξη της επιλογής και το καρτεσιανό γινόμενο.

Παράδειγμα 1.1.7 *Έστω οι ακόλουθες εξισώσεις της RA^+:*

$$\begin{aligned} &\mathtt{P} = \Pi_{1,3}((\mathtt{Q}_1 \bowtie_{1=2} \mathtt{Q}_2) \bowtie_{3=2} \mathtt{Q}_3) \\ &\mathtt{P} = \Pi_{1,3}\mathtt{Q}_4 \end{aligned}$$

με πληθικότητες των σχέσεων: $arity(P) = arity(Q_1) = arity(Q_2) = arity(Q_3) = 2$ *και* $arity(Q_4) = 3$. *Αυτές οι εξισώσεις έχουν σαν μετάφραση τους ακόλουθους κανόνες:*

$$\begin{aligned} &\mathtt{p}(\mathtt{X}_1,\mathtt{X}_3) \leftarrow \mathtt{q}_1(\mathtt{X}_1,\mathtt{X}_2),\mathtt{q}_2(\mathtt{X}_3,\mathtt{X}_4),\mathtt{q}_3(\mathtt{X}_5,\mathtt{X}_6),\mathtt{eq}(\mathtt{X}_1,\mathtt{X}_4),\mathtt{eq}(\mathtt{X}_3,\mathtt{X}_6). \\ &\mathtt{p}(\mathtt{X}_1\mathtt{X}_3) \leftarrow \mathtt{Q}_4(\mathtt{X}_1,\mathtt{X}_2,\mathtt{X}_3). \end{aligned}$$

Έτσι δείξαμε πως κάθε έκφραση της RA^+ μπορεί να μεταφραστεί σε ένα Datalog πρόγραμμα.

Κεφάλαιο 2

Υπάρχουσες μέθοδοι στην κατηγορία των μετασχηματισμών διάδοσης τιμών

Στο Κεφάλαιο αυτό θα παρουσιάσουμε τους σημαντικότερους εκπροσώπους της κατηγορίας των μετασχηματισμών *διάδοσης τιμών* οι οποίο αναφέρονται στα [49, 11, 7, 9, 43, 19].

2.1 Κατηγοριοποίηση των μεθόδων βελτιστοποίησης

Στην ενότητα αυτή θα δούμε μια κατηγοριοποίηση των μεθόδων βελτιστοποίησης επερωτήσεων Datalog ανάλογα με το κριτήριο που χρησιμοποιούμε. Η κατηγοριοποίηση αυτή στηρίζεται στο [11]. Σχηματικά οι διάφορες κατηγορίες βελτιστοποίησης φαίνονται στο παρακάτω σχήμα:

Κριτήριο	Εναλλακτική Μέθοδος	
Φορμαλισμός - Γλώσσα	Datalog	Σχεσιακή Άλγεβρα
Μέθοδος αναζήτησης	bottom-up	top-down
Αντικειμενικός στόχος	επανεγγραφή	καθαρός υπολογισμός

Οι μετασχηματισμοί διάδοσης τιμών ανήκουν στις κατηγορίες μεθόδων επανεγγραφής Datalog προγραμμάτων με bottom-up υπολογισμό.

2.1.1 Γλώσσα - Φορμαλισμός

Όπως είδαμε και σε προηγούμενη ενότητα, κάθε Datalog πρόγραμμα μπορεί να μετασχηματιστεί σε ένα ισοδύναμο σύστημα εξισώσεων της θετικής σχεσιακής άλγεβρας (RA^+). Επομένως, μπορεί κανείς να εφαρμόσει μεθόδους βελτιστοποίησης είτε στο αρχικό Datalog πρόγραμμα είτε στο ισοδύναμο μετασχηματισμένο σε σχεσιακή άλγεβρα. Στην εργασία αυτή μελετάμε μόνο βελτιστοποιήσεις σε Datalog προγράμματα.

2.1.2 Μέθοδος αναζήτησης

Ένα Datalog πρόγραμμα μπορεί να υπολογιστεί με δύο διαφορετικούς τρόπους: από-πάνω-προς-τα-κάτω (top-down) ή από-κάτω-προς-τα-πάνω (bottom-up) και θα εξηγηθούν παρακάτω εκτενέστερα. Στην πρώτη περίπτωση ξεκινάμε από τον στόχο που θέλουμε να απαντήσουμε (αρχικό πρόβλημα) και στη συνέχεια προσπαθούμε να λύσουμε τα υποπροβλήματα που προκύπτουν. Στη δεύτερη μέθοδο, ξεκινάμε από τα γνωστά άτομα και παράγονται νέα άτομα μέχρι να πάρουμε όλα τα άτομα που μπορούν να παραχθούν από το πρόγραμμα και τη βάση δεδομένων. Για κάθε μια από τις μεθόδους υπολογισμού υπάρχουν διάφορες τεχνικές βελτιστοποίησης. Στην παρούσα εργασία μελετώνται μόνο βελτιστοποιήσεις που αφορούν την από-κάτω-προς-τα-πάνω αποτίμηση. Για την ακρίβεια προσπαθούμε να εξομοιώσουμε την από-πάνω-προς-τα-κάτω αποτίμηση με μια από-κάτω-προς-τα-πάνω αποτίμηση (να χρησιμοποιούμε τις σταθερές που υπάρχουν στον αρχικό στόχο ώστε να μειώνουμε το χώρο αναζήτησης).

2.1.3 Αντικειμενικοί στόχοι της μεθόδου βελτιστοποίησης

Ανάλογα με τη πληροφορία που χρησιμοποιούμε έχουμε διαφορετικές μεθόδους βελτιστοποίησης. Μια κατηγορία χρησιμοποιεί μετασχηματισμό προγραμμάτων. Ειδικότερα, μετασχηματίζουμε το πρόγραμμα που μας δίνεται σε ένα ισοδύναμο στον ίδιο φορμαλισμό, το οποίο υπολογίζεται πιο αποδοτικά από το αρχικό. Αυτές οι τεχνικές θα ονομάζονται *τεχνικές βελτιστοποίησης επερωτήσεων (query optimization techniques)* [34, 2] ή *μέθοδοι επανεγγραφής όρων (term rewriting methods)* [8]. Αυτές οι μέθοδοι βρίσκονται στον αντίποδα των μεθόδων που βασίζονται στον *καθαρό υπολογισμό*. Για μια καλή περιγραφή και των δύο αυτών κατηγοριών μπορεί κανείς να ανατρέξει στο [8].

2.2 Κατηγοριοποίηση των προγραμμάτων Datalog

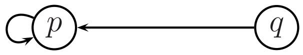
Στην ενότητα αυτή θα δώσουμε μια κατηγοριοποίηση των προγραμμάτων Datalog ανάλογα με το αν έχουν αναδρομή ή όχι και στη περίπτωση που έχουν αναδρομή, με το είδος της αναδρομής. Αρχικά θα ορίσουμε μερικές έννοιες που θα χρειαστούν παρακάτω. Έστω P ένα Datalog πρόγραμμα

$$p_i(a_1, \ldots, a_{n_i}) \leftarrow q_1(b_1, \ldots, b_k), \ldots, q_m(b_s, \ldots, b_h). \quad i = 1, \ldots, r,$$

που ορίζει n IDB κατηγορήματα $p_1, \ldots, p_n$ $(1 \leq n \leq r)$. Έστω X_P το σύνολο των IDB κατηγορημάτων του προγράμματος P.

Ορισμός 2.2.1 *Ο* γράφος εξάρτησης (dependency graph) *του P είναι ένας κατευθυνόμενος γράφος $G_{X_P} = (V, E)$, όπου το σύνολο των κορυφών V είναι το X_P, δηλ. οι κορυφές του γράφου είναι τα IDB κατηγορήματα και το σύνολο των ακμών του είναι $E = \{(p_i, p_j) \mid$ το p_j εμφανίζεται στο σώμα ενός κανόνα που έχει κεφαλή το $p_i\}$.*

Στη συνέχεια δίνεται ο γράφος εξάρτησης του προγράμματος του Παραδείγματος 1.1.5.



Ορισμός 2.2.2 *Δίνεται ένας κανόνας Datalog:*

$$C : p_i(a_1, \ldots, a_{n_i}) \leftarrow q_1(b_1, \ldots, b_k), \ldots, q_m(b_s, \ldots, b_h).$$

Τότε λέμε ότι ο κανόνας είναι αναδρομικός (recursive) *αν τουλάχιστον ένα από τα q_i ταυτίζεται με το p.*

Ορισμός 2.2.3 *Δίνεται ένα πρόγραμμα Datalog:*

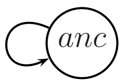
$$p_i(a_1, \ldots, a_{n_i}) \leftarrow q_1(b_1, \ldots, b_k), \ldots, q_m(b_s, \ldots, b_h). \quad i = 1, \ldots, r$$

Τότε λέμε πως είναι αναδρομικό (recursive) *αν ο γράφος εξάρτησης του περιέχει τουλάχιστον έναν κύκλο. Ένα πρόγραμμα που περιέχει αναδρομικούς κανόνες είναι αναδρομικό, ενώ το αντίστροφο δεν ισχύει (υπάρχουν προγράμματα που είναι αναδρομικά, αλλά δεν έχουν αναδρομικούς κανόνες).*

Παράδειγμα 2.2.1 *Έστω το παρακάτω πρόγραμμα, που υπολογίζει ζεύγη (X, Y), τέτοια ώστε ο Y να είναι πρόγονος του X:*

```
anc(X,Y) ← par(X,Y).
anc(X,Y) ← par(X,Z),anc(Z,Y).
```

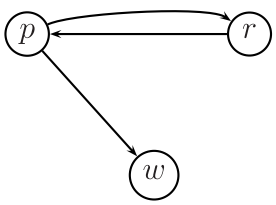
Ο δεύτερος κανόνας είναι αναδρομικός επομένως και το πρόγραμμα είναι αναδρομικό. Ο γράφος εξάρτησης του φαίνεται παρακάτω:



Παράδειγμα 2.2.2 *Στη συνέχεια δίνεται ένα παράδειγμα προγράμματος που είναι αναδρομικό, αλλά δεν έχει αναδρομικούς κανόνες:*

```
p(X,Y) ← r(X,Z),t(Z,Y).
p(X,Y) ← w(X,Y).
r(X,Y) ← s(X,Z),p(Z,Y).
r(X,Y) ← t(X,Y).
w(X,Y) ← s(X,Z),t(Z,Y).
```

Ο γράφος εξάρτησης είναι ο ακόλουθος:



Όπως μπορεί κανείς να δει από το γράφο εξάρτησης, υπάρχει κύκλος, επομένως το πρόγραμμα είναι αναδρομικό. Ωστόσο κανένας από τους κανόνες του δεν είναι αναδρομικός.

Ορισμός 2.2.4 *Ένα IDB κατηγόρημα p σε ένα Datalog πρόγραμμα P λέγεται* αναδρομικό (recursive), *αν η αντίστοιχη κορυφή του περιέχεται σε ένα κύκλο στο γράφο εξάρτησης του προγράμματος P.*

Ορισμός 2.2.5 *Δύο IDB κατηγορήματα p και q σε ένα Datalog πρόγραμμα P λέγονται* αμοιβαία αναδρομικά (mutually recursive), *αν οι αντίστοιχες κορυφές τους ανήκουν στον ίδιο κύκλο στο γράφο εξάρτησης του P.*

Συνεχίζοντας το προηγούμενο παράδειγμα, βλέπουμε πως τα κατηγορήματα p και r είναι αναδρομικά και μάλιστα αμοιβαία αναδρομικά, ενώ το w δεν είναι αναδρομικό.

Ορισμός 2.2.6 *Δίνεται ένας Datalog κανόνας:*

$$C : p_i(a_1, \ldots, a_{n_i}) \leftarrow q_1(b_1, \ldots, b_k), \ldots, q_m(b_s, \ldots, b_h).$$

Λέμε ότι ο C είναι γραμμικός (linear) ως προς το q_i, *αν υπάρχει το πολύ ένα q_i στο σώμα του κανόνα C.*

Στο παράδειγμα με το `anc` ο δεύτερος κανόνας είναι γραμμικός ως προς το `anc`. Αντίθετα, ένας κανόνας της μορφής: `anc(X,Y)` $\leftarrow$ `anc(X,Z),anc(Z,Y).` δεν είναι γραμμικός.

Ορισμός 2.2.7 *Δίνεται ένα Datalog πρόγραμμα P που περιλαμβάνει τον κανόνα:*

$$C : p_i(a_1, \ldots, a_{n_i}) \leftarrow q_1(b_1, \ldots, b_k), \ldots, q_m(b_s, \ldots, b_h).$$

Λέμε ότι ο κανόνας C είναι γραμμικός *αν υπάρχει το πολύ ένα q_i στο σώμα του κανόνα (πιθανώς το ίδιο το p), που είναι αμοιβαία αναδρομικό με το p στο P.*

Παρατηρήστε πως ένα πρόγραμμα μπορεί να είναι και γραμμικό και αναδρομικό. Στο παράδειγμα 2.2.1 με το `anc` στην πρώτη περίπτωση είναι γραμμικό και αναδρομικό, ενώ στη δεύτερη μόνο αναδρομικό.

Ορισμός 2.2.8 *Δίνεται ένα πρόγραμμα Datalog:*

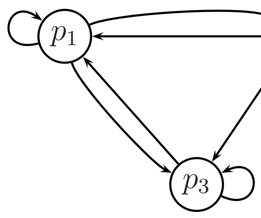
$$P : p_i(a_1, \ldots, a_{n_i}) \leftarrow q_1(b_1, \ldots, b_k), \ldots, q_m(b_s, \ldots, b_h). \quad i = 1, \ldots, r$$

Λέμε ότι το πρόγραμμα P είναι γραμμικό αν όλοι οι κανόνες του είναι γραμμικοί.

Παράδειγμα 2.2.3 *Έστω το πρόγραμμα:*

(1) $\texttt{p}_1\texttt{(X,Y)} \leftarrow \texttt{c}_1\texttt{(X,Y)}.$
(2) $\texttt{p}_1\texttt{(X,Y)} \leftarrow \texttt{p}_1\texttt{(X,Z)},\texttt{p}_3\texttt{(Z,Y)}.$
(3) $\texttt{p}_1\texttt{(X,Y)} \leftarrow \texttt{p}_2\texttt{(X,Y)}.$
(4) $\texttt{p}_2\texttt{(X,Y)} \leftarrow \texttt{p}_1\texttt{(X,Z)},\texttt{p}_3\texttt{(Z,Y)}.$
(5) $\texttt{p}_2\texttt{(X,Y)} \leftarrow \texttt{c}_3\texttt{(X,Y)}.$
(6) $\texttt{p}_3\texttt{(X,Y)} \leftarrow \texttt{p}_3\texttt{(X,Z)},\texttt{c}_2\texttt{(Z,Y)}.$
(7) $\texttt{p}_3\texttt{(X,Y)} \leftarrow \texttt{p}_1\texttt{(X,Y)}.$

Στη συνέχεια φαίνεται ο γράφος εξάρτησης του προγράμματος.



Σε αυτό το πρόγραμμα, τα κατηγορήματα $\mathrm{p}_1, \mathrm{p}_2$ *και* p_3 *είναι αμοιβαία αναδρομικά και επομένως αναδρομικά. Επιπλέον, μπορούμε να παρατηρήσουμε πως όλοι οι κανόνες είναι γραμμικοί ως προς όλα τα κατηγορήματα, εκτός από τους κανόνες 2 και 4 που δεν είναι γραμμικοί. Αυτό οφείλεται στο γεγονός πως στο σώμα του κανόνα 2 υπάρχουν τα* $\mathrm{p}_1, \mathrm{p}_3$ *που είναι αμοιβαία αναδρομικά με το* p_1. *Παρόμοια, ο κανόνας 4 δεν είναι γραμμικός, επειδή στο σώμα υπάρχουν τα κατηγορήματα* $\mathrm{p}_1, \mathrm{p}_3$, *τα οποία είναι αμοιβαία αναδρομικά με το* p_2. *Για αυτό, το πρόγραμμα είναι αναδρομικό.*

2.3 Σύγκριση ανάμεσα σε top-down αποτίμηση και bottom-up αποτίμηση

Η Prolog ως γνωστόν, χρησιμοποιεί την SLD-επίλυση για να αποτιμήσει ένα στόχο-ερώτηση. Η μέθοδος αυτή είναι μια top-down μέθοδος, η οποία αρχικά προσπαθεί να ενοποιήσει τον στόχο-ερώτηση με την κεφαλή κάποιου κανόνα, διαδικασία από την οποία προκύπτουν νέοι στόχοι, από το σώμα του κανόνα που επιλέχθηκε. Η διαδικασία επίλυσης των στόχων που υπάρχουν στο σώμα, ξεκινάει από το πρώτο υποστόχο που υπάρχει στο σώμα και συνεχίζεται από αριστερά προς τα δεξιά. Αυτό φαίνεται σχηματικά στο Σχήμα 2.1. Το αποτέλεσμα αυτού του τρόπου επίλυσης είναι να έχουμε μια πρώτα-κατά-βάθος αναζήτηση του χώρου αναζήτησης (depth-first-search). Το βασικό πλεονέκτημα μιας top-down προσέγγισης είναι πως στην περίπτωση που ο στόχος-ερώτηση έχει κάποια ορίσματα δεσμευμένα, δηλαδή είναι σταθερές που δίνονται και όχι μεταβλητές, τότε θα έχουμε μια εστίαση της μεθόδου επίλυσης μόνο σε γεγονότα που σχετίζονται με τον στόχο-ερώτηση.

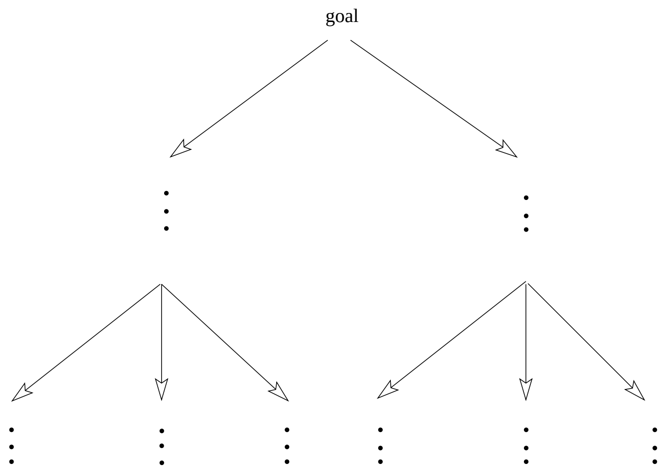
Ωστόσο, ένα βασικό μειονέκτημα των top-down προσεγγίσεων είναι πως εύκολα πέφτουν σε ατέρμονες επαναλήψεις, με αποτέλεσμα να μην τερματίζουν είτε υπολογίζουν τα ίδια πράγματα πολλές φορές.

Ένα χαρακτηριστικό παράδειγμα είναι το κλασικό παράδειγμα εύρεσης ενός μονοπατιού σε γράφο με αριστερή αναδρομή:

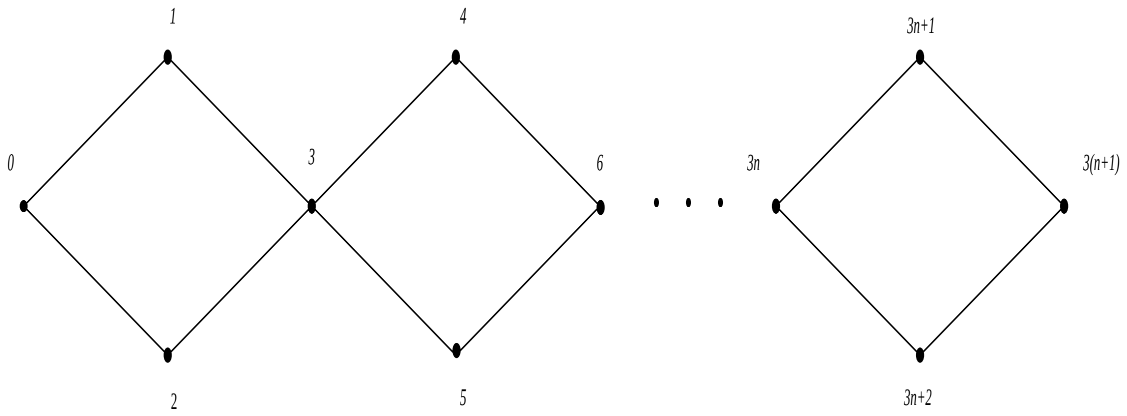
Παράδειγμα 2.3.1 *Έστω το ακόλουθο πρόγραμμα:*

$$\texttt{path(X,Y)} \leftarrow \texttt{edge(X,Y).}$$
$$\texttt{path(X,Y)} \leftarrow \texttt{path(X,Z),edge(Z,Y).}$$

όπου το άτομο $\texttt{edge}(\texttt{X},\texttt{Y})$ *δηλώνει πως υπάρχει ακμή από τον κόμβο* X *στον κόμβο* Y. *Η Prolog, θα δοκιμάζει να επιλύσει το αριστερότερο άτομο, με αποτέλεσμα να πέφτει σε έναν ατέρμονα βρόχο και να μην τερματίζει.*



Σχήμα 2.1: *Top-down εκτέλεση.*



Σχήμα 2.2: *Γράφος ασυμπτωτικού κάτω φράγματος.*

Στην δεύτερη περίπτωση, που υπολογίζουμε άτομα που έχουμε ήδη υπολογίσει, ένα παράδειγμα [45] είναι το ακόλουθο:

Παράδειγμα 2.3.2 *Έστω το ακόλουθο πρόγραμμα:*

$$\mathtt{t(X,Y)} \leftarrow \mathtt{e(X,W),t(W,Y).}$$
$$\mathtt{t(X,Y)} \leftarrow \mathtt{e(X,Y).}$$

με την ακόλουθη βάση δεδομένων, που φαίνεται στο Σχήμα 2.2:

$$\begin{array}{llll}
\mathtt{e}(0,1) & \mathtt{e}(0,2) & \mathtt{e}(1,3) & \mathtt{e}(2,3) \\
\vdots & \vdots & \vdots & \vdots \\
\mathtt{e}(3n,3n+1) & \mathtt{e}(3n,3n+2) & \mathtt{e}(3n+1,3(n+1)) & \mathtt{e}(3n+2,3(n+1))
\end{array}$$

Αποτιμώντας το πρόγραμμα με τον κλασική top-down μέθοδο, χρειαζόμαστε εκθετικό χρόνο ($\Omega(2^n)$) για να υπολογίσουμε όλες τις λύσεις. Στην περίπτωση *που υπολογίσουμε τις απαντήσεις με τον bottom-up τρόπο, αφού πρώτα το μετασχηματίσουμε με τον μετασχηματισμό magic sets, τότε χρειαζόμαστε πολυωνυμικό χρόνο.*

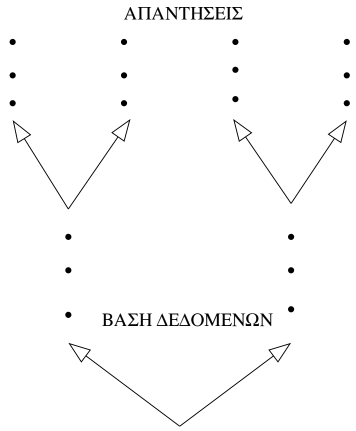
Από την άλλη πλευρά, οι κλασικές βάσεις δεδομένων, οι σχεσιακές (relational databases), αντιμετώπιζαν το πρόβλημα να εκφράζουν αναδρομικές σχέσεις και μεταβατικές κλειστότητες (transitive closures) [2]. Σύντομα, έγινε αντιληπτό πως οι σχέσεις στις βάσεις δεδομένων μπορούσαν να εκφρασθούν με κατηγορήματα που είναι αντικείμενα της λογικής πρώτης τάξης και ειδικότερα των γλωσσών λογικού προγραμματισμού. Το κύριο μέλημα ωστόσο για την γεφύρωση των δύο αυτών περιοχών, ήταν να έχουμε την δυνατότητα να προσπελαύνουμε γρήγορα τα δεδομένα από τη δευτερεύουσα μνήμη (δίσκος) καθώς ο όγκος των δεδομένων είναι μεγάλος και δεν αρκεί συνήθως η κύρια μνήμη για την εκτέλεση διαφόρων εργασιών. Προκειμένου το χάσμα αυτό να γεφυρωθεί, θα θέλαμε να περάσουμε από έναν υπολογισμό μιας πλειάδας-τη-φορά (tuple-at-a-time) σε μια προσέγγιση υπολογισμού ενός συνόλου-τη-φορά (set-at-a-time). Η δεύτερη προσέγγιση χρησιμοποιείται στο λογικό προγραμματισμό, οπότε θα πρέπει να τροποποιηθεί ο υπολογισμός αυτός, ώστε να είναι κατάλληλος για τον χειρισμό μεγάλου όγκου δεδομένων, κατάλληλου για την εκτέλεση επερωτήσεων (query processing) στις βάσεις δεδομένων.

Η ιδέα ήταν να δημιουργηθεί μια μέθοδος η οποία να διατηρούσε τα καλά χαρακτηριστικά της top-down μεθόδου αλλά να να μην έπεφτε σε ατέρμονες βρόχους ούτε να επαναλάμβανε υπολογισμούς γεγονότων που είχε ήδη υπολογίσει. Ακόμα η μέθοδος αυτή θα έπρεπε να είναι πλήρης (complete). Η λύση ήταν μια bottom-up προσέγγιση η οποία θα ξεκίναγε από τη βάση δεδομένων και θα κατέληγε στον στόχο-ερώτηση, όπως φαίνεται και στο Σχήμα 2.3. Επιπλέον χαρακτηριστικό της καινούργιας μεθόδου ήταν η αναζήτηση της βάσης δεδομένων πρώτα-κατά-πλάτος (breadth-first search). Παρόλα αυτά, το πρόβλημα που είχε μια απευθείας εφαρμογή αυτής της μεθόδου στο αρχικό πρόγραμμα, ήταν ότι θα παρήγαγε και γεγονότα που δεν θα ήταν σχετικά με τον στόχο-ερώτηση. Σύντομα δόθηκε λύση στο πρόβλημα αυτό, με τη χρήση τεχνικών μετασχηματισμού, οι οποίες διατηρούν το χαρακτηριστικό της εστίασης σε σχετικά με τον στόχο-ερώτηση γεγονότα. Επιπλέον, η νέα αυτή μέθοδος διατηρεί την δηλωτική σημασιολογία επιτρέποντας σημαντικές βελτιστοποιήσεις των προγραμμάτων. Από την άλλη μεριά, η top-down μέθοδος χρησιμοποιεί την λειτουργική σημασιολογία και βασίζεται σε αυτήν για να κάνει βελτιστοποιήσεις των προγραμμάτων, όπως για παράδειγμα η βελτιστοποίηση αναδρομής στην ουρά (tail-recursion optimization).

Το όφελος που μπορούμε να έχουμε από την χρήση της bottom-up μεθόδου έναντι της top-down μπορεί να φανεί καλύτερα στο παράδειγμα 2.3.2.

Θα πρέπει ωστόσο να σημειωθεί πως δεν είναι πάντοτε καλύτερη η bottom-up μέθοδος. Ενδεικτικά αναφέρουμε πως στην περίπτωση που αναζητούμε μόνο μία λύση, τότε με την top-down μέθοδο μπορούμε να βρούμε γρήγορο τρόπο να πάρουμε την απάντηση (χάνοντας σε δηλωτικότητα).

Η απλή bottom-up μέθοδος (ή fixpoint επανάληψη), όπως εκτελείται με τον τελεστή αμέσου επακολούθου T_P που ορίστηκε νωρίτερα, έχει το μειονέκτημα πως υπολογίζει μερικά γεγονότα περισσότερες φορές. Αυτή η μέθοδος είναι γνωστή και σαν *αφελής αποτίμηση (naive evaluation)*. Μια πιο εκλεπτυσμένη αποτίμη-



Σχήμα 2.3: *Bottom-up εκτέλεση.*

ση είναι η *ημιαφελής αποτίμηση (seminaive evaluation)*. Η αποτίμηση αυτή, που χρησιμοποιείται κατά κύριο λόγο στις επαγωγικές βάσεις δεδομένων, είναι πιο αποτελεσματική, διότι δεν υπολογίζει γεγονότα πολλές φορές, καθώς εξασφαλίζει σε κάθε επανάληψη, ότι μόνο κανόνες που χρησιμοποιούν τουλάχιστον ένα νέο γεγονός που παράχθηκε στην προηγούμενη επανάληψη, μπορούν να παράγουν νέα γεγονότα. Επιπλέον, απαλείφουμε τα γεγονότα που υπάρχουν πολλές φορές (duplicate elimination), που συνεπάγεται βέβαια ένα επιπλέον κόστος.

Πρώτα θα ορίσουμε ένα δυαδικό τελεστή, όμοιο με τον τελεστή αμέσου επακόλουθου Όπου υπάρχουν άγκιστρα ([...]) αναφερόμαστε σε πολυσύνολα (επιτρέπονται δηλαδή πολλαπλές εμφανίσεις του ίδιου στοιχείου).

$$\begin{aligned} W_P(X,Y) = [\quad & h\theta \mid h \leftarrow b_1,...,b_k \text{ είναι κανόνας του } P, \\ & \theta \text{ είναι ο μέγιστος κοινός ενοποιητής των } \{b_1,...,b_k\}, \{d_1,...,d_k\}, \\ & Y \subseteq X, \{d_1,...,d_k\} \subseteq X, \text{ και} \\ & k \geq 0,\ \{d_1,...,d_k\} \cap Y \neq \emptyset.] \end{aligned}$$

Διαισθητικά, ο τελεστής W_P επιτρέπει να παραχθούν μόνο γεγονότα από το σύνολο γεγονότων X που χρησιμοποιούν "νέα" γεγονότα από το Y, δηλαδή το X έχει όλα τα γεγονότα, ενώ το Y μόνο τα νέα γεγονότα. Επιτρέπονται πολλαπλότητες στα γεγονότα που παράγονται, καθώς μπορεί να υπάρχουν περισσότεροι κανόνες που αν πάρουμε ένα στιγμιότυπο τους, παράγουν το ίδιο γεγονός.

Ορισμός 2.3.1 *Ημι-αφελής αποτίμηση (Seminaine evaluation-SN):*

$$\begin{aligned} S_{-1} &= S_0 = \delta_0 = D \\ \delta_{n+1} &= dup\_elim(W_P(S_n, \delta_n - S_{n-1})) \\ S_{n+1} &= dup\_elim(S_n \cup \delta_{n+1}) \\ S &= lim_{n\to\infty} S_n \\ GC &= set(S). \end{aligned}$$

όπου το D η βάση δεδομένων, dup_elim είναι η συνάρτηση που βρίσκει και διαγράφει διπλότυπα, S_n είναι το σύνολο όλων των γεγονότων που έχουν παρα-

χθεί μέχρι το n βήμα, δ_n τα γεγονότα που μπήκαν στο n βήμα και GC είναι τα υπολογισμένα γεγονότα (generated consequences).

Με λίγα λόγια, θα μπορούσαμε να πούμε πώς το σύνολο των γεγονότων που παράγονται στην n επανάληψη (δ_n) συγκρίνεται με το σύνολο των γνωστών γεγονότων της προηγούμενης επανάληψης (S_{n-1}) για να βρεθούν τα νέα γεγονότα που παράγονται ($\delta_n - S_{n-1}$). Τα διπλότυπα διαγράφονται από την συνάρτηση *set*. Μόνο παραγωγές που χρησιμοποιούν νέα γεγονότα συνεχίζονται στην $n+1$ επανάληψη, αποφεύγοντας έτσι τις επαναλαμβανόμενες παραγωγές. Ο αλγόριθμος τερματίζει όταν στο $n+1$ βήμα φτάνουμε σε σταθερό σημείο, δηλαδή όταν $S_{n+1} = S_n$ (πρέπει να ελέγξουμε αν $\delta_{n+1} \subseteq S_n$). Η ημιαφελής αποτίμηση τερματίζει αν και μόνο αν το S είναι πεπερασμένο.

Παράδειγμα 2.3.3 *Έστω το ακόλουθο πρόγραμμα:*

```
d ← c.
c ← b.
b ← a.
a.
b.
```

Το παραπάνω πρόγραμμα δίνει σαν αποτέλεσμα τα ακόλουθα, αν εκτελεστεί με την ημιαφελή αποτίμηση: 1ο βήμα: `a, b`, *2ο βήμα:* `c`, *3ο βήμα:* `d`. *Από την άλλη, η αφελής αποτίμηση δίνει σαν αποτέλεσμα: 1ο βήμα:* `a, b`, *2ο βήμα:* `a,b,c`, *3ο βήμα:* `a,b,c,d`, *δηλαδή ότι παράγεται σε κάποιο βήμα k παράγεται και σε κάθε επόμενο βήμα $j > k$.*

Προκειμένου να μπορεί να χρησιμοποιηθεί η bottom-up αποτίμηση αποδοτικά, αναπτύχθηκαν διάφορες *τεχνικές επανεγγραφής* προγραμμάτων. Μια τέτοια κατηγορία είναι και οι *μετασχηματισμοί διάδοσης τιμών (value propagating techniques)* που περιγράφονται στις επόμενες ενότητες.

2.4 Μετασχηματισμός magic sets

Σε αυτή την ενότητα θα παρουσιάσουμε τον μετασχηματισμό Magic Sets, ο οποίος παρουσιάστηκε πρώτη φορά στο [7]. Διαισθητικά, ο μετασχηματισμός magic sets μιμείται μια top-down αποτίμηση, κατά την οποία οι τιμές που παράγονται σε ένα κατηγόρημα μεταφέρονται στα ορίσματα άλλων κατηγορημάτων, που είναι γνωστό ως *παράπλευρο πέρασμα πληροφοριών (sideways information passing)* [9]. Ας το δούμε αυτό λίγο πιο αναλυτικά: όταν καλείται ένας στόχος με κάποια ορίσματά του δεσμευμένα (δηλ. υπάρχουν κάποιες τιμές για αυτά τα ορίσματά του), τότε αυτή την πληροφορία μπορούμε να την χρησιμοποιήσουμε για να περιορίσουμε τον χώρο που θα αναζητήσουμε τις απαντήσεις στην αρχική μας επερώτηση. Αν ο στόχος με τα δεσμευμένα ορίσματα αντιστοιχεί σε ένα EDB άτομο, τότε στη βάση δεδομένων θα αναζητήσουμε μόνο πλειάδες που ταιριάζουν στις δοσμένες τιμές και θα αγνοήσουμε τις υπόλοιπες. Επιπλέον, οι επιστρεφόμενες τιμές θα δεσμεύσουν τις τιμές κάποιων άλλων υποστόχων του σώματος στο οποίο εμφανίζεται αυτός ο στόχος. Αν από την άλλη, πρόκειται για ένα IDB άτομο, τότε όταν θα απαντηθεί (μέσω των κανόνων που το ορίζουν), οι τιμές αυτές θα δεσμεύσουν κάποια άλλα ορίσματα ατόμων στο σώμα του κανόνα στο οποίο βρισκόταν αυτός ο στόχος, όταν κλήθηκε. Επομένως και στις δύο περιπτώσεις, οι τιμές των δεσμευμένων ορισμάτων θα δεσμεύσουν κάποια άλλα ορίσματα, περνώντας πληροφορίες παράπλευρα. Αυτός είναι ένας εναλλακτικός τρόπος για πέρασμα πληροφορίας στα προγράμματά μας, πέρα από την ενοποίηση.

Ας δούμε πως μπορεί να γίνει αυτό το παράπλευρο πέρασμα πληροφορίας μετασχηματίζοντας ένα πρόγραμμα. Η ιδέα είναι να προσθέσουμε κάποια νέα άτομα (σχέσεις) στους κανόνες, τα οποία θα λειτουργούν σαν φίλτρα, υπολογίζοντας με αυτό τον τρόπο μόνο τις τιμές που περνάνε παραπλεύρως. Τις σχέσεις αυτές, τις ονομάζουμε *μαγικά σύνολα (magic sets).* Σε μια top-down αποτίμηση, αν χρησιμοποιηθεί ένας κανόνας της μορφής:

$$q(\vec{IN}, \vec{OUT}) \leftarrow e(\cdots), p(\vec{IN}, \vec{OUT}), f(\cdots).$$

τότε ο υποστόχος $p(\vec{IN}, \vec{OUT})$ θα κληθεί, αν κληθεί το $q(\vec{IN}, \vec{OUT})$ και απαντηθούν όλοι οι υποστόχοι που βρίσκονται πριν από το p. Ακόμα, τα δεσμευμένα ορίσματα του p (εδώ συμβολίζονται με $\vec{IN}$) θα έχουν προκύψει από τα δεσμευμένα ορίσματα του q (που συμβολίζονται με $\vec{IN}$) και τις τιμές που προκύπτουν από τους υποστόχους που προηγούνται. Επομένως, σε μορφή κανόνων θα είχαμε:

$$magic\_p(\vec{IN}) \leftarrow magic\_q(\vec{IN}), e(\cdots).$$

Επιπλέον, επειδή και η ερώτηση-στόχος αποτελεί την αρχική κλήση, οι δεσμευμένες της τιμές, θα αποτελούν το γεγονότα που θα ξεκινούν τον bottom-up υπολογισμό (δηλ. θα είναι τιμές για το $magic\_s$, όπου το s είναι το κατηγόρημα της ερώτησης-στόχου).

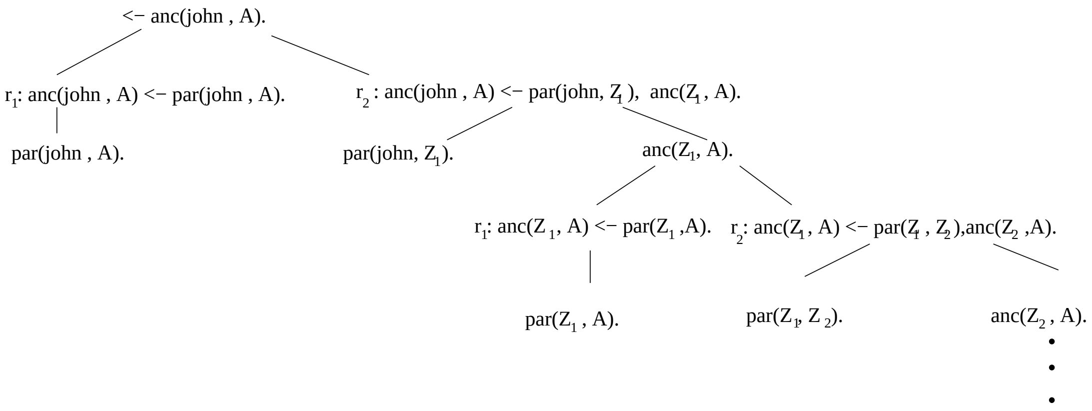
Συνεπώς, κάθε κανόνας που ορίζει ένα κατηγόρημα p, θα δεχτεί ένα άτομο $magic\_p$ στο σώμα του, όπου τα ορίσματά του θα είναι τα δεσμευμένα ορίσματα της κεφαλής του κανόνα.

Πριν περιγράψουμε τον αλγόριθμο magic sets, θα ορίσουμε μερικές έννοιες που είναι απαραίτητες προκειμένου να γίνει η περιγραφή.

Όταν υπολογίζεται ένας στόχος ως προς ένα δοσμένο πρόγραμμα με top-down τρόπο, δημιουργείται ένα δέντρο γνωστό ως *κανόνας/στόχος δέντρο (rule/goal tree).* Στο κλασικό Λ.Π. αυτό είναι το γνωστό δέντρο παραγωγής (derivation tree) που προκύπτει από την επίλυση (resolution) [35]. Κάθε κόμβος του δέντρου είναι είτε ένας στόχος είτε ένας κανόνας του προγράμματος. Το δέντρο αυτό, για έναν αρχικό στόχο G_0 κατασκευάζεται ως εξής:

1. Η ρίζα του δέντρου είναι ο κόμβος-στόχος G_0.

2. Έστω ότι ο κόμβος n είναι ένας κόμβος-στόχος για το στόχο G, με κατηγόρημα p. Τότε για κάθε κανόνα r που η κεφαλή του ενοποιείται με το p, φτιάχνουμε έναν κόμβο-κανόνα παιδί του κόμβου G, που περιέχει τον κανόνα r τροποποιημένο (μετονομασία κοινών μεταβλητών ανάμεσα στο κόμβο G και στις μεταβλητές της κεφαλής του r).



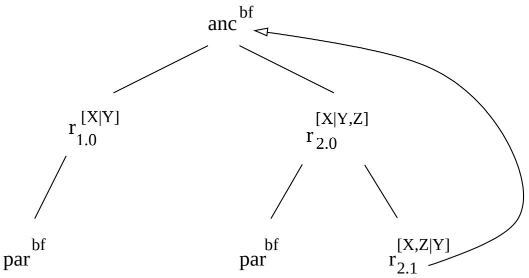
Σχήμα 2.4: *Το δέντρο κανόνα/στόχου του παραδείγματος anc*

3. Τα παιδιά ενός κόμβου-κανόνα είναι κόμβοι-στόχοι και αντιστοιχούν στους στόχους που υπάρχουν στο σώμα του κανόνα (με μετονομασία μεταβλητών).

Παράδειγμα 2.4.1 *Συνεχίζοντας το παράδειγμα 2.2.1 που ορίζει τους προγόνους ενός ατόμου με ερώτηση-στόχο* $\leftarrow$ `anc(john,A)`, *το δέντρο που παράγεται φαίνεται στο Σχήμα 2.4.*

Από το δέντρο αυτό φαίνεται πως περνάει η πληροφορία κατά την top-down εκτέλεση του προγράμματος. Καταρχήν, στο αριστερό παιδί της ρίζας, η σχέση `par` *υπολογίζεται έχοντας δεσμευμένο το πρώτο όρισμα με την τιμή* `john`. *Με άλλα λόγια, η σχέση αυτή υπολογίζεται με όλες εκείνες τις πλειάδες της βάσης δεδομένων που έχουν τη σταθερά* `john` *στη πρώτη θέση. Οι απαντήσεις που προκύπτουν, μεταφέρονται στον κόμβο γονέα, που είναι ένας κόμβος-κανόνας και από εκεί μεταβιβάζονται στον γονικό κόμβο, που είναι κόμβος-στόχος και αποτελεί το ερώτημα-στόχο. Το πιο ενδιαφέρον ωστόσο κομμάτι του υπολογισμού λαμβάνει χώρα στο δεξί παιδί της ρίζας. Σε αυτό, πάλι στο αριστερό παιδί αυτού του κόμβου-κανόνα, υπάρχει ένας κόμβος-στόχος με το EDB* `par`, *με το πρώτο όρισμα να είναι* `john`. *Όπως και προηγουμένως, θα υπολογιστούν κάποιες τιμές για το δεύτερο όρισμα της σχέσης* `par`. *Αυτές οι τιμές θα ανέβουν στο κόμβο γονέα και στη συνέχεια θα κατέβουν στο δεύτερο στόχο του σώματος του κανόνα, που είναι ο κόμβος-στόχος* `anc`. *Επομένως, αυτός ο στόχος θα υπολογιστεί στη συνέχεια με το πρώτο του όρισμα να περιορίζεται σε κάποιες τιμές. Με άλλα λόγια, έχουμε παράπλευρο πέρασμα πληροφορίας (sideways information passing), όπως περιγράφηκε παραπάνω.*

Στο παραπάνω παράδειγμα φάνηκε πως μπορούμε να έχουμε αυτό που ονομάζουμε παράπλευρο πέρασμα πληροφορίας. Ουσιαστικά, έχουμε τις ακόλουθες μορφές περάσματος πληροφορίας παράπλευρα: πέρασμα τιμών από κανόνα σε υποστόχους, πέρασμα από υποστόχους σε κανόνες και πέρασμα των απαντήσεων από ένα κανόνα στον υποστόχο που είχε καλέσει τον κανόνα [49]. Κατά την περιγραφή του μετασχηματισμού magic sets θα εξηγήσουμε το ρόλο κάθε νέας σχέσης ως προς αυτούς τους τρεις τρόπους περάσματος πληροφορίας. Το σημαντικό είναι πως καθώς καλούμε τους υποστόχους από αριστερά προς τα δεξιά, κάποια ορίσματα που ήταν ελεύθερα γίνονται δεσμευμένα. Ο αλγόριθμος που μας επιτρέπει να αποφασίζουμε πότε ένα όρισμα γίνεται δεσμευμένο σε έναν κανόνα της μορφής:



Σχήμα 2.5: *Ο γράφος κανόνα/στόχου του παραδείγματος anc*

$$r : H \leftarrow G_1, \ldots, G_k.$$

είναι απλός, με βάση το παράπλευρο πέρασμα πληροφορίας:

1. Μία μεταβλητή που εμφανίζεται σε δεσμευμένο όρισμα στη κεφαλή του κανόνα είναι δεσμευμένη πριν από την κλήση οποιουδήποτε υποστόχου.

2. Μία μεταβλητή είναι δεσμευμένη μετά την κλήση του υποστόχου G_i αν ήταν δεσμευμένη πριν από την κλήση του G_i ή αν εμφανίζεται στον υποστόχο G_i.

Από τα παραπάνω μπορούμε να κατασκευάσουμε το *γράφο κανόνα/στόχου (rule/goal graph)* ενός προγράμματος με μια δοσμένη ερώτηση-στόχο. Συμβολίζουμε με εκθέτες από ακολουθίες b και f τα *στολίσματα (adornments)* των κατηγορημάτων ενός προγράμματος (π.χ. p^{bfb}, σημαίνει πως το πρώτο και το τρίτο όρισμα του p είναι δεσμευμένα, ενώ το δεύτερο είναι ελεύθερο), ενώ τους στολισμούς των κανόνων τους συμβολίζουμε με ακολουθίες $[X_1, \ldots, X_n | Y_1, \ldots, Y_m]$, όπου τα X είναι τα δεσμευμένα ορίσματα, ενώ τα Y είναι τα ελεύθερα.

Παράδειγμα 2.4.2 *Συνεχίζοντας το παράδειγμα anc, έχουμε το γράφο κανόνα/στόχου που φαίνεται στο Σχήμα 2.5.*

Από ένα γράφο κανόνα/στόχου, μπορούμε να φτιάξουμε έναν γράφο στον οποίο κάθε IDB κατηγόρημα να εμφανίζεται σε όλους τους κανόνες με το ίδιο στόλισμα (adornment) (βλ. [49], παράγρ. 8.9, σελ. 799). Η ιδιότητα αυτή θα αναφέρεται στο εξής ως *ιδιότητα μοναδικής δέσμευσης (unique binding pattern)*.

Από αυτό το γράφο μπορούμε να φτιάξουμε τη στολισμένη εκδοχή ενός προγράμματος (adorned program), στην οποία κάθε IDB κατηγόρημα έχει σαν εκθέτη το (μοναδικό) στόλισμά του.

Παράδειγμα 2.4.3 *Το παράδειγμα anc στην στολισμένη εκδοχή του γίνεται ως εξής:*

$$\mathtt{anc}^{bf}\mathtt{(X,Y)} \leftarrow \mathtt{par(X,Y)}.$$
$$\mathtt{anc}^{bf}\mathtt{(X,Y)} \leftarrow \mathtt{par(X,Z)},\mathtt{anc}^{bf}\mathtt{(Z,Y)}.$$

και η ερώτηση-στόχος είναι: $\leftarrow \mathtt{anc}^{bf}\mathtt{(john,A)}$.

Αλγόριθμος: Δίνεται ένα σύνολο κανόνων και ένας στόχος $\leftarrow q(a_1, \ldots, a_n)$, όπου μερικά από τα a_i είναι δεσμευμένα. Θεωρούμε πως οι κανόνες έχουν την ιδιότητα της μοναδικής δέσμευσης ως προς αυτό το στόχο. Από αυτά, παράγουμε ένα νέο σύνολο κανόνων, ώστε η σχέση q να είναι η απάντηση στην επερώτηση. Για το μετασχηματισμό, χρειάζεται να ορίσουμε τα ακόλουθα νέα κατηγορήματα:

1. Μαγικά κατηγορήματα (magic predicates) ορίζονται για τα IDB κατηγορήματα. Για κάθε IDB κατηγόρημα p ορίζεται ένα μαγικό κατηγόρημα m_p. Τα ορίσματα του m_p είναι τα δεσμευμένα ορίσματα του p. Με τις σχέσεις που ορίζονται για τα μαγικά κατηγορήματα, φυλάμε τις τιμές των δεσμευμένων ορισμάτων της κεφαλής ενός κανόνα με κεφαλή p.

2. Συμπληρωματικά κατηγορήματα (supplementary predicates). Για κάθε κανόνα r_j με k υποστόχους, φτιάχνονται συμπληρωματικά κατηγορήματα $sup_{j.i}$ για $i = 0, \ldots, k-1$. Με τις σχέσεις που υπολογίζονται για τα συμπληρωματικά κατηγορήματα περνάμε από τους στόχους στους κανόνες.

Στη συνέχεια ορίζουμε κανόνες για τα νέα κατηγορήματα και τροποποιούμε τους ήδη υπάρχοντες κανόνες.

1. *Κανόνες για τα μαγικά κατηγορήματα.* Έστω p ένα IDB κατηγόρημα. Ενδιαφερόμαστε για όλους τους κανόνες που έχουν το p κατηγόρημα στο σώμα τους, καθώς το συμπληρωματικό κατηγόρημα της προηγούμενης θέσης που εμφανίζεται το p ενός τέτοιου κανόνα, θα καθορίσει την τιμή του μαγικού κατηγορήματος. Ας υποθέσουμε πως εμφανίζεται στον κανόνα r_j στον i υποστόχο και αυτός ο υποστόχος είναι ο $p(t_1, \ldots, t_n)$. Επιπλέον, ότι τα ορίσματα του $sup_{j.i-1}$ είναι οι μεταβλητές $U_1, \ldots, U_l$ και τα δεσμευμένα ορίσματα του p είναι τα $i_1, \ldots, i_m$. Τότε φτιάχνουμε τον κανόνα

   $$m\_p(t_{i_1}, \ldots, t_{i_m}) \leftarrow sup_{j.i-1}(U_1, \ldots, U_l).$$

   Ουσιαστικά, στη σχέση m_p φυλάμε τις τιμές του υποστόχου p που θα έχουν τα δεσμευμένα του ορίσματα, όταν κληθεί κάποιος κανόνας με αυτό το κατηγόρημα στη κεφαλή.

2. *Κανόνες για τα μηδενικά συμπληρωματικά κατηγορήματα.* Το μαγικό κατηγόρημα της κεφαλής του κανόνα που μας αφορά θα καθορίσει την τιμή του συμπληρωματικού κατηγορήματος. Έστω κανόνας r_j με κεφαλή $p(t_1, \ldots, t_n)$ και έστω $i_1, \ldots, i_m$ είναι τα δεσμευμένα ορίσματά του. Επιπλέον, $X_1, \ldots, X_n$ είναι οι μεταβλητές που είναι δεσμευμένες από την κεφαλή του κανόνα r_j, δηλ. εμφανίζονται μέσα στα ορίσματα $t_{i_1}, \ldots, t_{i_m}$. Τότε ο μοναδικός κανόνας για το $sup_{j.0}$ είναι

   $$sup_{j.0}(X_1, \ldots, X_n) \leftarrow m\_p(t_{i_1}, \ldots, t_{i_m}).$$

   Όταν κληθεί ένας κανόνας με κατηγόρημα κεφαλής p, τότε πριν κληθεί οποιοσδήποτε υποστόχος στο σώμα, οι μόνες δεσμευμένες τιμές θα είναι αυτές που υπάρχουν στη σχέση m_p.

3. *Κανόνες για τα υπόλοιπα συμπληρωματικά κατηγορήματα.* Για κάθε κανόνα r_j, με $k \geq 1$ υποστόχους, και για κάθε $i = 1, \ldots, k-1$, κατασκευάζουμε έναν κανόνα για το $sup_{j.i}$. Έστω ότι οι μεταβλητές του r_j που είναι ορίσματα στο $i-1$-οστό συμπληρωματικό κατηγόρημα του r_j είναι οι $U_1, \ldots, U_l$ και οι μεταβλητές που είναι ορίσματα στο i-στό συμπληρωματικό κατηγόρημα του r_j είναι οι $V_1, \ldots, V_m$. Τέλος, $p(t_1, \ldots, t_n)$ είναι ο i-στός υποστόχος του r_j. Τότε έχουμε τον κανόνα

$$sup_{j.i}(V_1, \ldots, V_m) \leftarrow sup_{j.i-1}(U_1, \ldots, U_l), p(t_1, \ldots, t_n).$$

   Οι τιμές που είχαν φυλαχθεί στο προηγούμενο συμπληρωματικό κατηγόρημα ($sup_{j.i-1}$) μαζί με τις τιμές που επιστρέφονται από τον τρέχοντα υποστόχο (p), δημιουργούν τις επόμενες χρήσιμες τιμές στη σχέση $sup_{j.i}$.

4. *Κανόνες για τους κανόνες του αρχικού προγράμματος.* Έστω p ένα IDB κατηγόρημα και r_j ένας κανόνας από τους κανόνες που ορίζουν το p. Έστω πως ο r_j έχει $k \geq 1$ υποστόχους. Οι κανόνες που ορίζουν το p καθορίζονται από το $sup_{j.k-1}$ (τελευταίο συμπληρωματικό κατηγόρημα του r_j) και τον τελευταίο υποστόχο του r_j. Αναλυτικά, αν $U_1, \ldots, U_l$ είναι οι μεταβλητές που σχηματίζουν τα ορίσματα του $k-1$-οστού συμπληρωματικού κατηγορήματος του r_j, $p(t_1, \ldots, t_n)$ είναι η κεφαλή για τον r_j και ο k-οστός (τελευταίος) υποστόχος είναι ο $q(v_1, ..., v_m)$, τότε έχουμε τον κανόνα:

$$p(t_1, \ldots, t_n) \leftarrow sup_{j.k-1}(U_1, \ldots, U_l), q(v_1, \ldots, v_m).$$

   Με αυτό τον τρόπο, οι απαντήσεις αυτού του κανόνα μπορούν να περάσουν στον στόχο που κάλεσε αυτό τον κανόνα.

5. *Κανόνας αρχικοποίησης.* Έστω πως στον ερώτηση-στόχο $\leftarrow q(a_1, \ldots, a_n)$ τα $i_1, \ldots, i_m$ ορίσματα είναι τα δεσμευμένα ορίσματα της επερώτησης. Τότε έχουμε τον κανόνα

$$m\_q(a_{i_1}, \ldots, a_{i_m}).$$

   που λέγεται *σπόρος (seed)*. Διαισθητικά, αυτός ο κανόνας εκφράζει το γεγονός πως μας ενδιαφέρουν μόνο απαντήσεις με αυτές τις σταθερές για τα δεσμευμένα ορίσματα του q. Επιπλέον, από αυτόν τον κανόνα αρχίζει και η από-κάτω-προς-τα-πάνω αποτίμηση (bottom-up evaluation).

Ο παραπάνω αλγόριθμος είναι ο βασικός μετασχηματισμός magic sets. Ωστόσο γίνονται κάποιες βελτιώσεις με τη χρήση άλλων τεχνικών βελτιστοποίησης, όπως είναι η απαλοιφή κατηγορημάτων (predicate elimination) και το ξεδίπλωμα (unfolding).

Παράδειγμα 2.4.4 *Το παράδειγμα 4.3.1 του [31] γίνεται με το μετασχηματισμό magic sets:*

```
m_q(a1).
m_q(X) ← sup3.0(X).
m_p(Y) ← sup2.1(X,Y).
sup1.0(X) ← m_q(X).
sup2.0(X) ← m_q(X).
sup3.0(X) ← m_p(X).
sup2.1(X,Y) ← sup2.0(X),e(X,Y).
sup3.1(X,W) ← sup3.0(X),q(X,W).
q(X,Z) ← sup1.0(X),f(X,Z).
q(X,Z) ← sup2.1(X,Y),p(Y,Z).
p(X,Z) ← sup3.1(X,W),g(X,W,Z).
```

Παράδειγμα 2.4.5 *Μετά τις βελτιώσεις, το πρόγραμμα γίνεται ως εξής:*

```
m_q(a1).
m_q(X) ← m_p(X).
m_p(Y) ← sup2.1(X,Y).
sup2.1(X,Y) ← m_q(X),e(X,Y).
q(X,Z) ← m_q(X),f(X,Z).
q(X,Z) ← m_q(X),e(X,Y),p(Y,Z).
p(X,Z) ← m_p(X),q(X,W),g(X,W,Z).
```

Μια παραλλαγή του μετασχηματισμού που μόλις περιγράφηκε, δεν χρησιμοποιεί τα συμπληρωματικά κατηγορήματα (supplemantary predicates). Σε αυτή την περίπτωση επαναλαμβάνουμε κάποιους υπολογισμούς στο σώμα κάποιων κανόνων, αλλά επειδή έχει επικρατήσει να θεωρείται αυτός ο μετασχηματισμός magic sets, από εδώ και στο εξής αυτόν θα χρησιμοποιούμε, ενώ ο προηγούμενος θα αναφέρεται ως supplementary magic sets.

Αλγόριθμος: Δίνεται ένα σύνολο κανόνων και ένας στόχος $\leftarrow q(a_1, \ldots, a_n)$, όπου μερικά από τα a_i είναι δεσμευμένα. Θεωρούμε πως οι κανόνες έχουν την ιδιότητα της μοναδικής δέσμευσης ως προς αυτό το στόχο. Από αυτά, παράγουμε ένα νέο σύνολο κανόνων, ώστε η σχέση q να είναι η απάντηση στην επερώτηση. Για το μετασχηματισμό, χρειάζεται να ορίσουμε τα ακόλουθα νέα κατηγορήματα: για κάθε IDB κατηγόρημα p ορίζουμε ένα νέο κατηγόρημα m_p. Σε κάθε κανόνα της μορφής:

$$r : p^a(\vec{A}) \leftarrow q_1^{a_1}(\vec{A_1}), \ldots, q_m^{a_m}(\vec{A_m}).$$

όπου τα a_i, $1 \leq i \leq m$ είναι τα στολίσματα των κατηγορημάτων, αν αυτά είναι IDB, αλλιώς είναι κενά, φτιάχνουμε έναν κανόνα για κάθε IDB. Έστω πως το $q_j^{a_j}$ είναι ένα IDB. Τότε ο κανόνας για το magic κατηγόρημα του είναι:

$$r : m\_q_j^{a_j}(\vec{A_j^b}) \leftarrow m\_p^a(\vec{A^b}), q_1^{a_1}(\vec{A_1}), \ldots, q_{j-1}^{a_{j-1}}(\vec{A_{j-1}}).$$

όπου το $\vec{A^b}$ δηλώνει τα δεσμευμένα ορίσματα του $\vec{A}$. Επιπλέον, ο κανόνας r γίνεται:

$$r : p^a(\vec{A}) \leftarrow m\_p^a(\vec{A^b}), q_1^{a_1}(\vec{A_1}), \ldots, q_m^{a_m}(\vec{A_m}).$$

2.5 Μέθοδος counting

Υπάρχουν κάποιες ειδικές περιπτώσεις στις οποίες ο magic sets μετασχηματισμός υστερεί και θα μπορούσε κάποιος άλλος μετασχηματισμός να έχει καλύτερα αποτέλεσμα.

Αυτό θα φανεί καλύτερα στο ακόλουθο παράδειγμα που είναι μια παραλλαγή του παραδείγματος εύρεσης των ατόμων που είναι στην ίδια γενιά με κάποιον.

Παράδειγμα 2.5.1

```
sg(X,Y) ← person(X,Y).
sg(X,Y) ← par(X,Xp),sg(Xp,Yp),rap(Yp,Y).
```

Το παράδειγμα αυτό μπορεί να μετασχηματιστεί στο ακόλουθο πρόγραμμα:

```
up_sg(0,a).
up_sg(succ(I),Xp) ← up_sg(I,X),par(X,Xp).
down_sg(I,Y) ← up_sg(I,X),person(X,Y).
down_sg(I,Y) ← down_sg(succ(I),Yp),rap(Yp,Y).
```

Έστω πως έχουμε την ακόλουθη βάση δεδομένων που είναι περίπου όπως στο [45]: `par(a,a`$_i$`)`, `par(a`$_i$`,c)`, *με* $i = 1, ..., m$, `par(a`$_m$`,f)`, `person(f,e`$_1$`)`, `person(c,d)`, `rap(d,e`$_j$`)`, `rap(e`$_j$`,e)`, *με* $j = 1, ..., n$, *και* `rap(e,b)`. *Το πρόγραμμα θα χρειαστεί να κατασκευάσει μόνο τα* `sg(f,e`$_1$`)`, `sg(c,d)` *ενδιάμεσα για να δώσει σαν απάντηση* `sg(a,e)`,`sg(a,b)`, *ενώ ο magic sets μετασχηματισμός θα χρειαστεί να παράγει* $m * n + 1$ *γεγονότα για την σχέση* `sg`, *δηλ. όλα τα* `sg(a`$_i$`,e`$_j$`)`, *για* $i = 1, ..., m, j = 1, ..., n$.

Λέμε ότι ένα λογικό πρόγραμμα είναι *μετρήσιμο-γραμμικό (counting-linear)*:

1. Αν υπάρχει μόνο ένα IDB κατηγόρημα p.

2. Υπάρχει ακριβώς ένας κανόνας που έχει αυτό το p σαν υποστόχο.

3. Μπορούμε να διατάξουμε τους υποστόχους μέσα στον κανόνα έτσι ώστε το p να έχει "στόλισμα" α, και οι υποστόχοι που βρίσκονται πριν από τον p-υποστόχο λέγονται *επάνω (up) υποστόχοι*, ενώ οι υποστόχοι που βρίσκονται μετά από τον p-υποστόχο λέγονται *κάτω (down) υποστόχοι*.

4. Ορίζουμε τα ορίσματα που εμφανίζονται είτε σε έναν επάνω υποστόχο είτε στις δεσμευμένες θέσεις του p-υποστόχου είτε στις δεσμευμένες θέσεις της κεφαλής να λέγονται *επάνω ορίσματα*. Αντίστοιχα τα ορίσματα που εμφανίζονται στους κάτω υποστόχους είτε στις ελεύθερες μεταβλητές του p-υποστόχου είτε στις ελεύθερες μεταβλητές της κεφαλής να λέγονται *κάτω ορίσματα*.

Όπως είναι προφανές, αυτός ο ορισμός είναι πολύ ειδικός και καλύπτει μια πολύ περιορισμένη κλάση λογικών προγραμμάτων, για το λόγο αυτό αργότερα προτάθηκε μια επέκταση που καλύπτει μια σαφώς πιο ευρεία κλάση προγραμμάτων.

Αλγόριθμος: Δίνεται ένα σύνολο κανόνων και ένας στόχος $\leftarrow q(a_1, \ldots, a_n)$, όπου μερικά από τα a_i είναι δεσμευμένα. Θεωρούμε πως οι κανόνες έχουν την ιδιότητα της μοναδικής δέσμευσης ως προς αυτό το στόχο. Από αυτά, παράγουμε ένα νέο σύνολο κανόνων, ώστε η σχέση q να είναι η απάντηση στην επερώτηση. Έστω πως έχουμε τον ακόλουθο κανόνα:

$$p(s_1, \ldots, s_n, t_1, \ldots, t_m) \leftarrow U_1, \ldots, U_k, p(u_1, \ldots, u_n, v_1, \ldots, v_m), D_1, \ldots, D_l.$$

όπου τα $s_1, \ldots, s_n, u_1, \ldots, u_n$ εμφανίζονται στις δεσμευμένες θέσεις του p στην κεφαλή και στο σώμα. Τα $U_1, \ldots, U_k$ είναι οι επάνω υποστόχοι και $D_1, \ldots, D_l$ είναι οι κάτω υποστόχοι.

Τότε φτιάχνουμε τους ακόλουθους κανόνες:

1. Για τα επάνω κατηγορήματα φτιάχνουμε τον βασικό κανόνα

$$up\_p(0, a_1, \ldots, a_n).$$

   όπου τα $a_1, \ldots, a_n$ είναι οι σταθερές του στόχου.

2. Ο αναδρομικός κανόνας για το επάνω κατηγόρημα είναι

$$up\_p(succ(I), u_1, \ldots, u_n) \leftarrow U_1, \ldots, U_k, up\_p(I, s_1, \ldots, s_n).$$

3. Για κάθε βασικό κανόνα της μορφής:

$$p(w_1, \ldots, w_n, w_{n+1}, \ldots, w_{n+m}) \leftarrow J_1, \ldots, J_j.$$

   κατασκεύασε τον κανόνα:

$$down\_p(I, w_{n+1}, \ldots, w_{n+m}) \leftarrow up\_p(I, w_1, \ldots, w_n), J_1, \ldots, J_j.$$

4. Ο αναδρομικός κανόνας για το κάτω κατηγόρημα είναι:

$$down\_p(I, t_1, \ldots, t_m) \leftarrow down\_p(succ(I), v_1, \ldots, v_m), D_1, \ldots, D_l.$$

5. Τέλος κατασκεύασε τον κανόνα που θα δώσει τις απαντήσεις:

$$p(a_1, \ldots, a_n, Y_1, \ldots, Y_m) \leftarrow down\_p(0, Y_1, \ldots, Y_m).$$

Πρέπει να παρατηρήσουμε πως το πρώτο όρισμα δείχνει την απόσταση των σταθερών από την απάντηση.

2.5.1 Γενικευμένη μέθοδος μέτρησης

Ωστόσο η παραπάνω μέθοδος εφαρμόζεται μόνο σε μια πολύ περιορισμένη κλάση προγραμμάτων. Για το λόγο αυτό, η παραπάνω μέθοδος επεκτάθηκε ώστε να χειρίζεται και πιο γενικά Datalog προγράμματα. Η επέκταση αυτή είναι γνωστή ως *γενικευμένη μέθοδος μέτρησης (generalized counting method).*

Στην μέθοδο αυτή, έχουμε δύο fixpoint υπολογισμούς: ο πρώτος που εκτελείται στους *counting κανόνες* και στους *supplementary counting κανόνες* ουσιαστικά διαδίδει τις δεσμεύσεις των μεταβλητών, και ο δεύτερος που εκτελείται στους *τροποποιημένους κανόνες (modified)* υπολογίζει τις απαντήσεις του προγράμματος. Για κάθε IDB κατηγόρημα p^α, που υπάρχει στο πρόγραμμα, όπου το στόλισμα α δείχνει ποιά ορίσματα είναι δεσμευμένα και ποιά ελεύθερα, αντιστοιχίζουμε ένα νέο κατηγόρημα p_ind^α, με 3 νέα ορίσματα. Τα νέα αυτά ορίσματα χρησιμοποιούνται για να κωδικοποιήσουμε: 1) το επίπεδο της αναδρομής, 2) τον αναδρομικό κανόνα που χρησιμοποιούμε και 3) το IDB κατηγόρημα που υπάρχει στο σώμα του κανόνα. Μπορούμε να κωδικοποιήσουμε με διάφορους τρόπους τους κανόνες και τα κατηγορήματα που χρησιμοποιούμε, ωστόσο θα χρησιμοποιήσουμε τον τρόπο που περιγράφουμε παρακάτω. Αν έχουμε m κανόνες, έστω $r_0, \ldots, r_m$, τότε η ακολουθία των κανόνων που χρησιμοποιούμε μπορεί να αναπαρασταθεί με μια ακολουθία αριθμών, όπου κάθε αριθμός παίρνει τιμές από 0 έως το m. Θα κωδικοποιήσουμε την ακολουθία $i_0, \ldots, i_k$ με την τιμή $\ldots(i_0 m + i_1)m + \ldots)m + i_k$. Μ' αυτόν τον τρόπο, αν ένας αριθμός παριστάνει μια ακολουθία, για να προσθέσουμε ένα στοιχείο, αρκεί να πολλαπλασιάσουμε τον δοσμένο αριθμό (δηλ. ακολουθία) με το m και μετά να προσθέσω το στοιχείο. Με την πράξη του υπολοίπου (modulo) μπορώ να πάρω όποιο στοιχείο θέλω. Παρόμοια κωδικοποίηση χρησιμοποιείται και για τα κατηγορήματα.

<u>*Αλγόριθμος:*</u> Δίνεται ένα σύνολο κανόνων και ένας στόχος $\leftarrow q(a_1, \ldots, a_n)$, όπου μερικά από τα a_i είναι δεσμευμένα. Θεωρούμε πως οι κανόνες έχουν την ιδιότητα της μοναδικής δέσμευσης ως προς αυτό το στόχο. Από αυτά, παράγουμε ένα νέο σύνολο κανόνων, ώστε η σχέση q να είναι η απάντηση στην επερώτηση. Έστω πως έχουμε έναν κανόνα της μορφής:

$$r_i \ : \ p^\alpha(\chi) \leftarrow q_1^{\alpha_1}(\theta_1), q_2^{\alpha_2}(\theta_2), \ldots, q_n^{\alpha_n}(\theta_n).$$

Προσθέτουμε έναν counting κανόνα που ορίζει το $cnt\_q\_ind_j^{\alpha_j}$, στην περίπτωση που η δέσμευση μεταβλητών μεταφέρεται από την κεφαλή στο κατηγόρημα $q_j^{\alpha_j}$. Η κεφαλή αυτού του κανόνα είναι το $cnt\_q\_ind_j^{\alpha_j}(I+1, K \times m + i, H \times t + j, \theta_j^b)$. Αν στο q_k δεσμεύονται μεταβλητές από κάποιο κατηγόρημα που βρίσκεται πριν από αυτό, τότε στο σώμα του counting κανόνα μπαίνει το $q\_ind_k^{\alpha_k}(I+1, K \times m+i, H \times t+k, \theta_k)$. Αν το q_k είναι ένα IDB κατηγόρημα και έχει στόλισμα α_k, τότε στο σώμα του counting κανόνα μπαίνει και το $cnt\_q\_ind_k^{\alpha_k}(I+1, K \times m+i, H \times t+k, \theta_k^b)$. Αν η κεφαλή δεσμεύει μεταβλητές, τότε στο σώμα του counting κανόνα προστίθεται το $cnt\_p\_ind^\alpha(I, K, H, \chi^b)$. Το m όπως είπαμε είναι το μέγιστο πλήθος κανόνων που υπάρχουν στο πρόγραμμα και t το μέγιστο πλήθος των κατηγορημάτων στο σώμα των κανόνων του προγράμματος (ή το πλήθος των αναδρομικών κατηγορημάτων, εναλλακτικά).

Ο τροποποιημένος κανόνας που αντιστοιχεί στον κανόνα r_i είναι:

$$p\_ind^{\alpha}(I, K, H/t, \chi) \leftarrow cnt\_p\_ind^{\alpha}(I, K, H/t, \chi^b),$$
$$q\_ind_1^{\alpha_1}(I+1, K \times m + i, H+1, \theta_1), \ldots, q\_ind_n^{\alpha_n}(I+1, K \times m + i, H+n, \theta_n).$$

Τέλος, αν η ερώτηση-στόχος είναι $\leftarrow q^{\alpha}(\eta)$, τότε προσθέτουμε το γεγονός $cnt\_q\_ind^{\alpha}(0, 0, 0, \eta^b)$, τον σπόρο (seed), που αποτελεί και το γεγονός που θα ξεκινήσει τον υπολογισμό.

Παράδειγμα 2.5.2 *Θεωρούμε το ακόλουθο παράδειγμα:*

```
← sg(john,Y).
sg(X,Y) ← flat(X,Y).
sg(X,Y) ← up(X,Z1),sg(Z1,Z2),flat(Z2,Z3),sg(Z3,Z4),down(Z4,Y).
```

το οποίο μετασχηματίζεται με την παραπάνω μέθοδο ως εξής:

```
cnt_sg_ind^bf(I+1,K × 2+2,H × 5 +2,Z1) ← cnt_sg_ind^bf(I,K,H,X),up(X,Z1).
cnt_sg_ind^bf(I+1,K × 2+2,H × 5 +4,Z3) ← cnt_sg_ind^bf(I,K,H,X),up(X,Z1),
sg_ind^bf(I+1,K × 2 + 2,H × 5 +2,Z1,Z2),flat(Z2,Z3).
sg_ind^bf(I,K,H/5,X,Y) ← cnt_ind^bf(I,K,H/5,X),flat(X,Y).
sg_ind^bf(I,K,H/5,X,Y) ← cnt_sg_inf^bf(I,K,H/5,X),up(X,Z1),
sg_ind^bf(I+1,K × 2 +2,H +2,Z1,Z2),flat(Z2,Z3),
sg_ind^bf(I+1,K × 2 +2,H+4,Z3,Z4),down(Z4,Y).
cnt_sg_ind^bf(0,0,0,john).
```

Σε γενικές γραμμές μπορούμε να πούμε πως η μέθοδος μέτρησης προτιμάται έναντι της μεθόδου magic sets, όταν μπορούμε να πετύχουμε μείωση του βαθμού των κατηγορημάτων ή όταν μπορούμε να παραλείψουμε κάποια λεκτήματα από το σώμα των κανόνων ή όταν ισχύουν και τα δύο ταυτόχρονα. Οι δύο αυτές συνθήκες και οι βελτιώσεις που μπορούμε να κάνουμε είναι γνωστές ως semijoin optimization [9]. Ο λόγος που τις κάνουμε είναι πως αφού χάνουμε κάτι λόγω της διατήρησης των δεικτών στην μέθοδο μέτρησης, θα πρέπει για αντιστάθμισμα να μην χρειάζεται να υπολογίζουμε κάποια άτομα ή να έχουν μικρότερο βαθμό μερικά κατηγορήματα.

Το βασικό μειονέκτημα αυτής της μεθόδου είναι πως δεν τερματίζει πάντα. Για παράδειγμα, αν η βάση δεδομένων είναι κυκλική ή αν έχουμε αριστερή αναδρομή σε μια σχέση. Παρόλα αυτά, έχουν προταθεί διάφορα κριτήρια ώστε να επιτυγχάνεται τερματισμός [43, 20, 42].

Παρόμοια με την εκδοχή των magic sets με συμπληρωματικά κατηγορήματα, μπορούμε να ορίσουμε και την μέθοδο *generalized supplementary counting.*

Αλγόριθμος: Δίνεται ένα σύνολο κανόνων και ένας στόχος $\leftarrow q(a_1, \ldots, a_n)$, όπου μερικά από τα a_i είναι δεσμευμένα. Θεωρούμε πως οι κανόνες έχουν την ιδιότητα της μοναδικής δέσμευσης ως προς αυτό το στόχο. Από αυτά, παράγουμε ένα νέο σύνολο κανόνων, ώστε η σχέση q να είναι η απάντηση στην επερώτηση. Έστω πως έχουμε έναν κανόνα της μορφής:

$$r_i \;:\; p^{\alpha}(\chi) \leftarrow q_1^{\alpha_1}(\theta_1), q_2^{\alpha_2}(\theta_2), \ldots, q_n^{\alpha_n}(\theta_n).$$

Έστω πως το τελευταίο IDB του κανόνα βρίσκεται στη θέση μ, δηλ. είναι το q_μ. Φτιάχνουμε για $j = 2, \ldots, \mu - 1$ τους παρακάτω συμπληρωματικούς κανόνες:

$$supcnt_j^{r_i}(I, K, H, \phi_j) \leftarrow cnt\_p\_ind^{\alpha}(I, K, H, \chi^b).$$

όπου τα ϕ_j είναι οι μεταβλητές που εμφανίζονται στα χ^b. Προσθέτουμε έναν counting κανόνα που ορίζει το $cnt\_q\_ind_j^{\alpha_j}$, στην περίπτωση που η δέσμευση μεταβλητών μεταφέρεται από την κεφαλή στο κατηγόρημα $q_j^{\alpha_j}$:

$$cnt\_q\_ind_j^{\alpha_j}(I + 1, K \times m + i, H \times t + j, \theta_j^b) \leftarrow supcnt_j^{r_i}(I, K, H, \phi_j).$$

Ο τροποποιημένος κανόνας που αντιστοιχεί στον κανόνα r_i είναι:

$$p\_ind^{\alpha}(I, K, H, \chi) \leftarrow supcnt_\mu^{r_i}(I, K, H, \phi_\mu),$$
$$q\_ind_\mu^{\alpha_\mu}(I + 1, K \times m + i, H \times t + \mu, \theta_\mu), \ldots, q\_ind_n^{\alpha_n}(\theta_n).$$

2.6 Η μέθοδος της στοίβας (Pushdown method)

Σε αυτή την ενότητα θα παρουσιάσουμε μια μέθοδο η οποία στηρίζεται στην αντιστοιχία που υπάρχει μεταξύ αλυσιδωτών επερωτήσεων (chain queries) και γραμματικών χωρίς συμφραζόμενα (context free grammars) [21, 55, 56]. Η μέθοδος αυτή μπορεί σε ορισμένες περιπτώσεις να αντικατασταθεί από την μέθοδο counting που περιγράφηκε στην προηγούμενη ενότητα. Πιο ειδικά, θα δούμε ότι η στοίβα (λίστα) που χρησιμοποιείται στην μέθοδο της στοίβας, μπορεί να αντικατασταθεί από έναν μετρητή. Ο Yannakakis [55] είχε αναπτύξει μια μέθοδο που βασιζόταν στη σχέση μεταξύ chain επερωτήσεων και γραμματικών χωρίς συμφραζόμενα (c.f.), η οποία χρησιμοποιούσε δυναμικό προγραμματισμό και υλοποιούσε την μέθοδο CYK (Cocke-Younger-Kasami) [21], για να αναγνωρίζει συμβολοσειρές γραμματικών χωρίς συμφραζόμενα (c.f.). Ωστόσο, παρότι η μέθοδος είναι αποδοτική για επερωτήσεις χωρίς δεσμευμένες τιμές στο στόχο, στην περίπτωση των επερωτήσεων που έχουμε δεσμευμένες τιμές, αυτές δεν χρησιμοποιούνται κατάλληλα για να περιοριστεί ο χώρος αναζήτησης.

Πιο συγκεκριμένα, λοιπόν, αντιστοιχούμε μια chain επερώτηση Q σε μια γραμματική χωρίς συμφραζόμενα (c.f.) G και στη συνέχεια από το αυτόματο που αναγνωρίζει τις συμβολοσειρές της $L(G)$ κατασκευάζουμε την καινούργια επερώτηση Q'.

Παρακάτω δίνουμε μερικούς ορισμούς για γραμματικές χωρίς συμφραζόμενα (c.f.) και για αυτόματα στοίβας, που είναι τα αυτόματα που αναγνωρίζουν αυτές τις γραμματικές, βλέπε [21, 56] για περισσότερες λεπτομέρειες. Μια γραμματική $G = \{V, T, \Pi, S\}$ λέγεται *χωρίς συμφραζόμενα* (context free-c.f.) αν οι κανόνες παραγωγής στο Π έχουν τη μορφή $A \rightarrow \beta$, όπου $\beta \in (V \cup T)^*$ και $A \in V$, όπου V είναι το σύνολο των *μη τερματικών συμβόλων (variables)* της γραμματικής και T είναι το σύνολο των *τερματικών συμβόλων (terminals)*, με $V \cap T = \emptyset$.

Παράδειγμα 2.6.1 $G = (V, T, \Pi, S), V = \{S, A, B\}, T = \{a, b\}$ *και* Π*:*
$S \rightarrow aB \mid bA \; A \rightarrow a \mid aS \mid bAA \; B \rightarrow b \mid bS \mid aBB$

η οποία παράγει την γλώσσα $L(G) = \{w \in T^+ \mid w \text{ έχει ίσο πλήθος από } a \text{ και } b\}$.

Ένα *αυτόματο στοίβας (push down automaton-PDA)* αποτελείται από μια ταινία εισόδου και επιπλέον από μια στοίβα. Η πρόσβαση στη στοίβα γίνεται μόνο στην κορυφή της στοίβας με τις εξής δύο λειτουργίες:

1. push: Τοποθετεί ένα στοιχείο στην κορυφή της στοίβας.
2. pop: Αφαιρεί ένα στοιχείο για χρήση από την κορυφή της στοίβας.

Στη συνέχεια δίνουμε τον τυπικό ορισμό ενός αυτομάτου στοίβας:

Ορισμός 2.6.1 *Ένα αυτόματο στοίβας (push down automaton ή για συντομία PDA) είναι μια πλειάδα $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, όπου:*

- *Q: πεπερασμένο σύνολο καταστάσεων,*
- *Σ: αλφάβητο εισόδου,*
- *Γ: αλφάβητο στοίβας,*
- *$\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow Pow(Q \times \Gamma^*)$ (πεπερασμένα υποσύνολα), η συνάρτηση μετάβασης (επιτρέπονται ϵ-κινήσεις και μη-ντετερμινισμός),*
- *$q_0 \in Q$: αρχική κατάσταση,*
- *$Z_0 \in \Gamma$: αρχικό σύμβολο στη στοίβα,*
- *$F \subseteq Q$: τελικές καταστάσεις.*

Αλγόριθμος: Δίνεται μια επερώτηση $Q = \{\leftarrow p(b, Y), P\}$, όπου το P είναι ένα σύνολο κανόνων σε chain μορφή. Από αυτά, παράγουμε μία νέα επερώτηση $Q' = \{\leftarrow p(Y, [\,]), P'\}$, όπου P' είναι ένα νέο σύνολο κανόνων, ώστε η σχέση $p(Y, [\,])$ να είναι η απάντηση στην επερώτηση. Θεωρούμε την γραμματική χωρίς συμφραζόμενα $G(Q) = \{V, T, \Pi, p\}$, όπου V το σύνολο των μη-τερματικών συμβόλων είναι τα IDB κατηγορήματα του προγράμματος P, T το σύνολο των τερματικών συμβόλων είναι τα EDB κατηγορήματα του P και Π είναι το σύνολο των κανόνων της γραμματικής, που κατασκευάζονται με τον ακόλουθο τρόπο:

Για κάθε chain κανόνα της μορφής:

$$p_0(X_0, Y_n) \leftarrow a_0(X_0, Y_0), p_1(Y_0, X_1), a_1(X_1, Y_1), \ldots, p_n(Y_{n-1}, X_n), a_n(X_n, Y_n).$$

με $n \geq 0$, υπάρχει ο κανόνας παραγωγής:

$$p_0 \rightarrow a_0, p_1, a_1, \ldots, p_n, a_n.$$

Η γλώσσα $L(G(Q))$ αναγνωρίζεται από ένα αυτόματο στοίβας δύο καταστάσεων (q_0 η αρχική κατάσταση και q η τελική κατάσταση), όπου ο πίνακας μεταβάσεων αποτελείται από μια στήλη για κάθε τερματικό σύμβολο και μια στήλη επιπλέον για το ϵ, μια γραμμή για το ζευγάρι (q_0, Z_0), όπου Z_0 είναι το αρχικό σύμβολο της στοίβας και μια γραμμή για κάθε ζευγάρι (q, v) (εκτός από το a_0), όπου v είναι σύμβολο κατηγορήματος του προγράμματος P (Παρατηρούμε πως το αλφάβητο της στοίβας δεν διαφέρει από το αλφάβητο εισόδου). Η πρώτη γραμμή του πίνακα μετάβασης, όπως φαίνεται και στο σχήμα, δείχνει ότι εισάγεται στην στοίβα το κατηγόρημα του στόχου, ενώ στη συνέχεια κάθε γραμμή μας δείχνει τι γίνεται με τα διάφορα σύμβολα, μια γραμμή για το p_0 και μια για κάθε a_i, για $1 \leq i \leq n$ (όχι το a_0). Στην περίπτωση που δεν έχουμε IDB κατηγορήματα στο σώμα ($n = 0$), τότε προσθέτουμε τον (q, ϵ) στον πίνακα για το a_0.

	a_0	a_1	$\ldots$	a_n	ε
(q_0, Z_0)					$(q, p\ Z_0)$
(q, p_0)	$(q, p_1 a_1 \ldots p_n a_n)$				
(q, a_1)		(q, ϵ)			
$\vdots$					
(q, a_n)				(q, ϵ)	

Ισοδύναμα, ο παρακάτω πίνακας μεταβάσεων μπορεί να παρασταθεί σε μορφή συνάρτησης με τον ακόλουθο τρόπο: $\delta(q_0, \epsilon, Z_0) = (q, p\ Z_0), \delta(q, a_0, p_0) = (q, p_1 a_1 \cdots p_n a_n)$, $\delta(q, a_1, a_1) = (q, \epsilon), \ldots, \delta(q, a_n, a_n) = (q, \epsilon)$. Στην περίπτωση ενός κανόνα με ένα EDB a στο σώμα έχουμε και $\delta(q, a, p) = (q, \epsilon)$.

Αν δίνεται μια ακολουθία $\alpha = a_{i_1} a_{i_2} \cdots a_{i_m}$, ένα *μονοπάτι συλλαβίζει* την α στο P, σημαίνει ότι έχουμε μια ακολουθία από $m + 1$ (όχι απαραίτητα διαφορετικές) σταθερές $b_0, \ldots, b_m$ τέτοιες ώστε για κάθε j, με $1 \leq j \leq m$, το $a_{i_j}(b_{j-1}, b_j)$ παράγεται από το P. Αν $m = 0$ τότε το μονοπάτι συλλαβίζει την κενή συμβολοακολουθία ϵ.

Είναι γνωστό πως το c ανήκει στο σύνολο λύσεων $A(Q)$, αν και μόνο αν υπάρχει μονοπάτι που αρχίζει από το b και καταλήγει στο c συλλαβίζοντας μια συμβολοακολουθία α της $L(G(Q))$ στο P. Για να υπολογίσουμε λοιπόν τις λύσεις $A(Q)$, αρκεί να αναγνωρίσουμε όλα τα μονοπάτια που αρχίζουν από το b και συλλαβίζουν μια συμβολοακολουθία α στο P. Αυτό μπορεί εύκολα να γίνει με ένα πρόγραμμα P' που υλοποιεί το αυτόματο. Το πρόγραμμα P' μπορεί να γραφτεί άμεσα από τον πίνακα μεταβάσεων, μεταφράζοντας κάθε γραμμή του πίνακα. Η αρχή του αυτομάτου μπορεί να γίνει με ένα γεγονός που θέτει σαν αρχικό κόμβο του μονοπατιού που συλλαβίζει μια συμβολοακολουθία της γλώσσας, τη σταθερά b και σαν κορυφή της στοίβας το κατηγόρημα του στόχου. Για την chain επερώτηση $Q = \{\leftarrow p(b, Y), P\}$, το πρόγραμμα P' που προκύπτει είναι:

$$
\begin{aligned}
&q(b, [p]).\\
&q(Y, [p_1, a_1, \ldots, p_n, a_n | T]) \leftarrow q(X, [p_0 | T]), a_0(X, Y).\\
&q(Y, T) \leftarrow q(X, [a_1 | T]), a_1(X, Y).\\
&\quad\vdots\\
&q(Y, T) \leftarrow q(X, [a_n | T]), a_n(X, Y).
\end{aligned}
$$

ενώ για κανόνα που έχει μόνο ένα EDB στο σώμα και τίποτα άλλο, έχουμε:

$$q(Y, T) \leftarrow q(X, [p_0 | T]), a_0(X, Y).$$

Η καινούργια επερώτηση είναι η $Q' = \{\leftarrow q(Y, [\,]), P'\}$ και είναι ισοδύναμη με την αρχική επερώτηση Q.

Παράδειγμα 2.6.2 *Δίνεται η chain επερώτηση* $Q = \{\leftarrow \texttt{sg(tom,Y)}, P\}$*, όπου το* P *είναι ένα μη-γραμμικό πρόγραμμα που ορίζει άτομα της ίδιας γενιάς:*

$$\texttt{sg(X,Y)} \leftarrow \texttt{flat(X,Y).}$$
$$\texttt{sg(X,Y)} \leftarrow \texttt{up(X,X}_1\texttt{),sg(X}_1\texttt{,Y}_1\texttt{),fl(Y}_1\texttt{,Y}_2\texttt{),sg(Y}_2\texttt{,Y}_3\texttt{),down(Y}_3\texttt{,Y).}$$

Το πρόγραμμα αυτό αντιστοιχεί σε μια c.f. γραμματική $G = \{V, T, \Pi, p\}$*, όπου* $V = \{\texttt{sg}\}, T = \{\texttt{flat}, \texttt{up}, \texttt{fl}, \texttt{down}\}$ *και κανόνες παραγωγής* Π*:*

$$\texttt{sg} \rightarrow \texttt{flat}$$
$$\texttt{sg} \rightarrow \texttt{up,sg,fl,sg,down}$$

Παρατηρούμε πως οι κανόνες παραγωγής προκύπτουν από το πρόγραμμα διώχνοντας τα ορίσματα και αντιστρέφοντας την φορά των βελών. Η γλώσσα $L(Q)$ *που προκύπτει από αυτή τη γραμματική μπορεί να αναγνωριστεί από το αυτόματο που φαίνεται παρακάτω.*

	up	fl	$down$	$flat$	ϵ
(q_0, Z_0)					$(q, sg\ Z_0)$
(q, sg)	$(q, sg\ fl\ sg\ down)$			(q, ϵ)	
(q, fl)		(q, ϵ)			
$(q, down)$			(q, ϵ)		

Ισοδύναμα, σε μορφή συνάρτησης μετάβασης, έχουμε: $\delta(q_0, \epsilon, Z_0) = (q, sg\ Z_0)$, $\delta(q, flat, sg) = (q, \epsilon), \delta(q, up, sg) = (q, sg\ fl\ sg\ down), \delta(q, fl, fl) = (q, \epsilon)$, $\delta(q, down, down) = (q, \epsilon)$.

Αυτό το αυτόματο μπορεί να μετατραπεί στο ακόλουθο πρόγραμμα P'*:*

```
q(tom,[sg]).
q(Y,T) ← q(X,[sg|T]),flat(X,Y).
q(Y,[sg,fl,sg,down|T]) ← q(X,[sg|T],up(X,Y).
q(Y,T) ← q(X,[fl|T],fl(X,Y).
q(Y,T) ← q(X,[down|T],down(X,Y).
```

Ο νέος στόχος είναι ο $\leftarrow \texttt{q(Y,[ ])}$*, ενώ η νέα επερώτηση είναι η* $Q' = \{\leftarrow \texttt{q(Y,[ ])}, P'\}$

Στην περίπτωση που κάθε αναδρομικός κανόνας είναι δεξιογραμμικός (δηλ. οι κανόνες είναι της μορφής $A \rightarrow \alpha B$ ή $A \rightarrow \alpha$, με $A, B \in V$ και $\alpha \in T^*$, τότε όπως ξέρουμε, έχουμε μια *δεξιογραμμική κανονική (rightlinear regular)* γραμματική και ένα πεπερασμένο αυτόματο (και όχι απαραίτητα στοίβα) μπορεί να αναγνωρίσει την γλώσσα που παράγεται από την γραμματική. Πιο συγκεκριμένα, η στοίβα μπορεί να προστεθεί σαν πληροφορία στην κατάσταση.

Παράδειγμα 2.6.3 *Έστω το ακόλουθο πρόγραμμα P, με στόχο $G :\leftarrow$* `p(x`$_0$`,Y)`*:*

$$\texttt{p(X,Y)} \leftarrow \texttt{b(X,Y).}$$
$$\texttt{p(X,Y)} \leftarrow \texttt{a(X,Z),p(Z,Y).}$$

Η αντίστοιχη γραμματική G είναι η ακόλουθη δεξιογραμμική:

$$\texttt{p} \rightarrow \texttt{b} \mid \texttt{a p}$$

Το αυτόματο στοίβας που αναγνωρίζει την γλώσσα $L(Q)$ είναι το:

	a	b	ε
(q_0, Z_0)			$(q, p\ Z_0)$
(q, p)	(q, p)	(q, ϵ)	

Ισοδύναμα, σε μορφή συνάρτησης: $\delta(q_0, \epsilon, Z_0) = (q, p\ Z_0), \delta(q, a, p) = (q, p)$, $\delta(q, b, p) = (q, \epsilon)$.

και το μετασχηματισμένο πρόγραμμα P' είναι:

$$\texttt{q(x}_0\texttt{,[p]).}$$
$$\texttt{q(Y,[p])} \leftarrow \texttt{q(X,[p]),a(X,Y).}$$
$$\texttt{q(Y,[ ])} \leftarrow \texttt{q(X,[p]),b(X,Y).}$$

Αφαιρώντας τη στοίβα και προσθέτοντας την πληροφορία στην κατάσταση, έχουμε:

$$\texttt{q}_p\texttt{([x}_0\texttt{]).}$$
$$\texttt{q}_p\texttt{(Y)} \leftarrow \texttt{q}_p\texttt{(X),a(X,Y).}$$
$$\texttt{q(Y)} \leftarrow \texttt{q}_p\texttt{(X),b(X,Y).}$$

Οι κανόνες παραγωγής της γραμματικής που αντιστοιχούν στο παραπάνω πρόγραμμα είναι:

$$\delta\texttt{(p,a)} \rightarrow \texttt{p}$$
$$\delta\texttt{(p,b)} \rightarrow \texttt{p}_F$$

όπου τα `p`,`p`$_F$ *είναι η αρχική και η τελική κατάσταση αντίστοιχα.*

Το νέο πρόγραμμα P' είναι:

$$\texttt{p(x}_0\texttt{).}$$
$$\texttt{p(Y)} \leftarrow \texttt{p(X),a(X,Y).}$$
$$\texttt{p}_F\texttt{(Y)} \leftarrow \texttt{p(X),b(X,Y).}$$

και ο νέος στόχος G' είναι ο $\leftarrow \texttt{p}_F\texttt{(Y)}$.

2.6.1 Αναγωγή της pushdown μεθόδου στην counting μέθοδο

Σε ορισμένες περιπτώσεις, η pushdown μέθοδος ανάγεται στην counting μέθοδο, που περιγράφηκε σε προηγούμενη ενότητα.

Αν η στοίβα περιέχει συμβολοακολουθίες της μορφής $\alpha^k\beta^n$, με $k = \{0, 1\}, n \geq 0$, τότε η στοίβα μπορεί να προσομοιωθεί με ένα *μετρητή* n και την εισαγωγή δύο νέων καταστάσεων q_α, q_β, που καταγράφουν ποιο στοιχείο βρίσκεται στην κορυφή της στοίβας. Αυτή η περίπτωση εμφανίζεται αν έχουμε: α) έναν αριθμό από μη αναδρομικούς κανόνες, β) έναν αριθμό από δεξιογραμμικούς κανόνες και γ) έναν ακριβώς γραμμικό αναδρομικό κανόνα που όμως δεν είναι αριστερός.

Παράδειγμα 2.6.4 *Έστω το ακόλουθο πρόγραμμα που ορίζει άτομα της ίδιας γενιάς, με στόχο G τον* ← `sg(b,Y)`*:*

```
sg(X,Y) ← flat(X,Y).
sg(X,Y) ← up(X,X1),sg(X1,Y1),down(Y1,Y).
```

Το πρόγραμμα P' που προκύπτει από την pushdown μέθοδο είναι:

```
q(b,[sg]).
q(Y,[sg,down|T]) ← q(X,[sg|T]),up(X,Y).
q(Y,T) ← q(X,[sg|T]),flat(X,Y).
q(Y,T) ← q(X,[down|T]),down(X,Y).
```

Η στοίβα περιλαμβάνει συμβολοακολουθίες της μορφής $\texttt{sg down}^n$ *ή της μορφής* $\texttt{down}^n$*, με $n \geq 0$. Μπορούμε να αντικαταστήσουμε την στοίβα με έναν μετρητή n και δύο νέες καταστάσεις* $\texttt{q}_{sg}$, $\texttt{q}_{down}$*, που υποδηλώνουν αν το* `sg` *ή το* `down` *είναι στην κορυφή της στοίβας, αντίστοιχα. Επομένως, το πρόγραμμα μπορεί να μετασχηματιστεί ως εξής:*

$$\begin{array}{l}
\texttt{q}_{down}\texttt{(b,0).}\\
\texttt{q}_{sg}\texttt{(Y,I+1)} \leftarrow \texttt{q}_{sg}\texttt{(X,I),up(X,Y).}\\
\texttt{q}_{down}\texttt{(Y,I)} \leftarrow \texttt{q}_{sg}\texttt{(X,I),flat(X,Y).}\\
\texttt{q}_{down}\texttt{(Y,I-1)} \leftarrow \texttt{q}_{down}\texttt{(X,I),down(X,Y).}
\end{array}$$

και ο νέος στόχος είναι ← $\texttt{q}_{down}\texttt{(Y,0)}$.

Μια ακόμα περίπτωση στην οποία η pushdown μέθοδος μπορεί να μετατραπεί στην counting μέθοδο είναι εκείνη στην οποία η στοίβα έχει συμβολοακολουθίες της μορφής $\alpha^k(\beta\alpha)^n$, με $k = \{0, 1\}, n \geq 0$. Αυτή η περίπτωση παρουσιάζεται όταν έχουμε: α) έναν αριθμό από μη αναδρομικούς κανόνες, β) έναν αριθμό από δεξιογραμμικούς κανόνες και γ) έναν ακριβώς διγραμμικό αναδρομικό κανόνα (δηλ. έχει δύο αναδρομικά κατηγορήματα στο σώμα του) που όμως κανένα δεν είναι αριστερογραμμικό.

Παράδειγμα 2.6.5 *Έστω το ακόλουθο πρόγραμμα που ορίζει κόκκινα-κίτρινα μονοπάτια με διπλή αναδρομή, με στόχο G τον* ← `path(b,Y)`*:*

```
path(X,X).
path(X,Y) ← red(X,V),path(V,W),yellow(W,Z),path(Z,Y).
```

Το πρόγραμμα P' που προκύπτει από την pushdown μέθοδο είναι:

```
q(b,[path]).
q(X,T) ← q(X,[path|T]).
q(Y,[path,yellow,path|T]) ← q(X,[path|T]),red(X,Y).
q(Y,T) ← q(X,[yellow|T]),yellow(X,Y).
```

Μπορούμε να αντικαταστήσουμε την στοίβα με έναν μετρητή n και δύο νέες καταστάσεις $\mathtt{q}_{path}$, $\mathtt{q}_{yellow}$, *που υποδηλώνουν αν το* `path` *ή το* `yellow` *είναι στην κορυφή της στοίβας, αντίστοιχα. Επομένως, το πρόγραμμα μπορεί να μετασχηματιστεί ως εξής:*

$$
\begin{array}{l}
\mathtt{q}_{path}\mathtt{(b,0).} \\
\mathtt{q}_{yellow}\mathtt{(X,I)} \leftarrow \mathtt{q}_{path}\mathtt{(X,I).} \\
\mathtt{q}_{path}\mathtt{(Y,I+1)} \leftarrow \mathtt{q}_{path}\mathtt{(X,I),red(X,Y).} \\
\mathtt{q}_{path}\mathtt{(Y,I-1)} \leftarrow \mathtt{q}_{yellow}\mathtt{(X,I),yellow(X,Y).}
\end{array}
$$

και ο νέος στόχος είναι $\leftarrow \mathtt{q}_{yellow}\mathtt{(Y,0)}$.

Παρατηρούμε πως σε αυτό το πρόγραμμα δεν μπορεί να εφαρμοστεί η counting μέθοδος, καθώς δεν έχει την binding passing property (BPP) ιδιότητα, που διαισθητικά λέει πως οι δεσμεύσεις δεν φτάνουν μέχρι την τελευταία εμφάνιση του `path`.

Κεφάλαιο 3

Μετασχηματισμός σε πολλαπλές καταναλώσεις

3.1 Εισαγωγή

Ο *μετασχηματισμός διακλαδιζόμενου χρόνου (branching-time transformation)* [36, 37] είναι ένας μετασχηματισμός που παρουσιάστηκε πρόσφατα και εφαρμόζεται σε Chain Datalog προγράμματα. Πιο συγκεκριμένα, η προσέγγιση διακλαδιζόμενου χρόνου ανήκει στην κλάση των βελτιστοποιήσεων Datalog προγραμμάτων μέσω *διάδοσης τιμής (value-propagating)* , στην οποία οι τιμές εισόδου του στόχου στο αρχικό πρόγραμμα διαδίδονται με σκοπό να περιοριστεί η παραγωγή των ατόμων στον bottom-up υπολογισμό του νέου προγράμματος. Τέτοιες τεχνικές είναι τα *μαγικά σύνολα (magic sets)* [7, 9, 45], η *μέθοδος μέτρησης (counting method)* [43], η *μέθοδος στοίβας (pushdown method)* [19], όπως παρουσιάστηκαν στο Κεφάλαιο 2 και άλλες. Ο μετασχηματισμός διακλαδιζόμενου χρόνου έχει τις ρίζες του σε μια παρόμοια τεχνική που έχει προταθεί στον συναρτησιακό προγραμματισμό [54, 52, 39, 40].

Σε αυτή την εργασία επεκτείνουμε την προσέγγιση με τον διακλαδιζόμενο χρόνο έτσι ώστε να μπορεί να εφαρμοστεί σε μια ευρύτερη κλάση well-moded Datalog προγραμμάτων (προγράμματα που τα ορίσματα των ατόμων στις προτάσεις έχουν καθορισμένο ρόλο, είτε είναι είσοδος είτε έξοδος), πέρα από τα Chain Datalog. Ένα από τα καινούργια χαρακτηριστικά των νέων προσεγγίσεων είναι πως η τελική γλώσσα είναι η Datalog_{nS} [13, 12] επεκταμένη με κατηγορήματα επιλογής [27], μια μη-ντετερμινιστική κατασκευή που αρχικά είχε οριστεί στην περιοχή των νοηματικών γλωσσών λογικού προγραμματισμού [26]. Η χρήση των κατηγορημάτων επιλογής επιτρέπει τον μετασχηματισμό προτάσεων που περιλαμβάνουν πολλαπλές καταναλώσεις μεταβλητών. Θεωρούμε πως η χρήση μη-ντετερμινιστικών κατασκευών ανοίγει νέους ορίζοντες στην περιοχή των βελτιστοποιήσεων Datalog προγραμμάτων. Για τα προγράμματα που προκύπτουν από το μετασχηματισμό ορίζουμε μια bottom-up αποδεικτική διαδικασία (proof procedure) που πάντοτε τερματίζει, παρά το γεγονός πως η βάση Herbrand του προγράμματος είναι άπειρη. Τέλος, ορίζουμε μια σειρά από βελτιστοποιήσεις στον τελικό κώδικα, που βελτιώνουν περαιτέρω την απόδοση του bottom-up υπολογισμού. Τα βασικά σημεία του Κεφαλαίου μπορούν να συνοψισθούν στα ακόλουθα:

- Προτείνουμε μία νέα τεχνική διάδοσης τιμών για μια νέα μεγάλη κλάση γλωσσών από moded Datalog προγράμματα και αποδεικνύουμε την ορθότητά της. Στην πραγματικότητα, οι κλάσεις των προγραμμάτων που μελετάμε είναι εκτενέστερες από τις κλάσεις που μελετώνται σε άλλους σχετικούς μετασχηματισμούς.

- Δείχνουμε ότι χρονικές γλώσσες όπως η Datalog$_{nS}$ μπορούν να επεκταθούν με μη-ντετερμινιστικές κατασκευές και να αποδειχθούν εξαιρετικά χρήσιμες για να ορισθούν νέοι ισχυροί μετασχηματισμοί Datalog προγραμμάτων (και πιθανώς και πιο γενικών λογικών προγραμμάτων). Ειδικότερα, παρουσιάζουμε πως μπορούμε να διαχειριστούμε αποτελεσματικά τις πολλαπλές καταναλώσεις μεταβλητών σε Datalog προγράμματα με την χρήση κατηγορημάτων επιλογής.

- Ορίζουμε μια αποδεικτική διαδικασία που μπορεί να εφαρμοστεί στα τελικά προγράμματα που προκύπτουν από τον μετασχηματισμό και αποδεικνύουμε πως η διαδικασία αυτή πάντοτε τερματίζει. Επιπλέον, προτείνουμε διάφορους κανόνες μετασχηματισμού του τελικού προγράμματος που αυξάνουν την αποδοτικότητά του.

3.2 Περιγραφή του μετασχηματισμού

Στην περιοχή των επαγωγικών βάσεων δεδομένων, προτιμάται η γλώσσα Datalog και ο bottom-up υπολογισμός, αντί του Λ.Π. και του κλασικού top-down υπολογισμού. Η αποδοτική εκτέλεση ενός προγράμματος με bottom-up τρόπο, βασίζεται σε τεχνικές βελτιστοποίησης που συχνά αναφέρονται ως *τεχνικές βελτιστοποίησης επερωτήσεων (query optimization techniques)* [34, 2] ή *συστήματα επανεγγραφής όρων (term rewriting systems)* [8], όπως έχει αναφερθεί και στην Ενότητα 2.1.3. Βελτιστοποίηση επερώτησης είναι ο μετασχηματισμός μίας επερώτησης (πρόγραμμα και στόχος) σε μία νέα επερώτηση, η οποία είναι σημασιολογικά ισοδύναμη με την αρχική, αλλά είναι σε τέτοια μορφή ώστε να γίνεται η αποτίμηση της με την bottom-up μέθοδο αποδοτικά. Μία γνωστή οικογένεια τέτοιων βελτιστοποιήσεων επερωτήσεων είναι οι *τεχνικές διάδοσης των τιμών (value-propagating techniques)*, οι οποίες χειρίζονται επερωτήσεις στις οποίες οι στόχοι έχουν κάποια ορίσματα δεσμευμένα με σταθερές τιμές. Τις δεσμευμένες αυτές τιμές προσπαθούμε να χρησιμοποιήσουμε με τέτοιο τρόπο, ώστε η αναζήτηση στη βάση δεδομένων να περιοριστεί σε μικρό μέρος της. Επίσης μπορούμε να πούμε πως οι τεχνικές αυτής της κατηγορίας ουσιαστικά προσομοιώνουν την εκτέλεση της top-down αποτίμησης με τέτοιο τρόπο, ώστε να διατηρούνται τα πλεονεκτήματα αυτής της μεθόδου (γρήγορη εστίαση στο αποτέλεσμα), αλλά ταυτόχρονα να εξαλείφουν τα αρνητικά της (μη τερματισμός, επανάληψη υπολογισμών).

3.3 Ο μετασχηματισμός διακλαδιζόμενου χρόνου

Μία τεχνική που ανήκει σε αυτήν την οικογένεια βελτιστοποιήσεων είναι η τεχνική διακλαδιζόμενου χρόνου και εφαρμόζεται σε προγράμματα αλυσίδα Datalog (Chain

Datalog) [36, 37]. Στην τεχνική αυτή, μετασχηματίζουμε κάθε πρόταση του αρχικού προγράμματος σε ένα σύνολο από νέες προτάσεις που παίζουν το ρόλο της αρχικής πρότασης. Διαισθητικά, οι προτάσεις που προκύπτουν, αντιστοιχούν με ένα πιο άμεσο τρόπο στη ροή των τιμών των μεταβλητών που πραγματοποιείται κατά την εκτέλεση του αρχικού προγράμματος σε μια top-down αποτίμηση. Το κύριο χαρακτηριστικό των νέων προτάσεων είναι πως κάθε μία έχει το πολύ ένα IDB στο σώμα της.

Στην τεχνική διακλαδιζόμενου χρόνου, για κάθε κατηγόρημα του αρχικού προγράμματος δύο νέα κατηγορήματα εισάγονται (αν και όπως θα δούμε παρακάτω, αυτό είναι απαραίτητο μόνο για τα IDB). Κάθε ένα από αυτά τα κατηγορήματα έχει δύο ακριβώς ορίσματα, ένα όρισμα ελέγχου του υπολογισμού (σε μορφή λίστας ακεραίων αριθμών) και ένα όρισμα δεδομένων που κωδικοποιεί το όρισμα που περνάει στο αρχικό κατηγόρημα. Το νόημα του ορίσματος ελέγχου είναι ότι συνδέει τα δύο νέα κατηγορήματα που προέκυψαν από ένα κατηγόρημα του αρχικού προγράμματος και τα συντονίζει κατά τέτοιο τρόπο ώστε να παράγονται οι σωστές απαντήσεις.

Για να παρουσιάσουμε την τεχνική διακλαδιζόμενου χρόνου στην αρχική της μορφή, ας θεωρήσουμε το ακόλουθο Chain Datalog πρόγραμμα, όπου το `p` είναι ένα IDB κατηγόρημα, ενώ το `e` είναι ένα EDB. Αν θεωρήσουμε πως το κατηγόρημα `e` παριστάνει τις ακμές ενός γράφου, τότε το κατηγόρημα `p` παριστάνει το μεταβατικό κλείσιμο (transitive closure) του γράφου:

```
← p(a,Y).
p(X,Z) ← e(X,Z).
p(X,Z) ← p(X,Y), p(Y,Z).
```

Στην κλάση των προγραμμάτων Chain Datalog, το πρώτο όρισμα κάθε κατηγορήματος θεωρείται σαν ένα όρισμα εισόδου, ενώ το δεύτερο σαν όρισμα εξόδου. Αυτό οφείλεται στην σύμβαση που έχουμε κάνει να μελετάμε μόνο προγράμματα που το πρώτο όρισμα του στόχου είναι δεσμευμένο και οι προτάσεις να έχουν σώμα που είναι σε μορφή αλυσίδας, οπότε σε μια top-down αποτίμηση η πρώτη θέση είναι η είσοδος. Τα νέα κατηγορήματα `p1` και `p2` που εισάγονται από τον μετασχηματισμό ενός κατηγορήματος του αρχικού προγράμματος `p`, αντιστοιχούν στις κλήσεις ή εισόδους (calls) και στις απαντήσεις ή εξόδους (answers) αντίστοιχα, για το κατηγόρημα `p`, σε έναν top-down υπολογισμό.

Θα παρουσιάσουμε τον μετασχηματισμό κοιτώντας κάθε πρόταση με την σειρά. Αρχικά, ο μετασχηματισμός του στόχου δίνει σαν αποτέλεσμα:

```
← p2([],Y).
p1([],a).
```

Παρατηρούμε πως το δεσμευμένο όρισμα του αρχικού στόχου έχει γίνει ένα όρισμα μιας μοναδιαίας πρότασης στο μετασχηματισμένο πρόγραμμα. Σαν αποτέλεσμα, ο bottom-up υπολογισμός του προκύπτοντος προγράμματος θα χρησιμοποιήσει αυτή την πρόταση στο πρώτο βήμα ώστε να στρέψει τον υπολογισμό σε σχετικές περιοχές του χώρου αναζήτησης και να περιορίσει το σύνολο των ατόμων που παράγονται στα επόμενα βήματα του υπολογισμού.

Ο μετασχηματισμός της πρώτης πρότασης του αρχικού προγράμματος δίνει ως αποτέλεσμα τις προτάσεις:

```
p2(L,Z) ← e2([1|L],Z).
e1([1|L],X) ← p1(L,X).
```

Είναι σημαντικό να παρατηρήσουμε την ετικέτα `1` που εμφανίζεται τόσο στο `e2([1|L],Z)` όσο και στο `e1([1|L],X)`. Η βασική ιδέα είναι πως η ετικέτα αυτή συσχετίζει τα δύο άτομα μεταξύ τους, μια και έχουν προκύψει από την ίδια κλήση στο αρχικό πρόγραμμα (την κλήση `e(X,Z)`). Είναι σημαντικό να παρατηρήσουμε ότι *οποιαδήποτε* ετικέτα μπορεί να χρησιμοποιηθεί, αρκεί να είναι διαφορετική από τις άλλες που έχουν απονεμηθεί στις άλλες κλήσεις του αρχικού προγράμματος.

Η δεύτερη πρόταση του δοθέντος προγράμματος μετασχηματίζεται ως εξής:

```
p2(L,Z) ← p2([3|L],Z).
p1([3|L],Y) ← p2([2|L],Y).
p1([2|L],X) ← p1(L,X).
```

Τέλος, ο μετασχηματισμός διακλαδιζόμενου χρόνου εισάγει και την ακόλουθη πρόταση:

```
e2(L,Y) ← e(X,Y),e1(L,X).
```

και παίζει τον ρόλο μιας διασύνδεσης με τα άτομα της βάσης δεδομένων, που έχουν κατηγορηματικό σύμβολο `e`.

Παρατηρούμε πως το πρόγραμμα που προέκυψε από τον μετασχηματισμό δεν είναι ένα Datalog πρόγραμμα. Στην πραγματικότητα είναι ένα Datalog_{nS} πρόγραμμα [13, 12]. Επίσης, πρέπει να σημειώσουμε πως στις αρχικές εργασίες που ορίζουν το μετασχηματισμό διακλαδιζόμενου χρόνου [36, 37], ακολουθίες από χρονικούς τελεστές είχαν χρησιμοποιηθεί αντί λίστες από φυσικούς αριθμούς οπότε και τα παραγόμενα προγράμματα ήταν *Branching Datalog*, όπου Branching Datalog είναι ένα υποσύνολο, χωρίς συναρτησιακά σύμβολα, της γλώσσας λογικού προγραμματισμού διακλαδιζόμενου χρόνου *Cactus* [38]. Ωστόσο είναι εύκολο να δειχθεί πως οι προσεγγίσεις αυτές είναι ισοδύναμες. Παρατηρούμε ακόμα, ότι η βάση Herbrand των προγραμμάτων που προκύπτουν από τον μετασχηματισμό δεν είναι πεπερασμένες, εξαιτίας της ύπαρξης λίστας στα άτομα που εισάγονται. Ωστόσο, όπως έχει δειχθεί στο [37] (βασισμένο στα αποτελέσματα του [12]) υπάρχει ένας bottom-up υπολογισμός που παράγει όλες τις απαντήσεις για τον στόχο. Για περισσότερες λεπτομέρειες για τον μετασχηματισμό διακλαδιζόμενου χρόνου μπορεί κάποιος να ανατρέξει στα [37, 36].

Ωστόσο, ο μετασχηματισμός διακλαδιζόμενου χρόνου όπως περιγράφτηκε παραπάνω δεν εφαρμόζεται σε Datalog προγράμματα στα οποία υπάρχουν πολλαπλές καταναλώσεις μεταβλητών (θα εξηγηθεί παρακάτω ο όρος κατανάλωση μεταβλητής) ούτε σε προγράμματα στα οποία οι καταναλώσεις των μεταβλητών δεν είναι συνεχόμενες. Το παρακάτω παράδειγμα δείχνει τα ζητήματα που προκύπτουν στην πρώτη περίπτωση. Έστω το Datalog πρόγραμμα που περιλαμβάνει την πρόταση:

```
p(X,Z) ← q(X,W),r(X,W,Z).
```

Το πρόβλημα με αυτή την πρόταση είναι πως η μεταβλητή `X` εμφανίζεται δύο φορές στο σώμα της πρότασης. Όταν θα προσπαθήσουμε να εφαρμόσουμε το μετασχηματισμό στο πρόγραμμα που περιλαμβάνει την παραπάνω πρόταση, η συσχέτιση ανάμεσα στις δύο εμφανίσεις της `X` θα χαθεί, με αποτέλεσμα το παραγόμενο πρόγραμμα να μην είναι σημασιολογικά ισοδύναμο με το αρχικό. Πιο συγκεκριμένα, μια απλοϊκή μετάφραση της παραπάνω πρότασης θα παρήγαγε, μεταξύ άλλων, τις ακόλουθες προτάσεις:

```
q1([l1|L],X) ← p1(L,X).
r1([l2|L],X) ← p1(L,X).
```

όπου οι `l1` και `l2` είναι φυσικοί αριθμοί. Παρατηρούμε πως οι δύο εμφανίσεις της `X` στο σώμα της αρχικής πρότασης έχουν αποκοπεί μεταξύ τους καθώς έχουν τοποθετηθεί σε διαφορετικές προτάσεις στο τελικό πρόγραμμα και για αυτό είναι δυνατόν να πάρουν διαφορετικές τιμές (κάτι που δεν ήταν δυνατόν να γίνει στο αρχικό πρόγραμμα). Με αυτό το τρόπο, το προκύπτον πρόγραμμα μπορεί να παράγει απαντήσεις που δεν συμπεριλαμβάνονται στο σύνολο απαντήσεων του αρχικού προγράμματος.

Στην εργασία αυτή, παρουσιάζω μία λύση για το παραπάνω πρόβλημα η οποία βασίζεται στα *κατηγορήματα επιλογής (choice predicates)* [27], μία μη-ντετερμινιστική κατασκευή που έχει προταθεί στην περιοχή του *νοηματικού λογικού προγραμματισμού (intensional logic programming)* [26]. Παρόμοιες μη-ντετερμινιστικές κατασκευές έχουν προταθεί επίσης και σε άλλες μορφές στις [17, 18]. Τα κατηγορήματα επιλογής έχουν δηλωτικό χαρακτήρα και η σημασιολογία τους είναι κομψή και καλά ορισμένη [27]. Η βασική ιδέα που κρύβεται πίσω από τα κατηγορήματα επιλογής είναι ότι μέσα σε ένα περιβάλλον (context), που στην περίπτωση μας παριστάνεται με μία λίστα φυσικών αριθμών `L`, ένα κατηγόρημα μπορεί να είναι αληθές για μια μόνο τιμή. Γι' αυτό, οι παραπάνω δύο προτάσεις μπορούν να γραφτούν ως εξής:

```
q1([l1|L],X) ← #p1(L,X).
r1([l2|L],X) ← #p1(L,X).
```

όπου το `#p1` είναι η έκδοση με επιλογή του κατηγορήματος `p1`, το οποίο σε κάθε δοσμένο περιβάλλον `L` μπορεί να είναι αληθές για μία μόνο τιμή του `X`. Αυτό αποκαθιστά τη σύνδεση ανάμεσα στις δύο εμφανίσεις του `X`, δίνοντας ένα τελικό πρόγραμμα που είναι ισοδύναμο με το αρχικό.

3.4 Η αρχική γλώσσα του μετασχηματισμού

Στο εξής, θεωρούμε πως ο αναγνώστης είναι εξοικειωμένος με τις βασικές έννοιες του λογικού προγραμματισμού, τις οποίες είδαμε και στο Κεφάλαιο 1 και μπορεί να συμβουλευτεί και το Βιβλίο [23], όπως και τις Σημειώσεις του [58].

Ένα σύνολο D από αποτιμημένα γεγονότα (ή μοναδιαίες προτάσεις) χωρίς συναρτησιακά σύμβολα αναφέρεται ως *επεκτατική βάση δεδομένων (extensional database)* ή πιο απλά ως *βάση δεδομένων*. Τα κατηγορήματα των ατόμων που εμφανίζονται στη βάση δεδομένων ονομάζονται *EDB κατηγορήματα*. Ένα *Datalog πρόγραμμα P* αποτελείται από ένα πεπερασμένο σύνολο από προτάσεις δίχως συναρτησιακά σύμβολα. Κατηγορήματα που εμφανίζονται στις κεφαλές των προτάσεων του P ονομάζονται *νοηματικά (intensional)* ή *IDB κατηγορήματα* (IDBs). Θεωρούμε πως τα EDB κατηγορήματα δεν εμφανίζονται στις κεφαλές των προτάσεων και επιπλέον θεωρούμε πως τα κατηγορήματα που εμφανίζονται μόνο στα σώματα των προτάσεων των Datalog προγραμμάτων είναι EDB κατηγορήματα. Ένα Datalog πρόγραμμα P μαζί με μία βάση δεδομένων D θα συμβολίζεται με P_D.

Η κλάση των προγραμμάτων που ο προτεινόμενος μετασχηματισμός μπορεί να χρησιμοποιηθεί είναι μία υποκλάση των Datalog προγραμμάτων και ορίζεται ως εξής:

Ορισμός 3.4.1 *Μια πρόταση*
$$p_0(\vec{v}_0, Z_n) \leftarrow p_1(\vec{v}_1, Z_1), p_2(\vec{v}_2, Z_2), \ldots, p_n(\vec{v}_n, Z_n).$$
με $n > 0$, λέγεται πρόταση σταθερού τρόπου (fixed mode clause) *ή (*fm-πρόταση (fm-clause) *για συντομία αν:*

1. *Κάθε $\vec{v}_i$, για $i = 0, \ldots, n$ είναι ένα μη κενό διάνυσμα από διακριτές μεταβλητές και οι $Z_1, \ldots, Z_n$ είναι διακριτές μεταβλητές.*

2. *$vars(\vec{v}_i) \subseteq vars(\vec{v}_0) \cup \{Z_1, \ldots, Z_{i-1}\}$, για $1 \le i \le n$.*

3. *Για κάθε $V \in vars(\vec{v}_0)$ υπάρχει ένα διάνυσμα $\vec{v}_i, 1 \le i \le n$ τέτοιο ώστε $V \in vars(\vec{v}_i)$.*

4. *Για κάθε Z_i με $1 \le i \le n-1$, υπάρχει ένα διάνυσμα $\vec{v}_j$ με $i < j \le n$ τέτοιο ώστε $Z_i \in vars(\vec{v}_j)$.*

Ένα πρόγραμμα P λέγεται fixed mode Datalog πρόγραμμα *ή* fm-Datalog πρόγραμμα *αν όλες οι προτάσεις του είναι fm-προτάσεις. Ένας* στόχος *G είναι της μορφής $\leftarrow q(\vec{e}, Z)$, όπου $\vec{e}$ είναι ένα μη κενό διάνυσμα από σταθερές, Z είναι μια μεταβλητή και q ένα IDB κατηγόρημα.*

Σε αυτό το σημείο, πρέπει να αναφέρουμε πως οι fm-προτάσεις έχουν καταστάσεις, δηλαδή οι όροι $\vec{v}_i$ του παραπάνω ορισμού αντιστοιχούν στα ορίσματα εισόδου, ενώ κάθε Z_i αντιστοιχεί στο μοναδικό όρισμα εξόδου κάθε ατόμου. Πάντως, γενικά στον λογικό προγραμματισμό δεν γίνεται αυτή η διάκριση σε ορίσματα εισόδου και ορίσματα εξόδου, αλλά κάθε όρισμα μπορεί σε κάποιες περιπτώσεις να έχει τιμή ενώ σε άλλες να ζητείται η τιμή του. Ένα κλασικό παράδειγμα αποτελεί το κατηγόρημα append το οποίο μπορεί να χρησιμοποιηθεί με διάφορους τρόπους (π.χ. για συνένωση δύο δοσμένων λιστών, αλλά και για τη διάσπαση μιας δοσμένης λίστας σε δύο άλλες).

Ορισμός 3.4.2 *Μία εμφάνιση μεταβλητής στα ορίσματα εισόδου της κεφαλής ή στα ορίσματα εξόδου ενός ατόμου στο σώμα μιας πρότασης λέγεται* παραγωγική (productive)*, ενώ στις υπόλοιπες περιπτώσεις θα λέγεται* καταναλωτική (consumptive).

Οι χαρακτηρισμοί αυτοί προκύπτουν άμεσα από τον τρόπο που εκτελείται ένα πρόγραμμα με top-down αποτίμηση, από τη μορφή των προτάσεων καθώς επίσης και από τη μορφή του στόχου (έχει δεσμευμένα ορίσματα).

Παράδειγμα 3.4.1 *Η ακόλουθη πρόταση είναι μια fixed mode πρόταση:*

$$\mathtt{p}(\overset{+}{\mathtt{X}},\overset{+}{\mathtt{Y}},\overset{-}{\mathtt{Z}}) \leftarrow \mathtt{q}(\overset{+}{\mathtt{X}},\overset{+}{\mathtt{Y}},\overset{-}{\mathtt{W}}),\mathtt{r}(\overset{+}{\mathtt{X}},\overset{-}{\mathtt{R}}),\mathtt{s}(\overset{+}{\mathtt{W}},\overset{+}{\mathtt{R}},\overset{+}{\mathtt{Y}},\overset{-}{\mathtt{Z}}).$$

όπου τα $+$ και $-$ πάνω από τις μεταβλητές δηλώνουν τα ορίσματα εισόδου και εξόδου αντίστοιχα και μεταφέρονται σαν εκθέτες στα νέα κατηγορήματα (για αυτό και μπορούν να παραλειφθούν). Παρατηρήστε ότι η τιμή της μεταβλητής X που παράγεται στη κεφαλή της πρότασης, καταναλώνεται πολλαπλά με συνεχόμενο τρόπο στα άτομα `q,r`*, ενώ η τιμή της μεταβλητής Y καταναλώνεται πολλαπλά, αλλά σε μη συνεχόμενα άτομα (στα* `q,s`*).*

Η διαίσθηση για την κλάση των fm-Datalog προγραμμάτων είναι πως κάθε τιμή που παράγεται από ένα άτομο, καταναλώνεται από ένα ή περισσότερα (όχι κατ' ανάγκη συνεχόμενα) άτομα που ακολουθούν το άτομο που παρήγαγε αυτή την τιμή. Η αρχή των παραγωγών γίνεται από την κεφαλή της πρότασης και ολοκληρώνεται με την κατανάλωση της τιμής που παράχθηκε στο τελευταίο άτομο στο σώμα της πρότασης, από το ελεύθερο όρισμα της κεφαλής. Πολλά φυσικά Datalog προγράμματα ανήκουν σε αυτή τη κλάση, για παράδειγμα, η κλάση των Datalog προγραμμάτων αλυσίδα, είναι ένα γνήσιο υποσύνολο αυτής της κλάσης. Στις επόμενες ενότητες θα δούμε μερικές υποκλάσεις αυτής της ισχυρής κλάσης: τα *συνεχόμενης κατανάλωσης (consecutive consumption)* Datalog προγράμματα και τα *παραγωγής-κατανάλωσης (productive-consumptive)* Datalog προγράμματα.

Στη συνέχεια θα δώσουμε μερικά παραδείγματα προγραμμάτων που ανήκουν στη κλάση fm-Datalog.

Παράδειγμα 3.4.2 *Έστω το ακόλουθο γενικευμένο κατηγόρημα* `path` *που ψάχνει για μονοπάτια ενός δοσμένου χρώματος σε έναν γράφο του οποίου οι ακμές είναι χρωματισμένες:*

```
path(X,Color,Z) ← edge(X,Color,Z).
path(X,Color,Z) ← edge(X,Color,W),path(W,Color,Z).
```

Το παραπάνω είναι προφανώς ένα πρόγραμμα fm-Datalog. Το EDB κατηγόρημα `edge(X,Color,Z)` *δηλώνει πως η ακμή (*`X,Z`*) έχει χρώμα* `Color`*. Ο στόχος* ← `path(a,red,Z)` *ρωτάει για κορυφές του γράφου που είναι προσβάσιμες από την κορυφή* `a` *μέσω* `red` *ακμών.*

Παράδειγμα 3.4.3 *Ο συνήθης ορισμός στο Λ.Π. του υπολοίπου και του μέγιστου κοινού διαιρέτη περιλαμβάνουν προτάσεις fm-Datalog:*

```
mod(X,Y,Z) ← mod_base(X,Y,Z).
mod(X,Y,Z) ← minus(X,Y,W),mod(W,Y,Z).
gcd(X,Y,Z) ← gcd_base(X,Y,Z).
gcd(X,Y,G) ← mod(X,Y,Z),gcd(Y,Z,G).
```

όπου τα `mod_base`, `gcd_base` *και* `minus` *είναι EDB κατηγορήματα που ορίζονται ως εξής:* `mod_base(X,Y,Z)` *αν* `X < Y` *και* `X = Z`, `gcd_base(X,Y,Z)` *αν* `X = Z` *και* `Y = 0` *και* `minus(X,Y,Z)` *αν* `Z = X - Y`.

Στη συνέχεια δίνουμε μία απλούστερη μορφή των fm-Datalog προγραμμάτων η οποία θα μας διευκολύνει στην παρουσίαση του μετασχηματισμού και στις αποδείξεις, μια και διατηρείται η ισοδυναμία με το αρχικό Datalog πρόγραμμα.

Ορισμός 3.4.3 *Ένα* απλό fm-Datalog πρόγραμμα *είναι ένα fm-Datalog πρόγραμμα στο οποίο κάθε πρόταση έχει το πολύ δύο άτομα στο σώμα του.*

Η σημασιολογία των fm-Datalog προγραμμάτων μπορεί να ορισθεί με ανάλογο τρόπο με αυτή του κλασικού λογικού προγραμματισμού. Οι έννοιες του *ελάχιστου μοντέλου Herbrand (minimum Herbrand model)* M_{P_D} του P_D, όπου P_D είναι ένα fm-Datalog πρόγραμμα με μια βάση δεδομένων D και *τελεστή αμέσου επακολούθου (immediate consequence operator)* T_{P_D} μεταφέρονται άμεσα, όπως είδαμε και στο Κεφάλαιο 1.

Η ακόλουθη πρόταση, η οποία μπορεί να αποδειχθεί χρησιμοποιώντας τεχνικές διπλώματος/ξεδιπλώματος (fold/unfold) [47, 15, 29], αποκαθιστούν την ισοδυναμία ανάμεσα σε fm-Datalog προγράμματα και σε απλά fm-Datalog προγράμματα. Σημειώνουμε με $M(p, P_D)$ το σύνολο των ατόμων του ελάχιστου μοντέλου Herbrand του P_D που έχουν κατηγόρημα το p.

Πρόταση 3.4.1 *Κάθε fm-Datalog πρόγραμμα P μπορεί να μετασχηματιστεί σε ένα απλό fm-Datalog πρόγραμμα, έτσι ώστε για κάθε κατηγόρημα p που εμφανίζεται στο P και για κάθε βάση δεδομένων D να ισχύει $M(p, P_D) = M(p, P'_D)$.*

Απόδειξη 3.4.1 *Έστω ότι n είναι ο μεγαλύτερος αριθμός ατόμων που υπάρχουν στο σώμα κάποιας πρότασης του προγράμματος P. Θα δείξουμε με επαγωγή στο n πως το P μπορεί να μετασχηματιστεί σε ένα απλό fm-Datalog πρόγραμμα P', τέτοιο ώστε $M(p, P_D) = M(p, P'_D)$. Για $n \leq 2$ είναι προφανές πως ισχύει, γιατί τότε κάθε πρόταση θα έχει το πολύ δύο άτομα στο σώμα της. Έστω ότι αυτή η ιδιότητα ισχύει για κάθε $n \geq 2$. Θα δείξουμε ότι αυτό ισχύει και για $n + 1$. Για μια fm-πρόταση C του P της μορφής:*

$$A_0 \leftarrow A_1, A_2, \ldots, A_{n+1}.$$

εισάγουμε μια νέα πρόταση C_{N+1} της μορφής:

$$new(\vec{v}, Z) \leftarrow A_2, \ldots, A_{n+1}.$$

όπου new είναι ένα νέο κατηγορηματικό σύμβολο, Z είναι η μεταβλητή εξόδου του A_{n+1} και $\vec{v}$ είναι το διάνυσμα των διακριτών μεταβλητών, τέτοιες ώστε $vars(\vec{v}) = (vars(A_2, \ldots, A_{n+1}) \cap vars(A_0, A_1)) - \{Z\}$.

Τότε, κάνουμε fold στην πρόταση C χρησιμοποιώντας την πρόταση C_{N+1} και παίρνουμε την πρόταση C':

$$A_0 \leftarrow A_1, new(\vec{v}, Z).$$

όπου η C αντικαθίσταται με $\{C_{N+1}, C'\}$. Η πρόταση C' έχει δύο άτομα στο σώμα της και η πρόταση C_{N+1} έχει n άτομα στο σώμα, ένα λιγότερο από ότι είχε πρόταση C. Αυτή η διαδικασία εφαρμόζεται σε κάθε πρόταση του προγράμματος P που το σώμα της έχει $n+1$ άτομα. Έστω P^i το προκύπτον πρόγραμμα. Μπορούμε να δούμε πως $M(p, P_D) = M(p, P_D^i)$ για κάθε κατηγόρημα p του P. Αυτό ισχύει, διότι οι νέες προτάσεις της μορφής C_{N+1} είναι Eureka ορισμοί *([29]), ενώ οι προτάσεις της μορφής C' προκύπτουν από folding ([47, 15, 29]) των προτάσεων C χρησιμοποιώντας προτάσεις της μορφής C_{N+1}. Τελικά στο πρόγραμμα P^i, όλες οι προτάσεις έχουν το πολύ n άτομα στο σώμα τους. Επομένως, μπορούμε να εφαρμόσουμε την επαγωγική υπόθεση και να πάρουμε το τελικό fm-Datalog πρόγραμμα P'.*

Στην παρουσίαση του προτεινόμενου μετασχηματισμού χρησιμοποιούμε απλά fm-Datalog προγράμματα σαν αρχική γλώσσα. Με βάση τις προηγούμενες προτάσεις, αυτό δεν αποτελεί περιορισμό της εφαρμογής του μετασχηματισμού, απλά γίνεται για λόγους ευκολίας της παρουσίασης.

Παράδειγμα 3.4.4 *Το ακόλουθο πρόγραμμα είναι ένα fm-Datalog πρόγραμμα:*

```
q(X,Z) ← f(X,Z).
q(X,Z) ← e(X,Y),q(Y,W),g(X,W,Z).
```

Το παραπάνω fm-Datalog πρόγραμμα δεν είναι απλό, καθώς η δεύτερη του πρόταση έχει περισσότερα από δύο άτομα στο σώμα της. Για να το μετατρέψουμε σε ένα απλό fm-Datalog πρόγραμμα, εισάγουμε μία νέα πρόταση:

```
p(X,Y,Z) ← q(Y,W),g(X,W,Z).
```

και την χρησιμοποιούμε για να κάνουμε fold την δεύτερη πρόταση του προγράμματος. Με αυτό τον τρόπο παίρνουμε το ακόλουθο απλό fm-Datalog πρόγραμμα:

```
q(X,Z) ← f(X,Z).
q(X,Z) ← e(X,Y),p(X,Y,Z).
p(X,Y,Z) ← q(Y,W),g(X,W,Z).
```

3.5 Η τελική γλώσσα του μετασχηματισμού

Η τελική γλώσσα του μετασχηματισμού είναι η γλώσσα Choice Datalog$_{nS}$ που είναι μία παραλλαγή της Datalog$_{nS}$ [13, 12] επεκταμένη με κατηγορήματα επιλογής [27]. Η Datalog$_{nS}$ είναι μία ισχυρή γλώσσα στις χρονικές επαγωγικές βάσεις δεδομένων και τα κατηγορήματα επιλογής είναι μη-ντετερμινιστικές κατασκευές χρήσιμες για χρονικές γλώσσες.

3.5.1 Κατηγορήματα επιλογής

Τα κατηγορήματα επιλογής [27] αρχικά είχαν εισαχθεί στην περιοχή των χρονικών λογικών προγραμμάτων σαν ένα μέσο για να εξασφαλίζουν ότι ένα δοσμένο κατηγόρημα έχει μοναδική τιμή σε μια δεδομένη χρονική στιγμή (γενικότερα σε ένα δοσμένο περιβάλλον). Ειδικότερα, με κάθε κατηγόρημα p ενός προγράμματος, ένα κατηγόρημα $\#p$, που ονομάζεται το κατηγόρημα επιλογής του p, συσχετίζεται με το p. Τα κατηγορήματα επιλογής μπορούν να εμφανίζονται μόνο στα σώματα των προτάσεων του προγράμματος (η αξιωματικοποίηση τους δίνεται αναλυτικά στο [27]).

Για να δώσουμε το κίνητρο των κατηγορημάτων επιλογής, ας δούμε ένα πρόγραμμα για να αναθέσουμε μία αίθουσα διδασκαλίας σε άτομα, διδάσκοντες, σε διαφορετικές χρονικές στιγμές. Το πρόβλημα είναι να βρούμε όλες τις διαφορετικές αναθέσεις με τέτοιο τρόπο ώστε σε κάθε χρονική στιγμή μόνο ένας διδάσκων να διαθέτει την αίθουσα. Το κατηγόρημα `requests_class(Time,Person)` εκφράζει το γεγονός ότι το `Person` *πραγματικά* χρησιμοποιεί την αίθουσα τη στιγμή `Time`.

```
requests_class(0,tom).
requests_class(0,nick).
requests_class(1,mary).
uses_class(Time,Person) ← #requests_class(Time,Person).
```

Στο παραπάνω πρόγραμμα, το `#requests_class` είναι το κατηγόρημα επιλογής που αντιστοιχεί στο (κλασικό) κατηγόρημα `requests_class`. Αυτό σημαίνει ότι είτε το άτομο `#requests_class(0,tom)` ή `#requests_class(0,nick)` αλλά όχι και τα δύο θα θεωρηθούν ως αληθή τη στιγμή `0`. Γι' αυτό, το παραπάνω πρόγραμμα δεν έχει ένα μοναδικό ελάχιστο μοντέλο, όπως συμβαίνει στον κλασικό λογικό προγραμματισμό, αντιθέτως έχει ένα σύνολο από ελαχιστικά μοντέλα (minimal models), ένα για κάθε διαφορετική (συναρτησιακή) ανάθεση των ατόμων στις διαφορετικές χρονικές στιγμές. Πιο συγκεκριμένα, τα δύο ελαχιστικά μοντέλα του παραπάνω προγράμματος είναι τα ακόλουθα:

$M_1 = \{$`#uses_class(0,tom),#uses_class(1,mary),uses_class(0,tom), uses_class(1,mary),#requests_class(0,tom),#requests_class(1,mary), requests_class(0,tom),requests_class(0,nick),requests_class(1,mary)`$\}$
and

$M_2 = \{$`#uses_class(0,nick),#uses_class(1,mary),uses_class(0,nick), uses_class(1,mary),#requests_class(0,nick),#requests_class(1,mary), requests_class(0,tom),requests_class(0,nick),requests_class(1,mary)`$\}$

Τα κατηγορήματα επιλογής δεν είναι απαραίτητα να περιορίζονται στην εφαρμογή απλών γλωσσών χρονικών λογικών προγραμμάτων, όπως χρησιμοποιήθηκε στο παράδειγμα (όπου ο χρόνος είναι γραμμικός). Εφαρμόζονται και σε πιο γενικές νοηματικές γλώσσες προγραμματισμού [26], που περιλαμβάνουν τη γλώσσα Datalog$_{nS}$ η οποία υιοθετήθηκε για να οριστεί ο προτεινόμενος μετασχηματισμός. Πιο συγκεκριμένα, στο [27], μια γενική σημασιολογική προσέγγιση δόθηκε για τα κατηγορήματα επιλογής που μπορεί να εφαρμοστεί σε μια ευρεία κλάση νοηματικών γλωσσών λογικού προγραμματισμού.

3.5.2 Το συντακτικό της γλώσσας Choice Datalog$_{nS}$

Η τελική γλώσσα του μετασχηματισμού είναι μια γλώσσα χρονικών επαγωγικών βάσεων δεδομένων που υποστηρίζει κατηγορήματα επιλογής. Θα αναφέρομαι σε αυτή τη τελική γλώσσα σαν *Choice Datalog$_{nS}$* μια και είναι μια εκδοχή της γλώσσας Datalog$_{nS}$ [12], με πρόσθετο στοιχείο τα κατηγορήματα επιλογής [27].

Η *Datalog$_{nS}$* είναι μια επέκταση της Datalog στην οποία τα άτομα έχουν ένα ξεχωριστό *χρονικό* όρισμα εκτός από το συνηθισμένο *όρισμα δεδομένων*. Ο χρόνος στην Datalog$_{nS}$ μπορεί να έχει αρκετά περίπλοκη δομή, δηλαδή δεν είναι κατ' ανάγκη γραμμικός. Στη περίπτωση που μελετώ, έχει χρησιμοποιηθεί μια παραλλαγή της Datalog$_{nS}$ στην οποία ο χρόνος έχει δενδροειδή/διακλαδιζόμενη δομή. Μπορούμε να δούμε ότι ο δενδροειδής/διακλαδιζόμενος χρόνος μπορεί να μοντελοποιηθεί αν αντικαταστήσουμε τις ακολουθίες χρονικών τελεστών με λίστες ακεραίων αριθμών[1] Επεκτείνουμε τη παραλλαγή αυτή της Datalog$_{nS}$ ώστε να μπορεί να υποστηρίζει κατηγορήματα επιλογής. Πριν προχωρήσω σε ένα τυπικό ορισμό του συντακτικού της γλώσσας, θα δώσω ένα παράδειγμα της μορφής μιας Choice Datalog$_{nS}$ πρότασης:

$$\texttt{q(L,Z)} \leftarrow \texttt{g(X,Y,Z),\#p([3|L],X),r([4,3|L],Y).}$$

Στο παραπάνω παράδειγμα, τα `L`, `[3|L]` και `[4,3|L]` είναι οι όροι που αντιστοιχούν στα ξεχωριστά χρονικά ορίσματα των ατόμων, τα `X`, `Y` και `Z` είναι συνηθισμένα ορίσματα δεδομένων Τα άτομα `q(L,Z)`, `#p([3|L],X)` και `r([4,3|L],Y)` είναι IDB άτομα, ενώ το `g(X,Y,Z)` είναι ένα EDB άτομο. Με άλλα λόγια, θεωρούμε πως μόνο τα IDB άτομα έχουν ένα χρονικό όρισμα, ενώ τα EDB άτομα δεν έχουν, μια και θεωρούμε πως ισχύουν πάντοτε (σε κάθε context). Το άτομο `#p([3|L],X)` είναι ένα παράδειγμα ενός κατηγορήματος επιλογής.

Ορίζω τώρα τυπικά το συντακτικό της τελικής γλώσσας του μετασχηματισμού. Υποθέτω ότι υπάρχει μια ξεχωριστή μεταβλητή L, που θα είναι η μόνη μεταβλητή που μπορεί να εμφανίζεται σε χρονικούς όρους ενός προγράμματος, ενώ όλες οι άλλες μεταβλητές που εμφανίζονται σε ένα πρόγραμμα πρέπει να είναι διαφορετικές από την L. Ένας χρονικός όρος ορίζεται ως εξής:

Ορισμός 3.5.1 *Ένας* χρονικός όρος (temporal term) *ορίζεται αναδρομικά ως εξής:*

$$tt ::= [\,] \mid L \mid [l|tt]$$

όπου L είναι μια χρονική μεταβλητή και $l \in \omega$.

Ένα *IDB άτομο* είναι ένα άτομο της μορφής $p(tt,t)$ ή της μορφής $\#p(tt,t)$, όπου tt είναι ένας χρονικός όρος και t είναι ένας συνηθισμένος όρος Datalog, δηλαδή είτε μία μεταβλητή είτε μία σταθερά. Τα πρώτα άτομα λέγονται *άτομα μη-επιλογής ή κλασικά άτομα*, ενώ τα επόμενα λέγονται *άτομα επιλογής*. Παρατηρούμε ότι στην εκδοχή της Datalog$_{nS}$ που χρησιμοποιώ, τα IDB άτομα έχουν μόνο ένα όρισμα δεδομένων (κάτι που δεν ίσχυε στα αρχικά IDBs). Ένα *EDB άτομο* είναι της

[1] Στην αρχική παρουσίαση του μετασχηματισμού [37] είχαν χρησιμοποιηθεί ακολουθίες από χρονικούς τελεστές. Εύκολα μπορεί να δειχθεί πως οι δύο αυτές προσεγγίσεις είναι ισοδύναμες.

μορφής $p(t_1, \ldots, t_n)$, όπου $t_1, \ldots, t_n$ είναι συνηθισμένοι Datalog όροι. Ως συνήθως, ένα άτομο λέγεται αποτιμημένο, αν είναι ελεύθερο από μεταβλητές (variable-free). Μία *πρόταση (clause)* στη γλώσσα Choice Datalog$_{nS}$ είναι της μορφής:

$$H \leftarrow B_1, B_2, \ldots, B_n.$$

όπου H είναι ένα IDB άτομο μη-επιλογής και $B_1, \ldots, B_n$ είναι άτομα (είτε IDB είτε EDB) και $n \geq 0$. Αν $n = 0$, τότε η πρόταση λέγεται *γεγονός (fact)* ή *μοναδιαία πρόταση (unit clause)*. Ένα Choice Datalog$_{nS}$ πρόγραμμα είναι ένα πεπερασμένο σύνολο από προτάσεις της παραπάνω μορφής. Ως συνήθως, P_D είναι ένα Choice Datalog$_{nS}$ πρόγραμμα P μαζί με μια βάση δεδομένων D. Ένας *στόχος (goal)* στην Choice Datalog$_{nS}$ είναι της μορφής $\leftarrow p([\,], X)$.

3.5.3 Η σημασιολογία της γλώσσας Choice Datalog$_{nS}$

Το σύμπαν Herbrand U_{P_D} ενός Choice Datalog$_{nS}$ προγράμματος P μαζί με μία βάση δεδομένων D περιλαμβάνει όλες τις σταθερές που εμφανίζονται στη βάση D ή σαν ορίσματα δεδομένων του P. Η βάση Herbrand B_{P_D} περιλαμβάνει:

- αποτιμημένα EDB άτομα που κατασκευάζονται χρησιμοποιώντας σταθερές από το U_{P_D}
- αποτιμημένα IDB άτομα (είτε επιλογής είτε μη) των οποίων το πρώτο όρισμα είναι μία λίστα από φυσικούς αριθμούς και το δεύτερο όρισμα είναι ένα στοιχείο του U_{P_D}.

Μια Herbrand ερμηνεία I του P_D είναι (κατά τα συνήθη) ένα υποσύνολο της βάσης Herbrand B_{P_D}. Η σημασιολογία των Choice Datalog$_{nS}$ προγραμμάτων μπορεί να οριστεί χρησιμοποιώντας τις βασικές αρχές που αναπτύχθηκαν στο [27]. Για λόγους πληρότητας υιοθετούμε τις βασικές έννοιες από το [27] για να ταιριάζει στους σκοπούς μας. Για μια αναλυτικότερη παρουσίαση του θέματος, ο ενδιαφερόμενος μπορεί να ανατρέξει στο [27].

Πριν να ορίσουμε την έννοια του μοντέλου χρειάζεται να ορίσουμε ένα σύνολο αξιωμάτων, ώστε να συνδέσουμε τα κατηγορήματα επιλογής με τα αντίστοιχα κλασικά κατηγορήματα.

Ορισμός 3.5.2 (Τύποι επιλογής) *Έστω P ένα Choice Datalog$_{nS}$ πρόγραμμα και p ένα κατηγόρημα που ορίζεται στο P. Τότε οι* τύποι επιλογής *που σχετίζονται με το p είναι οι εξής:*

$$\forall L \forall X (\#p(L, X) \rightarrow p(L, X)).$$
$$\forall L \forall X (\#p(L, X) \rightarrow \forall Y (\#p(L, Y) \rightarrow X = Y)).$$

Στη συνέχεια ορίζουμε την έννοια του μοντέλου για τα Choice Datalog$_{nS}$ προγράμματα:

Ορισμός 3.5.3 *Έστω P ένα Choice Datalog$_{nS}$ πρόγραμμα. Τότε ένα* μοντέλο (model) *του P είναι μια Herbrand ερμηνεία που ικανοποιεί τις προτάσεις του P μαζί με τους σχετικούς τύπους επιλογής για όλα τα κατηγορήματα που ορίζονται στο P.*

Μπορούμε να ορίσουμε έναν τελεστή T_{P_D} με το συνηθισμένο τρόπο. Ωστόσο, αυτή τη φορά θέλουμε κάτι πιο ισχυρό από τον συνήθη T_{P_D} για να ορίσουμε τη (λειτουργική) σημασιολογία των Choice Datalog$_{nS}$ προγραμμάτων. Για το λόγο αυτό, ορίζω έναν τελεστή NT_{P_D} που δουλεύει ακριβώς όπως ο T_{P_D}, αλλά δεν διώχνει τα άτομα επιλογής [27]. Πιο τυπικά:

Ορισμός 3.5.4 *Έστω P ένα Choice Datalog$_{nS}$ πρόγραμμα, D μια βάση δεδομένων και I μια Herbrand ερμηνεία του P_D. Τότε ο $NT_{P_D}(I)$ ορίζεται ως εξής:*

$$NT_{P_D}(I) = T_{P_D}(I) \cup choices(I)$$

όπου $choices(I)$ είναι το σύνολο των ατόμων επιλογής που ανήκουν στην ερμηνεία I.

Επιπλέον, η σημασιολογία των Choice Datalog$_{nS}$ προγραμμάτων απαιτούν έναν επιπλέον τελεστή, τον τελεστή C_{P_D}, που επιστρέφει όλες τις δυνατές άμεσες επεκτάσεις μιας Herbrand ερμηνείας του P_D, που καθορίζονται από τυχαίες επιλογές.

Έστω I μια Herbrand ερμηνεία του Choice Datalog$_{nS}$ προγράμματος μαζί με τη βάση δεδομένων P_D και έστω p ένα κατηγόρημα που εμφανίζεται στο P_D. Ορίζουμε:

$$E_{p,L}(I) = \{a \mid p(L, a) \in I\}$$

και επίσης:

$$E_{\#p,L}(I) = \{a \mid \#p(L, a) \in I\}$$

Ακόμα, έστω:

$$S_I = \{\langle p, L, a\rangle \mid E_{p,L}(I) \neq \emptyset,\ E_{\#p,L}(I) = \emptyset \ \textit{και}\ a \in E_{p,L}(I)\}$$

Με άλλα λόγια, S_I είναι ένα σύνολο, έτσι ώστε $\langle p, L, a\rangle \in S_I$, αν καμία επιλογή δεν έχει γίνει για το p στο περιβάλλον L και το a είναι ένα πιθανός υποψήφιος για αυτή την επιλογή. Ο τυπικός ορισμός του C_{P_D} δίνεται από κάτω:

Ορισμός 3.5.5 *Έστω P ένα Choice Datalog$_{nS}$ πρόγραμμα, D μια βάση δεδομένων και I μια Herbrand ερμηνεία του P_D. Τότε, $C_{P_D}(I)$ ορίζεται ως εξής:*

$$C_{P_D}(I) = \{I \cup \{\#p(L, a)\} \mid \langle p, L, a\rangle \in S_I\}$$

Μερικές χρήσιμες παρατηρήσεις μπορούν να γίνουν για τον τελεστή C_{P_D}. Η πρώτη, είναι ότι ο C_{P_D} διατηρεί τα άτομα που έχουν υπολογιστεί προηγουμένως, δηλαδή δεν απορρίπτει τίποτα, άρα είναι μονότονος. Το επόμενο είναι ότι C_{P_D} όταν εφαρμόζεται σε μία ερμηνεία I, επιλέγει *ακριβώς* ένα μεταξύ όλων των πιθανών $\langle p, L, a\rangle$ τριάδων στο S_I, και επιστρέφει το άτομο $\#p(L, a)$. Με απλό τρόπο, δοθείσας μιας ερμηνείας I, ο $C_{P_D}(I)$ είναι ένα σύνολο από ερμηνείες, όπου καθεμία αντιστοιχεί σε μια διαφορετική επιλογή που προστίθεται στο I.

Στη συνέχεια ορίζουμε την έννοια της P_D-αλυσίδας (P_D-chain), η οποία διαισθητικά περιγράφει έναν bottom-up υπολογισμό ενός Choice Datalog$_{nS}$ προγράμματος P μαζί με μια βάση δεδομένων D. Κατά τη διάρκεια του bottom-up υπολογισμού οι τελεστές $NT_{P_D} \uparrow \omega$ και C_{P_D} εναλλάσσονται όπως φαίνεται στον παρακάτω ορισμό:

Ορισμός 3.5.6 *Έστω P ένα Choice Datalog$_{nS}$ πρόγραμμα και D μια βάση δεδομένων. Μια P_D-αλυσίδα (P_D-chain) είναι μια ακολουθία $\langle M_0, M_1, M_2, \cdots \rangle$ που ικανοποιεί τις ακόλουθες συνθήκες:*

$$M_0 = \emptyset,$$

$$M_{2i+1} = NT_{P_D} \uparrow \omega(M_{2i}), \;\; i \geq 0,$$

και

$$M_{2i+2} \in C_{P_D}(M_{2i+1}), \;\; i \geq 0.$$

Παρατηρήστε πως στον προηγούμενο ορισμό, με $NT_{P_D} \uparrow \omega(M_{2i})$ συμβολίζουμε, ως συνήθως το σύνολο $\bigcup_{n \in \omega} NT^n_{P_D}(M_{2i})$.

Το λήμμα που αναφέρεται στη συνέχεια θα χρησιμοποιηθεί στις αποδείξεις παρακάτω και δηλώνει πως η P_D-chain είναι μονότονη.

Λήμμα 3.5.1 *Έστω P ένα Choice Datalog$_{nS}$ πρόγραμμα, D μια βάση δεδομένων και έστω $\langle M_0, M_1, \cdots \rangle$ μια P_D-chain. Τότε, για όλα τα $i \in \omega, M_i \subseteq M_{i+1}$.*

Απόδειξη 3.5.1 *Η απόδειξη του παραπάνω λήμματος είναι εύκολη και απορρέει από τον ορισμό της P_D-chain και της μονοτονικότητας των τελεστών που εναλλάσσονται σε αυτήν.*

Το ελάχιστο άνω φράγμα (least upper bound) μιας P_D-chain $\langle M_0, M_1, M_2, \cdots \rangle$ είναι το $M = \bigcup_{i<\omega} M_i$. Τα $M_0, M_1, \cdots$ λέγονται *προσεγγίσεις (approximations)* του M και το M λέγεται μία *οριακή προσέγγιση (limit interpretation)* του P_D. Δοθέντων δύο στοιχείων M_i και M_j μιας P_D-chain, λέμε ότι το M_i είναι ένας *πρόγονος (ancestor)* του M_j, αν $i < j$. Βασισμένος στους παραπάνω ορισμούς, το ακόλουθο θεώρημα είναι εύκολο να δειχθεί:

Θεώρημα 3.5.1 *Έστω P ένα Choice Datalog$_{nS}$ πρόγραμμα και D μια βάση δεδομένων. Τότε κάθε οριακή προσέγγιση του P_D είναι ένα μοντέλο του P_D.*

Παράδειγμα 3.5.1 *Έστω το ακόλουθο Choice Datalog$_{nS}$ πρόγραμμα:*

```
p([1|L],X) ← #q(L,X).
p([2|L],X) ← #q(L,X).
q([],a).
q([],b).
```

Εφαρμόζοντας τους τελεστές $NT_{P_D} \uparrow \omega$ και C_{P_D}, όπως φαίνεται στον Ορισμό 3.5.6 παίρνουμε τις δύο οριακές προσεγγίσεις:

$M = \{\#\mathtt{p}([1], \mathtt{a}), \#\mathtt{p}([2], \mathtt{a}), \mathtt{p}([1], \mathtt{a}), \mathtt{p}([2], \mathtt{a}), \#\mathtt{q}([\,], \mathtt{a}), \mathtt{q}([\,], \mathtt{a}), \mathtt{q}([\,], \mathtt{b})\}$, *και*

$M' = \{\#\mathtt{p}([1], \mathtt{b}), \#\mathtt{p}([2], \mathtt{b}), \mathtt{p}([1], \mathtt{b}), \mathtt{p}([2], \mathtt{b}), \#\mathtt{q}([\,], \mathtt{b}), \mathtt{q}([\,], \mathtt{a}), \mathtt{q}([\,], \mathtt{b})\}$.

Εύκολα μπορεί να επιβεβαιωθεί πως οι δύο αυτές ερμηνείες είναι μοντέλα του προγράμματος.

Στη συνέχεια δείχνουμε αναλυτικά τα βήματα για την παραγωγή των δύο αυτών μοντέλων:

$$
\begin{aligned}
&M_0 = \emptyset,\\
&M_1 = \{q([\,], a), q([\,], b)\},\\
&M_2 = M_1 \cup \{\#q([\,], a)\},\\
&M_2' = M_1 \cup \{\#q([\,], b)\},\\
&M_3 = M_2 \cup \{p([1], a), p([2], a)\},\\
&M_3' = M_2' \cup \{p([1], b), p([2], b)\},\\
&M_4 = M_3 \cup \{\#p([1], a)\},\\
&M_4' = M_3 \cup \{\#p([2], a)\},\\
&M_4'' = M_3' \cup \{\#p([1], b)\},\\
&M_4''' = M_3' \cup \{\#p([2], b)\},\\
&M_5 = M_4 \cup \{\#p([2], a)\},\\
&M_5' = M_4' \cup \{\#p([1], a)\},\\
&M_5'' = M_4'' \cup \{\#p([2], b)\},\\
&M_5''' = M_4''' \cup \{\#p([1], b)\}
\end{aligned}
$$

όπου $M = M_5 = M_5'$ και $M' = M_5'' = M_5'''$, δηλ. καθένα από τα δύο μοντέλα παράγονται με δύο τρόπους.

3.6 Ο αλγόριθμος μετασχηματισμού

Σε αυτή την ενότητα δίνεται ένας τυπικός ορισμός του αλγορίθμου μετασχηματισμού για την περίπτωση που έχουμε πολλαπλές καταναλώσεις μεταβλητών. Στη συνέχεια αυτός ο αλγόριθμος θα αναφέρεται ως *fixed mode μετασχηματισμός*. Ο αλγόριθμος επιδεικνύεται στη συνέχεια με ένα χαρακτηριστικό παράδειγμα.

Αλγόριθμος: Έστω P ένα απλό fm-Datalog πρόγραμμα και G ένας στόχος. Για κάθε IDB κατηγόρημα p με βαθμό $n+1$ στο P, εισάγω $n+1$ δυαδικά IDB κατηγορήματα $p_1^+, \ldots, p_n^+$, p^-, όπου το p_i^+ αντιστοιχεί στο i-οστό όρισμα εισόδου του p και p^- στο $n+1$-στό όρισμα του p, που είναι το όρισμα εξόδου του. Τα αντίστοιχα κατηγορήματα επιλογής μερικών από τα παραπάνω κατηγορήματα μπορεί επίσης να χρησιμοποιούνται στο τελικό πρόγραμμα. Η διαδικασία μετασχηματίζει το στόχο G και κάθε μία πρόταση του P και δίνει σαν έξοδο ένα νέο στόχο G^* μαζί με ένα Choice Datalog$_{nS}$ πρόγραμμα P^*. Θεωρούμε πως υπάρχει μια συνάρτηση ανάθεσης διαφορετικών ετικετών για κάθε άτομο που εμφανίζεται στο σώμα κάθε κανόνα του P. Οι ετικέτες είναι φυσικοί αριθμοί και συμβολίζονται με $l_1, l_2, \cdots$. Ο αλγόριθμος ορίζεται με ανάλυση περιπτώσεων, όπου κάθε περίπτωση εξαρτάται από τη μορφή της πρότασης που επεξεργαζόμαστε κάθε φορά:

Περίπτωση 1: Ο μετασχηματισμός του στόχου:

$$\leftarrow p(a_1, \ldots, a_n, Y).$$

δίνει σαν αποτέλεσμα ένα σύνολο από n νέες μοναδιαίες προτάσεις, που προστίθενται στο τελικό πρόγραμμα P^*:

$$p_i^+([\,], a_i).$$

όπου $i = 1, \ldots, n$. Ο νέος στόχος G^* είναι:

$$\leftarrow\ p^-([\,], Y).$$

Περίπτωση 2: Έστω C μια πρόταση της μορφής:

$$p(\vec{v}_0, Z)\ \leftarrow\ q(\vec{v}_1, Y), r(\vec{v}_2, Z).$$

και έστω l_1, l_2 οι ετικέτες των $q(\vec{v}_1, Y)$ και $r(\vec{v}_2, Z)$, αντίστοιχα. Τότε η C μετασχηματίζεται με τον ακόλουθο τρόπο:

1. Η ακόλουθη πρόταση προστίθεται στο P^*:

$$p^-(L, Z) \leftarrow\ r^-([l_2|L], Z).$$

2. Έστω X μία μεταβλητή που εμφανίζεται στην k-οστή θέση του $\vec{v}_0$ όπως επίσης και στην m-οστή θέση του $\vec{v}_1$. Τότε η ακόλουθη πρόταση προστίθεται στο P^*:

$$q_m^+([l_1|L], X) \leftarrow\ [\#]p_k^+(L, X).$$

όπου $[\#]p_k^+ = \#p_k^+$ αν X εμφανίζεται δύο φορές στο σώμα της C και $[\#]p_k^+ = p_k^+$ διαφορετικά. Παρόμοια, έστω X μία μεταβλητή που εμφανίζεται στην k-οστή θέση του $\vec{v}_0$ καθώς επίσης και στην m-οστή θέση του $\vec{v}_2$. Τότε η ακόλουθη πρόταση προστίθεται στο P^*:

$$r_m^+([l_2|L], X) \leftarrow\ [\#]p_k^+(L, X).$$

3. Αν η μεταβλητή εξόδου Y του q εμφανίζεται και στην m-οστή θέση του $\vec{v}_2$, τότε η ακόλουθη πρόταση προστίθεται στο P^*:

$$r_m^+([l_2|L], Y) \leftarrow\ q^-([l_1|L], Y).$$

Περίπτωση 3: Έστω η πρόταση C της μορφής:

$$p(\vec{v}_0, Z)\ \leftarrow\ q(\vec{v}_1, Z).$$

και έστω l_3 η ετικέτα του $q(\vec{v}_1, Z)$. Τότε η C μετασχηματίζεται ως εξής:

1. Η ακόλουθη πρόταση προστίθεται στο P^*:

$$p^-(L, Z) \leftarrow\ q^-([l_3|L], Z).$$

2. Έστω X η μεταβλητή που εμφανίζεται στην k-οστή θέση του $\vec{v}_0$ καθώς επίσης και στην m-οστή θέση του $\vec{v}_1$. Τότε η ακόλουθη πρόταση προστίθεται στο P^*:

$$q_m^+([l_3|L], X) \leftarrow \; p_k^+(L, X).$$

Περίπτωση 4: Για κάθε EDB κατηγόρημα p του P με n μεταβλητές εισόδου και μία μεταβλητή εξόδου, μία νέα πρόταση της ακόλουθης μορφής προστίθεται στο P^*:

$$p^-(L, Y) \leftarrow p(X_1, \ldots, X_n, Y), p_1^+(L, X_1), \ldots, p_n^+(L, X_n).$$

Παράδειγμα 3.6.1 *Έστω το ακόλουθο απλό fm-Datalog πρόγραμμα P, που προκύπτει από το Παράδειγμα 3.4.4, μαζί με ένα στόχο, όπου* `p`, `q` *είναι IDB κατηγορήματα και* `e`, `f`, `g` *είναι EDB κατηγορήματα:*

$$\begin{array}{l}
\leftarrow \mathtt{q(a,Z)}. \\
\mathtt{q(X,Z)} \leftarrow \mathtt{f(X,Z)}. \\
\mathtt{q(X,Z)} \leftarrow \mathtt{e(X,Y),p(X,Y,Z)}. \\
\mathtt{p(X,Y,Z)} \leftarrow \mathtt{q(Y,W),g(X,W,Z)}.
\end{array}$$

Το πρόγραμμα αυτό μετασχηματίζεται ως εξής:

$$\begin{array}{l}
\leftarrow \mathtt{q}^-\mathtt{([],Z)}. \\
\mathtt{q}_1^+\mathtt{([],a)}. \\
\mathtt{q}^-\mathtt{(L,Z)} \leftarrow \mathtt{f}^-\mathtt{([1|L],Z)}. \\
\mathtt{f}_1^+\mathtt{([1|L],X)} \leftarrow \mathtt{q}_1^+\mathtt{(L,X)}. \\
\mathtt{q}^-\mathtt{(L,Z)} \leftarrow \mathtt{p}^-\mathtt{([3|L],Z)}. \\
\mathtt{p}_2^+\mathtt{([3|L],Y)} \leftarrow \mathtt{e}^-\mathtt{([2|L],Y)}. \\
\mathtt{p}_1^+\mathtt{([3|L],X)} \leftarrow \mathtt{\#q}_1^+\mathtt{(L,X)}. \\
\mathtt{e}_1^+\mathtt{([2|L],X)} \leftarrow \mathtt{\#q}_1^+\mathtt{(L,X)}. \\
\mathtt{p}^-\mathtt{(L,Z)} \leftarrow \mathtt{g}^-\mathtt{([5|L],Z)}. \\
\mathtt{g}_2^+\mathtt{([5|L],W)} \leftarrow \mathtt{q}^-\mathtt{([4|L],W)}. \\
\mathtt{g}_1^+\mathtt{([5|L],X)} \leftarrow \mathtt{p}_1^+\mathtt{(L,X)}. \\
\mathtt{q}_1^+\mathtt{([4|L],Y)} \leftarrow \mathtt{p}_2^+\mathtt{(L,Y)}. \\
\mathtt{e}^-\mathtt{(L,Y)} \leftarrow \mathtt{e(X,Y)},\ \mathtt{e}_1^+\mathtt{(L,X)}. \\
\mathtt{f}^-\mathtt{(L,Y)} \leftarrow \mathtt{f(X,Y)},\ \mathtt{f}_1^+\mathtt{(L,X)}. \\
\mathtt{g}^-\mathtt{(L,Z)} \leftarrow \mathtt{g(X,Y,Z)},\ \mathtt{g}_1^+\mathtt{(L,X)},\ \mathtt{g}_2^+\mathtt{(L,Y)}.
\end{array}$$

3.7 Η ορθότητα του μετασχηματισμού

Σε αυτή την ενότητα παρουσιάζεται η ορθότητα του προτεινόμενου μετασχηματισμού, στην βελτιωμένη μορφή, δηλ. με κατηγορήματα επιλογής μόνο εκεί που χρειάζονται.

Έστω P ένα απλό fm-Datalog πρόγραμμα, D μια βάση δεδομένων και $\leftarrow p(a_1, \ldots, a_n, Y)$ ένας στόχος. Η απόδειξη της ορθότητας του μετασχηματισμού γίνεται ως εξής: πρώτα δείχνουμε (βλέπε Λήμμα 3.7.4 παρακάτω) πως αν ένα αποτιμημένο στιγμιότυπο $p(a_1, \ldots, a_n, b)$ του στόχου είναι ένα λογικό επακόλουθο του

P_D τότε το $p^-([\,], b)$ ανήκει σε μία οριακή ερμηνεία M του P^*_D, όπου $P^* \cup \{\leftarrow p^-([\,], Y)\}$ προκύπτει από την εφαρμογή του αλγορίθμου μετασχηματισμού στο $P \cup \{\leftarrow p(a_1, \ldots, a_n, Y)\}$. Για να αποδείξουμε το αποτέλεσμα, αποδεικνύουμε ένα πιο γενικό λήμμα (Λήμμα 3.7.3 πιο κάτω).

Το αντίστροφο του Λήμματος 3.7.4 είναι το Λήμμα 3.7.6. Πιο συγκεκριμένα, αποδεικνύουμε ότι, οποτεδήποτε το $p^-([\,], b)$ ανήκει σε μια οριακή ερμηνεία M του P^*_D, τότε το $p(a_1, \ldots, a_n, b)$ είναι ένα λογικό επακόλουθο του P_D. Πάλι, αποδεικνύουμε το αποτέλεσμα αποδεικνύοντας ένα πιο γενικό λήμμα (Λήμμα 3.7.5). Συνδυάζοντας τα παρακάτω αποτελέσματα παίρνουμε την απόδειξη της ορθότητας του μετασχηματισμού με το Θεώρημα 3.7.1.

Στη συνέχεια θα δώσουμε μερικούς ορισμούς που θα χρησιμοποιηθούν για τις αποδείξεις σε αυτή την ενότητα.

Ορισμός 3.7.1 *Έστω P ένα Choice Datalog$_{nS}$ πρόγραμμα και D μία βάση δεδομένων. Έστω I μία προσέγγιση σε μια οριακή ερμηνεία του P_D και έστω A ένα άτομο που ανήκει στην I. Τότε ένα* σύνολο παραγωγής (derivation set) *του A στην I ορίζεται αναδρομικά ως εξής:*

- *Αν το A είναι μια μοναδιαία πρόταση στο P_D, τότε το $\{A\}$ είναι ένα σύνολο παραγωγής για το A στην I.*
- *Αν τα $S_1, \ldots, S_n$ είναι σύνολα παραγωγής για τα $B_1, \ldots, B_n$, αντίστοιχα, στην I και $A \leftarrow B_1, \ldots, B_n$ είναι ένα αποτιμημένο στιγμιότυπο μιας πρότασης του P_D, τότε το $\{A\} \cup S_1 \cup \cdots \cup S_n$ είναι ένα σύνολο παραγωγής του A στο I.*
- *Αν το A είναι ένα κατηγόρημα επιλογής και το S είναι ένα σύνολο παραγωγής της εκδοχής χωρίς επιλογή του A που έχει εισαχθεί σε έναν πρόγονο J της I, τότε το $\{A\} \cup S$ είναι ένα σύνολο παραγωγής του A στην I.*

Η έννοια του συνόλου παραγωγής εκφράζει την έννοια των (πιθανώς πολλών) τρόπων που μπορούν να οδηγήσουν στην παραγωγή ενός ατόμου σε μια προσέγγιση. Γι' αυτό, ένα άτομο μπορεί να έχει πολλά διαφορετικά σύνολα παραγωγής σε μία δοσμένη προσέγγιση.

Οι ακόλουθοι ορισμοί είναι επίσης χρήσιμοι:

Ορισμός 3.7.2 *Ένα άτομο A* εξαρτάται από (depends on) *ένα άτομο B στην I αν υπάρχει ένα σύνολο παραγωγής S του A στην I τέτοιο ώστε το $B \in S$.*

Ορισμός 3.7.3 *Έστω P ένα Choice Datalog$_{nS}$ πρόγραμμα και D μία βάση δεδομένων. Έστω I μία προσέγγιση μιας οριακής ερμηνείας του P_D και έστω πως υπάρχουν A και B τέτοια ώστε το A να εξαρτάται από το B στην I. Έστω S ένα σύνολο παραγωγής για το A στην I που περιλαμβάνει το B. Ορίζουμε την* απόσταση (distance) *ανάμεσα στα A και B στην S να είναι ο ελάχιστος αριθμός από αποτιμημένα στιγμιότυπα προτάσεων του προγράμματος που χρησιμοποιούνται για να δημιουργηθεί η σχέση εξάρτησης του A από το B στην S. Ορίζουμε την* απόσταση (distance) *ανάμεσα στα A και B στην I να είναι η ελάχιστη απόσταση* *ανάμεσα στα A και B σε κάθε σύνολο παραγωγής S του A στην I που περιλαμβάνει το B. Όταν η απόσταση ανάμεσα στα A και B στην I είναι μεγαλύτερη του 0 λέμε πως το A* εξαρτάται ουσιωδώς (depends essentially) *από το B στην I.*

Ορισμός 3.7.4 *Έστω P ένα Choice Datalog$_{nS}$ πρόγραμμα και D μια βάση δεδομένων. Έστω I μια προσέγγιση σε μια οριακή ερμηνεία του P_D. Τότε η I λέγεται* μη περιοριστική ως προς το (non-restrictive with respect to) $S \subseteq I$, *αν ισχύουν τα ακόλουθα:*

1. *Κανένα άτομο επιλογής της I δεν εξαρτάται από κανένα στοιχείο του S.*
2. *Αν $p(L, a) \in S$ τότε για όλα τα $b \in U_{P_D}$, $\#p(L, b) \notin I$.*

Διαισθητικά, αν το δοσμένο πρόγραμμα μπορεί να παράγει το σύνολο S (ίσως με πολλούς τρόπους), τότε η ιδιότητα του μη περιορισμού, δηλώνει τι πρέπει να τηρηθεί κατά τη διάρκεια δημιουργίας των στοιχείων του S. Ειδικότερα, δηλώνει πως μπορείς να πάρεις οποιαδήποτε άτομα επιλογής θέλεις, αρκεί αυτά να μην είναι άτομα επιλογής μέσα από το S αλλά ούτε και άτομα επιλογής που εξαρτώνται από άτομα του S. Στην απόδειξη της ειδικότερης κλάσης 4.2.1, θα δούμε πως η ιδιότητα αυτή μπορεί να αντικατασταθεί από την ιδιότητα της *ελαχιστότητας (minimal)*, όπου μπορούμε να πάρουμε μόνο άτομα επιλογής που χρειάζονται για την παραγωγή των ατόμων του S. Προφανώς, η δεύτερη ιδιότητα είναι πιο αυστηρή σε σχέση με τη πρώτη.

Πριν προχωρήσω στην απόδειξη της ορθότητας του αλγορίθμου, χρειάζεται να αποδείξουμε τα Λήμματα 3.7.1 και 3.7.2.

Το πρώτο λήμμα που ακολουθεί, λέει πως μέσα ένα δοσμένο περιβάλλον L δεν είναι δυνατόν το $p_i^+(L, b)$ να εξαρτάται από το $p^-(L, a)$ (δηλαδή δοσμένης μιας προσέγγισης I, δεν υπάρχει σύνολο παραγωγής S για το άτομο $p_i^+(L, b)$ στην I τέτοιο ώστε $p^-(L, a) \in S$). Διαισθητικά αυτό σημαίνει ότι το όρισμα εξόδου του p εισάγεται σε κάποια προσέγγιση μιας οριακής ερμηνείας αργότερα από ότι εισήχθησαν όλα τα ορίσματα εισόδου του p (σε ένα δοσμένο περιβάλλον).

Λήμμα 3.7.1 *Έστω P ένα απλό fm-Datalog πρόγραμμα, D μία βάση δεδομένων και P^* το Choice Datalog$_{nS}$ πρόγραμμα που προκύπτει από το μετασχηματισμό. Για όλα τα κατηγορήματα p όπου ορίζονται στο P_D, όλες τις $L \in List(\mathcal{N})$, όλα τα $a, b \in U_{P_D}$, όλες τις θέσεις εισόδου i του p, και όλες τις προσεγγίσεις I στο P_D^*, το $p_i^+(L, b)$ δεν εξαρτάται από το $p^-(L, a)$ στην I.*

Απόδειξη 3.7.1 *Όταν η λίστα L είναι άδεια είναι εύκολο να ελέγξουμε πως το λήμμα ισχύει (καθώς δεν υπάρχει πρόταση στο P^*, εκτός από τις μοναδιαίες προτάσεις που προκύπτουν από το μετασχηματισμό του στόχου, που έχουν σαν στιγμιότυπα τα άτομα $p_i^+([\,], b)$). Για το λόγο αυτό χρειάζεται να εξετάσουμε την περίπτωση που το μήκος της λίστας L είναι μεγαλύτερο ή ίσο από 1. Υποθέτουμε ότι $L = [l_1|M]$. Η απόδειξη είναι με επαγωγή στην απόσταση ανάμεσα στα άτομα της I. Οι περιπτώσεις για 0 και 1 ισχύουν τετριμμένα (η πρώτη ισχύει λόγω των Ορισμών 3.7.2 και 3.7.1 και η δεύτερη εξαιτίας της μορφής των προτάσεων που παράγονται από τον μετασχηματισμό). Υποθέτουμε ότι το λήμμα ισχύει για αποστάσεις μέχρι k. Θα δείξουμε ότι ο ισχυρισμός ισχύει για αποστάσεις $k+1$.*

Υποθέτουμε, για χάρη της αντίφασης, ότι υπάρχει μία προσέγγιση I στην οποία το $p_i^+([l_1|M], b)$ εξαρτάται από το $p^-([l_1|M], a)$ στην I με απόσταση $k+1$. Αυτό σημαίνει ότι υπάρχει ένα σύνολο παραγωγής S για το $p_i^+([l_1|M], b)$ στην I για την οποία υπάρχει μία ακολουθία $k+1$ αποτιμημένων στιγμιοτύπων των προτάσεων του P_D^* που δημιουργεί την εξάρτηση ανάμεσα στα δύο άτομα. Η πρώτη πρόταση σε αυτή την ακολουθία πρέπει να περιλαμβάνει το $p^-([l_1|M], a)$ σαν άτομο στο σώμα του, ενώ η τελευταία πρόταση πρέπει να έχει το $p_i^+([l_1|M], b)$ σαν κεφαλή. Διακρίνουμε τις ακόλουθες περιπτώσεις:
Περίπτωση 1: *Η πρώτη πρόταση στην ακολουθία είναι της μορφής:*

$$q^-(M, a) \leftarrow \ p^-([l_1|M], a).$$

Αυτό είναι ένα αποτιμημένο στιγμιότυπο μιας πρότασης που έχει προκύψει μετά από τον μετασχηματισμό μιας πρότασης του αρχικού προγράμματος που είναι είτε της μορφής:

$$q(\cdots) \leftarrow r(\cdots), p(\cdots)$$

ή της μορφής:

$$q(\cdots) \leftarrow p(\cdots)$$

Τώρα, προκειμένου να πάμε από το $q^-(M, a)$ στο $p_i^+([l_1|M], b)$, η ετικέτα l_1 πρέπει να αποκατασταθεί από μια ενδιάμεση πρόταση της ακολουθίας. Αυτό μπορεί να επιτευχθεί είτε με ένα αποτιμημένο στιγμιότυπο μιας πρότασης της μορφής:

$$p_i^+([l_1|M], b) \leftarrow [\#] q_j^+(M, b).$$

είτε της μορφής:

$$p_i^+([l_1|M], b) \leftarrow \ r^-([l_2|M], b).$$

Αλλά, στην πρώτη περίπτωση αυτό συνεπάγεται ότι το $q_j^+(M, b)$ εξαρτάται από το $q^-(M, a)$ στην I και η απόσταση ανάμεσα στο $q_j^+(M, b)$ και στο $q^-(M, a)$ είναι $\leq k$ (αντίφαση εξαιτίας της επαγωγικής υπόθεσης). Στην δεύτερη περίπτωση, ο μόνος τρόπος που η ετικέτα l_2 μπορεί να έχει εισαχθεί είναι από ένα αποτιμημένο στιγμιότυπο μιας πρότασης της μορφής:

$$r_n^+([l_2|M], d) \leftarrow [\#] q_m^+(M, d).$$

Αλλά, τότε το $q_m^+(M, d)$ εξαρτάται από το $q^-(M, a)$ στην I και η απόσταση ανάμεσα στο $q_m^+(M, d)$ και στο $q^-(M, a)$ είναι $\leq k$ (αντίφαση λόγω της επαγωγικής υπόθεσης).
Περίπτωση 2: *Η πρώτη πρόταση στην ακολουθία είναι της μορφής:*

$$q_j^+([l_2|M], a) \leftarrow \ p^-([l_1|M], a).$$

Αυτό είναι ένα αποτιμημένο στιγμιότυπο μιας πρότασης που έχει προκύψει μετά από το μετασχηματισμό μιας πρότασης του αρχικού προγράμματος που είναι της μορφής:

$$r(\cdots) \leftarrow p(\cdots), q(\cdots)$$

Τώρα, προκειμένου να πάμε από το $q_j^+([l_2|M], a)$ στο $p_i^+([l_1|M], b)$, η ετικέτα l_2 πρέπει να απομακρυνθεί από μία ενδιάμεση πρόταση της ακολουθίας. Αυτό μπορεί να γίνει μόνο με ένα αποτιμημένο στιγμιότυπο μιας πρότασης της μορφής:

$$r^-(M, c) \leftarrow\ q^-([l_2|M], c).$$

Αλλά για να αποκαταστήσουμε την ετικέτα l_1 θα πρέπει στην τελευταία πρόταση να έχουμε ένα αποτιμημένο στιγμιότυπο μιας πρότασης της μορφής:

$$p_i^+([l_1|M], b) \leftarrow [\#]r_n^+(M, b).$$

Τότε το $r_n^+(M, b)$ εξαρτάται από το $r^-(M, c)$ στην I και η απόσταση ανάμεσα τους είναι $\leq k$ (αντίφαση εξαιτίας της επαγωγικής υπόθεσης).

Το επόμενο λήμμα δηλώνει πως κάτω από μια δοσμένη λίστα L δεν είναι δυνατόν το $p_i^+(L, b)$ να εξαρτάται από το $p_j^+(L, a)$ (όπου $p_i^+(L, b) \neq p_j^+(L, a)$). Διαισθητικά, αυτό σημαίνει ότι οι παραγωγές των ορισμάτων εισόδου του p (κάτω από ένα δοσμένο περιβάλλον) είναι ανεξάρτητες. Το λήμμα αυτό χρησιμοποιεί το προηγούμενο λήμμα και είναι ιδιαίτερα σημαντικό για να αποδειχθεί η ορθότητα του μετασχηματισμού.

Λήμμα 3.7.2 *Έστω P ένα απλό fm-Datalog πρόγραμμα και D μία βάση δεδομένων. Έστω P^* το Choice Datalog$_{nS}$ πρόγραμμα που προκύπτει από τον μετασχηματισμό. Τότε για όλα τα κατηγορήματα p που ορίζονται στο P_D, όλες τις $L \in List(\mathcal{N})$, όλα τα $a, b \in U_{P_D}$ και όλες τις θέσεις εισόδου i, j του p, δεν υπάρχει καμία προσέγγιση I σε καμία οριακή ερμηνεία του P_D^* τέτοια ώστε το $p_i^+(L, b)$ να εξαρτάται από το $p_j^+(L, a)$, όπου $p_i^+(L, b) \neq p_j^+(L, a)$.*

Απόδειξη 3.7.2 *Όταν η λίστα L είναι άδεια είναι εύκολο να ελέγξουμε πως το λήμμα ισχύει (καθώς δεν υπάρχει πρόταση στο P^*, εκτός από τις μοναδιαίες προτάσεις που προκύπτουν από το μετασχηματισμό του στόχου, που έχουν σαν στιγμιότυπα τα άτομα $p_i^+([\,], b)$). Για το λόγο αυτό χρειάζεται να εξετάσουμε την περίπτωση που το μήκος της λίστας L είναι μεγαλύτερο ή ίσο από 1. Υποθέτουμε ότι $L = [l_1|M]$. Η απόδειξη είναι με επαγωγή στην απόσταση ανάμεσα στα άτομα της I. Οι περιπτώσεις για 0 και 1 ισχύουν τετριμμένα (η πρώτη ισχύει λόγω των Ορισμών 3.7.2 και 3.7.1 και η δεύτερη εξαιτίας της μορφής των προτάσεων που παράγονται από τον μετασχηματισμό). Υποθέτουμε ότι το λήμμα ισχύει για αποστάσεις μέχρι k. Θα δείξουμε ότι ο ισχυρισμός ισχύει για αποστάσεις $k+1$.*

Υποθέτουμε, για χάρη της αντίφασης, ότι υπάρχει μία προσέγγιση I τέτοια ώστε το $p_i^+([l_1|M], b)$ να εξαρτάται από το $p_j^+([l_1|M], a)$ στην I με απόσταση $k+1$. Αυτό σημαίνει ότι υπάρχει μια ακολουθία από $k + 1$ αποτιμημένα στιγμιότυπα προτάσεων του P_D^* που δημιουργούν την εξάρτηση ανάμεσα στα δύο άτομα. Η πρώτη πρόταση σε αυτή την ακολουθία πρέπει να περιλαμβάνει το $p_j^+([l_1|M], a)$ σαν ένα άτομο στο σώμα, ενώ η τελευταία πρόταση πρέπει να περιλαμβάνει το $p_i^+([l_1|M], b)$ ως κεφαλή. Διακρίνουμε τις ακόλουθες περιπτώσεις:
<u>Περίπτωση 1:</u> *Η τελευταία πρόταση στην ακολουθία είναι της μορφής:*

$$p_i^+([l_1|M], b) \leftarrow [\#]q_k^+(M, b).$$

Αυτή είναι ένα αποτιμημένο στιγμιότυπο μιας πρότασης που έχει προκύψει μετά από τον μετασχηματισμό μιας πρότασης του αρχικού προγράμματος που είναι είτε της μορφής:

$$q(\cdots) \leftarrow p(\cdots).$$

ή της μορφής:

$$q(\cdots) \leftarrow r(\cdots), p(\cdots).$$

ή:

$$q(\cdots) \leftarrow p(\cdots), r(\cdots).$$

Τώρα, προκειμένου να πάμε από το $p_j^+([l_1|M], a)$ *στο* $q_k^+(M, b)$*, η ετικέτα* l_1 *πρέπει να διαγραφεί από μία ενδιάμεση πρόταση της ακολουθίας. Ένας τρόπος που αυτό μπορεί έχει γίνει είναι χρησιμοποιώντας ένα αποτιμημένο στιγμιότυπο μιας πρότασης της μορφής:*

$$q^-(M, c) \leftarrow\ p^-([l_1|M], c).$$

Αλλά, τότε το $q_k^+(M, b)$ *θα εξαρτιόταν από το* $q^-(M, c)$ *στην* I *όπως επίσης και το* $p_i^+([l_1|M], b)$ *θα εξαρτιόταν από το* $p^-([l_1|M], c)$ *(αντίφαση από το Λήμμα 3.7.1).*

Η διαγραφή της ετικέτας l_1 *μπορεί εναλλακτικά να έχει γίνει χρησιμοποιώντας ένα αποτιμημένο στιγμιότυπο μιας πρότασης της μορφής:*

$$r_m^+([l_2|M], c) \leftarrow\ p^-([l_1|M], c).$$

Αλλά τότε το $p_i^+([l_1|M], b)$ *θα εξαρτιόταν από το* $p^-([l_1|M], c)$ *στην* I *(αντίφαση λόγω Λήμματος 3.7.1).*

Περίπτωση 2: *Η τελευταία πρόταση στην ακολουθία είναι της μορφής:*

$$p_i^+([l_1|M], b) \leftarrow\ q^-([l_2|M], b).$$

Αυτή είναι ένα αποτιμημένο στιγμιότυπο μιας πρότασης που έχει προκύψει μετά τον μετασχηματισμό μιας πρότασης του αρχικού προγράμματος που είναι της μορφής:

$$r(\cdots) \leftarrow q(\cdots), p(\cdots).$$

Αλλά τότε, έπρεπε να έχει υπάρξει κάποιο ενδιάμεσο βήμα σε όλη την διαδικασία παραγωγής που απομάκρυνε την ετικέτα l_1 *από της αρχική λίστα* $[l_1|M]$ *του ατόμου* $p_j^+([l_1|M], a)$*. Αυτό μπορεί μόνο να έχει γίνει χρησιμοποιώντας ένα αποτιμημένο στιγμιότυπο μιας πρότασης της μορφής:*

$$r^-(M, c) \leftarrow\ p^-([l_1|M], c).$$

Σε αυτή την περίπτωση, το $p_i^+([l_1|M], b)$ *θα εξαρτάται από το* $p^-([l_1|M], c)$ *στην* I *(αντίφαση λόγω του Λήμματος 3.7.1).*

Το παραπάνω λήμμα είναι σημαντικό στο να αποδείξουμε την ορθότητα του αλγορίθμου μετασχηματισμού. Πράγματι, αν ήταν δυνατόν το $p_i^+(L, b)$ να εξαρτάται από το $p_i^+(L, a)$ τότε μπορούμε να σκεφτούμε ένα σενάριο στο οποίο σε όλες τις προσεγγίσεις το $\#p_i^+(L, a)$ θα χρειαζόταν να παραχθεί πριν από την παραγωγή του

$p_i^+(L,b)$. Αλλά τότε, δεν θα ήταν ποτέ δυνατό να παραχθεί το $\#p_i^+(L,b)$ και σαν αποτέλεσμα μπορεί να χάναμε συγκεκριμένες λύσεις του αρχικού προγράμματος. Με απλά λόγια, το γεγονός πως το παραπάνω λήμμα ισχύει, είναι σημαντικό στο να αποδείξουμε την ολική ορθότητα του αλγορίθμου μετασχηματισμού.

Ο ακόλουθος ορισμός θα χρησιμοποιηθεί για να απλοποιήσουμε τους συμβολισμούς που εμφανίζονται στις αποδείξεις.

Ορισμός 3.7.5 *Έστω C μια απλή fm-πρόταση και p, q δύο κατηγορήματα που εμφανίζονται στην C. Τότε, το $IN_{p^+q^+}$ είναι ένα σύνολο από θέσεις μεταβλητών εισόδου του p που είναι επίσης και μεταβλητές εισόδου του q. Επιπλέον, το $f_{p^+q^+} : IN_{p^+q^+} \rightarrow IN_{q^+p^+}$ είναι η συνάρτηση που παίρνει τη θέση μιας μεταβλητής εισόδου που εμφανίζεται στο κατηγόρημα p και επιστρέφει τη θέση στο κατηγόρημα q.*

Παράδειγμα 3.7.1 *Θεωρείστε την ακόλουθη fm-πρόταση:*

$\mathtt{p}(\overset{+}{\mathtt{X}},\overset{+}{\mathtt{Y}},\overset{-}{\mathtt{W}}) \leftarrow \mathtt{q}(\overset{+}{\mathtt{X}},\overset{-}{\mathtt{Z}}),\mathtt{r}(\overset{+}{\mathtt{Y}},\overset{+}{\mathtt{X}},\overset{+}{\mathtt{Z}},\overset{-}{\mathtt{W}})$.

Σε αυτή τη πρόταση, $IN_{\mathtt{p^+q^+}} = \{1\}$, $S_{\mathtt{q^+r^+}} = \{1\}$, $S_{\mathtt{r^+q^+}} = \{2\}$, και $f_{\mathtt{q^+r^+}}(2) = 2$.

Μπορούμε σε αυτό το σημείο να προχωρήσουμε στην απόδειξη των κύριων λημμάτων που αφορούν την ορθότητα του αλγορίθμου.

Λήμμα 3.7.3 *Έστω P ένα απλό fm-Datalog πρόγραμμα και D μια βάση δεδομένων. Έστω P^* το Choice Datalog$_{nS}$ πρόγραμμα που προκύπτει από το μετασχηματισμό. Για όλα τα κατηγορήματα p που ορίζονται στο P_D, όλες τις $L \in List(\mathcal{N})$ και όλες τις σταθερές $a_1,\ldots,a_n,b \in U_{P_D}$, αν $p(a_1,\ldots,a_n,b) \in T_{P_D} \uparrow \omega$ και υπάρχει μια προσέγγιση I σε μια οριακή ερμηνεία του P_D^* που είναι μη περιοριστική ως προς το $S = \{p_1^+(L,a_1),\ \ldots,\ p_n^+(L,a_n)\}$, τότε υπάρχει μια προσέγγιση $J \supseteq I$, τέτοια ώστε $p^-(L,b) \in J$ και το $p^-(L,b)$ εξαρτάται από κάθε άτομο $A \in S$ και από κάθε άτομο $B \in choices(J) - choices(I)$.*

Απόδειξη 3.7.3 *Θα δείξουμε το παραπάνω λήμμα με επαγωγή στις προσεγγίσεις του $T_{P_D} \uparrow \omega$.*

<u>Επαγωγική Βάση:</u> *Η επαγωγική βάση ισχύει προφανώς διότι $T_{P_D} \uparrow 0 = \emptyset$ και επομένως το $p(a_1,\ldots,a_n,b) \in T_{P_D} \uparrow 0$ είναι ψευδές.*

<u>Επαγωγική Υπόθεση:</u> *Υποθέτουμε ότι: για όλα τα κατηγορήματα p που ορίζονται στο P_D, όλες τις $L \in List(\mathcal{N})$ και όλες τις σταθερές $a_1,\ldots,a_n,b \in U_{P_D}$, αν $p(a_1,\ldots,a_n,b) \in T_{P_D} \uparrow k$ και υπάρχει μια προσέγγιση I σε μια οριακή ερμηνεία του P_D^* που είναι μη περιοριστική ως προς το $S = \{p_1^+(L,a_1),\ \ldots,\ p_n^+(L,a_n)\}$, τότε υπάρχει μια προσέγγιση $J \supseteq I$, τέτοια ώστε $p^-(L,b) \in J$ και το $p^-(L,b)$ εξαρτάται από κάθε άτομο $A \in S$ και κάθε άτομο $B \in choices(J) - choices(I)$.*

<u>Επαγωγικό Βήμα:</u> *Θα δείξουμε το επιθυμητό αποτέλεσμα για την $k+1$ επανάληψη του τελεστή T_{P_D}. Για αυτό θα αποδείξουμε ότι: αν $p(a_1,\ldots,a_n,b) \in T_{P_D} \uparrow (k+1)$ και υπάρχει μια προσέγγιση I σε μια οριακή ερμηνεία του P_D^* που είναι μη περιοριστική ως προς το $S = \{p_1^+(L,a_1),\ \ldots,\ p_n^+(L,a_n)\}$, τότε υπάρχει μια προσέγγιση $J \supseteq I$, τέτοια ώστε το $p^-(L,b) \in J$ και το $p^-(L,b)$ να εξαρτάται από κάθε άτομο $A \in S$ και κάθε άτομο $B \in choices(J) - choices(I)$.*

Θα χρησιμοποιήσουμε μια ανάλυση περιπτώσεων ως προς τον τρόπο που εισάγεται το $p(a_1,\ldots,a_n,b)$ στο $T_{P_D} \uparrow (k+1)$.

Περίπτωση 1: *Υποθέτουμε ότι το $p(a_1, \ldots, a_n, b)$ έχει εισαχθεί στο $T_{P_D} \uparrow (k+1)$ επειδή είναι ένα γεγονός στη βάση δεδομένων D. Με βάση το μετασχηματισμό, στο P^* υπάρχει μια πρόταση της μορφής:*

$$p^-(L,Y) \leftarrow p(X_1, \ldots, X_n, Y), p_1^+(L, X_1), \ldots, p_n^+(L, X_n).$$

Σημειώστε πως δεν υπάρχει άλλη πρόταση στο P^* που ορίζει το κατηγόρημα p^-. Επομένως, αφού $S = \{p_1^+(L,a_1), \ldots, p_n^+(L,a_n)\} \subseteq I$, τότε έχουμε $p^-(L,b) \in I$. Επιπλέον, το $p^-(L,b)$ εξαρτάται από κάθε άτομο $A \in S$ (εξαρτάται από κάθε $p_i^+(L,a_i)$) και κάθε $B \in choices(I) - choices(I) = \emptyset$.

Περίπτωση 2: *Υποθέτουμε ότι το $p(a_1, \ldots, a_n, b)$ έχει προστεθεί στο $T_{P_D} \uparrow (k+1)$ χρησιμοποιώντας μια πρόταση της μορφής:*

$$p(\cdots) \leftarrow q(\cdots), r(\cdots). \tag{3.1}$$

και υποθέτουμε ότι οι πληθικότητες των κατηγορήματων p, q και r είναι $n+1$, $u+1$ και $m+1$ αντίστοιχα. Τότε υπάρχει μια σταθερά c που ανατίθεται στη μεταβλητή εξόδου του $q(\cdots)$, τέτοια ώστε τα αντίστοιχα στιγμιότυπα των $q(\cdots)$ και $r(\cdots)$ στην πρόταση (3.1) να είναι στο $T_{P_D} \uparrow k$.

Θεωρούμε ένα όρισμα εισόδου του q (έστω το t-οστό) που μοιράζεται τη μεταβλητή του με το p (έστω το s-οστό). Τότε μια πρόταση της μορφής:

$$q_t^+([l_1|L], X) \leftarrow [\#]p_s^+(L, X). \tag{3.2}$$

όπου $s = f_{q^+p^+}(t)$, έχει προστεθεί στο P^* με τον αλγόριθμο μετασχηματισμού. Επιπλέον, για κάθε όρισμα εισόδου του r που μοιράζεται την μεταβλητή του με ένα όρισμα εισόδου του p, έχουμε στο P^* μια πρόταση της μορφής:

$$r_t^+([l_2|L], X) \leftarrow [\#]p_s^+(L, X). \tag{3.3}$$

όπου $s = f_{r^+p^+}(t)$. Ξεκινάμε από την I και εναλλάσσουμε το $C_{P_D^*}$ και το $NT_{P_D^*} \uparrow \omega^2$ μέχρι να φτάσουμε σε μια προσέγγιση I' που περιλαμβάνει τις εκδοχές με επιλογή των ατόμων του S (μόνο άτομα επιλογής της μορφής $\#p_i^+(L,a_i)$) που χρειάζονται για την παραγωγή των q_t^+ με λίστα $[l_1|L]$ από προτάσεις της μορφής (3.2) και r_t^+ με λίστα $[l_2|L]$ από προτάσεις της μορφής (3.3). Σημειώστε ότι αυτό μπορεί να γίνει επειδή κανένα από τα άτομα της μορφής $p_s^+(L, ...)$ δεν μπορεί να έχει την εκδοχή του με επιλογή στην I εξαιτίας του δεύτερου περιορισμού από την ιδιότητα του μη περιορισμού της I ως προς το S.

Χρησιμοποιώντας προτάσεις της μορφής (3.2), βλέπουμε ότι $q_t^+([l_1|L], a_s) \in I'$, για όλα τα $t = 1, \ldots, u$ and $s = f_{q^+p^+}(t)$. Επιπλέον, από προτάσεις της μορφής (3.3), παίρνουμε $r_t^+([l_2|L], a_s) \in I'$, για όλα τα $t \in IN_{r^+p^+}$ και $s = f_{r^+p^+}(t)$.

Τώρα, χρειάζεται να δείξουμε ότι η προσέγγιση I' είναι μη περιοριστική ως προς το $S' = \{q_1^+([l_1|L], a_{f_{q^+p^+}(1)}), \ldots, q_u^+([l_1|L], a_{f_{q^+p^+}(u)})\}$. Πράγματι αυτό ισχύει:

1. *Κανένα άτομο επιλογής της I' δεν εξαρτάται από το S'.*

[2] Η περίπτωση να ξεκινήσουμε τις εναλλαγές πρώτα από το $NT_{P_D^*} \uparrow \omega$ και μετά το C_{P_D} (δηλ. το I να έχει προκύψει από μια εφαρμογή ενός C_{P_D} τελεστή πάνω σε μια προσέγγιση) είναι ίδια.

2. *Κανένα άτομο της μορφής* $\#q_t^+([l_1|L], d)$, *για οποιοδήποτε* $d \in U_{P_D}$, *δεν ανήκει στην* I'.

Για να αποδείξουμε ότι η πρώτη συνθήκη ισχύει, υποθέτουμε ότι υπάρχει ένα άτομο επιλογής $A \in I'$, *τέτοιο ώστε το* A *να εξαρτάται από το* $q^+_{f_{p^+q^+}(j)}([l_1|L], a_j) \in S'$. *Υπάρχουν δύο περιπτώσεις:*
Υποπερίπτωση 1: *Αν* $A \in I'$ *και* $A \notin I$ *τότε το* A *είναι της μορφής* $\#p_i^+(L, a_i)$ *(εξαιτίας του περιορισμού που κάναμε σχετικά με την εναλλαγή των* $C_{P_D^*}$ *και* $NT_{P_D^*} \uparrow \omega$*). Αλλά αυτό σημαίνει πως το* $\#p_i^+(L, a_i)$ *εξαρτάται από το* $q^+_{f_{p^+q^+}(j)}([l_1|L], a_j)$ *το οποίο από την μορφή της πρότασης (3.2) εξαρτάται από το* $p_j^+(L, a_j)$. *Με άλλα λόγια, προκύπτει ότι κάποιο* $p_i^+(L, a_i)$ *εξαρτάται από κάποιο* $p_j^+(L, a_j)$, *πράγμα που δεν μπορεί να ισχύει από το Λήμμα 3.7.2 (σε περίπτωση που* $i = j$, *τότε το άτομο* $p_i^+(L, a_i)$ *εξαρτάται από τον εαυτό του μέσω κάποιας πρότασης, που δεν μπορεί να ισχύει εξαιτίας της μορφής των προτάσεων και από το γεγονός ότι ένα άτομο* A *μπορεί να εξαρτάται από τον εαυτό του μόνο με απόσταση 0).*
Υποπερίπτωση 2: *Το* $A \in I$. *Αλλά από τις προτάσεις της μορφής (3.2), το* $q^+_{f_{p^+q^+}(j)}([l_1|L], a_j)$ *εξαρτάται από το* $p_j^+(L, a_j)$. *Έτσι το άτομο επιλογής* A *θα εξαρτάται από κάποιο μέλος του* S, *που δεν μπορεί να ισχύει εξαιτίας της υπόθεσης ότι η* I *είναι μη περιοριστική ως προς το* S *(πρώτη συνθήκη).*

Επομένως, σε κάθε περίπτωση έχουμε αντίφαση.

Προκειμένου να αποδείξουμε τη δεύτερη συνθήκη του μη περιορισμού της I' *ως προς το* S', *πρέπει να πούμε ότι* I *είναι μη περιοριστική ως προς το* S, *επομένως δεν υπάρχουν άτομα επιλογής της μορφής* $\#p_i^+(L, a_i)$ *στην* I. *Επιπλέον, η μόνη πρόταση που μπορεί να ορίζει το* $q_j^+([l_1|L], a_j)$ *είναι της μορφής (3.2). Έτσι, εξαιτίας της υπόθεσης που κάνει για τα άτομα επιλογής που εισάγονται κατά την εναλλαγή των* $C_{P_D^*}$ *και* $NT_{P_D^*} \uparrow \omega$ *έχουμε ότι δεν υπάρχουν άτομα της μορφής* $\#q_j^+([l_1|L], a_j)$ *στην* I' *(δεν υπήρχαν ήδη στην* I *και δεν προστέθηκαν από την* I *στην* I'*). Πιο αναλυτικά, υποθέτουμε ότι υπάρχει ένα* $\#q_t^+([l_1|L], d) \in I'$. *Υπάρχουν δύο περιπτώσεις:*
Υποπερίπτωση 1: *Το* $\#q_t^+([l_1|L], d) \in I'$ *και* $\#q_t^+([l_1|L], d) \notin I$. *Αλλά αυτό δεν μπορεί να ισχύει εξαιτίας του τρόπου που πήραμε τα άτομα επιλογής από την* I *στην* I' *(μόνο της μορφής* $\#p_i^+(L, a_i)$*).*
Υποπερίπτωση 2: *Το* $\#q_t^+([l_1|L], d) \in I$. *Αλλά τότε αυτό άτομο επιλογής θα εξαρτιόταν από κάποιο* $p_i^+(L, a_i) \in S$, *αφού προτάσεις της μορφής (3.2) είναι οι μόνες προτάσεις που ορίζουν το* $q_t^+([l_1|L], X)$. *Αυτό έρχεται σε αντίφαση με τη πρώτη συνθήκη της ιδιότητας της μη περιοριστικότητας της* I *ως προς το* S.

Για αυτό μπορούμε να εφαρμόσουμε την επαγωγική υπόθεση στο q *με λίστα* $[l_1|L]$ *και στην προσέγγιση* I', *που δίνει:*

> *Αφού η προσέγγιση* I' *είναι μη προσεγγιστική ως προς το*
>
> $S' = \{q_1^+([l_1|L], a_{f_{q^+p^+}(1)}), \ldots, q_u^+([l_1|L], a_{f_{q^+p^+}(u)})\}$ *και*
>
> $q(a_{f_{q^+p^+}(1)}, \ldots, a_{f_{q^+p^+}(u)}, c) \in T_{P_D} \uparrow k$, *τότε υπάρχει μια προσέγγιση* $J' \supseteq I'$, *τέτοια ώστε* $q^-([l_1|L], c) \in J'$ *και το* $q^-([l_1|L], c)$ *εξαρτάται από κάθε άτομο* $A \in S'$ *και από κάθε άτομο* $B \in choices(J') - choices(I')$.

Αφού για όλα τα ορίσματα εισόδου του r που μοιράζονται τη μεταβλητή τους με το όρισμα εξόδου του q υπάρχει μια πρόταση στο P^* της μορφής:

$$r_w^+([l_2|L], Y) \leftarrow q^-([l_1|L], Y). \tag{3.4}$$

προκύπτει ότι $r_w^+([l_2|L], c) \in J'$.

Παρατηρούμε επίσης ότι η J' είναι μη περιοριστική ως προς το $S'' = \{r_1^+([l_2|L], a_{f_{r^+p^+}(1)}), \ldots, r_m^+([l_2|L], a_{f_{r^+p^+}(m)})\}$. *Αυτό ισχύει διότι:*

1. *Κανένα άτομο επιλογής της J' δεν εξαρτάται από κανένα μέλος του S''.*

2. *Κανένα άτομο της μορφής $\#r_s^+([l_2|L], d)$, για οποιοδήποτε $d \in U_{P_D}$, δεν ανήκει στο J'.*

Προκειμένου να αποδείξουμε την πρώτη συνθήκη υποθέτουμε ότι υπάρχει ένα άτομο επιλογής $A \in J'$ που εξαρτάται από κάποιο $r^+_{f_{p^+r^+}(j)}([l_2|L], a_j) \in S''$. Τότε υπάρχουν τρεις περιπτώσεις:
<u>Υποπερίπτωση 1:</u> *Αν $A \in J'$ και $A \notin I'$, τότε το $A \in choices(J') - choices(I')$. Αλλά το $r_w^+([l_2|L], c)$ εξαρτάται από το $q^-([l_1|L], c)$ που εξαρτάται από το A (από την επαγωγική υπόθεση για το q). Αλλά το A εξαρτάται από το $r^+_{f_{p^+r^+}(j)}([l_2|L], a_j)$, έτσι το $r_w^+([l_2|L], c)$ εξαρτάται από το $r^+_{f_{p^+r^+}(j)}([l_2|L], a_j)$ (αντίφαση από το Λήμμα 3.7.2).*
<u>Υποπερίπτωση 2:</u> *Αν $A \in I'$ και $A \notin I$, τότε το A είναι της μορφής $\#p_i^+(L, a_i)$. Αυτό σημαίνει ότι το $p_i^+(L, a_i)$ εξαρτάται από το $r^+_{f_{p^+r^+}(j)}([l_2|L], a_j)$, που είτε εξαρτάται από το $p_j^+(L, a_j)$ (αντίφαση από το Λήμμα 3.7.2) είτε από το $q^-([l_1|L], c)$. Αλλά από την επαγωγική υπόθεση για το q, το $q^-([l_1|L], c)$ εξαρτάται από κάθε άτομο $A \in S'$. Έτσι, από προτάσεις της μορφής (3.2), κάθε $A \in S'$ εξαρτάται από κάποιο $p_j^+(L, a_j)$. Για αυτό, το $p_i^+(L, a_i)$ εξαρτάται από το $p_j^+(L, a_j)$ (αντίφαση από το Λήμμα 3.7.2).*
<u>Υποπερίπτωση 3:</u> *Αν $A \in I$, τότε η I δεν θα ήταν μη περιοριστική ως προς το S (πρώτη συνθήκη), επειδή το $r^+_{f_{p^+r^+}(j)}([l_2|L], a_j)$ εξαρτάται από το $p_i^+(L, a_i)$ απευθείας από προτάσεις της μορφής (3.3) είτε μέσω του $q^-([l_1|L], c)$ από την πρόταση (3.4).*

Έτσι σε όλες τις περιπτώσεις έχουμε αντίφαση.

Για να αποδείξουμε την δεύτερη συνθήκη της μη περιοριστικότητας, υποθέτουμε ότι υπάρχει ένα άτομο επιλογής της μορφής $\#r_s^+([l_2|L], d)$ που ανήκει στην J'. Υπάρχουν τρεις περιπτώσεις:
<u>Υποπερίπτωση 1:</u> *Αν $\#r_s^+([l_2|L], d) \in I$, τότε η I δεν θα ήταν μη περιοριστική ως προς το S (πρώτη συνθήκη), επειδή το $r_s^+([l_2|L], d)$ εξαρτάται από το $p_i^+(L, a_i)$ ή από το $q^-([l_1|L], c)$ (στην πραγματικότητα προτάσεις της μορφής (3.3) και (3.4) είναι οι μόνες προτάσεις που ορίζουν το $r_s^+([l_2|L], X)$).*
<u>Υποπερίπτωση 2:</u> *Αν το $\#r_s^+([l_2|L], d) \in I'$ και $\#r_s^+([l_2|L], d) \notin I$. Δεν μπορεί να ισχύει αυτό, διότι, πήραμε μόνο άτομα της μορφής $\#p_i^+(L, a_i)$.*
<u>Υποπερίπτωση 3:</u> *Αν το $\#r_s^+([l_2|L], d) \in J'$ και $\#r_s^+([l_2|L], d) \notin I'$ (δηλ. $\#r_s^+([l_2|L], d) \in choices(J') - choices(I')$), τότε το $q^-([l_1|L], c)$ εξαρτάται από το $\#r_s^+([l_2|L], d)$*

(από την επαγωγική υπόθεση του q). Αλλά, το $r_w^+([l_2|L], c)$ εξαρτάται από το $q^-([l_1|L], c)$ από την πρόταση (3.4) που εξαρτάται από το $r_s^+([l_2|L], d)$ (αντίφαση από το Λήμμα 3.7.2).

Έτσι σε όλες τις περιπτώσεις έχουμε αντίφαση.

Επομένως μπορούμε να εφαρμόσουμε την επαγωγική υπόθεση στο r στη λίστα $[l_2|L]$ και στη προσέγγιση J', που μας δίνει:

> *Αφού η J' περιλαμβάνει το σύνολο των ατόμων του S'' που αντιστοιχούν στα ορίσματα εισόδου του r κάτω από το περιβάλλον $[l_2|L]$, η J' είναι μη περιοριστική ως προς το S'' και το $r(a_{r_1}, \ldots, a_{r_m}, b) \in T_{P_D} \uparrow k$, τότε υπάρχει μια προσέγγιση $J \supseteq J'$, τέτοια ώστε το $r^-([l_2|L], b) \in J$ και το $r^-([l_2|L], b)$ εξαρτάται από κάθε άτομο $A \in S''$ και κάθε άτομο $B \in choices(J) - choices(J')$.*

Τέλος, χρησιμοποιώντας το γεγονός πως το $r^-([l_2|L], b) \in J$ μαζί με με τη πρόταση:

$$p^-(L, Z) \leftarrow r^-([l_2|L], Z). \tag{3.5}$$

συμπεραίνουμε ότι το $p^-(L, b) \in J$. Είναι εύκολο να δούμε ότι το $p^-(L, b)$ εξαρτάται από κάθε άτομο $A \in S$ και κάθε άτομο $B \in choices(J) - choices(I)$. Από την πρόταση (3.5) έχουμε ότι το $p^-(L, b)$ εξαρτάται από το $r^-([l_2|L], b)$. Αλλά από την επαγωγική υπόθεση για το r έχουμε ότι το $r^-([l_2|L], b)$ εξαρτάται από κάθε άτομο $A'' \in S''$ και κάθε άτομο $B'' \in choices(J) - choices(J')$. Επιπλέον, κάθε άτομο $A'' \in S''$ είτε εξαρτάται από $p_i^+(L, a_i)$ ($\in S$) είτε το $\#p_i^+(L, a_i) \in choices(I') - choices(I)$ από προτάσεις της μορφής (3.3) ή $q^-([l_1|L], c)$ από την πρόταση (3.4) που εξαρτάται από κάθε άτομο $A' \in S'$ και κάθε άτομο $B' \in choices(J') - choices(I')$ (από την επαγωγική υπόθεση στο q). Αλλά από τις προτάσεις της μορφής (3.2) κάθε άτομο $A' \in S'$ εξαρτάται από κάποιο $p_j^+(L, a_j)$ ($\in S$) ή $\#p_j^+(L, a_j) \in choices(I') - choices(I)$. Αλλά από τη τρίτη συνθήκη του Ορισμού 3.4.1 μιας πρότασης σταθερού τρόπου αυτό καλύπτει όλα τα ορίσματα του p. Έτσι, το $p^-(L, b)$ εξαρτάται από κάθε άτομο $A \in S$. Επιπλέον, το $p^-(L, b)$ εξαρτάται από κάθε άτομο $B \in choices(J) - choices(I)$.

<u>Περίπτωση 3:</u> *Υποθέτουμε ότι το $p(a_1, \ldots, a_n, b)$ έχει προστεθεί στο $T_{P_D} \uparrow (k+1)$ χρησιμοποιώντας μια πρόταση της μορφής:*

$$p(\cdots) \leftarrow q(\cdots). \tag{3.6}$$

Η απόδειξη αυτής της περίπτωσης είναι μια απλουστευμένη εκδοχή της Περίπτωσης 2.

Λήμμα 3.7.4 *Έστω P ένα απλό fm-Datalog πρόγραμμα, D μια βάση δεδομένων και $\leftarrow p(a_1, \ldots, a_n, Y)$ ένας στόχος. Έστω P^* το Choice Datalog$_{nS}$ πρόγραμμα που προκύπτει εφαρμόζοντας τον αλγόριθμο μετασχηματισμού στο $P \cup \{\leftarrow p(a_1, \ldots, a_n, Y)\}$. Για όλα τα $b \in U_{P_D}$ το ακόλουθο ισχύει: αν $p(a_1, \ldots, a_n, b) \in T_{P_D} \uparrow \omega$, τότε υπάρχει μια οριακή ερμηνεία M του P_D^* έτσι ώστε το $p^-([\,], b) \in M$.*

Απόδειξη 3.7.4 *Αφού με το μετασχηματισμό του στόχου, τα γεγονότα* $p_i^+([\,], a_i)$*, για* $i = 1, \ldots, n$*, προστίθενται στο* P^**, αυτό το λήμμα είναι μια ειδική περίπτωση του Λήμματος 3.7.3.*

Το αντίστροφο του Λήμματος 3.7.3 δίνεται παρακάτω:

Λήμμα 3.7.5 *Έστω* P *ένα απλό fm-Datalog πρόγραμμα και* D *μια βάση δεδομένων. Έστω* P^* *το Choice Datalog*$_{nS}$ *πρόγραμμα που προκύπτει από το μετασχηματισμό. Έστω* M *η οριακή ερμηνεία του* P^*_D*. Τότε για όλα τα κατηγορήματα* p *που ορίζονται στο* P_D*, για όλες τις λίστες* $L \in List(\mathcal{N})$ *και για όλα τα* $b \in U_{P_D}$*, αν το* $p^-(L, b) \in M$*, τότε υπάρχουν σταθερές* $a_1, \ldots, a_n \in U_{P_D}$ *τέτοιες ώστε το* $p(a_1, \ldots, a_n, b) \in T_{P_D} \uparrow \omega$ *και τα* $p_i^+(L, a_i) \in M$*, για* $i = 1, \ldots, n$*.*

Απόδειξη 3.7.5 *Θα το αποδείξουμε με επαγωγή στις προσεγγίσεις του* M*.*
Επαγωγική Βάση: *Η επαγωγική βάση είναι: αν το* $p^-(L, b) \in M_0$*, τότε υπάρχουν σταθερές* $a_1, \ldots, a_n \in U_{P_D}$ *τέτοιες ώστε το* $p(a_1, \ldots, a_n, b) \in T_{P_D} \uparrow \omega$ *και τα* $p_i^+(L, a_i) \in M_0$*, για* $i = 1, \ldots, n$*. Αυτή η δήλωση ισχύει προφανώς γιατί* $M_0 = \emptyset$ *έτσι το* $p^-(L, b) \in M_0$ *είναι ψευδές.*
Επαγωγική Υπόθεση: *Αν το* $p^-(L, b) \in M_k$*, τότε υπάρχουν σταθερές* $a_1, \ldots, a_n \in U_{P_D}$ *τέτοιες ώστε το* $p(a_1, \ldots, a_n, b) \in T_{P_D} \uparrow \omega$ *και τα* $p_i^+(L, a_i) \in M_k$*, για* $i = 1, \ldots, n$*.*
Επαγωγικό Βήμα: *Θα αποδείξουμε ότι αν το* $p^-(L, b) \in M_{k+1}$*, τότε υπάρχουν σταθερές* $a_1, \ldots, a_n \in U_{P_D}$ *τέτοιες ώστε το* $p(a_1, \ldots, a_n, b) \in T_{P_D} \uparrow \omega$ *και τα* $p_i^+(L, a_i) \in M_{k+1}$*, για* $i = 1, \ldots, n$*.*

Σαν πρώτη παρατήρηση, σημειώστε πως αν η M_{k+1} *αντιστοιχεί σε ένα* $C_{P^*_D}$ *βήμα στην αλυσίδα που οδηγεί στο* M*, τότε το επαγωγικό βήμα ισχύει αμέσως (εξαιτίας της επαγωγικής υπόθεσης). Για αυτό χρειάζεται να εξετάσουμε μόνο την περίπτωση που* $M_{k+1} = NT_{P^*_D} \uparrow \omega(M_k)$*. Διακρίνουμε τις παρακάτω περιπτώσεις:*
Περίπτωση 1: *Υποθέτουμε ότι το* $p^-(L, b)$ *έχει εισαχθεί στην* M_{k+1} *από μια πρόταση της μορφής:*

$$p^-(L, Y) \leftarrow p(X_1, \ldots, X_n, Y), p_1^+(L, X_1), \ldots, p_n^+(L, X_n). \tag{3.1}$$

Τότε το p *είναι ένα EDB κατηγόρημα στην βάση δεδομένων* D *και υπάρχουν σταθερές* $a_1, \ldots, a_n \in U_{P_D}$ *τέτοιες ώστε το* $p(a_1, \ldots, a_n, b) \in T_{P_D} \uparrow 1$ *και τα* $p_i^+(L, a_i) \in M_{k+1}$*, για* $i = 1, \ldots, n$*.*
Περίπτωση 2: *Υποθέτουμε ότι υπάρχει στο* P *μια πρόταση της μορφής:*

$$p(\cdots) \leftarrow q(\cdots), r(\cdots). \tag{3.2}$$

τέτοια ώστε οι πληθικότητες των κατηγορημάτων p*,* q *και* r *είναι* $n+1$*,* $u+1$ *και* $m+1$ *αντίστοιχα, και οι ετικέτες των* $q(\cdots)$ *και* $r(\cdots)$ *είναι* l_1 *και* l_2 *αντίστοιχα. Η μετάφραση της παραπάνω πρότασης δίνει σαν αποτέλεσμα προτάσεις της μορφής:*

$$p^-(L, Y) \leftarrow \; r^-([l_2|L], Y). \tag{3.3}$$
$$r_t^+([l_2|L], Y) \leftarrow \; q^-([l_1|L], Y). \tag{3.4}$$
$$r_t^+([l_2|L], Y) \leftarrow [\#]p_s^+(L, Y). \tag{3.5}$$
$$q_t^+([l_1|L], Y) \leftarrow [\#]p_s^+(L, Y). \tag{3.6}$$

στο P^* (όπου το εύρος των δεικτών t και s μπορεί εύκολα να προκύψει από την περιγραφή του μετασχηματισμού). Υποθέτουμε τώρα ότι το $p^-(L,b) \in M_{k+1}$ και ότι το $p^-(L,b)$ έχει εισαχθεί στην M_{k+1} χρησιμοποιώντας τη πρόταση (3.3) που υπάρχει παραπάνω.

Ας θυμηθούμε τώρα ότι $M_{k+1} = NT_{P_D^*} \uparrow \omega(M_k)$. Θα κάνουμε μια εσωτερική επαγωγή, δηλ. θα δείξουμε ότι: για όλα τα $\nu \geq 0$, αν το $p^-(L,b) \in NT^{\nu}_{P_D^*}(M_k)$, τότε υπάρχουν σταθερές $a_1, \ldots, a_n \in U_{P_D}$ τέτοιες ώστε το $p(a_1, \ldots, a_n, b) \in T_{P_D} \uparrow \omega$ και τα $p_i^+(L, a_i) \in NT^{\nu}_{P_D^*}(M_k)$, για $i = 1, \ldots, n$. Η επαγωγική βάση για $\nu = 0$ είναι προφανής καθώς συμπίπτει με την εξωτερική επαγωγική υπόθεση (αφού $NT^0_{P_D^*}(M_k) = M_k$). Για $\nu = 1$ επίσης ισχύει εξαιτίας της μορφής των προτάσεων που παράγονται από τον αλγόριθμο μετασχηματισμού. Ας υποθέσουμε πως η δήλωση ισχύει για $\nu > 1$. Θα δείξουμε ότι είναι έγκυρη για $\nu + 1$. Αφού το $p^-(L,b) \in NT^{\nu+1}_{P_D^*}(M_k)$ έχουμε ότι το $r^-([l_2|L], b) \in NT^{\nu}_{P_D^*}(M_k)$. Για αυτό μπορούμε να εφαρμόσουμε την εσωτερική επαγωγική υπόθεση στο r για να πάρουμε:

> *Αφού το $r^-([l_2|L], b) \in NT^{\nu}_{P_D^*}(M_k)$, τότε υπάρχουν σταθερές $c_1, \ldots, c_m \in U_{P_D}$ τέτοιες ώστε το $r(c_1, \ldots, c_m, b) \in T_{P_D} \uparrow \omega$ και τα $r_j^+([l_2|L], c_j) \in NT^{\nu}_{P_D^*}(M_k)$, για $j = 1, \ldots, m$.*

Παρατηρήστε ότι ο μόνος τρόπος που το $r_t^+([l_2|L], c_t)$, για όλα τις θέσεις ορισμάτων εισόδου t του r, μπορεί να έχει εισαχθεί στην $NT^{\nu}_{P_D^*}(M_k)$, είναι είτε χρησιμοποιώντας μια πρόταση της μορφής (3.4) είτε μια πρόταση της μορφής (3.5) παραπάνω (όλες οι άλλες προτάσεις που ορίζουν το r_t^+ έχουν διαφορετική κεφαλή και κατά συνέπεια δεν μπορούν να χρησιμοποιηθούν). Έχουμε δύο περιπτώσεις:
Υποπερίπτωση 2.1: *Αν το $r_t^+([l_2|L], c_t)$, με $t \in IN_{r^+p^+}$, έχει εισαχθεί χρησιμοποιώντας την πρόταση (3.5) παραπάνω, τότε το $p_s^+(L, c_t) \in NT^{\nu}_{P_D^*}(M_k)$, για όλα τα $s \in IN_{p^+r^+}$.*
Υποπερίπτωση 2.2: *Αν το $r_t^+([l_2|L], c_t)$, όπου το t είναι η θέση εισόδου του r που μοιράζεται την μεταβλητή του με τη θέση εξόδου του q στην πρόταση (3.2), έχει εισαχθεί χρησιμοποιώντας την πρόταση (3.4), τότε συμπεραίνουμε πως το $q^-([l_1|L], c_t) \in NT^{\nu}_{P_D^*}(M_k)$. Χρησιμοποιώντας την επαγωγική υπόθεση για το q, παίρνουμε:*

> *Αφού το $q^-([l_1|L], c_t) \in NT^{\nu}_{P_D^*}(M_k)$, τότε υπάρχουν σταθερές $d_1, \ldots, d_u \in U_{P_D}$ τέτοιες ώστε το $q(d_1, \ldots, d_u, c_t) \in T_{P_D} \uparrow \omega$ και τα $q_j^+([l_1|L], d_j) \in NT^{\nu}_{P_D^*}(M_k)$, για $j = 1, \ldots, u$.*

Χρησιμοποιώντας προτάσεις της μορφής (3.6) παραπάνω, παίρνουμε ότι τα $p_s^+(L, d_{f_{p^+q^+}(s)}) \in NT^{\nu}_{P_D^*}(M_k)$, για όλα τις $s \in IN_{p^+q^+}$.

Για αυτό, έχουμε ότι το $q(d_1, \ldots, d_u, c_t) \in T_{P_D} \uparrow \omega$ και το $r(c_1, \ldots, c_m, b) \in T_{P_D} \uparrow \omega$. Προκειμένου αυτά τα δύο αποτιμημένα άτομα να συνδυάζονται χρησιμοποιώντας τη πρόταση (3.2) για να παράγουν ένα άτομο για το p, πρέπει να σιγουρευτούμε ότι αν το d_i και το c_j αντιστοιχούν σε θέσεις ορισμάτων εισόδου του q και του r που στην πρόταση (3.2) μοιράζονται την ίδια μεταβλητή, τότε $d_i = c_j$. Αλλά είναι εύκολο να δούμε ότι αυτό έχει εξασφαλισθεί, επειδή και οι *δύο τιμές προκύπτουν από το ίδιο* $\#p_s^+$ *(πράγμα που ισχύει επειδή τα κατηγορήματα επιλογής έχουν μοναδική τιμή σε ένα δοσμένο περιβάλλον).*

Για αυτό, τα $q(d_1,\ldots,d_u,c_t)$ *και* $r(c_1,\ldots,c_m,b)$ *μπορούν να συνδυαστούν χρησιμοποιώντας τη πρόταση (3.2) προκειμένου να δώσουν το* $p(a_1,\ldots,a_n,b) \in T_{P_D} \uparrow \omega$.

<u>Περίπτωση 3:</u> *Ας υποθέσουμε ότι το στο* P *υπάρχει μια πρόταση της μορφής:*

$$p(\cdots) \leftarrow q(\cdots). \tag{3.7}$$

Η απόδειξη αυτής της περίπτωσης είναι παρόμοια (και στην πραγματικότητα ευκολότερη) από αυτή της Περίπτωσης 2.

Το Λήμμα 3.7.6 είναι μια ειδική περίπτωση του Λήμματος 3.7.5.

Λήμμα 3.7.6 *Έστω* P *ένα απλό fm-Datalog πρόγραμμα,* D *μια βάση δεδομένων και* $\leftarrow p(a_1,\ldots,a_n,Y)$ *ένας στόχος. Έστω* P^* *το Choice Datalog*$_{nS}$ *πρόγραμμα που προκύπτει από την εφαρμογή του μετασχηματισμού στο* $P \cup \{\leftarrow p(a_1,\ldots,a_n,Y)\}$. *Για όλα τα* $b \in U_{P_D}$ *τα ακόλουθα ισχύουν: αν υπάρχει μια οριακή ερμηνεία* M *του* P_D^* *τέτοια ώστε το* $p^-([\,],b) \in M$, *τότε το* $p(a_1,\ldots,a_n,b) \in T_{P_D} \uparrow \omega$.

Απόδειξη 3.7.6 *Καθώς το* $p^-([\,],b) \in M$, *από το Λήμμα 3.7.5 έχουμε ότι υπάρχουν σταθερές* $c_1,\ldots,c_n \in U_{P_D}$ *τέτοιες ώστε το* $p(c_1,\ldots,c_n,b) \in T_{P_D} \uparrow \omega$ *και τα* $p_i^+([\,],c_i) \in M$, *για* $i = 1,\ldots,n$. *Αλλά, καθώς τα μόνα στιγμιότυπα του* $p_i^+([\,],X)$ *στην* M *είναι τα* $p_i^+([\,],a_i)$, *συμπεραίνουμε πως* $c_i = a_i$, *για* $i = 1,\ldots,n$.

Το ακόλουθο θεώρημα δείχνει την ορθότητα του μετασχηματισμού.

Θεώρημα 3.7.1 *Έστω* P *ένα απλό fm-Datalog πρόγραμμα,* D *μια βάση δεδομένων* $\leftarrow p(a_1,\ldots,a_n,Y)$ *ο στόχος. Έστω* P^* *το Choice Datalog*$_{nS}$ *πρόγραμμα που προκύπτει από την εφαρμογή του μετασχηματισμού στο* $P \cup \{\leftarrow p(a_1,\ldots,a_n,Y)\}$ *και* $\leftarrow p^-([\,],Y)$ *ο νέος στόχος. Τότε για όλα τα* $b \in U_{P_D}$ *το ακόλουθο ισχύει: υπάρχει μια οριακή ερμηνεία* M *του* P_D^* *τέτοια ώστε το* $p^-([\,],b) \in M$ *ανν* $p(a_1,\ldots,a_n,b) \in T_{P_D} \uparrow \omega$.

Απόδειξη 3.7.7 *Άμεση συνέπεια των Λημμάτων 3.7.4 και 3.7.6.*

Κεφάλαιο 4

Μετασχηματισμός σε συνεχόμενες πολλαπλές καταναλώσεις

4.1 Εισαγωγή

Σε αυτό το Κεφάλαιο θα παρουσιάσουμε μια επέκταση του *μετασχηματισμού διακλαδιζόμενου χρόνου (branching time transformation)* ([37]). Πιο συγκεκριμένα, ο προτεινόμενος μετασχηματισμός εφαρμόζεται σε όλα τα Datalog προγράμματα που επιτρέπουν συνεχόμενες πολλαπλές καταναλώσεις μεταβλητών, σε αντίθεση με τον αρχικό που εφαρμόζεται μόνο σε Chain Datalog προγράμματα. Η κλάση προγραμμάτων στην οποία εφαρμόζεται ο προτεινόμενος μετασχηματισμός είναι μια ειδική περίπτωση της κλάσης που αναφέρθηκε στην Ενότητα 3.4. Ειδικότερα, στην κλάση των Datalog προγραμμάτων που θα παρουσιάσουμε επιτρέπονται πολλαπλές καταναλώσεις, που γίνονται όμως μόνο σε συνεχόμενα άτομα, ένας περιορισμός που δεν υπήρχε στην κλάση των fm-Datalog προγραμμάτων [32]. Θα παρουσιάσουμε την απόδειξη της ορθότητας του μετασχηματισμού, που είναι ανάλογη με αυτή που παρουσιάστηκε στην Ενότητα 3.7. Στις επόμενες ενότητες θα μελετηθούν τα παρακάτω θέματα:

- Ορίζεται μια νέα κλάση προγραμμάτων Datalog, η οποία είναι μια επέκταση των Chain Datalog προγραμμάτων. Η κλάση αυτή επιτρέπει να εκφραστούν μια σειρά από ενδιαφέρουσες επερωτήσεις στις επαγωγικές βάσεις δεδομένων.

- Ο προτεινόμενος μετασχηματισμός διατηρεί τις ιδιότητες του μετασχηματισμό που παρουσιάστηκε στο Κεφάλαιο 3. Η τελική γλώσσα είναι ίδια με αυτή που παρουσιάστηκε στην Ενότητα 3.5.

4.2 Η αρχική γλώσσα του μετασχηματισμού

Η αρχική γλώσσα του μετασχηματισμού ορίζεται παρακάτω:

Ορισμός 4.2.1 *Μια πρόταση*

$$p_0(\vec{v}_0, Z_n) \leftarrow p_1(\vec{v}_1, Z_1), p_2(\vec{v}_2, Z_2), \ldots, p_n(\vec{v}_n, Z_n).$$

με $n > 0$, λέγεται συνεχόμενης κατανάλωσης (consecutive consumption) πρόταση *(ή* cc-πρόταση *για συντομία) αν:*

1. *Κάθε* $\vec{v}_i$, *για* $i = 0, \ldots, n$ *είναι ένα μη κενό διάνυσμα από διακριτές μεταβλητές και οι* $Z_1, \ldots, Z_n$ *είναι διακριτές μεταβλητές.*

2. $vars(\vec{v}_0) = vars(\vec{v}_1)$ *και* $vars(\vec{v}_i) = \{Z_{i-1}\} \cup u_{i-1}$, *για* $i = 2, \ldots, n$, *όπου* $u_{i-1} \subseteq vars(\vec{v}_{i-1})$.

3. $Z_i \notin \bigcup_{j \le i} vars(\vec{v}_j)$, *για* $i = 1, \ldots, n$.

Ένα πρόγραμμα P λέγεται συνεχόμενης κατανάλωσης Datalog πρόγραμμα *ή* cc-Datalog πρόγραμμα *αν όλες οι προτάσεις του είναι cc-προτάσεις. Ένας* στόχος *G είναι της μορφής* $\leftarrow q(\vec{e}, Z)$, *όπου* $\vec{e}$ *είναι ένα μη κενό διάνυσμα από σταθερές, Z είναι μια μεταβλητή και q ένα IDB κατηγόρημα.*

Παράδειγμα 4.2.1 *Το ακόλουθο πρόγραμμα είναι ένα fm-Datalog πρόγραμμα:*

$$\begin{array}{l} \mathtt{q}(\overset{+}{\mathtt{X}}, \overset{-}{\mathtt{Z}}) \leftarrow \mathtt{f}(\overset{+}{\mathtt{X}}, \overset{-}{\mathtt{Z}}). \\ \mathtt{q}(\overset{+}{\mathtt{X}}, \overset{-}{\mathtt{Z}}) \leftarrow \mathtt{e}(\overset{+}{\mathtt{X}}, \overset{-}{\mathtt{Y}}), \mathtt{q}(\overset{+}{\mathtt{Y}}, \overset{-}{\mathtt{W}}), \mathtt{g}(\overset{+}{\mathtt{Y}}, \overset{+}{\mathtt{W}}, \overset{-}{\mathtt{Z}}). \end{array}$$

όπου τα + και − πάνω από τις μεταβλητές δηλώνουν τα ορίσματα εισόδου και εξόδου αντίστοιχα και μεταφέρονται σαν εκθέτες στα νέα κατηγορήματα (για αυτό και μπορούν να παραλειφθούν).

Η διαίσθηση για την κλάση των cc-Datalog προγραμμάτων είναι πως κάθε τιμή που παράγεται από ένα άτομο, καταναλώνεται από ένα ή περισσότερα συνεχόμενα άτομα που ακολουθούν το άτομο που παρήγαγε αυτή την τιμή. Η αρχή των παραγωγών γίνεται από την κεφαλή της πρότασης και ολοκληρώνεται με την κατανάλωση της τιμής που παράχθηκε στο τελευταίο άτομο στο σώμα της πρότασης, από το ελεύθερο όρισμα της κεφαλής. Πολλά φυσικά Datalog προγράμματα ανήκουν σε αυτή τη κλάση, για παράδειγμα, η κλάση των Datalog προγραμμάτων αλυσίδα, είναι ένα γνήσιο υποσύνολο αυτής της κλάσης.

Τα παραδείγματα 3.4.2, 3.4.3 που παρουσιάστηκαν στην Ενότητα 3.4 είναι στην πραγματικότητα cc-Datalog προγράμματα. Ακόμα ένα ενδιαφέρον πρόγραμμα που ανήκει στη κατηγορία των cc-Datalog προγραμμάτων είναι το ακόλουθο:

Παράδειγμα 4.2.2 *Το ακόλουθο πρόγραμμα μπορεί να βρίσκει όλα τα σημεία που ανήκουν στο ίδιο ημιεπίπεδο ως προς μια ευθεία γραμμή:*

```
same_side(X,Line,Z) ← same(X,Line,Z).
same_side(X,Line,Z) ← same(X,Line,W),same_side(W,Line,Z).
```

όπου το κατηγόρημα `same(X,Line,Z)` *δηλώνει πως τα σημεία X και Z βρίσκονται στο ίδιο ημιεπίπεδο ως προς την ευθείας* `Line`.

Όπως και στα fm-Datalog προγράμματα, έτσι και εδώ κάθε cc-Datalog πρόγραμμα μπορεί να μετατραπεί σε ένα απλό cc-Datalog πρόγραμμα, με μια ανάλογη πρόταση με την Πρόταση 3.4.1. Επομένως από εδώ και στο εξής θα χρησιμοποιούμε μόνο απλές cc-προτάσεις.

Παράδειγμα 4.2.3 *Το ακόλουθο πρόγραμμα είναι ένα cc-Datalog πρόγραμμα:*

```
q(X,Z) ← f(X,Z).
q(X,Z) ← e(X,Y),q(Y,W),g(Y,W,Z).
```

Το παραπάνω cc-Datalog πρόγραμμα δεν είναι απλό καθώς η δεύτερη του πρόταση έχει περισσότερα από δύο άτομα στο σώμα της. Για να το μετατρέψουμε σε ένα απλό cc-Datalog πρόγραμμα, εισάγουμε μία νέα πρόταση:

```
p(Y,Z) ← q(Y,W),g(Y,W,Z).
```

και την χρησιμοποιούμε για να κάνουμε fold την δεύτερη πρόταση του προγράμματος. Με αυτό τον τρόπο παίρνουμε το ακόλουθο απλό cc-Datalog πρόγραμμα:

```
q(X,Z) ← f(X,Z).
q(X,Z) ← e(X,Y),p(Y,Z).
p(Y,Z) ← q(Y,W),g(Y,W,Z).
```

4.3 Ο αλγόριθμος μετασχηματισμού

Αλγόριθμος (Πρώτη Προσέγγιση): Έστω P ένα απλό cc-Datalog πρόγραμμα και G ένας στόχος. Για κάθε IDB ή EDB κατηγόρημα p με βαθμό $n+1$ στο P, εισάγω $n+1$ δυαδικά IDB κατηγορήματα $p_1^+, \ldots, p_n^+$, p^-, όπου το p_i^+ αντιστοιχεί στο i-οστό όρισμα εισόδου του p και p^- στο $n+1$-στό όρισμα του p, που είναι το όρισμα εξόδου του. Τα αντίστοιχα κατηγορήματα επιλογής των παραπάνω κατηγορημάτων επίσης χρησιμοποιούνται στο τελικό πρόγραμμα. Η διαδικασία μετασχηματίζει το στόχο G και κάθε μία πρόταση του P και δίνει σαν έξοδο ένα νέο στόχο G^* μαζί με ένα Choice Datalog$_{nS}$ πρόγραμμα P^*. Θεωρούμε πως υπάρχει μια συνάρτηση ανάθεσης διαφορετικών ετικετών για κάθε άτομο που εμφανίζεται στο σώμα κάθε κανόνα του P. Οι ετικέτες είναι φυσικοί αριθμοί και συμβολίζονται με $l_1, l_2, \cdots$. Ο αλγόριθμος ορίζεται με ανάλυση περιπτώσεων, όπου κάθε περίπτωση εξαρτάται από τη μορφή της πρότασης που επεξεργαζόμαστε κάθε φορά:

Περίπτωση 1: Ο μετασχηματισμός του στόχου:

$$\leftarrow p(a_1, \ldots, a_n, Y).$$

δίνει σαν αποτέλεσμα ένα σύνολο από n νέες μοναδιαίες προτάσεις, που προστίθενται στο τελικό πρόγραμμα P^*:

$$p_i^+([\,], a_i).$$

όπου $i = 1, \ldots, n$. Ο νέος στόχος G^* είναι:

$$\leftarrow\ p^-([\,], Y).$$

<u>*Περίπτωση 2:*</u> Έστω C μια πρόταση της μορφής:

$$p(\vec{v}_0, Z) \ \leftarrow \ q(\vec{v}_1, Y), r(\vec{v}_2, Z).$$

και έστω l_1, l_2 οι ετικέτες των $q(\vec{v}_1, Y)$ και $r(\vec{v}_2, Z)$, αντίστοιχα. Τότε η C μετασχηματίζεται με τον ακόλουθο τρόπο:

1. Η ακόλουθη πρόταση προστίθεται στο P^*:

$$p^-(L, Z) \leftarrow \ \#r^-([l_2|L], Z).$$

2. Έστω X μία μεταβλητή που εμφανίζεται στην k-οστή θέση του $\vec{v}_0$ όπως επίσης και στην m-οστή θέση του $\vec{v}_1$. Τότε η ακόλουθη πρόταση προστίθεται στο P^*:

$$q_m^+([l_1|L], X) \leftarrow \ \#p_k^+(L, X).$$

   Παρόμοια, έστω X μία μεταβλητή που εμφανίζεται στην k-οστή θέση του $\vec{v}_0$ καθώς επίσης και στην m-οστή θέση του $\vec{v}_2$. Τότε, η ακόλουθη πρόταση προστίθεται στο P^*:

$$r_m^+([l_2|L], X) \leftarrow \ \#p_k^+(L, X).$$

3. Αν η μεταβλητή εξόδου Y του q εμφανίζεται και στην m-οστή θέση του $\vec{v}_2$, τότε η ακόλουθη πρόταση προστίθεται στο P^*:

$$r_m^+([l_2|L], Y) \leftarrow \ \#q^-([l_1|L], Y).$$

<u>*Περίπτωση 3:*</u> Έστω η πρόταση C της μορφής:

$$p(\vec{v}_0, Z) \ \leftarrow \ q(\vec{v}_1, Z).$$

και έστω l_3 η ετικέτα του $q(\vec{v}_1, Z)$. Τότε η C μετασχηματίζεται ως εξής:

1. Η ακόλουθη πρόταση προστίθεται στο P^*:

$$p^-(L, Z) \leftarrow \ \#q^-([l_3|L], Z).$$

2. Έστω X η μεταβλητή που εμφανίζεται στην k-οστή θέση του $\vec{v}_0$ καθώς επίσης και στην m-οστή θέση του $\vec{v}_1$. Τότε, η ακόλουθη πρόταση προστίθεται στο P^*:

$$q_m^+([l_3|L], X) \leftarrow \ \#p_k^+(L, X).$$

<u>*Περίπτωση 4:*</u> Για κάθε EDB κατηγόρημα p του P με n μεταβλητές εισόδου και μία μεταβλητή εξόδου, μία νέα πρόταση της ακόλουθης μορφής προστίθεται στο P^*:

$$p^-(L, Y) \leftarrow p(X_1, \ldots, X_n, Y), \#p_1^+(L, X_1), \ldots, \#p_n^+(L, X_n).$$

Παράδειγμα 4.3.1 *Έστω το ακόλουθο απλό cc-Datalog πρόγραμμα P, που προκύπτει από το Παράδειγμα 4.2.3, μαζί με ένα στόχο, όπου* `p`, `q` *είναι IDB κατηγορήματα και* `e`, `f` *και* `g` *είναι EDB κατηγορήματα:*

$$\begin{array}{l}
\leftarrow \texttt{q(a1,Z).}\\
\texttt{q(X,Z)} \leftarrow \texttt{f(X,Z).}\\
\texttt{q(X,Z)} \leftarrow \texttt{e(X,Y),p(Y,Z).}\\
\texttt{p(Y,Z)} \leftarrow \texttt{q(Y,W),g(Y,W,Z).}
\end{array}$$

Το τελικό πρόγραμμα, μαζί με το νέο στόχο, που προκύπτει από την εφαρμογή του μετασχηματισμού στο $P \cup \{\leftarrow \texttt{q(a1,Z)}\}$ *είναι:*

$$\begin{array}{l}
\leftarrow \texttt{q}^{-}\texttt{([],Z).}\\
\texttt{q}_1^{+}\texttt{([],a1).}\\
\texttt{q}^{-}\texttt{(L,Z)} \leftarrow \texttt{\#f}^{-}\texttt{([1|L],Z).}\\
\texttt{f}_1^{+}\texttt{([1|L],X)} \leftarrow \texttt{\#q}_1^{+}\texttt{(L,X).}\\
\texttt{q}^{-}\texttt{(L,Z)} \leftarrow \texttt{\#p}^{-}\texttt{([3|L],Z).}\\
\texttt{p}_1^{+}\texttt{([3|L],Y)} \leftarrow \texttt{\#e}^{-}\texttt{([2|L],Y).}\\
\texttt{e}_1^{+}\texttt{([2|L],X)} \leftarrow \texttt{\#q}_1^{+}\texttt{(L,X).}\\
\texttt{p}^{-}\texttt{(L,Z)} \leftarrow \texttt{\#g}^{-}\texttt{([5|L],Z).}\\
\texttt{g}_2^{+}\texttt{([5|L],W)} \leftarrow \texttt{\#q}^{-}\texttt{([4|L],W).}\\
\texttt{g}_1^{+}\texttt{([5|L],Y)} \leftarrow \texttt{\#p}_1^{+}\texttt{(L,Y).}\\
\texttt{q}_1^{+}\texttt{([4|L],Y)} \leftarrow \texttt{\#p}_1^{+}\texttt{(L,Y).}\\
\texttt{e}^{-}\texttt{(L,Y)} \leftarrow \texttt{e(X,Y), \#e}_1^{+}\texttt{(L,X).}\\
\texttt{f}^{-}\texttt{(L,Y)} \leftarrow \texttt{f(X,Y), \#f}_1^{+}\texttt{(L,X).}\\
\texttt{g}^{-}\texttt{(L,Z)} \leftarrow \texttt{g(X,Y,Z), \#g}_1^{+}\texttt{(L,X),\#g}_2^{+}\texttt{(L,Y).}
\end{array}$$

Θεωρούμε την ακόλουθη βάση δεδομένων D:

`e(a1,a2).`	`f(a2,a).`	`g(a2,a,b3).`
`e(a1,b1).`	`f(b1,b).`	`g(b1,a,a3).`

Τόσο το αρχικό πρόγραμμα όσο και το τελικό έχουν μία λύση, την Z = b3. *Ωστόσο, αν αντικαταστήσουμε τα κατηγορήματα επιλογής με τα αντίστοιχα κλασικά, τότε το προκύπτον πρόγραμμα θα έχει μια επιπλέον, λανθασμένη, λύση την* Z = a3.

Ο αλγόριθμος που μόλις παρουσιάστηκε, αποτελεί την αρχική προσέγγιση [31], όπου χρησιμοποιούνται κατηγορήματα επιλογής παντού, ανεξαιρέτως αν χρειάζονται σε όλες τις περιπτώσεις. Οι λόγοι που μας οδήγησαν σε αυτή την προσέγγιση ήταν λόγοι απλούστευσης της απόδειξης της ορθότητας του αλγορίθμου μετασχηματισμού. Στον αλγόριθμο που παρουσιάζουμε στη συνέχεια, χρησιμοποιούμε κατηγορήματα επιλογής μόνο εκεί που είναι απαραίτητα. Ο λόγος που αναγκαστήκαμε να μειώσουμε την χρήση των κατηγορημάτων επιλογής ήταν πως, όπως παρατηρήθηκε, τα πλεονάζοντα αυτά κατηγορήματα προκαλούν προβλήματα στην απόδοση της bottom-up αποτίμησης του τελικού προγράμματος, μια και αυξάνουν την διακλάδωση που πρέπει να κάνουμε όταν εφαρμόζουμε τον τελεστή C_{P_D}, όπως επίσης και τα βήματα υπολογισμού του τελικού αποτελέσματος.

Αλγόριθμος (Δεύτερη Προσέγγιση): Ακριβώς όπως στο μετασχηματισμό 4.3, αλλά βάζουμε κατηγορήματα επιλογής μόνο όπου χρειάζονται, επομένως η Περίπτωση 2 τροποποιείται ως εξής:

1. *Περίπτωση 2':* Έστω C μια πρόταση της μορφής:

$$p(\vec{v}_0, Z) \ \leftarrow \ q(\vec{v}_1, Y), r(\vec{v}_2, Z).$$

και έστω l_1, l_2 οι ετικέτες των $q(\vec{v}_1, Y)$ και $r(\vec{v}_2, Z)$, αντίστοιχα. Τότε η C μετασχηματίζεται με τον ακόλουθο τρόπο:

(α) Η ακόλουθη πρόταση προστίθεται στο P^*:

$$p^-(L, Z) \leftarrow \ r^-([l_2|L], Z).$$

(β) Έστω X μία μεταβλητή που εμφανίζεται στην k-οστή θέση του $\vec{v}_0$ όπως επίσης και στην m-οστή θέση του $\vec{v}_1$. Τότε η ακόλουθη πρόταση προστίθεται στο P^*:

$$q^+_m([l_1|L], X) \leftarrow \ [\#]p^+_k(L, X).$$

όπου $[\#]p^+_k = \#p^+_k$ αν X εμφανίζεται δύο φορές στο σώμα της C και $[\#]p^+_k = p^+_k$ διαφορετικά. Παρόμοια, έστω X μία μεταβλητή που εμφανίζεται στην k-οστή θέση του $\vec{v}_0$ καθώς επίσης και στην m-οστή θέση του $\vec{v}_2$. Τότε η ακόλουθη πρόταση προστίθεται στο P^*:

$$r^+_m([l_2|L], X) \leftarrow \ \#p^+_k(L, X).$$

(γ) Αν η μεταβλητή εξόδου Y του q εμφανίζεται και στην m-οστή θέση του $\vec{v}_2$, τότε η ακόλουθη πρόταση προστίθεται στο P^*:

$$r^+_m([l_2|L], Y) \leftarrow \ q^-([l_1|L], Y).$$

Παράδειγμα 4.3.2 *Το πρόγραμμα του Παραδείγματος 4.3.1 μετασχηματίζεται με βάση τη νέα προσέγγιση όπως φαίνεται παρακάτω:*

$$\begin{array}{l}
\leftarrow \texttt{q}^-\texttt{([],Z).} \\
\texttt{q}^+_1\texttt{([],a1).} \\
\texttt{q}^-\texttt{(L,Z)} \leftarrow \texttt{f}^-\texttt{([1|L],Z).} \\
\texttt{f}^+_1\texttt{([1|L],X)} \leftarrow \texttt{q}^+_1\texttt{(L,X).} \\
\texttt{q}^-\texttt{(L,Z)} \leftarrow \texttt{p}^-\texttt{([3|L],Z).} \\
\texttt{p}^+_1\texttt{([3|L],Y)} \leftarrow \texttt{e}^-\texttt{([2|L],Y).} \\
\texttt{e}^+_1\texttt{([2|L],X)} \leftarrow \texttt{q}^+_1\texttt{(L,X).} \\
\texttt{p}^-\texttt{(L,Z)} \leftarrow \texttt{g}^-\texttt{([5|L],Z).} \\
\texttt{g}^+_2\texttt{([5|L],W)} \leftarrow \texttt{q}^-\texttt{([4|L],W).} \\
\texttt{g}^+_1\texttt{([5|L],Y)} \leftarrow \texttt{\#p}^+_1\texttt{(L,Y).} \\
\texttt{q}^+_1\texttt{([4|L],Y)} \leftarrow \texttt{\#p}^+_1\texttt{(L,Y).} \\
\texttt{e}^-\texttt{(L,Y)} \leftarrow \texttt{e(X,Y), e}^+_1\texttt{(L,X).} \\
\texttt{f}^-\texttt{(L,Y)} \leftarrow \texttt{f(X,Y), f}^+_1\texttt{(L,X).} \\
\texttt{g}^-\texttt{(L,Z)} \leftarrow \texttt{g(X,Y,Z), g}^+_1\texttt{(L,X), g}^+_2\texttt{(L,Y).}
\end{array}$$

Τόσο το αρχικό πρόγραμμα όσο και το τελικό με την ίδια ΒΔ D έχουν μία λύση, την $\mathtt{Z} = \mathtt{b3}$, *ενώ αν αντικαταστήσουμε τα κατηγορήματα επιλογής με τα αντίστοιχα κλασικά, τότε το προκύπτον πρόγραμμα θα έχει μια επιπλέον, λανθασμένη, λύση, την* $\mathtt{Z} = \mathtt{a3}$. *Αυτό φαίνεται αν κάνουμε μια από-κάτω-προς-τα-πάνω αποτίμηση των δύο προγραμμάτων (έχουμε πάρει άτομα επιλογής μόνο για εκείνα τα άτομα που εμφανίζονται με # στο σώμα και όχι και για τα επτά άτομα που θα έπρεπε να πάρουμε, κάτι που όπως ξέρουμε γίνεται, με βάση τον Ορισμό 5.1.2 της Ενότητας 5.1), όπως φαίνεται στο Σχήμα 4.1.*

$$\mathtt{q}_1^{+}([\,],\mathtt{a1}).$$
$$\downarrow$$
$$\mathtt{f}_1^{+}([1],\mathtt{a1}).\ \mathtt{e}_1^{+}([2],\mathtt{a1}).$$
$$\downarrow$$
$$\mathtt{e}^{-}([2],\mathtt{a2}).\ \mathtt{e}^{-}([2],\mathtt{b1}).$$
$$\downarrow$$
$$\mathtt{p}_1^{+}([3],\mathtt{a2}). \qquad \mathtt{p}_1^{+}([3],\mathtt{b1}).$$
$$\swarrow \quad \searrow$$

$$\#\mathtt{p}_1^{+}([3],\mathtt{a2}).$$
$$\downarrow$$
$$\mathtt{g}_1^{+}([5,3],\mathtt{a2}).\ \mathtt{q}_1^{+}([4,3],\mathtt{a2}).$$
$$\downarrow$$
$$\mathtt{f}_1^{+}([1,4,3],\mathtt{a2}).\ \mathtt{e}_1^{+}([2,4,3],\mathtt{a2}).$$
$$\downarrow$$
$$\mathtt{f}^{-}([1,4,3],\mathtt{a}).$$
$$\downarrow$$
$$\mathtt{q}^{-}([4,3],\mathtt{a}).$$
$$\downarrow$$
$$\mathtt{g}_2^{+}([5,3],\mathtt{a}).$$
$$\downarrow$$
$$\mathtt{g}^{-}([5,3],\mathtt{b3}).$$
$$\downarrow$$
$$\mathtt{p}^{-}([3],\mathtt{b3}).$$
$$\downarrow$$
$$\mathtt{q}^{-}([\,],\mathtt{b3}).$$

$$\#\mathtt{p}_1^{+}([3],\mathtt{b1}).$$
$$\downarrow$$
$$\mathtt{g}_1^{+}([5,3],\mathtt{b1}).\ \mathtt{q}_1^{+}([4,3],\mathtt{b1}).$$
$$\downarrow$$
$$\mathtt{f}_1^{+}([1,4,3],\mathtt{b1}).\ \mathtt{e}_1^{+}([2,4,3],\mathtt{b1}).$$
$$\downarrow$$
$$\mathtt{f}^{-}([1,4,3],\mathtt{b}).$$
$$\downarrow$$
$$\mathtt{q}^{-}([4,3],\mathtt{b}).$$
$$\downarrow$$
$$\mathtt{g}_2^{+}([5,3],\mathtt{b}).$$

Σχήμα 4.1: *Υπολογισμός του τρέχοντος παραδείγματος με άτομα επιλογής*

Αντίθετα, χωρίς τα άτομα επιλογής θα είχαμε το αποτέλεσμα του Σχήματος 4.2, όπου φαίνεται πως υπολογίζουμε και τη λάθος απάντηση a3.

Η αποδοτικότητα του bottom-up υπολογισμού των προγραμμάτων θα συζητηθεί στις Ενότητες 5.1 και 5.2.

$$
\begin{gathered}
\mathtt{q}_1^+([\,],\mathtt{a1}).\\
\downarrow\\
\mathtt{f}_1^+([1],\mathtt{a1}).\ \mathtt{e}_1^+([2],\mathtt{a1}).\\
\downarrow\\
\mathtt{e}^-([2],\mathtt{a2}).\ \mathtt{e}^-([2],\mathtt{b1}).\\
\downarrow\\
\mathtt{p}_1^+([3],\mathtt{a2}).\ \mathtt{p}_1^+([3],\mathtt{b1}).\\
\downarrow\\
\mathtt{g}_1^+([5,3],\mathtt{a2}).\ \mathtt{g}_1^+([5,3],\mathtt{b1}).\ \mathtt{q}_1^+([4,3],\mathtt{a2}).\ \mathtt{q}_1^+([4,3],\mathtt{b1}).\\
\downarrow\\
\mathtt{f}_1^+([1,4,3],\mathtt{a2}).\ \mathtt{f}_1^+([1,4,3],\mathtt{b1}).\ \mathtt{e}_1^+([2,4,3],\mathtt{a2}).\ \mathtt{e}_1^+([2,4,3],\mathtt{b1}).\\
\downarrow\\
\mathtt{f}^-([1,4,3],\mathtt{a}).\ \mathtt{f}^-([1,4,3],\mathtt{b}).\\
\downarrow\\
\mathtt{q}^-([4,3],\mathtt{a}).\ \mathtt{q}^-([4,3],\mathtt{b}).\\
\downarrow\\
\mathtt{g}_2^+([5,3],\mathtt{a}).\ \mathtt{g}_2^+([5,3],\mathtt{b}).\\
\downarrow\\
\mathtt{g}^-([5,3],\mathtt{b3}).\ \mathtt{g}^-([5,3],\mathtt{a3}).\\
\downarrow\\
\mathtt{p}^-([3],\mathtt{b3}).\ \mathtt{p}^-([3],\mathtt{a3}).\\
\downarrow\\
\mathtt{q}^-([\,],\mathtt{b3}).\ \mathtt{q}^-([\,],\mathtt{a3}).
\end{gathered}
$$

Σχήμα 4.2: *Υπολογισμός του τρέχοντος παραδείγματος χωρίς τα άτομα επιλογής*

4.4 Η ορθότητα του μετασχηματισμού

Στη παρουσίαση της ορθότητας του μετασχηματισμού θα δανειστούμε πολλούς από τους ορισμούς που χρησιμοποιήθηκαν στην Ενότητα 3.7. Ειδικότερα, η έννοια του συνόλου παραγωγής για ένα άτομο A σε μία προσέγγιση I, η έννοια της εξάρτησης και της απόστασης ενός ατόμου A από ένα άτομο B σε μία προσέγγιση I, μεταφέρονται αυτούσια. Αυτό που πρέπει να προσαρμοστεί, για να καλύπτει την παρούσα κλάση προγραμμάτων, είναι η έννοια που συσχετίζει μία προσέγγιση I με ένα $S \subseteq I$. Το σύνολο ατόμων S περιέχει τα άτομα τα οποία θα πρέπει να υπάρχουν σε μια προσέγγιση I, ενώ η εκάστοτε ιδιότητα (μη περιορισμός ή ελαχιστικότητας) μας καθορίζει ποιές προσεγγίσεις την ικανοποιούν και κατά κάποιο τρόπο τον τρόπο που παράγονται τα άτομα του S (από τους πιθανώς πολλούς διαφορετικούς που υπάρχουν).

Ορισμός 4.4.1 *Έστω P ένα Choice Datalog$_{nS}$ πρόγραμμα και D μία βάση δεδομένων. Έστω I μια προσέγγιση μιας οριακής ερμηνείας του P_D και έστω $S \subseteq I$. Τότε, η I θα λέγεται ελαχιστική* (minimal) *ως προς το S αν το σύνολο των ατόμων επιλογής του I ταυτίζεται με το σύνολο των ατόμων επιλογής από τα οποία* *τα μέλη του S εξαρτώνται στην I.*

Με άλλα λόγια, τα άτομα επιλογής που περιλαμβάνει η I είναι όλα “σχετικά” με τις παραγωγές των ατόμων του S. Όπως είχαμε δει και την απόδειξη της ορθότητας της κλάσης σταθερού τρόπου (fm-Datalog), αυτή η ιδιότητα είναι πιο αυστηρή, καθώς μας επιτρέπει να παίρνουμε μόνο εκείνα τα άτομα επιλογής που χρειάζονται για την παραγωγή των ατόμων του S.

Η απόδειξη των παρακάτω Λημμάτων είναι μια παραλλαγή των αποδείξεων που δόθηκαν στην Ενότητα 3.7

Λήμμα 4.4.1 *Έστω P ένα απλό cc-Datalog πρόγραμμα και D μια βάση δεδομένων. Έστω P^* το Choice Datalog$_{nS}$ πρόγραμμα που προκύπτει από τον μετασχηματισμό. Για όλα τα κατηγορήματα p που ορίζονται στο P_D, όλες τις $L \in List(\mathcal{N})$ και όλες τις σταθερές $a_1, \ldots, a_n, b \in U_{P_D}$, αν το $p(a_1, \ldots, a_n, b) \in T_{P_D} \uparrow \omega$ και υπάρχει μία προσέγγιση I σε μια οριακή ερμηνεία P^*_D που είναι ελαχιστική ως προς το σύνολο $S = \{p_1^+(L, a_1), \ \ldots, \ p_n^+(L, a_n)\}$, τότε υπάρχει μία προσέγγιση J, όπου $I \subseteq J$, που είναι ελαχιστική ως προς το σύνολο $\{p^-(L, b)\}$.*

Απόδειξη 4.4.1 *Παρόμοια με την απόδειξη του Λήμματος 3.7.3.*

Το Λήμμα 4.4.2 είναι μια ειδική περίπτωση του Λήμματος 4.4.1.

Λήμμα 4.4.2 *Έστω P ένα απλό cc-Datalog πρόγραμμα, D μία βάση δεδομένων και $\leftarrow p(a_1, \ \ldots, a_n, Y)$ ένας στόχος. Έστω P^* το Choice Datalog$_{nS}$ πρόγραμμα που προκύπτει εφαρμόζοντας τον αλγόριθμο μετασχηματισμού στο $P \cup \{\leftarrow p(a_1, \ldots, a_n, Y)\}$. Για όλα τα $b \in U_{P_D}$ ισχύει το ακόλουθο: αν το $p(a_1, \ldots, \ a_n, b) \in T_{P_D} \uparrow \omega$, τότε υπάρχει μία οριακή ερμηνεία M του P^*_D τέτοια ώστε το $p^-([\,], b) \in M$.*

Απόδειξη 4.4.2 *Αφού μετασχηματίζοντας το στόχο, τα γεγονότα $p_i^+([\,], a_i)$, για $i = 1, \ldots, n$, εισάγονται στο P^*, αυτό το λήμμα είναι μια ειδική περίπτωση του Λήμματος 4.4.1.*

Λήμμα 4.4.3 *Έστω P ένα απλό cc-Datalog πρόγραμμα και D μία βάση δεδομένων. Έστω P^* το Choice Datalog$_{nS}$ πρόγραμμα που προκύπτει από το μετασχηματισμό. Έστω M μία οριακή ερμηνεία του P^*_D. Τότε για όλα τα κατηγορήματα p που ορίζονται στο P_D, για όλες τις $L \in List(\mathcal{N})$ και για όλα τα $b \in U_{P_D}$, αν το $p^-(L, b) \in M$, τότε υπάρχουν σταθερές $a_1, \ldots, a_n \in U_{P_D}$ τέτοιες ώστε το $p(a_1, \ldots, a_n, b) \in T_{P_D} \uparrow \omega$ και τα $p_i^+(L, a_i) \in M$, για $i = 1, \ldots, n$.*

Απόδειξη 4.4.3 *Το Λήμμα αυτό αποδεικνύεται με παρόμοιο τρόπο με το Λήμμα 3.7.5.*

Το Λήμμα 4.4.4 είναι μια ειδική περίπτωση του Λήμματος 4.4.3.

Λήμμα 4.4.4 *Έστω P ένα απλό cc-Datalog πρόγραμμα, D μια βάση δεδομένων και $\leftarrow p(a_1, \ldots, a_n, Y)$ ένας στόχος. Έστω P^* το Choice Datalog$_{nS}$ πρόγραμμα που προκύπτει εφαρμόζοντας τον αλγόριθμο μετασχηματισμού στο $P \cup \{\leftarrow p(a_1, \ldots, a_n, Y)\}$. Για όλα τα $b \in U_{P_D}$ ισχύει το ακόλουθο: αν υπάρχει μια οριακή ερμηνεία M του P^*_D τέτοια ώστε το $p^-([\,], b) \in M$ τότε το $p(a_1, \ldots, a_n, b) \in T_{P_D} \uparrow \omega$.*

Απόδειξη 4.4.4 *Καθώς το $p^-([\,], b) \in M$, από το Λήμμα 3.7.5 έχουμε ότι υπάρχουν σταθερές $c_1, \ldots, c_n \in U_{P_D}$ τέτοιες ώστε το $p(c_1, \ldots, c_n, b) \in T_{P_D} \uparrow \omega$ και τα $p_i^+([\,], c_i) \in M$, για $i = 1, \ldots, n$. Αλλά, καθώς τα μόνα στιγμιότυπα των $p_i^+([\,], X)$ στο M, είναι τα $p_i^+([\,], a_i)$, συμπεραίνουμε ότι $c_i = a_i$, για $i = 1, \ldots, n$.*

Το ακόλουθο θεώρημα δείχνει την ορθότητα του αλγορίθμου μετασχηματισμού.

Θεώρημα 4.4.1 *Έστω P ένα απλό cc-Datalog program, D μια βάση δεδομένων και $\leftarrow p(a_1, \ldots, a_n, Y)$ ένας στόχος. Έστω P^* ένα Choice Datalog$_{nS}$ πρόγραμμα που προκύπτει εφαρμόζοντας τον αλγόριθμο μετασχηματισμού στο $P \cup \{\leftarrow p(a_1, \ldots, a_n, Y)\}$ και $\leftarrow p^-([\,], Y)$ ο νέος στόχος. Για όλα τα $b \in U_{P_D}$ ισχύει το ακόλουθο: υπάρχει μια οριακή ερμηνεία M του P_D^* τέτοια ώστε το $p^-([\,], b) \in M$ αν και μόνο αν το $p(a_1, \ldots, a_n, b) \in T_{P_D} \uparrow \omega$.*

Απόδειξη 4.4.5 *Η απόδειξη είναι μια άμεση συνέπεια των Λημμάτων 4.4.2 και 4.4.4.*

Κεφάλαιο 5

Υπολογιστικά θέματα

Σε αυτό το Κεφάλαιο θα παρουσιάσουμε μια βελτιστοποιημένη από-κάτω-προς-τα-πάνω στρατηγική υπολογισμού για τα Choice Datalog$_{nS}$ που προκύπτουν από το μετασχηματισμό, που αποδεικνύεται ότι τερματίζει. Στη συνέχεια παρουσιάζουμε μια σειρά βελτιστοποιήσεων οι οποίες εφαρμόζονται στο τελικό πρόγραμμα. Η εφαρμογή τους οφείλεται στη μορφή που έχει ο μετασχηματισμός και οδηγούν σε πιο αποδοτική εκτέλεση των προγραμμάτων που παράγονται. Επιπλέον, συζητάμε το ζήτημα των λιστών που χρησιμοποιούνται και κατά πόσο αυτές επιβραδύνουν την εκτέλεση των τελικών προγραμμάτων. Στο τέλος του Κεφαλαίου υπάρχει μια σύγκριση των προτεινόμενων μεθόδων με άλλες διαδεδομένες τεχνικές της περιοχής.

5.1 Μία βελτιστοποιημένη από-κάτω-προς-τα-πάνω (bottom-up) αποτίμηση που τερματίζει

Σε αυτή την ενότητα παρουσιάζουμε μια bottom-up αποτίμηση για τα Choice Datalog$_{nS}$ προγράμματα που προκύπτουν από το μετασχηματισμό, η οποία τερματίζει. Η βασική ιδέα πίσω από αυτή τη διαδικασία που προτείνουμε, είναι πως κατά τη διάρκεια του bottom-up υπολογισμού χρειάζεται να λαμβάνουμε υπόψη μας μόνο άτομα που έχουν λίστα (χρονικοί όροι) με φραγμένο μήκος από μια σταθερά που εξαρτάται από τα χαρακτηριστικά του αρχικού προγράμματος P και την βάση δεδομένων D.

Ο ακόλουθος ορισμός είναι απαραίτητος, προκειμένου να περιγράψουμε αυτή τη διαδικασία:

Ορισμός 5.1.1 *Ένα σύνολο S ατόμων είναι* φραγμένης-λίστας (list-bounded) *από το $k < \omega$ αν η λίστα κάθε μέλους του S έχει μήκος που είναι μικρότερο ή ίσο από το k.*

Είμαστε πλέον στη θέση να αναφέρουμε το κύριο θεώρημα αυτής της ενότητας. Διαισθητικά, το θεώρημα λέει πως αν ένα άτομο $p(a_1, \ldots, a_n, b)$ είναι μια απάντηση για το στόχο G του P_D και αυτή η απάντηση μπορεί να προκύψει σε λιγότερα ή ακριβώς k επαναλήψεις του τελεστή T_{P_D}, τότε μια αντίστοιχη λύση μπορεί να βρεθεί με έναν bottom-up υπολογισμό στο P^*_D λαμβάνοντας υπόψη μόνο τα άτομα των οποίων οι λίστες έχουν μήκος μικρότερο ή ίσο με το k.

Θεώρημα 5.1.1 *Έστω P ένα απλό cc-Datalog πρόγραμμα, D μια βάση δεδομένων και $\leftarrow p(a_1, \ldots, a_n, Y)$ ένας στόχος. Έστω $P^* \cup \{\leftarrow p^-([\,], Y)\}$ το Choice Datalog$_{nS}$ πρόγραμμα και ο στόχος που προκύπτουν από τον μετασχηματισμό. Υποθέτουμε ότι το $p(a_1, \ldots, a_n, b) \in T_{P_D} \uparrow k$, όπου $k < \omega$. Τότε υπάρχει μία οριακή ερμηνεία M του P^*_D τέτοια ώστε $p^-([\,], b) \in M$ και το $p^-([\,], b)$ έχει ένα σύνολο παραγωγής στο M που είναι φραγμένης-λίστας από το k.*

Η απόδειξη του θεωρήματος είναι μία άμεση συνέπεια του ακόλουθου λήμματος:

Λήμμα 5.1.1 *Έστω p ένα κατηγόρημα στο P_D και υποθέτουμε ότι το $p(a_1, \ldots, a_n, b) \in T_{P_D} \uparrow k$, με $k < \omega$. Επιπλέον, υποθέτουμε ότι υπάρχει μια προσέγγιση I σε μια οριακή ερμηνεία του P^*_D που είναι ελαχιστική ως προς το σύνολο $S = \{p^+_1(L, a_1), \ldots, p^+_n(L, a_n)\}$ και ότι κάθε μέλος του S έχει ένα σύνολο παραγωγής στην I που είναι φραγμένης λίστας από το $|L| + k$. Τότε υπάρχει μια προσέγγιση J, όπου $I \subseteq J$, που είναι ελαχιστική ως προς το σύνολο $\{p^-(L, b)\}$ και το $p^-(L, b)$ έχει ένα σύνολο παραγωγής στην J που είναι φραγμένης-λίστας από το $|L| + k$.*

Απόδειξη 5.1.1 *Η απόδειξη του λήμματος γίνεται με επαγωγή στο k. Στην πραγματικότητα, η απόδειξη είναι σχεδόν ίδια με την απόδειξη του Λήμματος 3.7.3, η μόνη διαφορά είναι ότι σε κάθε περίπτωση του επαγωγικού βήματος πρέπει να επιβεβαιωθεί ότι το σύνολο παραγωγής για το $p^-(L, b)$ στην J είναι φραγμένης-λίστας από το $|L| + k$. Οι λεπτομέρειες της απόδειξης είναι προφανείς για αυτό και παραλείπονται.*

Το παραπάνω θεώρημα υποδηλώνει ότι είναι δυνατόν να βρεθεί ένας φυσικός αριθμός (που εξαρτάται από τα χαρακτηριστικά του αρχικού προγράμματος P και της βάσης δεδομένων D) έτσι ώστε η διαδικασία απόδειξης για το τελικό πρόγραμμα P^* να λαμβάνει υπόψη της μόνο άτομα φραγμένης-λίστας από αυτή τη σταθερά. Πιο συγκεκριμένα, αρκεί να εξάγουμε μια σταθερά k τέτοια ώστε $T_{P_D} \uparrow k$ είναι το ελάχιστο σταθερό σημείο (least fixpoint) του T_{P_D}. (Επειδή το αρχικό πρόγραμμα ανήκει στην Datalog, η βάση Herbrand του είναι πεπερασμένη, άρα πάντα θα υπάρχει μια σταθερά στην οποία ο T_{P_D} θα τερματίζει. Αξίζει να σημειώσουμε πως Datalog $\subset \mathcal{P}$, όπου $\mathcal{P}$ η κλάση πολυπλοκότητας).

Πρόταση 5.1.1 *Έστω P ένα απλό cc-Datalog πρόγραμμα και D μια βάση δεδομένων. Επιπλέον, έστω c το πλήθος των διαφορετικών σταθερών που εμφανίζονται στη βάση δεδομένων D, π το πλήθος των IDB κατηγορημάτων στο P και α ο μέγιστος βαθμός των IDB κατηγορημάτων στο P. Έστω $k = \pi \cdot c^\alpha$. Τότε, το $T_{P_D} \uparrow k$ είναι το ελάχιστο σταθερό σημείο του T_{P_D}.*

Απόδειξη 5.1.2 *Στην χειρότερη περίπτωση, όλα τα IDB κατηγορήματα έχουν τον ίδιο βαθμό α και στο ελάχιστο σταθερό σημείο του T_{P_D} μπορούν να εμφανίζονται όλοι οι πιθανοί συνδυασμοί από σταθερές σε όλα τα IDB κατηγορήματα. Επιπλέον, στην χειρότερη περίπτωση, μόνο ένα νέο IDB άτομο εισάγεται σε κάθε βήμα στον bottom-up υπολογισμό. Αυτό δίνει ένα $\pi \cdot c^\alpha$ φράγμα χειρότερης περίπτωσης για τον αριθμό επαναλήψεων που απαιτούνται για να φτάσουμε στο ελάχιστο σταθερό σημείο του T_{P_D}.*

Το παραπάνω θεώρημα εύκολα οδηγεί σε μια αποδεικτική διαδικασία για τα τελικά προγράμματα που προκύπτουν από τον μετασχηματισμό, που τερματίζει. Πιο συγκεκριμένα, κατά τη διάρκεια του bottom-up υπολογισμού μπορούμε να απορρίψουμε όλα τα άτομα των οποίων οι λίστες έχουν μήκος μεγαλύτερο από το όριο που καθορίζεται από την Πρόταση 5.1.1. Τότε, προφανώς, η αποδεικτική διαδικασία θα τερματίζει μια και το σύνολο των ατόμων που ανήκουν στη βάση Herbrand ενός Choice Datalog$_{nS}$ προγράμματος και τα οποία είναι φραγμένης-λίστας από μια σταθερά, είναι πεπερασμένο.

Υπάρχει μια ακόμα παρατήρηση που μπορεί να χρησιμοποιηθεί για να βελτιστοποιήσουμε την bottom-up εκτέλεση των τελικών προγραμμάτων. Έστω P ένα πρόγραμμα που έχει προκύψει από το μετασχηματισμό και έστω S_P το σύνολο των κατηγορημάτων επιλογής που εμφανίζονται σε προτάσεις του P. Είναι εύκολο να δούμε ότι κατά τη διάρκεια του bottom-up υπολογισμού του προγράμματος, μπορούμε να λαμβάνουμε υπόψη μας μόνο άτομα επιλογής των οποίων τα κατηγορήματα ανήκουν στο S_P. Με άλλα λόγια, τα κατηγορήματα επιλογής που δεν ανήκουν στο S_P δεν είναι σχετικά με την παραγωγή των σωστών απαντήσεων του στόχου του P. Αυτή η βελτιστοποίηση μειώνει σημαντικά τις διαφορετικές διακλαδώσεις που χρειάζεται να λάβουμε υπόψη μας κατά τη διάρκεια του bottom-up υπολογισμού.

Η παραπάνω συζήτηση οδηγεί σε μια βελτιστοποιημένη αποδεικτική διαδικασία που τερματίζει και περιγράφεται παρακάτω: έστω $\widehat{NT}_{P_D}$ να είναι ακριβώς όπως ο NT_{P_D} με την μόνη διαφορά ότι ο $\widehat{NT}_{P_D}$ δεν εισάγει άτομα των οποίων οι λίστες είναι μεγαλύτερες από την σταθερά k που σχετίζεται με το P_D και καθορίζεται από την Πρόταση 5.1.1. Επιπλέον, έστω $\widehat{C}_{P_D}$ να είναι ακριβώς σαν το C_{P_D} με την μόνη διαφορά να είναι ότι το $\widehat{C}_{P_D}$ επιστρέφει ερμηνείες των οποίων τα άτομα επιλογής έχουν κατηγορήματα που ανήκουν στο S_P. Τότε η έννοια της P_D-αλυσίδας μπορεί να οριστεί ως εξής:

Ορισμός 5.1.2 *Έστω P ένα Choice Datalog$_{nS}$ πρόγραμμα και D μια βάση δεδομένων. Μία φραγμένης-λίστας* (list-bounded) *P_D-αλυσίδα είναι μια ακολουθία $\langle M_0, M_1, M_2, \ldots\rangle$ που ικανοποιεί τις ακόλουθες συνθήκες:*

$$M_0 = \emptyset,$$

$$M_{2i+1} = \widehat{NT}_{P_D} \uparrow \omega(M_{2i}),\ \ i \geq 0,$$

και

$$M_{2i+2} \in \widehat{C}_{P_D}(M_{2i+1}),\ \ i \geq 0.$$

Η ακόλουθη πρόταση είναι εύκολο να αποδειχθεί:

Πρόταση 5.1.2 *Έστω P ένα Choice Datalog$_{nS}$ πρόγραμμα που προκύπτει από τον μετασχηματισμό και D μια βάση δεδομένων. Έστω $\langle M_0, M_1, M_2, \ldots\rangle$ φραγμένης-λίστας P_D-αλυσίδα. Τότε, για όλα τα $i \in \omega$ υπάρχει $m \in \omega$ τέτοιο ώστε $M_{2i+1} = \widehat{NT}^{m}_{P_D}(M_{2i})$. Επιπλέον, υπάρχει $n \in \omega$ τέτοιο ώστε $M_n = M_{n+1}$.*

Η παραπάνω πρόταση δείχνει ότι μπορούμε να υπολογίσουμε το όριο μιας φραγμένης-λίστας P_D-αλυσίδα σε πεπερασμένο χρόνο. Αυτό δίνει μια αποδεικτική διαδικασία που τερματίζει για το τελικό πρόγραμμα του μετασχηματισμού.

Ένα λογικό ερώτημα που προκύπτει από την παραπάνω συζήτηση είναι πόσο πρακτική είναι η παραπάνω αποδεικτική διαδικασία. Προφανώς, η σταθερά που καθορίζεται από την Πρόταση 5.1.1 μπορεί να είναι πολύ μεγάλη. Επιπλέον, ο αριθμός των ατόμων που παράγονται στον bottom-up υπολογισμό και είναι φραγμένης-λίστας από αυτή τη σταθερά, μπορεί να είναι εξαιρετικά μεγάλος. (Μπορούμε ωστόσο να δούμε, ότι ο αριθμός των διαφορετικών ετικετών που χρησιμοποιούμε μπορεί να μειωθεί σημαντικά.) Τα ακόλουθα επιχειρήματα μπορούν να δοθούν για να υπερασπιστούμε την προτεινόμενη προσέγγιση:

- Πολλές γνωστές τεχνικές βελτιστοποίησης διάδοσης τιμών οδηγούν σε τελικά προγράμματα των οποίων η bottom-up εκτέλεση υποφέρει από το πρόβλημα του μη-τερματισμού ([49]). Για το λόγο αυτό, ακόμα και η ύπαρξη της Πρότασης 5.1.1 είναι ένα βήμα προς την σωστή κατεύθυνση.

- Όπως φαίνεται στην πράξη, υπάρχουν πολλές περιπτώσεις που η bottom-up εκτέλεση του τελικού προγράμματος τερματίζει ακόμα και αν δεν καταφύγουμε στην παραπάνω φραγμένης-λίστας αποδεικτική διαδικασία.

- Όπως θα γίνει ξεκάθαρο στην επόμενη ενότητα, υπάρχουν μετασχηματισμοί που μπορούν να μειώσουν τον αριθμό των ετικετών των λιστών που εμφανίζονται στο τελικό πρόγραμμα του μετασχηματισμού. Αυτό σημαίνει πως και ο αριθμός των ατόμων φραγμένης-λίστας μειώνεται και για αυτό η παραπάνω αποδεικτική διαδικασία γίνεται πιο αποδοτική. Στην πραγματικότητα, σε πολλές ενδιαφέρουσες περιπτώσεις οι ετικέτες λιστών μπορούν να απαλειφθούν εντελώς και έτσι το τελικό πρόγραμμα να είναι στην ουσία ένα βελτιστοποιημένο Datalog πρόγραμμα που μπορεί να εκτελεστεί πολύ αποδοτικά ([37]).

Στην επόμενη ενότητα θα συζητήσουμε ορισμένους μετασχηματισμούς που αυξάνουν την απόδοση του τελικού κώδικα του προτεινόμενου μετασχηματισμού.

5.2 Βελτιστοποιώντας το τελικό πρόγραμμα

Σε αυτή την ενότητα θα δείξουμε ότι τα προγράμματα που προκύπτουν από το μετασχηματισμό της Ενότητας 3.6 μπορούν να απλοποιηθούν σημαντικά χρησιμοποιώντας ξεδίπλωμα (unfolding) και απαλοιφή περιττών προτάσεων. Για αυτό ορίζουμε την ορθότητα του ξεδιπλώματος (unfolding) και των κανόνων απαλοιφής για τα Choice Datalog$_{nS}$ προγράμματα στην Υποενότητα 5.2.1 και στην συνέχεια στην Υποενότητα 5.2.2, θα αποδείξουμε πως αυτοί οι κανόνες μπορούν να χρησιμοποιηθούν για να απλοποιήσουμε τα προγράμματα που προκύπτουν από τον αλγόριθμο της Ενότητας 3.6. Με αυτό τον τρόπο, αυξάνουμε λίγο το compile-time, ώστε να μειωθεί το run-time.

5.2.1 Ξεδίπλωμα και απαλοιφή περιττών προτάσεων στην Choice Datalog$_{nS}$

Ο κανόνας ξεδιπλώματος (unfolding) και ο κανόνας απαλοιφής των περιττών προτάσεων που θα περιγράψουμε παρακάτω είναι παρόμοιοι με αυτούς που ορίζονται στον κλασικό λογικό προγραμματισμό [47, 15, 29], η μόνη διαφορά είναι η ύπαρξη των ατόμων επιλογής στις προτάσεις.

Ορισμός 5.2.1 *Έστω P ένα Choice Datalog$_{nS}$ πρόγραμμα και C μια πρόταση στο P της μορφής:*

$$A \leftarrow A_1, \ldots, A_{i-1}, A_i, A_{i+1}, \ldots, A_n.$$

όπου A_i είναι ένα άτομο μη-επιλογής (κλασικό). Έστω $C_1, C_2, \ldots, C_m$ όλες οι προτάσεις στο P, που οι κεφαλές τους είναι ενοποιήσιμες με το A_i με μέγιστους κοινούς ενοποιητές (mgu) $\theta_1, \theta_2, \ldots, \theta_m$ αντίστοιχα. Το αποτέλεσμα του ξεδιπλώματος (unfolding) της C στο *A_i είναι το σύνολο των προτάσεων $\{C'_1, \ldots, C'_m\}$ έτσι ώστε, για κάθε j, με $1 \leq j \leq m$, αν C_j είναι η πρόταση:*

$$B_j \leftarrow B_{j_1}, \ldots, B_{j_h}.$$

με $h \geq 0$ και $B_j\theta_j = A_i\theta_j$, τότε η C'_j είναι η πρόταση:

$$(A \ \leftarrow \ A_1, \ldots, A_{i-1}, B_{j_1}, \ldots, B_{j_h}, A_{i+1}, \ldots, A_n)\theta_j.$$

Τότε το πρόγραμμα που προκύπτει από το ξεδίπλωμα (unfolding) είναι το πρόγραμμα $P' = (P - \{C\}) \cup \{C'_1, \ldots, C'_m\}$. Η πρόταση C λέγεται η unfolded clause, *τα $C_1, \ldots, C_m$ λέγονται οι* unfolding clauses *και το A_i λέγεται το* unfolded atom.

Παρατηρείστε ότι οι κανόνες ξεδιπλώματος που ορίστηκαν παραπάνω μπορούν να εφαρμοστούν μόνο σε άτομα μη-επιλογής (κλασικά) του σώματος των κανόνων. Το ακόλουθο λήμμα δείχνει την ορθότητα του κανόνα ξεδιπλώματος (unfolding).

Λήμμα 5.2.1 *Έστω P ένα Choice Datalog$_{nS}$ πρόγραμμα και P' το πρόγραμμα που προκύπτει από το ξεδίπλωμα μιας πρότασης C στο P. Τότε, για κάθε βάση δεδομένων D, τα P_D και P'_D έχουν τα ίδια σύνολα οριακών ερμηνειών.*

Απόδειξη 5.2.1 *(Περίγραμμα απόδειξης) Έστω S_k και S'_k τα σύνολα των προσεγγίσεων στις οριακές ερμηνείες του P_D και P'_D αντίστοιχα, που προκύπτουν κατά τη διάρκεια της k-οστής επανάληψης στον bottom-up υπολογισμό (δηλαδή το σύνολο των k-οστών στοιχείων όλων των P_D-αλυσίδων όπως περιγράφονται στον Ορισμό 3.5.6). Θα χρησιμοποιήσουμε επαγωγή στο k για να αποδείξουμε ότι $S_k = S'_k$ για όλα τα $k \in \omega$.*
<u>Βάση Επαγωγής</u>: *Το λήμμα ισχύει για $k = 0$, αφού $S_0 = S'_0 = \{\emptyset\}$.*
<u>Επαγωγική Υπόθεση</u>: *Υποθέτουμε ότι $S_k = S'_k$ για όλα τα $k \leq n$, με $n \in \omega$.*
<u>Βήμα Επαγωγής</u>: *Θα αποδείξουμε ότι $S_{n+1} = S'_{n+1}$. Διακρίνουμε τις ακόλουθες περιπτώσεις:*
<u>Περίπτωση 1</u>: *Έστω $n = 2l + 1$, $l \geq 0$. Είναι εύκολο να δούμε ότι $S_{n+1} = S'_{n+1}$ $= \{M \mid M \in C_{P_D}(M'),\ M' \in S_n\}$.*
<u>Περίπτωση 2</u>: *Έστω $n = 2l+2$ για κάποιο $l \geq 0$ και έστω $J \in S_n$. Τότε, εξαιτίας της επαγωγικής υπόθεσης, $J \in S'_n$. Είναι εύκολο να δούμε ότι οι προσεγγίσεις που προκύπτουν από την J στην S_{n+1} και στην S'_{n+1} είναι οι $I = NT_{P_D} \uparrow \omega(J)$ και $I' = NT_{P'_D} \uparrow \omega(J)$, αντίστοιχα. Θα αποδείξουμε ότι $I = I'$ αποδεικνύοντας ότι* a) *$I \subseteq I'$ και* b) *$I' \subseteq I$.*

<u>Απόδειξη του (a):</u> *Θα χρησιμοποιήσουμε επαγωγή στο* i *για να αποδείξουμε ότι* $NT^{i}_{P_D}(J) \subseteq I'$.
<u>Βάση Επαγωγής (του a):</u> *Για* $i = 0$ *έχουμε:* $NT^{0}_{P_D}(J) = NT^{0}_{P'_D}(J) = J$. *Εξαιτίας του Λήμματος 3.5.1,* $J \subseteq I'$ *και για αυτό συμπεραίνουμε ότι* $NT^{0}_{P_D}(J) \subseteq I'$.
<u>Επαγωγική Υπόθεση (του a):</u> *Αν το* $A \in NT^{m}_{P_D}(J)$, *τότε* $A \in I'$.
<u>Βήμα Επαγωγής (του a):</u> *Θα αποδείξουμε ότι αν* $A \in NT^{m+1}_{P_D}(J)$, *τότε το* $A \in I'$. *Ας υποθέσουμε ότι το* A *έχει εισαχθεί στο* $NT^{m+1}_{P_D}(J)$ *εφαρμόζοντας μια πρόταση* C *του* P_D.
<u>Περίπτωση 1 (του a):</u> *Η* C *επίσης ανήκει στο* P'_D. *Τότε, καθώς τα άτομα του σώματος στο στιγμιότυπο του* C *χρησιμοποιούνται για να εισάγουν το* A *στην* $NT^{m+1}_{P_D}(J)$ *ανήκουν στην* $NT^{m}_{P_D}(J)$, *ανήκουν επίσης στην* I', *εξαιτίας της επαγωγικής υπόθεσης. Για αυτό, το* A *είναι άτομο που ανήκει επίσης και στην* I' *καθώς μπορεί να παραχθεί χρησιμοποιώντας την πρόταση* C.
<u>Περίπτωση 2 (του a):</u> *Η πρόταση* C *δεν ανήκει στο πρόγραμμα* P' *επειδή έχει ξεδιπλωθεί (unfolded) στο* P. *Έστω ότι η* C *είναι της μορφής:*

$$A_0 \leftarrow A_1, \ldots, A_{i-1}, A_i, A_{i+1}, \ldots, A_n.$$

όπου το A_i *είναι το ξεδιπλωμένο άτομο (unfolded atom). Έστω ότι* $C_1, C_2, \ldots, C_s$ *είναι όλες οι προτάσεις στο πρόγραμμα* P, *των οποίων οι κεφαλές είναι ενοποιήσιμες με το* A_i *με μέγιστους κοινούς ενοποιητές (mgu)* $\theta_1, \theta_2, \ldots, \theta_s$, *αντίστοιχα. Είναι εύκολο να παρατηρήσουμε ότι αφού ένα αποτιμημένο στιγμιότυπο της πρότασης* C *έχει χρησιμοποιηθεί για να εισαχθεί το* A *στην* $NT^{m+1}_{P_D}(J)$, *το αντίστοιχο στιγμιότυπο του* A_i *είναι στην* $NT^{m}_{P_D}(J)$ *και έχει παραχθεί χρησιμοποιώντας μια από τις προτάσεις* $C_1, C_2, \ldots, C_s$, *καθώς αυτές είναι όλες οι προτάσεις του* P, *των οποίων οι κεφαλές είναι ενοποιήσιμες με το* A_i. *Υποθέτουμε ότι αυτή η πρόταση είναι η* C_j:

$$B_j \leftarrow B_{j_1}, \ldots, B_{j_h}.$$

με $h \geq 0$. *Καθώς η πρόταση* C_j *είναι μια unfolding πρόταση, μια πρόταση* C'_j *της παρακάτω μορφής:*

$$(A_0 \;\leftarrow\; A_1, \ldots, A_{i-1}, B_{j_1}, \ldots, B_{j_h}, A_{i+1}, \ldots, A_n)\theta_j.$$

θα έχει εισαχθεί στο πρόγραμμα P', *όπου* $\theta_j = mgu(B_j, A_i)$ *και για αυτό* $B_j\theta_j = A_i\theta_j$. *Είναι εύκολο να δούμε ότι το* A *επίσης ανήκει στην* I' *μια και μπορεί να εισαχθεί στην* I' *από το κατάλληλο στιγμιότυπο της πρότασης* C'_j. *Τα άτομα στο σώμα αυτής της πρότασης είναι τα στιγμιότυπα των ατόμων στα σώματα των στιγμιοτύπων των προτάσεων* C *και* C_j, *που έχουν χρησιμοποιηθεί την εισαγωγή του* A *στην* $NT^{m+1}_{P_D}(J)$. *Αυτά τα άτομα επίσης ανήκουν στην* I', *εξαιτίας της επαγωγικής υπόθεσης (καθώς ανήκουν στην* $NT^{m}_{P'_D}(J)$*).*
<u>Απόδειξη του (b):</u> *Όπως και στην απόδειξη του* (a) *χρησιμοποιούμε επαγωγή στο* i *για να αποδείξουμε ότι* $NT^{i}_{P'_D}(J) \subseteq I$.
<u>Βάση Επαγωγής (του b):</u> *Όπως και στην Βάση Επαγωγής του* (a) *θα αποδείξουμε ότι* $NT^{0}_{P'_D}(J) \subseteq I$.

Επαγωγική Υπόθεση (του b): *Αν το* $A \in NT^{m}_{P'_D}(J)$*, τότε το* $A \in I$*.*
Βήμα Επαγωγής (του b): *Θα αποδείξουμε ότι αν* $A \in NT^{m+1}_{P'_D}(J)$*, τότε* $A \in I$*. Υποθέτουμε ότι το* A *έχει εισαχθεί στην* $NT^{m+1}_{P_D}(J)$ *εφαρμόζοντας μια πρόταση* E *του* P'*.*
Περίπτωση 1 (του b): *Η πρόταση* E *ανήκει επίσης στο πρόγραμμα* P*. Τότε, το* A *έχει εισαχθεί επίσης στην* I *χρησιμοποιώντας την πρόταση* E *(η απόδειξη είναι παρόμοια με την Περίπτωση 1 του (a)).*
Περίπτωση 2 (του b): *Η πρόταση* E *είναι ανάμεσα στις προτάσεις του προγράμματος* P' *που προέκυψαν με το ξεδίπλωμα (unfolding) μιας πρότασης* C *του προγράμματος* P*. Είναι εύκολο να αποδείξουμε (με παρόμοιο τρόπο με την απόδειξη της Περίπτωσης 2 του (a)), ότι το άτομο* A *έχει εισαχθεί επίσης στην* I *χρησιμοποιώντας μια πρόταση* C *και την πρόταση ξεδιπλώματος (unfolding clause) που χρησιμοποιήθηκε για να πάρουμε την* E*.*

Όπως θα δούμε στην Υποενότητα 5.2.2, όταν εφαρμόζουμε τον κανόνα ξεδιπλώματος (unfolding rule) μερικές προτάσεις ξεδιπλώματος (unfolding clauses) γίνονται περιττές (καθώς ενδιαφερόμαστε να διατηρήσουμε την ισοδυναμία μόνο σε σχέση με συγκεκριμένα κατηγορήματα). Για το λόγο αυτό, μπορούμε να απαλείψουμε τις προτάσεις αυτές. Ο κανόνας απαλοιφής δίνεται τυπικά στους επόμενους δύο ορισμούς.

Ορισμός 5.2.2 *Έστω* P *ένα Choice Datalog*$_{nS}$ *πρόγραμμα. Ο* γράφος εξάρτησης κατηγορημάτων (predicate dependency graph) G_P *του* P *είναι ένας κατευθυνόμενος γράφος* (V, E)*, όπου το* V*, το σύνολο των κόμβων του γράφου, είναι το σύνολο των κατηγορηματικών συμβόλων του* P *και το* E*, το σύνολο των ακμών του, είναι τέτοιο ώστε* $(p, q) \in E$ *αν και μόνο αν υπάρχει μια πρόταση* C *στο* P*, της οποίας το κατηγόρημα κεφαλής είναι το* p *και υπάρχει ένα άτομο ή ένα άτομο επιλογής στο σώμα της* C*, του οποίου το κατηγόρημα είναι το* q*.*

Ο παραπάνω ορισμός είναι μια επέκταση του Ορισμού 2.2.1, όπου οι κόμβοι είναι μόνο τα IDB κατηγορήματα.

Ορισμός 5.2.3 *Έστω* P *ένα Choice Datalog*$_{nS}$ *πρόγραμμα και* S *ένα σύνολο από κατηγορήματα στο* P*. Μια πρόταση* C *στο* P *λέγεται* περιττή ως προς το S στο P*, αν και μόνο αν δεν υπάρχει μονοπάτι στον γράφο εξάρτησης κατηγορημάτων* G_P *του* P *που να οδηγεί από ένα κατηγόρημα του* S *στο κατηγόρημα κεφαλής της* C*.*

Το ακόλουθο λήμμα μπορεί λοιπόν εύκολα να αποδειχθεί.

Λήμμα 5.2.2 *Έστω* P *ένα Choice Datalog*$_{nS}$ *πρόγραμμα και* S *ένα σύνολο από κατηγορήματα στο* P*. Έστω ότι* P' *είναι το πρόγραμμα που προκύπτει διαγράφοντας όλες τις προτάσεις του* P *που είναι περιττές ως προς το* S*. Έστω ότι το* A *είναι ένα κλασικό άτομο ή ένα άτομο επιλογής του οποίου το κατηγόρημα επιλογής είναι στο* S*. Τότε, για κάθε βάση δεδομένων* D*, αν υπάρχει μια οριακή ερμηνεία* M *του* P_D *τέτοια ώστε το* $A \in M$*, τότε υπάρχει μια οριακή ερμηνεία* M' *του* P'_D *τέτοια ώστε* $A \in M'$*.*

Αν κατασκευάσουμε τον γράφο εξάρτησης κατηγορημάτων του προγράμματος που μας δίνεται, μπορούμε να βρούμε τα κατηγορήματα που είναι προσβάσιμα από το κατηγόρημα του στόχου, άρα και αυτά που δεν είναι. Όμως, γνωρίζουμε πως το *πρόβλημα να βρούμε αν δύο σημεία ενός κατευθυνόμενου γράφου συνδέονται μεταξύ τους με κάποιο μονοπάτι (reachability problem)* μπορεί να λυθεί σε πολυωνυμικό χρόνο και λογαριθμικό χώρο ως προς το μέγεθος της εισόδου [28, 55]. Επομένως, μπορούμε αποδοτικά να βρούμε το πρόγραμμα που προκύπτει, αν διαγράψουμε από το αρχικό πρόγραμμα όλες τις περιττές προτάσεις ως προς το κατηγόρημα του στόχου.

5.2.2 Βελτιστοποιώντας το τελικό πρόγραμμα χρησιμοποιώντας ξεδίπλωμα και απαλοιφές

Σε αυτή την υποενότητα δείχνουμε ότι μπορούμε να χρησιμοποιήσουμε τους μετασχηματισμούς της προηγούμενης υποενότητας για να βελτιώσουμε το τελικό πρόγραμμα.

5.2.3 Απαλοιφή όλων των IDB κατηγορημάτων που προκύπτουν από EDB άτομα του σώματος του αρχικού προγράμματος

Όλα τα IDB κατηγορήματα που εμφανίζονται στο τελικό πρόγραμμα P^*, που αντιστοιχούν σε EDB άτομα που εμφανίζονται στα σώματα των προτάσεων του αρχικού προγράμματος P, μπορούν να απαλειφθούν χρησιμοποιώντας ξεδίπλωμα (unfolding). Πιο συγκεκριμένα, το κατηγόρημα στις κεφαλές των προτάσεων που μπήκαν στο P^* στη Περίπτωση 4 του αλγορίθμου μετασχηματισμού 4 (που είναι κατηγορήματα εξόδου) μπορούν να εμφανίζονται μόνο στα σώματα (μερικών από τις) προτάσεις του P^*, που προστέθηκαν στις Περιπτώσεις 2 και 3 του αλγορίθμου. Όλες οι προτάσεις που περιλαμβάνουν αυτά τα άτομα στα σώματά τους μπορούν να ξεδιπλωθούν χρησιμοποιώντας τις προτάσεις που εισήχθησαν από την Περίπτωση 4 του αλγορίθμου.

Οι προτάσεις που προκύπτουν από αυτά τα βήματα ξεδιπλώματος περιέχουν εμφανίσεις κατηγορημάτων εισόδου που αντιστοιχούν σε EDB κατηγορήματα. Αυτές οι εμφανίσεις μπορούν να ξεδιπλωθούν περαιτέρω δίνοντας σαν αποτέλεσμα την πλήρη απαλοιφή όλων των κατηγορημάτων (εισόδου και εξόδου) που αντιστοιχούν σε EDB κατηγορήματα του αρχικού προγράμματος. Είναι εύκολο να δούμε, εξαιτίας της δομής του προγράμματος P^*, ότι κανένα από τα unfolded άτομα δεν εμφανίζεται στην εκδοχή του με κατηγόρημα επιλογής και για αυτό όλα τα βήματα ξεδιπλώματος είναι έγκυρα με βάση τον Ορισμό 5.2.1. Παρατηρούμε ότι όλες οι ετικέτες που αντιστοιχούν σε EDB άτομα του σώματος εξαφανίζονται επίσης, μέσω αυτής της διαδικασίας.

Παράδειγμα 5.2.1 *Ας δούμε το πρόγραμμα P^* που προκύπτει από το Παράδειγ-*

μα 4.3.1.

$$
\begin{array}{ll}
(G) & \leftarrow \texttt{q}^{-}\texttt{([],Z)}. \\
(C_1) & \texttt{q}_1^{+}\texttt{([],a1)}. \\
(C_2) & \texttt{q}^{-}\texttt{(L,Z)} \leftarrow \texttt{f}^{-}\texttt{([1|L],Z)}. \\
(C_3) & \texttt{f}_1^{+}\texttt{([1|L],X)} \leftarrow \texttt{q}_1^{+}\texttt{(L,X)}. \\
(C_4) & \texttt{q}^{-}\texttt{(L,Z)} \leftarrow \texttt{p}^{-}\texttt{([3|L],Z)}. \\
(C_5) & \texttt{p}_1^{+}\texttt{([3|L],Y)} \leftarrow \texttt{e}^{-}\texttt{([2|L],Y)}. \\
(C_6) & \texttt{e}_1^{+}\texttt{([2|L],X)} \leftarrow \texttt{q}_1^{+}\texttt{(L,X)}. \\
(C_7) & \texttt{p}^{-}\texttt{(L,Z)} \leftarrow \texttt{g}^{-}\texttt{([5|L],Z)}. \\
(C_8) & \texttt{g}_2^{+}\texttt{([5|L],W)} \leftarrow \texttt{q}^{-}\texttt{([4|L],W)}. \\
(C_9) & \texttt{g}_1^{+}\texttt{([5|L],Y)} \leftarrow \texttt{\#p}_1^{+}\texttt{(L,Y)}. \\
(C_{10}) & \texttt{q}_1^{+}\texttt{([4|L],Y)} \leftarrow \texttt{\#p}_1^{+}\texttt{(L,Y)}. \\
(C_{11}) & \texttt{e}^{-}\texttt{(L,Y)} \leftarrow \texttt{e(X,Y),e}_1^{+}\texttt{(L,X)}. \\
(C_{12}) & \texttt{f}^{-}\texttt{(L,Y)} \leftarrow \texttt{f(X,Y),f}_1^{+}\texttt{(L,X)}. \\
(C_{13}) & \texttt{g}^{-}\texttt{(L,Z)} \leftarrow \texttt{g(X,Y,Z),g}_1^{+}\texttt{(L,X),g}_2^{+}\texttt{(L,Y)}.
\end{array}
$$

Θα απαλείψουμε όλα τα IDB άτομα στο P^* (και τις προτάσεις που τα ορίζουν) που αντιστοιχούν στα EDB κατηγορήματα `e`, `f` *και* `g`, *χρησιμοποιώντας τους κανόνες μετασχηματισμού που ορίζονται στην Υποενότητα 5.2.1.*

Ξεδιπλώνοντας την (C_5) χρησιμοποιώντας την (C_{11}) παίρνουμε:

$$(C_{14})\quad \texttt{p}_1^{+}\texttt{([3|L],Y)} \leftarrow \texttt{e(X,Y), e}_1^{+}\texttt{([2|L],X)}.$$

Τώρα, ξεδιπλώνοντας την (C_{14}) χρησιμοποιώντας την (C_6) παίρνουμε:

$$(C_{15})\quad \texttt{p}_1^{+}\texttt{([3|L],Y)} \leftarrow \texttt{e(X,Y), q}_1^{+}\texttt{(L,X)}.$$

Στην συνέχεια μπορούμε να αντικαταστήσουμε την (C_{15}) στη θέση της (C_5) στο P^*. Οι προτάσεις (C_6) και (C_{11}) μπορούν να απαλειφθούν καθώς είναι περιττές ως προς το σύνολο $S = \{\texttt{q2}\}$ που περιλαμβάνει το κατηγόρημα του στόχου.

Εφαρμόζοντας την ίδια διαδικασία μπορούμε να απαλείψουμε τα κατηγορήματα `f2` *και* `f1` *και τις προτάσεις που τα ορίζουν. Πιο συγκεκριμένα, ξεδιπλώνουμε την (C_2) χρησιμοποιώντας την (C_{12}) και στη συνέχεια ξεδιπλώνουμε την πρόταση που προέκυψε από την (C_3). Με αυτό τον τρόπο παίρνουμε την πρόταση:*

$$(C_{16})\quad \texttt{q}^{-}\texttt{(L,Z)} \leftarrow \texttt{f(X,Z), q}_1^{+}\texttt{(L,X)}.$$

που αντικαθιστά την πρόταση (C_2) στο P^*. Μπορούμε πάλι να απαλείψουμε τις περιττές προτάσεις (C_3) και (C_{12}).

Εφαρμόζοντας την ίδια διαδικασία, μπορούμε να απαλείψουμε τα κατηγορήματα `g1`, `g2` *και* `g3` *που αντιστοιχούν στο EDB κατηγόρημα* `g`. *Για να το κάνουμε αυτό, ξεδιπλώνουμε την (C_7) χρησιμοποιώντας την (C_{13}) και στη συνέχεια ξεδιπλώνουμε την προκύπτουσα πρόταση χρησιμοποιώντας τις προτάσεις (C_8) και (C_9). Με αυτό τον τρόπο παίρνουμε την πρόταση:*

$$(C_{17})\quad \texttt{p}^{-}\texttt{(L,Z)} \leftarrow \texttt{g(X,Y,Z), \#p}_1^{+}\texttt{(L,X), q}^{-}\texttt{([4|L],Y)}.$$

Η πρόταση (C_{17}) μπορεί να αντικαταστήσει την πρόταση (C_7), ενώ οι προτάσεις (C_8), (C_9) και (C_{13}) μπορούν να διαγραφούν καθώς γίνονται περιττές. Το πρόγραμμα που προκύπτει εφαρμόζοντας τον παραπάνω μετασχηματισμό είναι το ακόλουθο:

(G) $\leftarrow$ `q`$^-$`([],Z).`
(C_1) `q`$_1^+$`([],a1).`
(C_4) `q`$^-$`(L,Z)` $\leftarrow$ `p`$^-$`([3|L],Z).`
(C_{10}) `q`$_1^+$`([4|L],Y)` $\leftarrow$ `#p`$_1^+$`(L,Y).`
(C_{15}) `p`$_1^+$`([3|L],Y)` $\leftarrow$ `e(X,Y),q`$_1^+$`(L,X).`
(C_{16}) `q`$^-$`(L,Z)` $\leftarrow$ `f(X,Z),q`$_1^+$`(L,X).`
(C_{17}) `p`$^-$`(L,Z)` $\leftarrow$ `g(X,Y,Z),#p`$_1^+$`(L,X),q`$^-$`([4|L],Y).`

Παρατηρούμε ότι έχουμε καταφέρει να απαλείψουμε IDB άτομα στο P^*, όπως επίσης και τις προτάσεις που τα ορίζουν, που αντιστοιχούν στα EDB κατηγορήματα `e`, `f` και `g`. Επιπλέον, οι ετικέτες 1, 2, και 5, που αντιστοιχούν στα EDB άτομα του σώματος του αρχικού προγράμματος έχουν επίσης διαγραφεί.

5.2.4 Περισσότερες βελτιστοποιήσεις χρησιμοποιώντας ξεδίπλωμα

Στην προηγούμενη υποενότητα είδαμε πως μπορούμε να απαλείψουμε τα IDB κατηγορήματα που αντιστοιχούν στα EDB άτομα του σώματος από το πρόγραμμα, που προκύπτουν από τον αλγόριθμο μετασχηματισμού της Ενότητας 3.6. Ωστόσο, όπως θα δούμε στο Παράδειγμα 5.2.2, μπορούμε να εφαρμόσουμε μερικά ακόμα βήματα ξεδιπλώματος για να απλοποιήσουμε περαιτέρω το πρόγραμμα.

Παράδειγμα 5.2.2 *Ας θεωρήσουμε το πρόγραμμα που προκύπτει από το Παράδειγμα 5.2.1. Παρατηρούμε ότι μπορούμε να ξεδιπλώσουμε την πρόταση (C_4) στο άτομο* `p2([3|L],Z)` *χρησιμοποιώντας την πρόταση (C_{17}). Με αυτό τον τρόπο παίρνουμε την πρόταση:*

(C_{18}) `q`$^-$`(L,Z)` $\leftarrow$ `g(X,Y,Z), #p`$_1^+$`([3|L],X), q`$^-$`([4,3|L],Y).`

Η πρόταση (C_{18}) μπορεί τώρα να αντικαταστήσει την πρόταση (C_4) στο πρόγραμμα και την πρόταση (C_{17}) που είναι περιττή ως προς το σύνολο $S = \{$`q2`*$\}$ και μπορεί να διαγραφεί. Με αυτό τον τρόπο παίρνουμε το πρόγραμμα:*

(G) $\leftarrow$ `q`$^-$`([],Z).`
(C_1) `q`$_1^+$`([],a1).`
(C_{10}) `q`$_1^+$`([4|L],Y)` $\leftarrow$ `#p`$_1^+$`(L,Y).`
(C_{15}) `p`$_1^+$`([3|L],Y)` $\leftarrow$ `e(X,Y),q`$_1^+$`(L,X).`
(C_{16}) `q`$^-$`(L,Z)` $\leftarrow$ `f(X,Z),q`$_1^+$`(L,X).`
(C_{18}) `q`$^-$`(L,Z)` $\leftarrow$ `g(X,Y,Z),#p`$_1^+$`([3|L],X),q`$^-$`([4,3|L],Y).`

Μπορούμε ακόμα να ξεδιπλώσουμε τις προτάσεις (C_{15}) και (C_{16}) στο άτομο `q1(L,X)`*. Με αυτό τον τρόπο παίρνουμε τις προτάσεις:*

(C_{19}) `p`$_1^+$`([3],Y)` $\leftarrow$ `e(a1,Y).`
(C_{20}) `p`$_1^+$`([3,4|L],Y)` $\leftarrow$ `e(X,Y),#p`$_1^+$`(L,X).`
(C_{21}) `q`$^-$`([],Z)` $\leftarrow$ `f(a1,Z).`
(C_{22}) `q`$^-$`([4|L],Z)` $\leftarrow$ `f(X,Z),#p`$_1^+$`(L,X).`

Οι προτάσεις (C_{19}), (C_{20}), (C_{21}) και (C_{22}) μπορούν να αντικαταστήσουν τις προτάσεις (C_{15}) και (C_{16}) στο τελικό πρόγραμμα και οι προτάσεις (C_1) και (C_{10}) μπορούν να διαγραφούν καθώς γίνονται περιττές. Κατά αυτόν τον τρόπο παίρνουμε το πρόγραμμα:

(G) $\leftarrow$ `q`$^-$`([],Z).`
(C_{19}) `p`$_1^+$`([3],Y)` $\leftarrow$ `e(a1,Y).`
(C_{20}) `p`$_1^+$`([3,4|L],Y)` $\leftarrow$ `e(X,Y),#p`$_1^+$`(L,X).`
(C_{21}) `q`$^-$`([],Z)` $\leftarrow$ `f(a1,Z).`
(C_{22}) `q`$^-$`([4|L],Z)` $\leftarrow$ `f(X,Z),#p`$_1^+$`(L,X).`
(C_{18}) `q`$^-$`(L,Z)` $\leftarrow$ `g(X,Y,Z),#p`$_1^+$`([3|L],X),q`$^-$`([4,3|L],Y).`

Ο υπολογισμός αυτού του προγράμματος φαίνεται στο Σχήμα 5.1 (από την βελτιωμένη αποτίμηση, έχουμε πάρει άτομα επιλογής μόνο για εκείνα τα άτομα που εμφανίζονται με # στο σώμα, κάτι που όπως θα δούμε παρακάτω γίνεται):

$$
\begin{array}{ccc}
 & \mathtt{p}_1^+([3],\mathtt{a2}).\ \mathtt{p}_1^+([3],\mathtt{b1}). & \\
 & \swarrow \quad \searrow & \\
\#\mathtt{p}_1^+([3],\mathtt{a2}). & & \#\mathtt{p}_1^+([3],\mathtt{b1}). \\
\downarrow & & \downarrow \\
\mathtt{q}^-([4,3],\mathtt{a}). & & \mathtt{q}^-([4,3],\mathtt{b}). \\
\downarrow & & \\
\mathtt{q}^-([\ ],\mathtt{b3}). & &
\end{array}
$$

Σχήμα 5.1: *Υπολογισμός του τρέχοντος παραδείγματος μετά τις βελτιστοποιήσεις*

Είναι σημαντικό να σημειώσουμε σε αυτό το σημείο, ότι στην περίπτωση της απαλοιφής των IDB ατόμων του σώματος στο αρχικό πρόγραμμα, διαθέτουμε έναν αλγόριθμο, που παρουσιάστηκε στην Υποενότητα 5.2.3, ο οποίος μπορεί να εφαρμοστεί για να πραγματοποιηθεί η βελτιστοποίηση. Από την άλλη μεριά, για τα βήματα ξεδιπλώματος του Παραδείγματος 5.2.2, δεν έχουμε έναν συστηματικό τρόπο, ένα συγκεκριμένο αλγόριθμο. Ωστόσο, η πεποίθηση μας είναι πως κάποιος μπορεί να καθοδηγηθεί από την δομή του αρχικού προγράμματος ώστε να μπορέσει να πραγματοποιήσει τις περισσότερες από τις βελτιστοποιήσεις με έναν αλγοριθμικό τρόπο. Ωστόσο, δεν θίγουμε περισσότερο αυτά τα ζητήματα.

5.2.5 Απαλοιφή κατηγορημάτων επιλογής

Η από-κάτω-προς-τα-πάνω εκτέλεση των τελικών προγραμμάτων μπορεί να βελτιωθεί αν μπορούμε να απαλείψουμε συγκεκριμένα άτομα επιλογής ή αν τα αντικαταστήσουμε με την κλασική εκδοχή τους. Αυτό είναι εφικτό αν γνωρίζουμε ότι ένα δοσμένο κατηγόρημα είναι ντετερμινιστικό:

Ορισμός 5.2.4 *Έστω P ένα Choice Datalog$_{nS}$ πρόγραμμα και D μια βάση δεδομένων. Ένα IDB κατηγόρημα p του P θα λέγεται* ντετερμινιστικό *ως προς μια βάση δεδομένων D, αν για κάθε L, όποτε δύο άτομα $p(L,a)$ και $p(L,b)$ παράγονται σε μια από-κάτω-προς-τα-πάνω αποτίμηση του P_D, τότε $a = b$. Ένα EDB κατηγόρημα p θα λέγεται* ντετερμινιστικό *αν το όρισμα εξόδου του ορίζεται μονοσήμαντα από τα ορίσματα εισόδου.*

Ο εντοπισμός του ντετερμινισμού σε ένα πρόγραμμα που πραγματοποιείται με τη τεχνική μας (αφού έχουμε εφαρμόσει όλες τις βελτιστοποιήσεις που αναφέρονται στην Ενότητα 5.2), είναι πολύ σημαντικός, μια και μπορούμε να αντικαταστήσουμε τα άτομα επιλογής ενός ντετερμινιστικού κατηγορήματος που εμφανίζεται σε ένα πρόγραμμα με την κλασική του μορφή. Ένας τέτοιος μετασχηματισμός οδηγεί σε λιγότερους κλάδους που πρέπει να ακολουθηθούν κατά τη διάρκεια ενός από-κάτω-προς-τα-πάνω υπολογισμού. Επιπλέον, επιτρέπει να εφαρμοστούν περισσότερες βελτιστοποιήσεις που προτάθηκαν στη προηγούμενη ενότητα, καθώς υπάρχουν περισσότερα υποψήφια άτομα για unfolding.

Υπάρχουν πολλές περιπτώσεις ντετερμινισμού που μπορούν να εντοπιστούν με στατικό τρόπο. Έστω μια βάση δεδομένων D και ένα πρόγραμμα P που έχει προκύψει από το μετασχηματισμό και την εφαρμογή των βελτιστοποιήσεων που έχουν περιγραφεί μέχρι στιγμής. Ο παρακάτω μετασχηματισμός και το λήμμα που ακολουθεί μας δίνει ένα τρόπο να εντοπίζουμε μια όχι και τόσο προφανή κατηγορία ντετερμινιστικών κατηγορημάτων.

Ορισμός 5.2.5 *Έστω μια λίστα L.* Επίθεμα (suffix) *L' της L είναι μια λίστα τα στοιχεία της οποίας εμφανίζονται συνεχόμενα στην αρχή της λίστας L.*

Ορισμός 5.2.6 *Ένα IDB κατηγόρημα p λέγεται* εν δυνάμει ντετερμινιστικό *αν ο ορισμός του αποτελείται από προτάσεις της μορφής:*

i) $p([], a)$.

ii) $p([l], Y) \leftarrow e(a_1, ..., a_n, Y)$.

iii) $p(L, Y) \leftarrow [\#]q(L_1, Y)$.

iv) $p(L, Y) \leftarrow e(X_1, ..., X_n, Y), [\#]q_1(L_1, X_1), ..., [\#]q_n(L_n, X_n)$.

όπου κάθε L_i, για $i = 1, ..., n$, είναι ένα γνήσιο επίθεμα του L.

Το παρακάτω λήμμα δίνει ένα τρόπο ελέγχου αν ένα σύνολο από κατηγορήματα είναι ντετερμινιστικό.

Λήμμα 5.2.3 (Τεστ ντετερμινισμού) *Έστω S ένα σύνολο κατηγορημάτων του P και υποθέτουμε πως οι παρακάτω συνθήκες ισχύουν για κάθε μέλος p του S:*

1. *Αν p είναι ένα EDB κατηγόρημα, τότε είναι ντετερμινιστικό και αν το p είναι ένα IDB κατηγόρημα τότε είναι εν δυνάμει ντετερμινιστικό.*

2. *Τα χρονικά ορίσματα των ατόμων στις κεφαλές όλων των κανόνων που ορίζουν το p καθορίζουν διαφορετικά χρονικά σημεία.*

3. *Το κατηγόρημα κάθε ατόμου που εμφανίζεται στο σώμα ενός κανόνα που ορίζει το p είναι μέλος του S.*

Απόδειξη 5.2.2 *Η απόδειξη θα γίνει με επαγωγή στο μήκος του χρονικού ορίσματος (μήκος λίστας).*
Επαγωγική Βάση: *Αν το μήκος του χρονικού ορίσματος είναι 0, τότε ισχύει, γιατί μόνο η περίπτωση (i) του Ορισμού 5.2.6 μπορεί να εφαρμοστεί (εξαιτίας του περιορισμού (2) δεν μπορούν οι λίστες L και L_i ταυτόχρονα να έχουν μήκος 0, δηλ. να είναι και οι δύο κενές). Επομένως το p έχει μοναδική τιμή στο όρισμα δεδομένων.*
Επαγωγική Υπόθεση: *Έστω ότι ισχύει για μήκος $k \geq 0$, δηλ. όλα τα κατηγορήματα p του S που ικανοποιούν τις συνθήκες (i), (ii), (iii) και (iv) έχουν μοναδική τιμή στο όρισμα δεδομένων για όλα τα χρονικά ορίσματα μήκους μέχρι k.*
Επαγωγική Βήμα: *Θα δείξουμε πως ισχύει και για όλα τα χρονικά ορίσματα μήκους $> k$, δηλ. όλα τα κατηγορήματα p του S που ικανοποιούν τις συνθήκες (i), (ii), (iii) και (iv) έχουν μοναδική τιμή στο όρισμα δεδομένων για όλα τα χρονικά ορίσματα μήκους $> k$. Αφού το χρονικό όρισμα έχει μήκος > 0, είτε προτάσεις της μορφής (ii), είτε της μορφής (iii) είτε της μορφής (iv) μπορούν να ορίσουν το κατηγόρημα p του S. Στη πρώτη περίπτωση έχουμε:*

$$p([l], Y) \leftarrow e(a_1, ..., a_n, Y).$$

όπου το e είναι μέλος του S από τη συνθήκη (3) και είναι ντετερμινιστικό από τη συνθήκη (1), για αυτό και το p θα έχει μοναδική τιμή στο όρισμα δεδομένων για κάθε χρονικό όρισμα $[l]$.

Στη δεύτερη περίπτωση, έχουμε:

$$p(L, Y) \leftarrow [\#]q(L_1, Y).$$

όπου η L_1 είναι γνήσιο αρχικό επίθεμα της L, επομένως από την επαγωγική υπόθεση και αφού το q είναι μέλος του S από τη συνθήκη (3), έχουμε πως το q έχει μοναδική τιμή στο όρισμα δεδομένων για το χρονικό όρισμα L_1. Επομένως και το p έχει μοναδική τιμή στο όρισμα δεδομένων για το χρονικό όρισμα L.

Στη τρίτη περίπτωση, έχουμε:

$$p(L, Y) \leftarrow e(X_1, ..., X_n, Y), [\#]q_1(L_1, X_1), ..., [\#]q_n(L_n, X_n).$$

όπου κάθε L_i, για $i = 1, ..., n$, είναι ένα γνήσιο επίθεμα του L. Αλλά από την συνθήκη (3) έχουμε πως τα $e, q_1, \ldots, q_n$ ανήκουν στο S. Η επαγωγική υπόθεση ισχύει για αυτά, επομένως κάθε q_i έχει μοναδική τιμή για το όρισμα δεδομένων του στο χρονικό όρισμα L_i. Επιπλέον, το e είναι ντετερμινιστικό από τη συνθήκη (1), επομένως το όρισμα εξόδου του έχει μοναδική τιμή για το δοσμένο διάνυσμα των ορισμάτων εισόδου του (που έχει παραχθεί από τα q_i). Επομένως, το p έχει μοναδική τιμή για το όρισμα δεδομένων του στο το χρονικό όρισμα L. Στη περίπτωση που υπάρχουν περισσότερες από μια προτάσεις που ορίζουν το p, τότε *από τη συνθήκη (2), αυτές ορίζονται σε διαφορετικά χρονικά σημεία. Επομένως το p είναι ντετερμινιστικό, δηλ. έχει μοναδική τιμή για το όρισμα δεδομένων του σε κάθε χρονικό όρισμα.*

Παράδειγμα 5.2.3 *Έστω το πρόγραμμα που προκύπτει από το Παράδειγμα 5.2.2. Μπορούμε να δούμε ότι το τεστ που ορίστηκε παραπάνω εφαρμόζεται στο σύνολο* {$\mathtt{p}_1^+$, e} *σε όλες τις περιπτώσεις που το EDB* e *είναι ντετερμινιστικό. Σε αυτές τις περιπτώσεις μπορούμε να αντικαταστήσουμε τα άτομα επιλογής με τα αντίστοιχα κλασικά που έχει σαν αποτέλεσμα το πρόγραμμα:*

(G) $\leftarrow$ `q2([],Z).`
(C_{19}) `p1([3],Y)` $\leftarrow$ `e(a1,Y).`
$(C_{20'})$ `p1([3,4|L],Y)` $\leftarrow$ `e(X,Y), p1(L,X).`
(C_{21}) `q2([],Z)` $\leftarrow$ `f(a1,Z).`
$(C_{22'})$ `q2([4|L],Z)` $\leftarrow$ `f(X,Z), p1(L,X).`
$(C_{18'})$ `q2(L,Z)` $\leftarrow$ `g(X,Y,Z), p1([3|L],X), q2([4,3|L],Y).`

Παρατηρήστε ωστόσο πως αν το e *ήταν μη-ντετερμινιστικό τότε αυτή η βελτιστοποίηση δεν θα μπορούσε να εφαρμοστεί.*

5.2.6 Μειώνοντας το κόστος από τη διαχείριση των λιστών

Ένα θέμα που τίθεται από το μετασχηματισμό διακλαδιζόμενου χρόνου που φαίνεται (με μια πρώτη ματιά) πως θέτει σοβαρούς περιορισμούς ως προς την απόδοση, είναι το γεγονός πως το τελικό πρόγραμμα βασίζεται σε μεγάλο βαθμό σε υπολογισμούς σε λίστες. Για παράδειγμα, ας υποθέσουμε πως κατά τη διάρκεια του υπολογισμού ενός προγράμματος δημιουργείται ένα άτομο. Για να επαληθεύσουμε πως το άτομο αυτό δεν έχει ήδη παραχθεί σε κάποιο προηγούμενο βήμα, πρέπει να το συγκρίνουμε με όλα τα άτομα που έχουν ήδη παραχθεί και έχουν ίδιο κατηγορηματικό σύμβολο. Ωστόσο, καθώς ένα άτομο μπορεί να έχει μια τυχαία μεγάλη λίστα, αυτό μοιάζει να σημαίνει πως η πράξη σύγκρισης δύο ατόμων κοστίζει ακριβά. Ευτυχώς υπάρχει μια απλή και κομψή λύση για αυτό το πρόβλημα, μια λύση που χρησιμοποιείται εκτενώς στην αντίστοιχη εκδοχή του μετασχηματισμού διακλαδιζόμενου χρόνου στις συναρτησιακές γλώσσες προγραμματισμού ([39]). Η ιδέα είναι πως οι λίστες μπορούν να κωδικοποιηθούν με μικρούς φυσικούς αριθμούς χρησιμοποιώντας μια τεχνική γνωστή ως *hash-consing*. Το κύριο πλεονέκτημα του hash-consing είναι πως δύο λίστες μπορούν να ελεχθούν για ισότητα με μια απλή πράξη, παρά με ένα επαναληπτικό βρόχο που ελέγχει τις λίστες και συγκρίνει αντίστοιχα στοιχεία. Το μόνο (μικρό) μειονέκτημα είναι ότι με κάθε cons πράξη πρέπει να συμβουλευόμαστε ένα πίνακα κατακερματισμού (hash table) για να ελέγχουμε ότι η λίστα που κατασκευάζουμε δεν έχει ήδη μια αναπαράσταση.

Αποθηκεύουμε τις αναπαραστάσεις των λιστών σε ένα πίνακα του οποίου κάθε γραμμή είναι ένα ζευγάρι *(head,index of tail)*. Η λίστα αναπαρίσταται (κωδικοποιείται) με το δείκτη του κατάλληλου ζευγαριού στον πίνακα. Οι παρακάτω πρωταρχικές συναρτήσεις χρησιμοποιούνται για να υλοποιήσουμε το hash-consing:

1. $hashcons(head, tail\_code)$: Χρησιμοποιεί μια συνάρτηση κατακερματισμού για να ελέγχει αν το ζευγάρι $(head, tail\_code)$ υπάρχει ήδη στο πίνακα κατα-

κερματισμού. Αν δεν υπάρχει, τότε το εισάγει. Τελικά, επιστρέφει την θέση του ζευγαριού στον πίνακα.

2. *hashhead*(*list_code*): Επιστρέφει το πρώτο στοιχείο από το ζευγάρι που βρίσκει στην θέση *list_code* του πίνακα κατακερματισμού.

3. *hashtail*(*list_code*): Επιστρέφει το δεύτερο στοιχείο του ζευγαριού που βρίσκει στην θέση *list_code* του πίνακα κατακερματισμού.

Οι παραπάνω πράξεις μπορούν να πραγματοποιηθούν αποδοτικά και ο χώρος που καταλαμβάνει ο πίνακας κατακερματισμού είναι λογικός. Χρησιμοποιώντας hash-consing και τις παραπάνω πρωταρχικές διαδικασίες, μια αποδοτική υλοποίηση της τεχνικής μπορεί να γίνει, η οποία θα αποτρέπει μια ακριβή διαχείριση των λιστών. Για περισσότερες λεπτομέρειες σχετικά με την τεχνική hash-consing μπορεί κάποιος να ανατρέξει στην [39].

5.2.7 Συζήτηση σχετικά με τις βελτιστοποιήσεις στο τελικό πρόγραμμα

Όπως φαίνεται, το τελικό πρόγραμμα επιδέχεται έναν μεγάλο αριθμό από βελτιστοποιήσεις που μειώνουν το χρόνο της εκτέλεσής του. Αυτό οφείλεται κατά κύριο λόγο στο γεγονός πως κάθε αρχικό κατηγόρημα παράγει ένα σύνολο από νέα κατηγορήματα (όσα η πληθικότητά του) πράγμα που έχει σαν αποτέλεσμα να έχουμε περισσότερη ελευθερία στο τελικό πρόγραμμα. Πολλοί από αυτούς τους περαιτέρω μετασχηματισμούς μπορούν να εκφραστούν με έναν αλγοριθμικό τρόπο και να εφαρμοστούν σε μια ευρεία κλάση από τελικά προγράμματα.

Είναι σημαντικό να αναφέρουμε ότι η απαλοιφή περιττών ατόμων επιλογής που περιγράφηκε παραπάνω είναι μεγάλης σημασίας, διότι η παρουσία μη-ντετερμινισμού είναι μια πηγή αναποτελεσματικότητας. Όπως είδαμε και στην Υποενότητα 5.2.5 η αναγκαιότητα της χρήσης ατόμων επιλογής εξαρτάται συνήθως από τη δομή της βάσης δεδομένων πάνω στην οποία εφαρμόζεται το πρόγραμμα (και όχι στην δομή του ίδιου του προγράμματος). Το κριτήριο που προτείνουμε στην Υποενότητα 5.2.5 είναι ένα πρώτο βήμα προς την κατεύθυνση της μείωσης της επιβάρυνσης που προκύπτει από την εισαγωγή του μη-ντετερμινισμού. Η επινόηση επιπλέον τεστ που θα μπορούν να εφαρμοστούν σε ευρύτερες κλάσεις προγραμμάτων είναι ένα από τα θέματα που θα μας απασχολήσει σε μελλοντικές εργασίες.

Ένα εξίσου σημαντικό θέμα βελτιστοποίησης είναι η απαλοιφή μη απαραιτήτων ετικετών από τις λίστες. Αυτή η βελτιστοποίηση είναι σημαντική καθώς μειώνει το χρόνο εκτέλεσης του προγράμματος. Μερικές ακόμα βελτιστοποιήσεις που είναι πολλά υποσχόμενες στο να μειώσουν τις ετικέτες που εμφανίζονται στις λίστες μπορούν να προκύψουν από την εκλέπτυνση των μεθόδων *Απαλοιφής περιττών τελεστών next* και *Απαλοιφής χρονικών τελεστών που αφορούν αριστερή αναδρομή* που έχουν προταθεί στο [37]. Ωστόσο η εφαρμογή αυτών των μεθόδων δεν μπορεί να γίνει απευθείας καθώς υπάρχουν τα άτομα επιλογής που μας περιορίζουν. Τα θέματα αυτά είναι αντικείμενο μελέτης.

Συμπερασματικά θα μπορούσαμε να πούμε πως πιστεύουμε πως το θέμα της βελτιστοποίησης των τελικών προγραμμάτων του μετασχηματισμού μας μπορεί να παράγει ενδιαφέροντα αποτελέσματα. Παρόλα αυτά, ακόμα και με τους τρέχοντες μετασχηματισμούς η δική μας προσέγγιση συναγωνίζεται πολύ καλά τις ήδη υπάρχουσες προσεγγίσεις (οι οποίες υπάρχουν και δέχονται βελτιώσεις εδώ και αρκετά χρόνια).

Κεφάλαιο 6

Ο μετασχηματισμός διακλαδιζόμενου χρόνου σε μη συνεχόμενες καταναλώσεις

6.1 Η δεύτερη επέκταση του αρχικού μετασχηματισμού

Σε αυτό το Κεφάλαιο προτείνουμε μία επέκταση του *μετασχηματισμού διακλαδιζόμενου χρόνου (branching-time transformation)* [37] που μπορεί να χειριστεί μια σημαντικά ευρύτερη κλάση Datalog προγραμμάτων. Ο αρχικός μετασχηματισμός μπορούσε να εφαρμοστεί μόνο σε Chain Datalog, μια ιδιαίτερα χρήσιμη κλάση προγραμμάτων. Σε αυτή την ενότητα παρουσιάζουμε μια επέκταση του μετασχηματισμού [37], ώστε να μπορεί να χειρίζεται Datalog προγράμματα τα οποία επιτρέπουν στις καταναλώσεις να μην είναι συνεχείς, αλλά να μην είναι και πολλαπλές. Πιο συγκεκριμένα, επιτρέπουμε απλές καταναλώσεις μεταβλητών, χωρίς αυτές κατ' ανάγκη να είναι συνεχόμενες. Θα παρουσιάσουμε την ορθότητα του νέου μετασχηματισμού και θα προτείνουμε συγκεκριμένες βελτιστοποιήσεις των τελικών προγραμμάτων ώστε να έχουμε πιο αποδοτική από-κάτω-προς-τα-πάνω αποτίμηση. Η κλάση που θα προτείνουμε είναι μια ειδική περίπτωση της κλάσης των fm-Datalog προγραμμάτων που παρουσιάστηκε στην Ενότητα 3.4.

6.2 Περιγραφή του μετασχηματισμού

Πιο συγκεκριμένα, η τεχνική που περιγράφεται στο [37] εφαρμόζεται σε Chain Datalog προγράμματα, ένα υποσύνολο της Datalog. Η σύνταξη των Chain Datalog προγραμμάτων είναι κάπως περιοριστική, με την έννοια πως η τιμή μιας μεταβλητής που παράγεται σε ένα άτομο πρέπει να καταναλωθεί από το αμέσως επόμενο άτομο της πρότασης. Σε αυτήν την ενότητα προτείνουμε μια μέθοδο η οποία ξεπερνάει αυτό το περιορισμό. Ειδικότερα, κάθε μεταβλητή μια πρότασης μπορεί να καταναλώνεται σε οποιοδήποτε από τα άτομα, που ακολουθούν το άτομο που την έχει παράγει (και όχι αποκλειστικά μόνο από το επόμενο). Αυτή η νέα κλάση είναι προφανώς ένα υπερσύνολο της Chain Datalog και μας δίνει μεγαλύτερη ελευθερία στην έκφραση επερωτήσεων. Πιο ειδικά, επιτρέποντας μη συνεχόμενες καταναλώσεις μεταβλητών φεύγουμε από τις γραμματικές χωρίς συμφραζόμενα (context free grammars) και ανεβαίνουμε στην ιεραρχία Chomsky ίσως στις γραμματικές με συμφραζόμενα (context sensitive) (ή ένα υποσύνολό τους), δηλαδή αυξάνεται η εκφραστικότητα. Η μελέτη αυτής της επέκτασης, δίνει έναυσμα να μελετήσουμε τα ακόλουθα θέματα:

- Θα παρουσιάσουμε πως ο μετασχηματισμός διακλαδιζόμενου χρόνου μπορεί να επεκταθεί, ώστε να εφαρμόζεται σε μια κλάση Datalog προγραμμάτων παραγωγής-κατανάλωσης. Αυτή είναι μια ευρύτερη κλάση από αυτή των Chain Datalog προγραμμάτων και επιτρέπει να εκφράζονται πιο ενδιαφέρουσες ερωτήσεις.

- Ο νέος μετασχηματισμός είναι εξίσου απλός με τον [37]. Χρησιμοποιεί την ίδια τελική γλώσσα και τα προγράμματα που προκύπτουν έχουν τις ίδιες επιθυμητές ιδιότητες με αυτά που προκύπτουν από τον [37]. Πρέπει να σημειωθεί, πως αν επιτραπούν πολλαπλές καταναλώσεις μεταβλητών, τότε η τελική γλώσσα πρέπει να ενισχυθεί με κατηγορήματα επιλογής, όπως είδαμε στην Ενότητα 3.3 και στην [31].

6.3 Η αρχική γλώσσα του μετασχηματισμού

Η αρχική γλώσσα του νέου μετασχηματισμού ορίζεται παρακάτω:

Ορισμός 6.3.1 *Μια πρόταση*

$$p_0(\vec{v}_0, Z_n) \leftarrow p_1(\vec{v}_1, Z_1), p_2(\vec{v}_2, Z_2), \dots, p_n(\vec{v}_n, Z_n).$$

με $n > 0$, λέγεται πρόταση παραγωγής-κατανάλωσης (productive-consumptive clause) *(ή* pc-*πρόταση για συντομία) αν:*

1. *Κάθε $\vec{v}_i$, για $i = 0, \dots, n$ είναι ένα μη κενό διάνυσμα που περιλαμβάνει διακριτές μεταβλητές. Επιπλέον, τα $Z_1, \dots, Z_n$ είναι διακριτές μεταβλητές.*

2. *$vars(\vec{v}_i) \subseteq vars(\vec{v}_0) \cup \{Z_1, \dots, Z_{i-1}\}$, για $1 \leq i \leq n$.*

3. *για κάθε $V \in vars(\vec{v}_0)$, υπάρχει ακριβώς ένα διάνυσμα $\vec{v}_i, 1 \leq i \leq n$ τέτοιο ώστε $V \in vars(\vec{v}_i)$.*

4. *για κάθε Z_i με $1 \leq i \leq n-1$, υπάρχει ακριβώς ένα διάνυσμα $\vec{v}_j$ με $i < j \leq n$ τέτοιο ώστε $Z_i \in vars(\vec{v}_j)$.*

Ένα πρόγραμμα P λέγεται pc-Datalog πρόγραμμα *αν όλες οι προτάσεις του είναι pc-προτάσεις. Ένας στόχος G είναι της μορφής $\leftarrow q(\vec{e}, Z)$, όπου το $\vec{e}$ είναι ένα μη κενό διάνυσμα από σταθερές, Z είναι μια μεταβλητή και το q είναι ένα IDB κατηγόρημα.*

Αξίζει να σημειωθεί εδώ, πως οι pc-προτάσεις είναι moded. Ειδικότερα, υποθέτουμε ότι κάθε κατηγόρημα έχει μόνο μια λειτουργία, δηλαδή κάθε θέση κατηγορήματος χρησιμοποιείται είτε ως είσοδος είτε ως έξοδος, αλλά όχι και τα δύο. Πιο συγκεκριμένα οι όροι $\vec{v}_i$ του παραπάνω ορισμού αντιστοιχούν στα ορίσματα εισόδου, ενώ κάθε Z_i αντιστοιχεί στο μοναδικό όρισμα εξόδου του κάθε ατόμου.

Παράδειγμα 6.3.1 *Η ακόλουθη πρόταση είναι μια pc-πρόταση:*

$$\mathtt{p}(\overset{+}{\mathtt{X}}, \overset{+}{\mathtt{Y}}, \overset{-}{\mathtt{Z}}) \leftarrow \mathtt{q}(\overset{+}{\mathtt{Y}}, \overset{-}{\mathtt{W}}), \mathtt{r}(\overset{+}{\mathtt{X}}, \overset{-}{\mathtt{R}}), \mathtt{s}(\overset{+}{\mathtt{W}}, \overset{+}{\mathtt{R}}, \overset{-}{\mathtt{Z}}).$$

όπου τα σύμβολα $+$ και $-$ πάνω από τις μεταβλητές δηλώνουν τα ορίσματα εισόδου και εξόδου αντίστοιχα.

Επιπλέον οι έννοιες των παραγωγών και των καταναλώσεων δίνονται στον ορισμό 3.4.2. Η διαίσθηση που κρύβεται πίσω από τα pc-Datalog προγράμματα είναι ότι η τιμή που παράγεται από ένα άτομο πρέπει να καταναλωθεί σε ακριβώς ένα άτομο που το ακολουθεί, όχι απαραίτητα στο αμέσως επόμενο από αυτό που το παρήγαγε (εκτός από την παραγωγή στο τελευταίο άτομο που επιστρέφει την τιμή του στην κεφαλή). Για αυτό, κάθε μεταβλητή εμφανίζεται ακριβώς δύο φορές σε μια pc-πρόταση. Πολλά φυσικά Datalog προγράμματα ανήκουν σε αυτή την κλάση, όπως για παράδειγμα η κλάση των Chain Datalog προγραμμάτων που είναι ένα γνήσιο υποσύνολο αυτής της κλάσης.

Ορισμός 6.3.2 *Ένα* απλό (simple) pc-Datalog πρόγραμμα *είναι ένα pc-Datalog πρόγραμμα στο οποίο κάθε πρόταση έχει το πολύ δύο άτομα στο σώμα της.*

Η ακόλουθη πρόταση, που μπορεί εύκολα να αποδειχθεί χρησιμοποιώντας μετασχηματισμούς διπλώματος/ξεδιπλώματος (unfold/fold) όπως υπάρχουν στα [15, 29], αποκαθιστά την ισοδυναμία ανάμεσα στα pc-Datalog προγράμματα και στα απλά pc-Datalog.

Πρόταση 6.3.1 *Κάθε pc-Datalog πρόγραμμα P μπορεί να μετασχηματιστεί σε ένα απλό pc-Datalog πρόγραμμα P' τέτοιο ώστε για κάθε κατηγόρημα p που εμφανίζεται στο P και για κάθε βάση δεδομένων D, να ισχύει $M(p, P_D) = M(p, P'_D)$.*

Απόδειξη 6.3.1 *Παρόμοια με την Απόδειξη 3.4.1.*

Παράδειγμα 6.3.2 *Ας θεωρήσουμε το ακόλουθο pc-Datalog πρόγραμμα P:*

$$\begin{array}{ll} (1) & \mathtt{p(X,Y,Z)} \leftarrow \mathtt{e(X,W)},\ \mathtt{p(W,Y,R)},\mathtt{f(R,Z)}. \\ (2) & \mathtt{p(X,Y,Z)} \leftarrow \mathtt{g(X,Y,Z)}. \end{array}$$

Το αντίστοιχο απλό pc-Datalog πρόγραμμα P' είναι:

$$\begin{array}{ll} (1') & \mathtt{p(X,Y,Z)} \leftarrow \mathtt{e(X,W)},\ \mathtt{q(W,Y,Z)}. \\ (\mathrm{E}) & \mathtt{q(W,Y,Z)} \leftarrow \mathtt{p(W,Y,R)},\mathtt{f(R,Z)}. \\ (2) & \mathtt{p(X,Y,Z)} \leftarrow \mathtt{g(X,Y,Z)}. \end{array}$$

Το P' έχει προκύψει από το P εισάγοντας ένα νέο ορισμό (definition), την πρόταση E, και στην συνέχεια διπλώνοντας την πρόταση 1 χρησιμοποιώντας την πρόταση E, ώστε να προκύψει η $1'$.

Αφού από την Πρόταση 6.3.1 έχουμε ότι για κάθε pc-Datalog πρόγραμμα μπορούμε να πάρουμε ένα ισοδύναμο απλό pc-Datalog πρόγραμμα, για πρακτικούς λόγους θα ορίσουμε τον αλγόριθμο μετασχηματισμού και θα αποδείξουμε την ορθότητα του σε απλά pc-Datalog προγράμματα.

6.4 Η τελική γλώσσα του μετασχηματισμού

Τελική γλώσσα του μετασχηματισμού είναι η $Datalog_{nS}$, η οποία είναι μια γλώσσα χρονικού λογικού προγραμματισμού που υποστηρίζει μια διακλαδιζόμενη έννοια του χρόνου. Οι βασικές έννοιες της γλώσσας παρουσιάζονται στην ενότητα 3.5.2. Μια πιο αναλυτική περιγραφή της γλώσσας μπορεί κανείς να βρει στις [13, 12].

6.5 Ο αλγόριθμος μετασχηματισμού

Σε αυτή την ενότητα θα ορίσουμε τον αλγόριθμο μετασχηματισμού τυπικά.

Αλγόριθμος: Έστω P ένα απλό pc-Datalog πρόγραμμα και G ένας στόχος. Για κάθε IDB ή EDB κατηγόρημα p βαθμού $(n+1)$ στο P, εισάγουμε $n+1$ δυαδικά IDB κατηγορήματα $p_1^+, \ldots, p_n^+, p^-$, όπου το p_i^+ αντιστοιχεί στο i-οστό όρισμα εισόδου του p και το p^- στο $(n+1)$-οστό όρισμα του p, που είναι το όρισμα εξόδου. Ο μετασχηματισμός επεξεργάζεται το στόχο G και κάθε πρόταση στο P και δίνει σαν έξοδο ένα νέο στόχο G^* και ένα Datalog$_{nS}$ πρόγραμμα P^*. Θεωρούμε πως υπάρχει μια συνάρτηση ανάθεσης διαφορετικών ετικετών για κάθε άτομο που εμφανίζεται στο σώμα κάθε κανόνα του P. Οι ετικέτες είναι φυσικοί αριθμοί και συμβολίζονται με $l_1, l_2, \cdots$.

Ο αλγόριθμος επεξεργάζεται το στόχο και το πρόγραμμα με τον ακόλουθο τρόπο:

Περίπτωση 1: Ο μετασχηματισμός του στόχου

$$\leftarrow p(a_1, \ldots, a_n, Y).$$

δίνει σαν αποτέλεσμα ένα σύνολο από n νέες μοναδιαίες προτάσεις, που προστίθενται στο P^*

$$p_i^+([\,], a_i).$$

για $i = 1, \ldots, n$.

Ο νέος στόχος G^* είναι:

$$\leftarrow p^-([\,], Y).$$

Περίπτωση 2: Έστω C μια πρόταση της μορφής:

$$p(\vec{v}_0, Z) \;\leftarrow\; q(\vec{v}_1, Y), r(\vec{v}_2, Z).$$

και έστω ότι l_1, l_2 είναι οι ετικέτες του $q(\vec{v}_1, Y)$ και $r(\vec{v}_2, Z)$, αντίστοιχα. Τότε η C μετασχηματίζεται με τον ακόλουθο τρόπο:

a) Η ακόλουθη πρόταση προστίθεται στο P^*:

$$p^-(L, Z) \leftarrow r^-([l_2|L], Z).$$

b) Έστω X η μεταβλητή που εμφανίζεται στην k-οστή θέση του $\vec{v}_0$ και επίσης στην m-οστή θέση του $\vec{v}_1$. Τότε, η ακόλουθη πρόταση προστίθεται στο P^*:

$$q_m^+([l_1|L], X) \leftarrow p_k^+(L, X).$$

Μεταβλητές που εμφανίζονται τόσο στο $\vec{v}_0$ όσο και στο $\vec{v}_2$, αντιμετωπίζονται με τον ίδιο τρόπο.

c) Αν η μεταβλητή εξόδου Y του q εμφανίζεται στην m-οστή θέση του $\vec{v}_2$, τότε η ακόλουθη πρόταση εισάγεται στο P^*:

$$r^+_m([l_2|L], Y) \leftarrow q^-([l_1|L], Y).$$

Περίπτωση 3: Έστω C πρόταση της μορφής:

$$p(\vec{v}_0, Z) \ \leftarrow \ q(\vec{v}_1, Z).$$

και έστω ότι l_3 είναι ο τελεστής διακλαδιζόμενου χρόνου του $q(\vec{v}_1, Z)$. Τότε η C μετασχηματίζεται με τον ακόλουθο τρόπο:

a) Η ακόλουθη πρόταση προστίθεται στο P^*:

$$p^-(L, Z) \leftarrow q^-([l_3|L], Z).$$

b) Έστω X μια μεταβλητή που εμφανίζεται στην k-οστή θέση του $\vec{v}_0$ και επίσης στην m-οστή θέση του $\vec{v}_1$. Τότε, η ακόλουθη πρόταση εισάγεται στο P^*:

$$q^+_m([l_3|L], X) \leftarrow p^+_k(L, X).$$

Περίπτωση 4: Για κάθε EDB κατηγόρημα p με βαθμό $n+1$ του P μια νέα πρόταση της ακόλουθης μορφής εισάγεται στο P^*:

$$p^-(L, Y) \leftarrow p(X_1, \ldots, X_n, Y), p^+_1(L, X_1), \ldots, p^+_n(L, X_n).$$

Παράδειγμα 6.5.1 *Ας θεωρήσουμε το πρόγραμμα P' που προκύπτει από το Παράδειγμα 6.3.2 μαζί με το στόχο (G), όπου τα κατηγορήματα e, f και g είναι EDBs.*

$$(G): \quad \leftarrow \texttt{p(a,1,Y).}$$

Μετασχηματίζοντας το $P' \cup \{G\}$ παίρνουμε ένα νέο στόχο G^*:

$$\leftarrow \texttt{p}^-\texttt{([],Y).}$$

και το πρόγραμμα P^*:

$$
\begin{array}{l}
\texttt{p}_1^+\texttt{([],a).} \\
\texttt{p}_2^+\texttt{([],1).} \\
\texttt{p}^-\texttt{(L,Z)} \leftarrow \texttt{q}^-\texttt{([2|L],Z).} \\
\texttt{q}_2^+\texttt{([2|L],Y)} \leftarrow \texttt{p}_2^+\texttt{(L,Y).} \\
\texttt{q}_1^+\texttt{([2|L],W)} \leftarrow \texttt{e}^-\texttt{([1|L],W).} \\
\texttt{e}_1^+\texttt{(1|L],X)} \leftarrow \texttt{p}_1^+\texttt{(L,X).} \\
\texttt{q}^-\texttt{(L,Z)} \leftarrow \texttt{f}^-\texttt{([4|L],Z).} \\
\texttt{f}_1^+\texttt{([4|L],R)} \leftarrow \texttt{p}^-\texttt{([3|L],R).} \\
\texttt{p}_2^+\texttt{([3|L],Y)} \leftarrow \texttt{q}_2^+\texttt{(L,Y).} \\
\texttt{p}_1^+\texttt{([3|L],W)} \leftarrow \texttt{q}_1^+\texttt{(L,W).} \\
\texttt{p}^-\texttt{(L,Z)} \leftarrow \texttt{g}^-\texttt{([5|L],Z).} \\
\texttt{g}_2^+\texttt{([5|L],Y)} \leftarrow \texttt{p}_2^+\texttt{(L,Y).} \\
\texttt{g}_1^+\texttt{([5|L],X)} \leftarrow \texttt{p}_1^+\texttt{(L,X).} \\
\texttt{e}^-\texttt{(L,Y)} \leftarrow \texttt{e(X,Y),e}_1^+\texttt{(L,X).} \\
\texttt{f}^-\texttt{(L,Y)} \leftarrow \texttt{f(X,Y),f}_1^+\texttt{(L,X).} \\
\texttt{g}^-\texttt{(L,Z)} \leftarrow \texttt{g(X,Y,Z),g}_1^+\texttt{(L,X),g}_2^+\texttt{(L,Y).}
\end{array}
$$

6.6 Η ορθότητα του μετασχηματισμού

Σε αυτή την ενότητα παρουσιάζεται η ορθότητα του μετασχηματισμού.

Έστω P ένα απλό pc-Datalog πρόγραμμα, D μια βάση δεδομένων και $\leftarrow p(a_1, \ldots, a_n, Y)$ ένας στόχος. Η απόδειξη της ορθότητας του μετασχηματισμού γίνεται σε δύο βήματα: πρώτα αποδεικνύουμε ότι κάθε λύση του αρχικού προγράμματος είναι λύση και του τελικού προγράμματος, δηλ. την *πληρότητα (completeness)* του μετασχηματισμού και στη συνέχει ότι κάθε λύση του τελικού είναι λύση και του αρχικού προγράμματος, δηλ. την *ορθότητα (soundness)* του μετασχηματισμού. Ειδικότερα, για να αποδείξουμε τις δύο αυτές ιδιότητες αποδεικνύουμε τα παρακάτω δύο γενικότερα λήμματα από τα οποία εύκολα μπορούν να προκύψουν οι ιδιότητες της ορθότητας και της πληρότητας και κατά συνέπεια πως ο μετασχηματισμός είναι σωστός.

Η ειδίκευση του πρώτου λήμματος δείχνει πως αν ένα αποτιμημένο στιγμιότυπο $p(a_1, \ldots, a_n, b)$ του στόχου είναι ένα λογικό επακόλουθο του P_D, τότε το $p^-([\,], b)$ είναι λογικό επακόλουθο του P^*_D, όπου το $P^* \cup \{\leftarrow p^-([\,], Y)\}$ προκύπτει από την εφαρμογή του μετασχηματισμού στο $P \cup \{\leftarrow p(a_1, \ldots, a_n, Y)\}$. Το αντίστροφο δείχνει πως αν το $p^-([\,], b)$ είναι λογικό επακόλουθο του P^*_D, τότε το $p(a_1, \ldots, a_n, b)$ είναι λογικό επακόλουθο του P_D.

Λήμμα 6.6.1 *Έστω P ένα απλό pc-Datalog πρόγραμμα και D μια βάση δεδομένων. Έστω P^* το Datalog$_{nS}$ πρόγραμμα που προκύπτει από τον μετασχηματισμό. Για όλα τα κατηγορήματα p που ορίζονται στο P_D, όλες τις $L \in List(\mathcal{N})$ και όλες τις σταθερές $a_1, \ldots, a_n, b \in U_{P_D}$, αν $p(a_1, \ldots, a_n, b) \in T_{P_D} \uparrow \omega$ τότε $p^-(L, b) \in T_{P^*_D} \uparrow \omega$.*

Απόδειξη 6.6.1 *Αποδεικνύεται με επαγωγή στις προσεγγίσεις του $T_{P_D} \uparrow \omega$ (βλ. αντίστοιχο Λήμμα στα [31, 37]).*

Η ειδική περίπτωση αυτού του λήμματος είναι:

Λήμμα 6.6.2 *Έστω P ένα απλό pc-Datalog πρόγραμμα, D μία βάση δεδομένων και $\leftarrow p(a_1, \ldots, a_n, Y)$ ένας στόχος. Έστω P^* το Datalog$_{nS}$ πρόγραμμα που προκύπτει εφαρμόζοντας τον αλγόριθμο μετασχηματισμού στο $P \cup \{\leftarrow p(a_1, \ldots, a_n, Y)\}$. Για όλα τα $b \in U_{P_D}$ ισχύει το ακόλουθο: αν $p(a_1, \ldots, a_n, b) \in T_{P_D} \uparrow \omega$, τότε $p^-([\,], b) \in T_{P^*_D} \uparrow \omega$.*

Απόδειξη 6.6.2 *Αφού μετασχηματίζοντας το στόχο, τα γεγονότα $p_i^+([\,], a_i)$, για $i = 1, \ldots, n$, εισάγονται στο P^*, αυτό το λήμμα είναι μια ειδική περίπτωση του Λήμματος 6.6.1.*

Το αντίστροφο λήμμα είναι το ακόλουθο:

Λήμμα 6.6.3 *Έστω P ένα απλό pc-Datalog πρόγραμμα και D μία βάση δεδομένων. Έστω P^* το Datalog$_{nS}$ πρόγραμμα που προκύπτει από το μετασχηματισμό. Για όλα τα κατηγορήματα p που ορίζονται στο P_D, για όλες τις $L \in List(\mathcal{N})$ και για όλα τα $b \in U_{P_D}$, αν $p^-(L, b) \in T_{P^*_D} \uparrow \omega$, τότε υπάρχουν σταθερές $a_1, \ldots, a_n \in U_{P_D}$ τέτοιες ώστε $p(a_1, \ldots, a_n, b) \in T_{P_D} \uparrow \omega$ και $p_i^+(L, a_i) \in T_{P^*_D} \uparrow \omega$, για $i = 1, \ldots, n$.*

Απόδειξη 6.6.3 *Η απόδειξη γίνεται με επαγωγή στις προσεγγίσεις του* $T_{P_D^*} \uparrow \omega$ *(βλ. αντίστοιχο Λήμμα στα [31, 37]).*

Ειδική περίπτωση του λήμματος είναι το ακόλουθο:

Λήμμα 6.6.4 *Έστω P ένα απλό pc-Datalog πρόγραμμα, D μια βάση δεδομένων και $\leftarrow p(a_1, \ldots, a_n, Y)$ ένας στόχος. Έστω P^* το Datalog$_{nS}$ πρόγραμμα που προκύπτει εφαρμόζοντας τον αλγόριθμο μετασχηματισμού στο $P \cup \{\leftarrow p(a_1, \ldots, a_n, Y)\}$. Για όλα τα $b \in U_{P_D}$ ισχύει το ακόλουθο: αν $p^-([\,], b) \in T_{P_D^*} \uparrow \omega$, τότε $p(a_1, \ldots, a_n, b) \in T_{P_D} \uparrow \omega$.*

Απόδειξη 6.6.4 *Καθώς το $p^-([\,], b) \in T_{P_D^*} \uparrow \omega$, από το Λήμμα 6.6.3 έχουμε ότι υπάρχουν σταθερές $c_1, \ldots, c_n \in U_{P_D}$ τέτοιες ώστε το $p(c_1, \ldots, c_n, b) \in T_{P_D} \uparrow \omega$ και τα $p_i^+([\,], c_i) \in T_{P_D^*} \uparrow \omega$, για $i = 1, \ldots, n$. Αλλά, καθώς τα μόνα στιγμιότυπα των $p_i^+([\,], X)$ στο $T_{P_D^*} \uparrow \omega$, είναι τα $p_i^+([\,], a_i)$, συμπεραίνουμε ότι $c_i = a_i$, για $i = 1, \ldots, n$.*

Η ορθότητα του αλγορίθμου μετασχηματισμού αποδεικνύεται από το ακόλουθο θεώρημα.

Θεώρημα 6.6.1 *Έστω P ένα απλό pc-Datalog πρόγραμμα, D μια βάση δεδομένων και $\leftarrow p(a_1, \ldots, a_n, Y)$ ένας στόχος. Έστω P^* το Datalog$_{nS}$ πρόγραμμα που προκύπτει εφαρμόζοντας τον αλγόριθμο μετασχηματισμού στο $P \cup \{\leftarrow p(a_1, \ldots, a_n, Y)\}$. Για όλα τα $b \in U_{P_D}$ ισχύει το ακόλουθο: $p^-([\,], b) \in T_{P_D^*} \uparrow \omega$ αν και μόνο αν $p(a_1, \ldots, a_n, b) \in T_{P_D} \uparrow \omega$.*

Απόδειξη 6.6.5 *Η απόδειξη προκύπτει από τα Λήμματα 6.6.2 και 6.6.4.*

6.7 Βελτιώσεις του μετασχηματισμού

Σε αυτή την ενότητα παρουσιάζουμε συγκεκριμένες προτάσεις που βελτιώνουν το πρόγραμμα που προκύπτει από το μετασχηματισμό, λαμβάνοντας υπόψη μας συγκεκριμένα χαρακτηριστικά του αρχικού και του τελικού προγράμματος. Οι βελτιστοποιήσεις αυτής της ενότητας είναι παρόμοιες με αυτές που παρουσιάζονται στο [37] και στην Ενότητα 5.2.

(a) Όλα τα κατηγορήματα του προγράμματος που προκύπτουν από τον μετασχηματισμό που αντιστοιχούν σε EDB κατηγορήματα του αρχικού προγράμματος, όπως επίσης και οι προτάσεις που τα ορίζουν στο P^*, μπορούν να απαλειφθούν με τη χρήση του ξεδιπλώματος (unfolding) [16]. Για περισσότερες λεπτομέρειες μπορείτε να συμβουλευτείται το [37] και την Ενότητα 5.2.

Παράδειγμα 6.7.1 *Εφαρμόζοντας αυτή την βελτίωση στο τελικό πρόγραμμα P^* του Παραδείγματος 6.5.1 παίρνουμε:*

$$
\begin{array}{l}
\leftarrow \texttt{p}^-\texttt{([],Y)}.\\
\texttt{p}_1^+\texttt{([],a)}.\\
\texttt{p}_2^+\texttt{([],1)}.\\
\texttt{p}^-\texttt{(L,Z)} \leftarrow \texttt{q}^-\texttt{([2|L],Z)}.\\
\texttt{q}_2^+\texttt{([2|L],Y)} \leftarrow \texttt{p}_2^+\texttt{(L,Y)}.\\
\texttt{q}_1^+\texttt{([2|L],W)} \leftarrow \texttt{e(X,W)},\ \texttt{p}_1^+\texttt{(L,X)}.\\
\texttt{q}^-\texttt{(L,Z)} \leftarrow \texttt{f(R,Z)},\ \texttt{p}^-\texttt{([3|L],R)}.\\
\texttt{p}_2^+\texttt{([3|L],Y)} \leftarrow \texttt{q}_2^+\texttt{(L,Y)}.\\
\texttt{p}_1^+\texttt{([3|L],W)} \leftarrow \texttt{q}_1^+\texttt{(L,W)}.\\
\texttt{p}^-\texttt{(L,Z)} \leftarrow \texttt{g(X,Y,Z)},\ \texttt{p}_1^+\texttt{(L,X)},\ \texttt{p}_2^+\texttt{(L,Y)}.
\end{array}
$$

(b) Στην παρουσίαση του μετασχηματισμού υποθέσαμε ότι σε κάθε άτομο στο σώμα έχει ανατεθεί μία διαφορετική ετικέτα l_i. Ωστόσο, μπορούμε εύκολα να δούμε ότι μερικές ετικέτες είναι περιττές. Ειδικότερα, δεν είναι απαραίτητο να αναθέσουμε μία ετικέτα l_i σε άτομα των οποίων τα κατηγορήματα εμφανίζονται μόνο μια φορά μεταξύ των ατόμων των σωμάτων του $P \cup \{G\}$. Επομένως, μπορούμε να αναδιατυπώσουμε τις Περιπτώσεις 2 και 3 του αλγορίθμου μετασχηματισμού, με τον παρακάτω τρόπο:

Περίπτωση 2': Από μια πρόταση της μορφής:

$$p(\vec{v}_0, Z) \ \leftarrow \ q(\vec{v}_1, Y), r(\vec{v}_2, Z).$$

αντί για τις προτάσεις στα a), b) και c) της Περίπτωσης 2 του αλγορίθμου μετασχηματισμού, τώρα παίρνουμε προτάσεις της ακόλουθης μορφής, αντίστοιχα:

$$
\begin{array}{l}
p^-(L,Z) \leftarrow r^-(Lab_j, Z).\\
q_m^+(Lab_i, X) \leftarrow p_k^+(L, X).\\
r_m^+(Lab_j, Y) \leftarrow q^-(Lab_i, Y).
\end{array}
$$

όπου το Lab_j είναι $[l_j|L]$ αν υπάρχει κάποιο άλλο άτομο στο σώμα του $P \cup \{G\}$, με το ίδιο κατηγόρημα r, διαφορετικά το Lab_j είναι L. Παρόμοια για το Lab_i. Η Περίπτωση 3 του αλγορίθμου μπορεί να εκλεπτυνθεί με παρόμοιο τρόπο.

Παράδειγμα 6.7.2 *Αν εφαρμόσουμε το παραπάνω στο Παράδειγμα 6.7.1, παίρνουμε το πρόγραμμα που ακολουθεί, στο οποίο όλες οι ετικέτες, εκτός από την 3, έχουν απαλειφθεί. Παρατηρούμε ότι η ετικέτα 3 δεν μπορεί να απαλειφθεί, διότι το αντίστοιχο κατηγόρημα p εμφανίζεται δύο φορές στο*

$P \cup \{G\}$:

$\leftarrow$ `p`$^-$`([],Y).`
`p`$_1^+$`([],a).`
`p`$_2^+$`([],1).`
`p`$^-$`(L,Z)` $\leftarrow$ `q`$^-$`(L,Z).`
`q`$_2^+$`(L,Y)` $\leftarrow$ `p`$_2^+$`(L,Y).`
`q`$_1^+$`(L,W)` $\leftarrow$ `e(X,W), p`$_1^+$`(L,X).`
`q`$^-$`(L,Z)` $\leftarrow$ `f(R,Z), p`$^-$`([3|L],R).`
`p`$_2^+$`([3|L],Y)` $\leftarrow$ `q`$_2^+$`(L,Y).`
`p`$_1^+$`([3|L],W)` $\leftarrow$ `q`$_1^+$`(L,W).`
`p`$^-$`(L,Z)` $\leftarrow$ `g(X,Y,Z), p`$_1^+$`(L,X), p`$_2^+$`(L,Y).`

(c) Στο μετασχηματισμό διακλαδιζόμενου χρόνου για Chain Datalog προγράμματα που παρουσιάστηκε στο [37], είχε δειχθεί ότι οι χρονικοί τελεστές (αντίστοιχα ετικέτες) που αντιστοιχούν σε αριστερές αναδρομικές κλήσεις μπορούν να διαγραφούν. Δυστυχώς κάτι τέτοιο δεν ισχύει, γενικά, στο νέο μετασχηματισμό. Μπορούμε μόνο να απαλείψουμε τέτοιες ετικέτες, όταν αυτές προκύπτουν από το μετασχηματισμό προτάσεων της παρακάτω μορφής:

$$p(X_1, X_2, ..., X_n, Z) \leftarrow p(X_1, X_2, ..., X_n, Y), q(Y, Z).$$

δηλαδή, όταν η σειρά εμφάνισης των μεταβλητών εισόδου της κεφαλής είναι η ίδια με αυτή της σειράς εμφάνισης στο πρώτο άτομο στο σώμα της πρότασης. Δυστυχώς, στην πιο γενική περίπτωση, όπου οι εμφανίσεις των μεταβλητών εισόδου `p` είναι "ανακατεμένες", αυτή η βελτίωση δεν μπορεί να εφαρμοστεί. Για παράδειγμα, αν έχουμε την πρόταση:

`p(X,Y,Z)` $\leftarrow$ `p(Y,X,W),f(W,Z).`

τότε η ετικέτα που αντιστοιχεί στο `p(Y,X,W)` δεν μπορεί να απαλειφθεί. Ακολουθεί ένα παράδειγμα που δείχνει ότι απαλείφοντας τις ετικέτες σε μια τέτοια περίπτωση θα παίρναμε και λάθος απαντήσεις:

Παράδειγμα 6.7.3

$\leftarrow$ `p(a,1,Y).`
`p(X,Y,Z)` $\leftarrow$ `p(Y,X,W),f(W,Z).`
`p(X,Y,Z)` $\leftarrow$ `e(X,Y,Z).`

και βάση δεδομένων D:

`e(a,1,56).`   `f(56,20).`

Το αρχικό πρόγραμμα έχει μια μοναδική λύση, την $Y = 56$.

Αρχικά, από τον μετασχηματισμό θα πάρουμε το ακόλουθο Datalog$_{nS}$ *πρόγραμμα:*

$$
\begin{array}{l}
\texttt{p}_1^+\texttt{([],a).} \\
\texttt{p}_2^+\texttt{([],1).} \\
\texttt{p}^-\texttt{(L,Z)} \leftarrow \texttt{f}^-\texttt{([2|L],Z).} \\
\texttt{f}_1^+\texttt{([2|L],W)} \leftarrow \texttt{p}^-\texttt{([2|L],W).} \\
\texttt{p}_2^+\texttt{([1|L],X)} \leftarrow \texttt{p}_1^+\texttt{(L,X).} \\
\texttt{p}_1^+\texttt{([1|L],Y)} \leftarrow \texttt{p}_2^+\texttt{(L,Y).} \\
\texttt{p}^-\texttt{(L,Z)} \leftarrow \texttt{e}^-\texttt{([3|L],Z).} \\
\texttt{e}_1^+\texttt{([3|L],X)} \leftarrow \texttt{p}_1^+\texttt{(L,X).} \\
\texttt{e}_2^+\texttt{([3|L],Y)} \leftarrow \texttt{p}_2^+\texttt{(L,Y).} \\
\texttt{e}^-\texttt{(L,Z)} \leftarrow \texttt{e(X,Y,Z),e}_1^+\texttt{(L,X),e}_2^+\texttt{(L,Y).}
\end{array}
$$

Σε περίπτωση που μετασχηματίσουμε το πρόγραμμα εφαρμόζοντας τον προτεινόμενο μετασχηματισμό (απαλείφοντας την ετικέτα 1 που πρέπει να τοποθετηθεί στο `p` *που είναι στο σώμα του πρώτου κανόνα), παίρνουμε:*

$$
\begin{array}{l}
\texttt{p}_1^+\texttt{([],a).} \\
\texttt{p}_2^+\texttt{([],1).} \\
\texttt{p}^-\texttt{(L,Z)} \leftarrow \texttt{f}^-\texttt{([2|L],Z).} \\
\texttt{f}_1^+\texttt{([1|L],W)} \leftarrow \texttt{p}^-\texttt{(L,W).} \\
\texttt{p}_2^+\texttt{(L,X)} \leftarrow \texttt{p}_1^+\texttt{(L,X).} \\
\texttt{p}_1^+\texttt{(L,Y)} \leftarrow \texttt{p}_2^+\texttt{(L,Y).} \\
\texttt{p}^-\texttt{(L,Z)} \leftarrow \texttt{e}^-\texttt{([3|L],Z).} \\
\texttt{e}_1^+\texttt{([3|L],X)} \leftarrow \texttt{p}_1^+\texttt{(L,X).} \\
\texttt{e}_2^+\texttt{([3|L],Y)} \leftarrow \texttt{p}_2^+\texttt{(L,Y).} \\
\texttt{e}^-\texttt{(L,Z)} \leftarrow \texttt{e(X,Y,Z),e}_1^+\texttt{(L,X),e}_2^+\texttt{(L,Y).}
\end{array}
$$

οπότε το τελικό πρόγραμμα θα έχει μια επιπλέον (λανθασμένη) απάντηση, την $Y = 20$.

Η ορθότητα όλων των παραπάνω εκλεπτύνσεων του αλγορίθμου μπορεί εύκολα να αποδειχθεί με κατάλληλες τροποποιήσεις της απόδειξης ορθότητας του μετασχηματισμού. Είναι εύκολο να επιβεβαιώσουμε ότι υπάρχουν συγκεκριμένες υποκλάσεις των pc-Datalog προγραμμάτων για τις οποίες ο μετασχηματισμός παράγει κλασικά Datalog προγράμματα (δηλαδή προγράμματα που δεν περιέχουν καθόλου λίστες). Μια τέτοια υποκλάση περιλαμβάνει κάθε μη-αναδρομικό pc-Datalog πρόγραμμα P στο οποίο κάθε IDB κατηγόρημα εμφανίζεται μόνο μια φορά στα σώματα των προτάσεων του προγράμματος του $P \cup \{\leftarrow G\}$, όπου G είναι ο στόχος. Παρατηρούμε ότι οι εμφανίσεις των EDB κατηγορημάτων του P δεν επηρεάζουν αυτή την ιδιότητα, καθώς όλες οι ετικέτες που εισάγονται από αυτά τα άτομα, μαζί με τα αντίστοιχα IDB κατηγορήματα, απαλείφονται, εφαρμόζοντας την βελτίωση (a).

6.8 Συμπεράσματα

Σε αυτό το κεφάλαιο παρουσιάσαμε μια ειδική περίπτωση του προτεινόμενου μετασχηματισμού σε μια ενδιαφέρουσα κλάση προγραμμάτων Datalog στην οποία επιτρέπονται καταναλώσεις μεταβλητών όχι μόνο σε συνεχόμενα άτομα, χωρίς να επιτρέπονται πολλαπλές καταναλώσεις μεταβλητών (κάτι που θα απαιτούσε κατηγορήματα επιλογής).

Κεφάλαιο 7

Σύγκριση των προτεινόμενων μετασχηματισμών με σχετικές μεθόδους

Σε αυτό το Κεφάλαιο θα παρουσιάσουμε μια σύγκριση των προτεινόμενων μετασχηματισμών με τις ήδη υπάρχουσες τεχνικές. Σκοπός της σύγκρισης είναι να φανούν τα πλεονεκτήματα και τα μειονεκτήματα της κάθε μεθόδου και να γίνει φανερό ποια είναι η πιο κατάλληλη σε κάθε περίπτωση. Τα αποτελέσματα της σύγκρισης αυτής μας επιτρέπουν να οδηγηθούμε στο συμπέρασμα πως οι προτεινόμενοι μετασχηματισμοί πλεονεκτούν έναντι των άλλων μεθόδων σε μια μεγάλη κλάση προγραμμάτων και βάσεων δεδομένων. Αυτό τις καθιστά ενδιαφέρουσες και από μια πρακτική σκοπιά, πέρα από το θεωρητικό ενδιαφέρον που έχουν.

Η δουλειά που παρουσιάστηκε σε αυτή την εργασία συνεισφέρει στην περιοχή των βελτιστοποιήσεων προγραμμάτων στις επαγωγικές βάσεις δεδομένων. Πιο συγκεκριμένα, ο προτεινόμενος μετασχηματισμός ανήκει στην γνωστή κατηγορία τεχνικών, όπου οι τιμές εισόδου του στόχου της επερώτησης διαδίδονται με σκοπό να περιορίσουμε την παραγωγή άσχετων ατόμων κατά τη διάρκεια του bottom-up υπολογισμού της ζητούμενης λύσης. Σε αυτή την ενότητα παρουσιάζουμε μια σύγκριση της προτεινόμενης τεχνικής με τις πιο γνωστές τεχνικές αυτής της κατηγορίας. Η σύγκριση δεν είναι εξαντλητική, καθώς αυτό θα απαιτούσε να διαθέτουμε μια αξιόπιστη υλοποίηση για όλες τις σχετικές τεχνικές και πειραματισμούς με ένα αριθμό προγραμμάτων και βάσεων δεδομένων. Ωστόσο, στη συνέχεια θα προσπαθήσουμε να διακρίνουμε τα βασικά πλεονεκτήματα και μειονεκτήματα της δικιάς μας προσέγγισης και των ανταγωνιστών της.

Μεταξύ των υπαρχουσών Datalog βελτιστοποιήσεων, αυτές που είναι περισσότερο σχετικές με την παρούσα προσέγγιση είναι ο *μετασχηματισμός μαγικών συνόλων (magic sets transformation)* [7, 9, 45], η *μέθοδος μέτρησης (counting method)* [41, 43] και η *προσέγγιση στοίβας (pushdown approach)* [19] που περιγράφηκαν στις Ενότητες 2.4, 2.5 και 2.6. Στα ακόλουθα, συζητάμε τα σχετικά οφέλη και τα μειονεκτήματα καθεμιάς μεθόδου σε σχέση με την δική μας προσέγγιση.

7.1 Σύγκριση με τη μέθοδο magic sets

Ο πιο γνωστός αντιπρόσωπος στην περιοχή των βελτιστοποιήσεων Datalog προγραμμάτων μέσω διάδοσης τιμής είναι ο μετασχηματισμός magic sets, ο οποίος παρουσιάστηκε στην Ενότητα 2.4. Σε αυτόν, για κάθε IDB κατηγόρημα του αρχικού προγράμματος εισάγεται ένα νέο κατηγόρημα, που ονομάζεται *μαγικό κατηγόρημα (magic predicate)*. Τα ορίσματα του μαγικού κατηγορήματος είναι τα δεσμευμένα ορίσματα του αντίστοιχου IDB κατηγορήματος του αρχικού προγράμματος και ο σκοπός είναι να "σπρώξουμε τις επιλογές μέσα στις προτάσεις" ("push selections into clauses"). Ο μετασχηματισμός magic sets μπορεί να εφαρμοστεί σε γενικά Datalog προγράμματα (και σε λογικά προγράμματα) [9, 49, 45] και για αυτό το λόγο είναι ο πιο γενικός από όλους τους παρεμφερείς μετασχηματισμούς. Αυτή η γενικότητα του, ωστόσο, μπορεί να αποδειχθεί και το αδύνατό του σημείο σε ορισμένες περιπτώσεις: όπως ισχυριζόμαστε παρακάτω, ο προτεινόμενος μετασχηματισμός έχει τα ίδια πλεονεκτήματα ως προς τον magic sets όπως και η τεχνική μέτρησης. Πιο ειδικά, είναι γνωστό [49] ότι σε μια ποικιλία βάσεων δεδομένων, η τεχνική counting πλεονεκτεί έναντι του magic sets. Πιο συγκεκριμένα, υπάρχουν προγράμματα που με δεδομένες βάσεις δεδομένων, τερματίζουν σε $O(n)$ χρόνο χρησιμοποιώντας counting, ενώ θέλουν $\Omega(n^2)$ χρόνο με το magic sets. Όπως θα δείξουμε στο τέλος αυτής της ενότητας, αυτό συμβαίνει και με τη δική μας προσέγγιση.

7.2 Σύγκριση με τη μέθοδο counting

Η τεχνική counting χρησιμοποιεί ακέραιους δείκτες για να κωδικοποιήσει δύο υπολογισμούς σταθερού σημείου: έναν για τη διάδοση των δεσμεύσεων που γίνεται στην top-down φάση και τον άλλο για την παραγωγή του επιθυμητού αποτελέσματος στον bottom-up υπολογισμό, όπως είδαμε και στην Ενότητα 2.5. Με άλλα λόγια, οι ακέραιοι δείκτες παίζουν το ρόλο των λιστών του δικού μας μετασχηματισμού. Στην αρχική τη ς εκδοχή, η counting εφαρμόζονταν μόνο σε μια περιορισμένη κλάση επερωτήσεων [49, 20]. Αργότερα επεκτάθηκε στην *γενικευμένη μέθοδο μέτρησης (generalized counting)* [9], που εφαρμόζεται σε μια ευρύτερη κλάση από επερωτήσεις που έχουν την λεγόμενη *ιδιότητα περάσματος δεσμεύσεων (binding-passing property) (BPP)*[43]. Διαισθητικά, η BPP εξασφάλιζε ότι οι "δεσμεύσεις μπορούν να περάσουν σε όλα τα επίπεδα της αναδρομής". Ωστόσο, η κλάση των BPP προγραμμάτων δεν περιλαμβάνει όλα τα Chain Datalog προγράμματα όπως επίσης ούτε όλα τα cc-Datalog. Για παράδειγμα, η διάτρεξη ενός γράφου με χρήση διπλής αναδρομής (το παράδειγμα στην Ενότητα 3.3). Η γενικευμένη counting μέθοδος δεν μπορεί να χειριστεί αυτή την περίπτωση, καθώς το αντίστοιχο πρόγραμμα δεν έχει την BPP ιδιότητα. Αλλά αυτό το πρόγραμμα είναι ένα Chain Datalog πρόγραμμα και για αυτό μπορεί να μετασχηματιστεί από τον παρόντα μετασχηματισμό. Με άλλα λόγια, η counting μέθοδος είναι λιγότερο γενική τόσο από τον branching-time μετασχηματισμό όσο και από τον προτεινόμενο. Ωστόσο, οι δύο προσεγγίσεις έχουν πολλά κοινά χαρακτηριστικά: οι τελικές τους γλώσσες είναι και στις δύο περιπτώσεις μια επέκταση της Datalog και αυτό δημιουργεί μια σειρά από ενδιαφέροντα ερωτήματα που αφορούν την bottom-up εκτέλεση του τελικού κώδικα.

7.3 Σύγκριση με τη μέθοδο pushdown

Η μέθοδος pushdown βασίζεται στη σχέση που έχουν οι επερωτήσεις αλυσίδας (chain queries) με τις γραμματικές χωρίς συμφραζόμενα (context-free languages) [55]. Επιπλέον, η context-free γλώσσα που αντιστοιχεί σε μια επερώτηση σχετίζεται με ένα αυτόματο στοίβας (pushdown automaton). Με αυτό τον τρόπο η αρχική επερώτηση επανεγγράφεται σε μια πιο κατάλληλη μορφή για αποδοτικό bottom-up υπολογισμό. Η *pushdown μέθοδος* εφαρμόζεται σε όλες τις αλυσιδωτές επερωτήσεις [19] και γενικά καλύπτει μια υποκλάση της προτεινόμενης τεχνικής, καθώς δεν μπορεί να χειριστεί την περίπτωση που έχουμε πολλαπλή κατανάλωση μεταβλητών. Η πλήρης περιγραφή της υπάρχει στην Ενότητα 2.6.

7.4 Σύγκριση μέσω παραδειγμάτων

Στη συνέχεια θα συγκρίνουμε τη μέθοδο μας με τις άλλες τεχνικές χρησιμοποιώντας τέσσερα παραδείγματα. Το πρώτο αφορά το γνωστό πρόγραμμα `same generation` (εύρεσης ατόμων που είναι στην ίδια γενιά, που στην πραγματικότητα είναι ένα Chain Datalog πρόγραμμα). Το δεύτερο παράδειγμα είναι το πρόγραμμα `path` (μονοπάτι σε γράφο) που παρουσιάστηκε στην Ενότητα 3.4, που έχει πολλαπλές καταναλώσεις μεταβλητών. Το τρίτο παράδειγμα είναι ένα διπλά αναδρομικό πρόγραμμα που υπολογίζει τα περιττού μήκους μονοπάτια ενός δοσμένου χρώματος σε έναν γράφο. Τέλος, το τελευταίο παράδειγμα είναι το τρέχον παράδειγμα του προηγούμενου Κεφαλαίου.

Το παράδειγμα same generation

Ας θεωρήσουμε το παρακάτω πρόγραμμα εύρεσης των ατόμων ίδιας γενιάς, που είναι κοινό στην περιοχή των επαγωγικών ΒΔ:

$$\begin{array}{l} \leftarrow \texttt{sg(a,Z)}. \\ \texttt{sg(X,Y)} \leftarrow \texttt{equal(X,Y)}. \\ \texttt{sg(X,Y)} \leftarrow \texttt{par(X,Xp),sg(Xp,Yp),rap(Yp,Y)}. \end{array}$$

μαζί με τη βάση δεδομένων που αποτελείται από τις μοναδιαίες προτάσεις $\texttt{par(a,b}_i\texttt{)}$, $\texttt{par(b}_i\texttt{,c)}$, $\texttt{rap(b}_i\texttt{,a)}$ και $\texttt{rap(c,b}_i\texttt{)}$, όπου $1 \leq i \leq n$ και `equal` είναι το συνηθισμένο κατηγόρημα ισότητας (αυτό είναι μια ελαφρώς διαφοροποιημένη εκδοχή του προγράμματος που δίνεται στο [49], σελίδα 831). Παρατηρούμε ότι το πρόγραμμα είναι ένα Chain Datalog πρόγραμμα (που δεν έχει πολλαπλές καταναλώσεις μεταβλητών). Ο προτεινόμενος μετασχηματισμός (fixed mode) όταν εφαρμόζεται στο παραπάνω πρόγραμμα δίνει σαν αποτέλεσμα, μετά και από τις κατάλληλες βελτιστοποιήσεις που περιγράφονται στην Υποενότητα 5.2.2:

```
← sg2([],Z).
sg1([],a).
sg2(L,Y) ← equal(X,Y),sg1(L,X).
sg2(L,Y) ← rap(X,Y),sg2([5,2|L],X).
sg1([5,2|L],Xp) ← par(X,Xp),sg1(L,X).
```

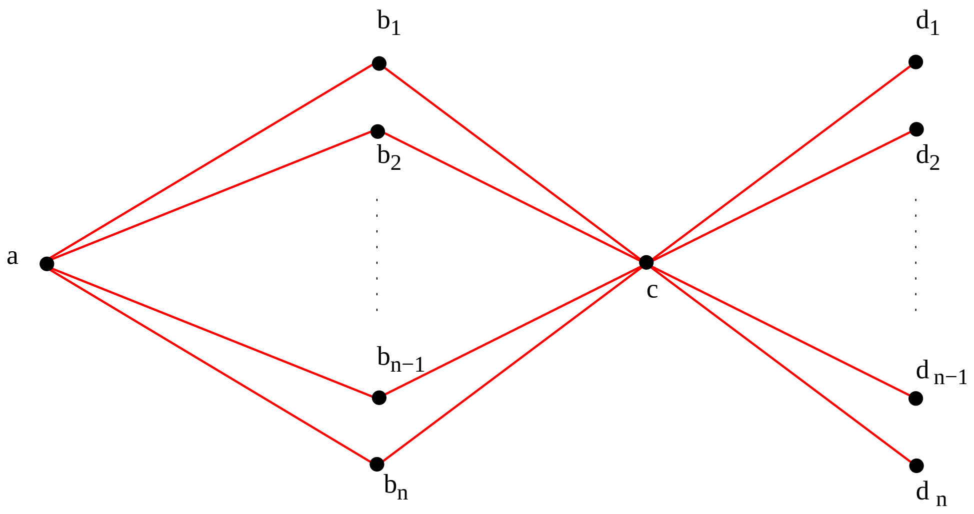
Μπορούμε εύκολα να παρατηρήσουμε ότι το παραπάνω πρόγραμμα τερματίζει παράγοντας την επιθυμητή απάντηση (Z = a) σε χρόνο $O(n)$. Από την άλλη μεριά, το αντίστοιχο magic sets πρόγραμμα (από [49], σελίδα 856) απαιτεί χρόνο $\Omega(n^2)$ (βλέπε επίσης τη συζήτηση στο [49], σελίδα 947). Με άλλα λόγια, ο μετασχηματισμός μας φαίνεται να δίνει ενθαρρυντικά αποτελέσματα σε σχέση με τον magic sets.

Το πρόγραμμα colored path

Έστω το ακόλουθο cc-Datalog πρόγραμμα του Παραδείγματος 3.4.2:

```
← path(a,red,Z).
path(X,Color,Z) ← edge(X,Color,Z).
path(X,Color,Z) ← edge(X,Color,W),path(W,Color,Z).
```

μαζί με τη βάση δεδομένων: edge(a,red,b_i), edge(b_i,red,c), edge(c,red,d_i), $i = 1, ..., n$, που φαίνεται στο Σχήμα 7.1.



Σχήμα 7.1: *Ο γράφος του παραδείγματος colored path*

Το τελικό πρόγραμμα που προκύπτει από την εφαρμογή του προτεινόμενου μετασχηματισμού (fixed mode μετασχηματισμός) και τα κατάλληλα unfoldings είναι το ακόλουθο:

$\leftarrow$ `path`$^-$`([],Z).`
`path`$_1^+$`([],a).`
`path`$_2^+$`([],red).`
`path`$^-$`(L,Z)` $\leftarrow$ `edge(X,Color,Z),path`$_1^+$`(L,X),path`$_2^+$`(L,Color).`
`path`$^-$`(L,Z)` $\leftarrow$ `path`$^-$`([3|L],Z).`
`path`$_1^+$`([3|L],W)` $\leftarrow$ `edge(X,Color,W),path`$_1^+$`(L,X),#path`$_2^+$`(L,Color).`
`path`$_2^+$`([3|L],Color)` $\leftarrow$ `#path`$_2^+$`(L,Color).`

Είναι εύκολο να δούμε πως το Λήμμα 5.2.3 μπορεί να εφαρμοστεί στο σύνολο $\{$`path`$_2^+\}$, για αυτό το κατηγόρημα `path`$_2^+$ είναι ντετερμινιστικό ως προς τη δοσμένη βάση δεδομένων (στην πραγματικότητα ως προς οποιαδήποτε βάση δεδομένων), γιατί σε κάθε περιβάλλον, το όρισμα δεδομένων έχει την τιμή `red`. Για αυτό τα άτομα επιλογής μπορούν να αντικατασταθούν από τις κλασικές τους εκδόσεις δίνοντας ένα πρόγραμμα που είναι απαλλαγμένο από κατηγορήματα επιλογής.

$\leftarrow$ `path`$^-$`([],Z).`
`path`$_1^+$`([],a).`
`path`$_2^+$`([],red).`
`path`$^-$`(L,Z)` $\leftarrow$ `edge(X,Color,Z),path`$_1^+$`(L,X),path`$_2^+$`(L,Color).`
`path`$^-$`(L,Z)` $\leftarrow$ `path`$^-$`([3|L],Z).`
`path`$_1^+$`([3|L],W)` $\leftarrow$ `edge(X,Color,W),path`$_1^+$`(L,X),path`$_2^+$`(L,Color).`
`path`$_2^+$`([3|L],Color)` $\leftarrow$ `path`$_2^+$`(L,Color).`

Επιπλέον είναι εύκολο να δούμε ότι τα άτομα που το κατηγόρημά τους είναι το `path`$_2^+$ μπορούν να φύγουν τελείως, δίνοντας έτσι το ακόλουθο πρόγραμμα:

$\leftarrow$ `path`$^-$`([],Z).`
`path`$_1^+$`([],a).`
`path`$^-$`(L,Z)` $\leftarrow$ `edge(X,red,Z),path`$_1^+$`(L,X).`
`path`$^-$`(L,Z)` $\leftarrow$ `path`$^-$`([3|L],Z).`
`path`$_1^+$`([3|L],W)` $\leftarrow$ `edge(X,red,W),path`$_1^+$`(L,X).`

Οι απαντήσεις στον στόχο (`Z = b`$_i$`, c, d`$_i$, για $i = 1, ..., n$) παράγονται σε $O(n)$ χρόνο. Αυτό φαίνεται στο Σχήμα 7.2.

Από την άλλη μεριά, το πρόγραμμα που προκύπτει από το μετασχηματισμό magic sets, που δίνεται παρακάτω:

```
m_path(a,red).
m_path(W,Color) ← m_path(X,Color),edge(X,Color,W).
path(X,Color,Z) ← m_path(X,Color),edge(X,Color,Z).
path(X,Color,Z) ← m_path(X,Color),edge(X,Color,W),path(W,Color,Z).
```

παράγει όλα τα γεγονότα `path(b`$_i$`,red,d`$_j$`)`, για $i, j = 1, .., n$, δηλ. όλα τα μονοπάτια ανάμεσα στα `b`$_i$ και `d`$_j$, για $i, j = 1, ..., n$ και για αυτό χρειάζεται χρόνο $\Omega(n^2)$. Αυτό φαίνεται παρακάτω στο Σχήμα 7.3.

Ακόμα και οι πιο εξειδικευμένες λύσεις των magic sets, όπως οι *supplementary magic sets* [9], δεν προσφέρουν κάποια ουσιώδη βελτίωση, καθώς πάλι $\Omega(n^2)$

$$\begin{array}{c}
\mathtt{path}_1^+([\,],\mathtt{a}).\\
\downarrow\\
\mathtt{path}^-([\,],\mathtt{b}_1).\ \cdots\ \mathtt{path}^-([\,],\mathtt{b}_n).\\
\mathtt{path}_1^+([3],\mathtt{b}_1).\ \cdots\ \mathtt{path}_1^+([3],\mathtt{b}_n).\\
\downarrow\\
\mathtt{path}^-([3],\mathtt{c}).\\
\mathtt{path}_1^+([3,3],\mathtt{c}).\\
\downarrow\\
\mathtt{path}^-([\,],\mathtt{c}).\\
\mathtt{path}^-([3,3],\mathtt{d}_1).\ \cdots\ \mathtt{path}^-([3,3],\mathtt{d}_n).\\
\mathtt{path}_1^+([3,3,3],\mathtt{d}_1).\ \cdots\ \mathtt{path}_1^+([3,3,3],\mathtt{d}_n).\\
\downarrow\\
\mathtt{path}^-([3],\mathtt{d}_1).\ \cdots\ \mathtt{path}^-([3],\mathtt{d}_n).\\
\downarrow\\
\mathtt{path}^-([\,],\mathtt{d}_1).\ \cdots\ \mathtt{path}^-([\,],\mathtt{d}_n).
\end{array}$$

Σχήμα 7.2: *Υπολογισμός του παραδείγματος colored path με την προτεινόμενη μέθοδο*

$$\begin{array}{c}
\mathtt{m_path}(\mathtt{a},\mathtt{red}).\\
\downarrow\\
\mathtt{path}(\mathtt{a},\mathtt{red},\mathtt{b}_1).\ \cdots\ \mathtt{path}(\mathtt{a},\mathtt{red},\mathtt{b}_n).\\
\mathtt{m_path}(\mathtt{b}_1,\mathtt{red}).\ \cdots\ \mathtt{m_path}(\mathtt{b}_n,\mathtt{red}).\\
\downarrow\\
\mathtt{m_path}(\mathtt{c},\mathtt{red}).\\
\mathtt{path}(\mathtt{b}_1,\mathtt{red},\mathtt{c}).\ \cdots\ \mathtt{path}(\mathtt{b}_n,\mathtt{red},\mathtt{c}).\\
\downarrow\\
\mathtt{m_path}(\mathtt{d}_1,\mathtt{red}).\ \cdots\ \mathtt{m_path}(\mathtt{d}_n,\mathtt{red}).\\
\mathtt{path}(\mathtt{c},\mathtt{red},\mathtt{d}_1).\ \cdots\ \mathtt{path}(\mathtt{c},\mathtt{red},\mathtt{d}_n).\\
\mathtt{path}(\mathtt{a},\mathtt{red},\mathtt{c}).\\
\downarrow\\
\mathtt{path}(\mathtt{b}_1,\mathtt{red},\mathtt{d}_1).\ \cdots\ \mathtt{path}(\mathtt{b}_i,\mathtt{red},\mathtt{d}_j).\ \cdots\ \mathtt{path}(\mathtt{b}_n,\mathtt{red},\mathtt{d}_n).\\
\downarrow\\
\mathtt{path}(\mathtt{a},\mathtt{red},\mathtt{d}_1).\ \cdots\ \mathtt{path}(\mathtt{a},\mathtt{red},\mathtt{d}_n).
\end{array}$$

Σχήμα 7.3: *Υπολογισμός του παραδείγματος colored path με τη μέθοδο magic sets*

άτομα πρέπει να παραχθούν. Στην ουσία σε αυτή την παραλλαγή των magic sets, οι βοηθητικές σχέσεις `sup` φυλάνε κάποιες ενδιάμεσες τιμές προκειμένου να μην έχουμε επανάληψη υπολογισμών.

$$
\begin{array}{l}
\texttt{m\_path(a,red).} \\
\texttt{m\_path(W,Color)} \leftarrow \texttt{sup}_2^2\texttt{\_path(X,Color,W).} \\
\texttt{sup}_2^2\texttt{\_path(X,Color,W)} \leftarrow \texttt{m\_path(X,Color),edge(X,Color,W).} \\
\texttt{path(X,Color,Z)} \leftarrow \texttt{m\_path(X,Color),edge(X,Color,Z).} \\
\texttt{path(X,Color,Z)} \leftarrow \texttt{sup}_2^2\texttt{\_path(X,Color,W),path(W,Color,Z).}
\end{array}
$$

Η αποτίμηση του προγράμματος φαίνεται στο Σχήμα 7.4.

$$
\begin{array}{c}
\texttt{m\_path}(\mathrm{a},\texttt{red}). \\
\downarrow \\
\texttt{sup}_2^2(\mathrm{a},\texttt{red},\mathrm{b_1}).\ \cdots\ \texttt{sup}_2^2(\mathrm{a},\texttt{red},\mathrm{b_n}). \\
\texttt{path}(\mathrm{a},\texttt{red},\mathrm{b_1}).\ \cdots\ \texttt{path}(\mathrm{a},\texttt{red},\mathrm{b_n}). \\
\downarrow \\
\texttt{m\_path}(\mathrm{b_1},\texttt{red}).\ \ldots\ \texttt{m\_path}(\mathrm{b_n},\texttt{red}). \\
\downarrow \\
\texttt{sup}_2^2(\mathrm{b_1},\texttt{red},\mathrm{c}).\ \cdots\ \texttt{sup}_2^2(\mathrm{b_n},\texttt{red},\mathrm{c}). \\
\texttt{path}(\mathrm{b_1},\texttt{red},\mathrm{c}).\ \cdots\ \texttt{path}(\mathrm{b_n},\texttt{red},\mathrm{c}). \\
\downarrow \\
\texttt{m\_path}(\mathrm{c},\texttt{red}). \\
\texttt{path}(\mathrm{a},\texttt{red},\mathrm{c}). \\
\downarrow \\
\texttt{sup}_2^2(\mathrm{c},\texttt{red},\mathrm{d_1}).\ \cdots\ \texttt{sup}_2^2(\mathrm{c},\texttt{red},\mathrm{d_n}). \\
\texttt{path}(\mathrm{c},\texttt{red},\mathrm{d_1}).\ \cdots\ \texttt{path}(\mathrm{c},\texttt{red},\mathrm{d_n}). \\
\downarrow \\
\texttt{m\_path}(\mathrm{d_1},\texttt{red}).\ \ldots\ \texttt{m\_path}(\mathrm{d_n},\texttt{red}). \\
\texttt{path}(\mathrm{b_1},\texttt{red},\mathrm{d_1}).\ \cdots\ \texttt{path}(\mathrm{b_n},\texttt{red},\mathrm{d_n}). \\
\downarrow \\
\texttt{path}(\mathrm{a},\texttt{red},\mathrm{d_1}).\ \cdots\ \texttt{path}(\mathrm{a},\texttt{red},\mathrm{d_n}).
\end{array}
$$

Σχήμα 7.4: *Υπολογισμός του παραδείγματος colored path με τη μέθοδο supplementary magic sets*

Η μέθοδος counting δίνει το ακόλουθο πρόγραμμα:

$$
\begin{array}{l}
\texttt{c\_path(0,a,red).} \\
\texttt{c\_path(I+1,W,Color)} \leftarrow \texttt{c\_path(I,X,Color),edge(X,Color,W).} \\
\texttt{path(I,Z)} \leftarrow \texttt{c\_path(I,X,Color),edge(X,Color,Z).} \\
\texttt{path(I,Z)} \leftarrow \texttt{path(I+1,Z).}
\end{array}
$$

Μια από-κάτω-προς-τα-πάνω αποτίμηση του προγράμματος υπολογίζει τις απαντήσεις στο στόχο $\leftarrow$ `path(0,Z)` σε χρόνο $O(n)$, όπως φαίνεται και στο Σχήμα 7.5. Ακόμα και η supplementary counting method χρειάζεται $O(n)$ χρόνο:

$$
\begin{array}{c}
\mathtt{c\_path}(0,\mathtt{a},\mathtt{red}).\\
\downarrow\\
\mathtt{path}(0,\mathtt{b_1}).\ \cdots\ \mathtt{path}(0,\mathtt{b_n}).\\
\mathtt{c\_path}(1,\mathtt{b_1},\mathtt{red}).\ \cdots\ \mathtt{c\_path}(1,\mathtt{b_n},\mathtt{red}).\\
\downarrow\\
\mathtt{c\_path}(2,\mathtt{c},\mathtt{red}).\\
\mathtt{path}(1,\mathtt{c}).\\
\downarrow\\
\mathtt{c\_path}(3,\mathtt{d_1},\mathtt{red}).\ \cdots\ \mathtt{c\_path}(3,\mathtt{d_n},\mathtt{red}).\\
\mathtt{path}(2,\mathtt{d_1}).\ \cdots\ \mathtt{path}(2,\mathtt{d_n}).\\
\mathtt{path}(0,\mathtt{c}).\\
\downarrow\\
\mathtt{path}(1,\mathtt{d_1}).\ \cdots\ \mathtt{path}(1,\mathtt{d_n}).\\
\downarrow\\
\mathtt{path}(0,\mathtt{d_1}).\ \cdots\ \mathtt{path}(0,\mathtt{d_n}).
\end{array}
$$

Σχήμα 7.5: *Υπολογισμός του παραδείγματος colored path με τη μέθοδο counting*

```
c_path(0,a,red).
supcntc₂²(I,X,Color,W) ← c_path(I,X,Color),edge(X,Color,W).
c(I,W,Color) ← supcnt₂²(I,X,Color,W).
path(I,Z) ← c_path(I,X,Color),edge(X,Color,Z).
path(I,Z) ← path(I+1,Z).
```

Μια από-κάτω-προς-τα-πάνω αποτίμηση του προγράμματος υπολογίζει τις απαντήσεις όπως δείχνει και το Σχήμα 7.6.

Το πρόγραμμα odd length colored path

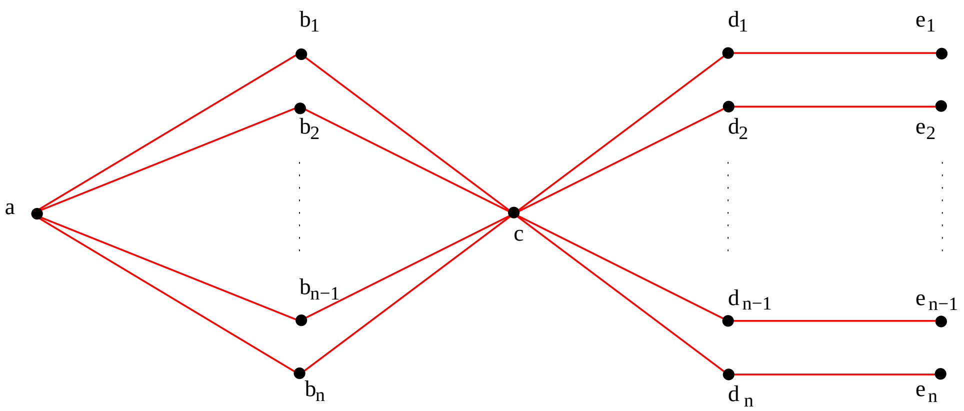
Έστω το cc-Datalog πρόγραμμα που βρίσκει όλες τις κορυφές ενός γράφου που είναι προσβάσιμες από μια κορυφή a μέσω περιττού μήκους μονοπάτια (με διπλή αναδρομή):

```
← path(a,red,Y).
path(X,C,Y) ← edge(X,C,Y).
path(X,C,Y) ← edge(X,C,Z), path(Z,C,R), path(R,C,Y).
```

μαζί με τη βάση δεδομένων που φαίνεται στο Σχήμα 7.7: $\mathtt{edge(a,red,b}_i\mathtt{)}$, $\mathtt{edge(b}_i\mathtt{,red,c)}$, $\mathtt{edge(c,red,d}_i\mathtt{)}$ και $\mathtt{edge(d}_i\mathtt{,red,e}_i\mathtt{)}$, για $i = 1, ..., n$.

$$
\begin{array}{c}
\texttt{c\_path}(0,\texttt{a},\texttt{red}).\\
\downarrow\\
\texttt{sup}_2^2(0,\texttt{a},\texttt{red},\texttt{b}_1).\ \cdots\ \texttt{sup}_2^2(0,\texttt{a},\texttt{red},\texttt{b}_\texttt{n}).\\
\texttt{path}(0,\texttt{b}_1).\ \cdots\ \texttt{path}(0,\texttt{b}_\texttt{n}).\\
\downarrow\\
\texttt{c\_path}(1,\texttt{b}_1,\texttt{red}).\ \ldots\ \texttt{c\_path}(1,\texttt{b}_\texttt{n},\texttt{red}).\\
\downarrow\\
\texttt{sup}_2^2(1,\texttt{b}_1,\texttt{red},\texttt{c}).\ \cdots\ \texttt{sup}_2^2(1,\texttt{b}_\texttt{n},\texttt{red},\texttt{c}).\\
\texttt{path}(1,\texttt{red},\texttt{c}).\\
\downarrow\\
\texttt{c\_path}(2,\texttt{c},\texttt{red}).\\
\texttt{path}(0,\texttt{c}).\\
\downarrow\\
\texttt{sup}_2^2(2,\texttt{c},\texttt{red},\texttt{d}_1).\ \cdots\ \texttt{sup}_2^2(2,\texttt{c},\texttt{red},\texttt{d}_\texttt{n}).\\
\texttt{path}(2,\texttt{d}_1).\ \cdots\ \texttt{path}(2,\texttt{d}_\texttt{n}).\\
\downarrow\\
\texttt{c\_path}(3,\texttt{d}_1,\texttt{red}).\ \ldots\ \texttt{c\_path}(3,\texttt{d}_\texttt{n},\texttt{red}).\\
\texttt{path}(1,\texttt{d}_1).\ \cdots\ \texttt{path}(1,\texttt{d}_\texttt{n}).\\
\downarrow\\
\texttt{path}(0,\texttt{d}_1).\ \cdots\ \texttt{path}(0,\texttt{d}_\texttt{n}).
\end{array}
$$

Σχήμα 7.6: *Υπολογισμός του παραδείγματος colored path με την supplementary counting μέθοδο*



Σχήμα 7.7: *Ο γράφος του παραδείγματος odd length colored path*

Το τελικό πρόγραμμα ύστερα από την εφαρμογή του προτεινόμενου μετασχηματισμού (στο αρχικό και όχι στο απλό πρόγραμμα, όπως ορίστηκε στον Ορισμό 3.4.3) και των κατάλληλων unfoldings είναι:

$\leftarrow$ `path⁻([],Z).`
$\texttt{path}_1^+\texttt{([],a).}$
$\texttt{path}_2^+\texttt{([],red).}$
$\texttt{path}^-\texttt{(L,Y)} \leftarrow \texttt{edge(X,C,Y), path}_1^+\texttt{(L,X), path}_2^+\texttt{(L,C).}$
$\texttt{path}^-\texttt{(L,Y)} \leftarrow \texttt{path}^-\texttt{([4|L],Y).}$
$\texttt{path}_2^+\texttt{([4|L],C)} \leftarrow \texttt{\#path}_2^+\texttt{(L,C).}$
$\texttt{path}_2^+\texttt{([3|L],C)} \leftarrow \texttt{\#path}_2^+\texttt{(L,C).}$
$\texttt{path}_1^+\texttt{([4|L],R)} \leftarrow \texttt{path}^-\texttt{([3|L],R).}$
$\texttt{path}_1^+\texttt{([3|L],Z)} \leftarrow \texttt{edge(X,C,Z), path}_1^+\texttt{(L,X), \#path}_2^+\texttt{(L,C).}$

ή μετά την αντικατάσταση των κατηγορημάτων επιλογής από τα αντίστοιχα κλασικά τους είναι το ακόλουθο:

$\leftarrow \texttt{path}^-\texttt{([],Z).}$
$\texttt{path}_1^+\texttt{([],a).}$
$\texttt{path}_2^+\texttt{([],red).}$
$\texttt{path}^-\texttt{(L,Y)} \leftarrow \texttt{edge(X,C,Y), path}_1^+\texttt{(L,X), path}_2^+\texttt{(L,C).}$
$\texttt{path}^-\texttt{(L,Y)} \leftarrow \texttt{path}^-\texttt{([4|L],Y).}$
$\texttt{path}_2^+\texttt{([4|L],C)} \leftarrow \texttt{path}_2^+\texttt{(L,C).}$
$\texttt{path}_2^+\texttt{([3|L],C)} \leftarrow \texttt{path}_2^+\texttt{(L,C).}$
$\texttt{path}_1^+\texttt{([4|L],R)} \leftarrow \texttt{path}^-\texttt{([3|L],R).}$
$\texttt{path}_1^+\texttt{([3|L],Z)} \leftarrow \texttt{edge(X,C,Z), path}_1^+\texttt{(L,X), path}_2^+\texttt{(L,C).}$

και μετά την απομάκρυνση του κατηγορήματος $\texttt{path}_2^+$, με παρόμοιο τρόπο όπως στο παράδειγμα colored path, έχουμε το ακόλουθο:

$\leftarrow \texttt{path}^-\texttt{([],Z).}$
$\texttt{path}_1^+\texttt{([],a).}$
$\texttt{path}^-\texttt{(L,Y)} \leftarrow \texttt{edge(X,red,Y), path}_1^+\texttt{(L,X).}$
$\texttt{path}^-\texttt{(L,Y)} \leftarrow \texttt{path}^-\texttt{([4|L],Y).}$
$\texttt{path}_1^+\texttt{([4|L],R)} \leftarrow \texttt{path}^-\texttt{([3|L],R).}$
$\texttt{path}_1^+\texttt{([3|L],Z)} \leftarrow \texttt{edge(X,red,Z), path}_1^+\texttt{(L,X).}$

Οι απαντήσεις στο στόχο ($\texttt{Z = b}_i\texttt{, d}_i$, για $i = 1, ..., n$) παράγονται σε χρόνο $O(n)$ (καθώς μόνο $O(n)$ άτομα παράγονται κατά τη διάρκεια του υπολογισμού) όπως φαίνεται στο Σχήμα 7.8.

Από την άλλη, το αντίστοιχο magic sets πρόγραμμα:

$\texttt{m\_path(a,red).}$
$\texttt{m\_path(Z,C)} \leftarrow \texttt{m\_path(X,C), edge(X,C,Z).}$
$\texttt{m\_path(R,C)} \leftarrow \texttt{m\_path(X,C), edge(X,C,Z), path(Z,C,R).}$
$\texttt{path(X,C,Y)} \leftarrow \texttt{m\_path(X,C), edge(X,C,Y).}$
$\texttt{path(X,C,Y)} \leftarrow \texttt{m\_path(X,C), edge(X,C,Z), path(Z,C,R), path(R,C,Y).}$

χρειάζεται χρόνο $\Omega(n^2)$ για να υπολογίσει τις απαντήσεις, μια και παράγει όλα τα γεγονότα της μορφής $\texttt{path}(\texttt{b}_\texttt{i}, \texttt{red}, \texttt{e}_\texttt{j})$, για $i, j = 1, ..., n$. Δηλ. παράγει όλα τα

$$
\begin{array}{c}
\mathtt{path}_1^+([\,],\mathtt{a}).\\
\downarrow\\
\mathtt{path}^-([\,],\mathtt{b_1}).\ \cdots\ \mathtt{path}^-([\,],\mathtt{b_n}).\\
\mathtt{path}_1^+([3],\mathtt{b_1}).\ \cdots\ \mathtt{path}_1^+([3],\mathtt{b_n}).\\
\downarrow\\
\mathtt{path}^-([3],\mathtt{c}).\\
\mathtt{path}_1^+([3,3],\mathtt{c}).\\
\downarrow\\
\mathtt{path}_1^+([4],\mathtt{c}).\\
\mathtt{path}^-([3,3],\mathtt{d_1}).\ \cdots\ \mathtt{path}^-([3,3],\mathtt{d_n}).\\
\mathtt{path}_1^+([3,3,3],\mathtt{d_1}).\ \cdots\ \mathtt{path}_1^+([3,3,3],\mathtt{d_n}).\\
\downarrow\\
\mathtt{path}^-([4],\mathtt{d_1}).\ \cdots\ \mathtt{path}^-([4],\mathtt{d_n}).\\
\mathtt{path}_1^+([3,4],\mathtt{d_1}).\ \cdots\ \mathtt{path}_1^+([3,4],\mathtt{d_n}).\\
\mathtt{path}_1^+([4,3],\mathtt{d_1}).\ \cdots\ \mathtt{path}_1^+([4,3],\mathtt{d_n}).\\
\mathtt{path}^-([3,3,3],\mathtt{e_1}).\ \cdots\ \mathtt{path}^-([3,3,3],\mathtt{e_n}).\\
\mathtt{path}_1^+([3,3,3,3],\mathtt{e_1}).\ \cdots\ \mathtt{path}_1^+([3,3,3,3],\mathtt{e_n}).\\
\downarrow\\
\mathtt{path}^-([\,],\mathtt{d_1}).\ \cdots\ \mathtt{path}^-([\,],\mathtt{d_n}).\\
\mathtt{path}^-([3,4],\mathtt{e_1}).\ \cdots\ \mathtt{path}^-([3,4],\mathtt{e_n}).\\
\mathtt{path}_1^+([3,3,4],\mathtt{e_1}).\ \cdots\ \mathtt{path}_1^+([3,3,4],\mathtt{e_n}).\\
\mathtt{path}^-([4,3],\mathtt{e_1}).\ \cdots\ \mathtt{path}^-([4,3],\mathtt{e_n}).\\
\mathtt{path}_1^+([4,3,3],\mathtt{e_1}).\ \cdots\ \mathtt{path}_1^+([4,3,3],\mathtt{e_n}).\\
\mathtt{path}_1^+([3,4,3],\mathtt{e_1}).\ \cdots\ \mathtt{path}_1^+([3,4,3],\mathtt{e_n}).\\
\downarrow\\
\mathtt{path}_1^+([4,4],\mathtt{e_1}).\ \cdots\ \mathtt{path}_1^+([4,4],\mathtt{e_n}).\\
\mathtt{path}^-([3],\mathtt{e_1}).\ \cdots\ \mathtt{path}^-([3],\mathtt{e_n}).\\
\downarrow\\
\mathtt{path}_1^+([4],\mathtt{e_1}).\ \cdots\ \mathtt{path}_1^+([4],\mathtt{e_n}).
\end{array}
$$

Σχήμα 7.8: *Υπολογισμός του παραδείγματος odd length colored path με την προτεινόμενη μέθοδο*

μονοπάτια μήκους 3 (περιττού μήκους) μεταξύ των κόμβων $\mathtt{b}_i$ και $\mathtt{e}_j$ για $i, j = 1, ..., n$. Ο από-κάτω-προς-τα-πάνω υπολογισμός φαίνεται στο Σχήμα 7.9.

Παρόμοια, η counting μέθοδος παράγει το ακόλουθο πρόγραμμα:

```
c_path(0,0,a,red).
c_path(I+1,2*H,Z,C) ← c_path(I,H,X,C), edge(X,C,Z).
c_path(I+1,2*H+1,R,C) ← c_path(I,H,X,C), edge(X,C,Z),
path(I+1,2*H,Z,C,R).
path(I,H/2,X,C,Y) ← c_path(I,H/2,X,C), edge(X,C,Y).
path(I,H/2,X,C,Y) ← c_path(I,H/2,X,C), edge(X,C,Z),
path(I+1,H,Z,C,R), path(I+1,H+1,R,C,Y).
```

$$\begin{array}{c}
\mathtt{m_path(a,red).}\\
\downarrow\\
\mathtt{m_path(b_1,red).}\ \cdots\ \mathtt{m_path(b_n,red).}\\
\mathtt{path(a,red,b_1).}\ \cdots\ \mathtt{path(a,red,b_n).}\\
\downarrow\\
\mathtt{m_path(c,red).}\\
\mathtt{path(b_1,red,c).}\ \cdots\ \mathtt{path(b_n,red,c).}\\
\downarrow\\
\mathtt{m_path(d_1,red).}\ \cdots\ \mathtt{m_path(d_n,red).}\\
\mathtt{path(c,red,d_1).}\ \cdots\ \mathtt{path(c,red,d_n).}\\
\downarrow\\
\mathtt{m_path(e_1,red).}\ \cdots\ \mathtt{m_path(e_n,red).}\\
\mathtt{path(d_1,red,e_1).}\ \cdots\ \mathtt{path(d_n,red,e_n).}\\
\mathtt{path(a,red,d_1).}\ \cdots\ \mathtt{path(a,red,d_n).}\\
\downarrow\\
\mathtt{path(b_1,red,e_1).}\ \cdots\ \mathtt{path(b_i,red,e_j).}\ \cdots\ \mathtt{path(b_n,red,e_n).}
\end{array}$$

Σχήμα 7.9: *Υπολογισμός του παραδείγματος odd length colored path με την μέθοδο magic sets*

Όπως φαίνεται στο πρόγραμμα, η μεταβλητή C, που είναι δεσμευμένη, εμφανίζεται σε όλα τα άτομα του σώματος του αναδρομικού κανόνα. Επομένως, όλα αυτά τα άτομα χρειάζονται στο μετασχηματισμένο πρόγραμμα. Συνεπώς, στο πρόγραμμα αυτό δεν ισχύει η βελτιστοποίηση semi-join που θα μας επέτρεπε να μειώσουμε την πληθικότητα των κατηγορημάτων και να διώξουμε κάποια από τα λεκτήματα στο σώμα μερικών κανόνων. Αυτό έχει σαν αποτέλεσμα ο υπολογισμός να μην είναι πολύ αποδοτικός με την μέθοδο counting. Όπως φαίνεται, ο υπολογισμός του τελικού προγράμματος παράγει όλα τα άτομα της μορφής $\mathtt{path(1,0,b_i,red,e_j)}$ για $i,j = 1,...,n$. Επομένως, χρειάζεται χρόνος $\Omega(n^2)$. Παρατηρήστε πως το counting πρόγραμμα χρειάζεται δύο δείκτες (αντί για έναν που χρειάζεται το πρόγραμμα που προκύπτει από το μετασχηματισμό διακλαδιζόμενου χρόνου). Ο από-κάτω-προς-τα-πάνω υπολογισμός απεικονίζεται στο Σχήμα 7.10.

Το τρέχον παράδειγμα

Ας δούμε πάλι το τρέχον παράδειγμα (του προηγούμενου Κεφαλαίου) μαζί με τη βάση δεδομένων: $\mathtt{e(a1,b}_i\mathtt{)}$, $\mathtt{f(b}_i\mathtt{,c}_i\mathtt{)}$ και $\mathtt{g(b}_i\mathtt{,c}_i\mathtt{,d}_i\mathtt{)}$, για $i = 1,...,n$. Οι απαντήσεις ($\mathtt{Z} = \mathtt{d_i}$, για $i = 1,...,n$) για το τελικό πρόγραμμα που δίνεται στο Παράδειγμα 5.2.3 παράγονται σε χρόνο $O(n)$, καθώς παράγονται $O(n)$ άτομα.

Το αντίστοιχο magic sets πρόγραμμα είναι:

```
m_q(a1).
m_q(Y) ← m_q(X),e(X,Y).
q(X,Z) ← m_q(X),f(X,Z).
q(X,Z) ← m_q(X),e(X,Y),q(Y,W),g(Y,W,Z).
```

$$
\begin{array}{c}
\mathtt{c\_path(0,0,a,red).} \\
\downarrow \\
\mathtt{c\_path(1,0,b_1,red).} \cdots \mathtt{c\_path(1,0,b_n,red).} \\
\mathtt{path(0,0,a,red,b_1).} \cdots \mathtt{path(0,0,a,red,b_n).} \\
\downarrow \\
\mathtt{c\_path(2,0,c,red).} \\
\mathtt{path(1,0,b_1,red,c).} \cdots \mathtt{path(1,0,b_n,red,c).} \\
\downarrow \\
\mathtt{c\_path(3,0,d_1,red).} \cdots \mathtt{c\_path(3,0,d_n,red).} \\
\mathtt{path(2,0,c,red,d_1).} \cdots \mathtt{path(2,0,c,red,d_n).} \\
\mathtt{c\_path(1,1,c,red).} \\
\downarrow \\
\mathtt{c\_path(4,0,e_1,red).} \cdots \mathtt{c\_path(4,0,e_n,red).} \\
\mathtt{c\_path(2,1,d_1,red).} \cdots \mathtt{c\_path(d,1,d_n,red).} \\
\mathtt{c\_path(2,2,d_1,red).} \cdots \mathtt{c\_path(2,2,d_n,red).} \\
\mathtt{c\_path(3,1,e_1,red).} \cdots \mathtt{c\_path(3,1,e_n,red).} \\
\mathtt{path(3,0,d_1,red,e_1).} \cdots \mathtt{path(3,0,d_n,red,e_n).} \\
\mathtt{path(1,1,c,red,d_1).} \cdots \mathtt{path(1,1,c,red,d_n).} \\
\downarrow \\
\mathtt{c\_path(3,2,e_1,red).} \cdots \mathtt{c\_path(3,2,e_n,red).} \\
\mathtt{c\_path(3,4,e_1,red).} \cdots \mathtt{c\_path(3,4,e_n,red).} \\
\mathtt{path(0,0,a,red,d_1).} \cdots \mathtt{path(0,0,a,red,d_n).} \\
\mathtt{path(2,1,b_1,red,e_1).} \cdots \mathtt{path(2,1,b_i,red,e_j).} \cdots \mathtt{path(2,1,b_n,red,e_n).} \\
\mathtt{path(2,2,b_1,red,e_1).} \cdots \mathtt{path(2,2,b_i,red,e_j).} \cdots \mathtt{path(2,2,b_n,red,e_n).} \\
\mathtt{path(1,0,b_1,red,e_1).} \cdots \mathtt{path(1,0,b_i,red,e_j).} \cdots \mathtt{path(1,0,b_n,red,e_n).} \\
\downarrow \\
\mathtt{c\_path(2,3,e_1,red).} \cdots \mathtt{c\_path(2,3,e_n,red).} \\
\mathtt{c\_path(1,1,e_1,red).} \cdots \mathtt{c\_path(1,1,e_n,red).}
\end{array}
$$

Σχήμα 7.10: *Υπολογισμός του παραδείγματος odd length colored path με την μέθοδο counting*

και χρειάζεται χρόνο $O(n)$.

Τέλος, το πρόγραμμα που προκύπτει από τη μέθοδο counting είναι:

```
c_q(0,a1).
c_q(I+1,Y) ← c_q(I,X),e(X,Y).
q(I,X,Z) ← c_q(I,X),f(X,Z).
q(I,X,Z) ← c_q(I,X),e(X,Y),q(I+1,Y,W),g(Y,W,Z).
```

και χρειάζεται επίσης χρόνο $O(n)$ για να παράγει τις απαντήσεις.

7.4.1 Αποτελέσματα σύγκρισης

Ανακεφαλαιώνοντας, η σύγκριση των τριών τεχνικών φαίνεται στον παρακάτω πίνακα:

Μέθοδος	Magic Sets	Counting	Fixed mode
Same generation	$\Omega(n^2)$	$O(n)$	$O(n)$
Colored path	$\Omega(n^2)$	$O(n)$	$O(n)$
Odd length colored path	$\Omega(n^2)$	$\Omega(n^2)$	$O(n)$
Τρέχον παράδειγμα	$O(n)$	$O(n)$	$O(n)$

Σαν τελευταίο σχόλιο, ο προτεινόμενος μετασχηματισμός (fixed mode) φαίνεται να έχει παρόμοια συμπεριφορά με την counting μέθοδο για τα προγράμματα που παρουσιάσαμε (όπως επίσης και για άλλα προγράμματα που δοκιμάσαμε). Ένα πλεονέκτημα της προτεινόμενης μεθόδου είναι ότι έχει ένα ξεκάθαρο κριτήριο τερματισμού που είναι ρεαλιστικό για πολλά παραδείγματα. Τέλος, τόσο ο μετασχηματισμός counting όσο και ο fixed mode (επέκταση του διακλαδιζόμενου χρόνου) φαίνεται να πλεονεκτούν έναντι του magic sets σε πολλά παραδείγματα (ωστόσο ο τελευταίος είναι σαφώς γενικότερος).

Γενικά, από τη σύγκριση που έχει γίνει, είναι φανερό πως κάθε μέθοδος έχει τα δικά της δυνατά και αδύνατα σημεία, επομένως δεν μπορεί να θεωρηθεί πως κάποια μέθοδος είναι η καλύτερη σε όλα τα επίπεδα. Ανάλογα με το πρόγραμμα και τη βάση δεδομένων, εξαρτάται ποια μέθοδος είναι η καλύτερη έναντι των άλλων.

Κεφάλαιο 8

Σύνοψη-Συμπεράσματα

Συνοψίζοντας, θα μπορούσαμε να αναφέρουμε πως στην εργασία αυτή μελετήθηκαν ζητήματα χρήσης του *νοηματικού λογικού προγραμματισμού (intensional logic programming)* για την βελτιστοποίηση επερωτήσεων στις *επαγωγικές βάσεις δεδομένων (deductive databases)*. Στα αρχικά κεφάλαια δόθηκαν οι βασικοί ορισμοί και η σχετική ορολογία που χρησιμοποιείται σε όλη την εργασία και παρουσιάστηκε η σχέση μεταξύ του λογικού προγραμματισμού Λ.Π. και των βάσεων δεδομένων ΒΔ. Στη συνέχεια αναλύθηκαν οι διάφορες τεχνικές στην περιοχή των μετασχηματισμών *διάδοσης των σταθερών τιμών των στόχων των επερωτήσεων (value propagating transformation techniques)* σε επαγωγικές βάσεις δεδομένων. Προτάθηκε μια νέα τεχνική βελτιστοποιήσεων που εφαρμόζεται σε μια μεγάλη υποκατηγορία Κ προγραμμάτων, η οποία είναι επέκταση του μετασχηματισμού διακλαδιζόμενου χρόνου [37]. Η υποκλάση αυτή είναι σημαντική, καθώς περιλαμβάνει την κλάση των Chain Datalog προγραμμάτων, αλλά και πολλά άλλα προγράμματα που έχουν εξαιρετικό ενδιαφέρον στις επαγωγικές βάσεις δεδομένων.

Στην προσπάθεια να συγκριθεί η παρούσα τεχνική με τις ήδη υπάρχουσες, προέκυψαν διάφορα ερωτήματα σχετικά με την αποδοτική εκτέλεση των τελικών προγραμμάτων. Τα υπολογιστικά αυτά ζητήματα οδήγησαν στην εύρεση μιας αποδοτικής αποδεικτικής διαδικασίας για την από-κάτω-προς-τα-πάνω (bottom-up) αποτίμηση του τελικού προγράμματος που τερματίζει, καθώς επίσης και στη παρουσίαση μιας σειράς βελτιστοποιήσεων του τελικού προγράμματος, ώστε αυτό να εκτελείται πιο αποδοτικά.

Επίσης, από τη σύγκριση της παρούσας τεχνικής με τις καθιερωμένες τεχνικές της περιοχής, προέκυψε πως η προτεινόμενη τεχνική εκτός από θεωρητικό ενδιαφέρον έχει και πρακτική αξία. Σε πολλά προγράμματα και μια μεγάλη κλάση βάσεων δεδομένων, ο μετασχηματισμός που παρουσιάστηκε είναι ταχύτερος από τις άλλες μεθόδους. Βέβαια, προκειμένου να γίνει μια αναλυτικότερη σύγκριση θα ήταν απαραίτητα μια υλοποίηση όλων των τεχνικών, ώστε να δοκιμαστούν με μια μεγάλη ποικιλία προγραμμάτων και βάσεων δεδομένων. Παρόλα αυτά, τα θεωρητικά αποτελέσματα είναι άκρως ενθαρρυντικά.

Τα συμπεράσματα που προκύπτουν από αυτή την εργασία μπορούν να συγκεντρωθούν στα παρακάτω σημεία:

- Προτάθηκε μία νέα τεχνική διάδοσης σταθερών τιμών για μια νέα ευρεία κλάση γλωσσών από moded Datalog προγράμματα και αποδείχθηκε η ορθότητά της. Στην πραγματικότητα, οι κλάσεις των προγραμμάτων που μελετήθηκαν είναι εκτενέστερες από τις κλάσεις που μελετώνται σε άλλους σχετικούς μετασχηματισμούς.

- Έγινε φανερό ότι χρονικές γλώσσες όπως η Datalog$_{nS}$ μπορούν να επεκταθούν με μη-ντετερμινιστικές κατασκευές και να αποδειχθούν εξαιρετικά χρήσιμες προκειμένου να ορισθούν νέοι ισχυροί μετασχηματισμοί Datalog προγραμμάτων (και πιθανώς και πιο γενικών λογικών προγραμμάτων). Ειδικότερα, παρουσιάστηκε πως μπορεί να γίνει αποτελεσματική διαχείριση των πολλαπλών καταναλώσεων μεταβλητών σε Datalog προγράμματα με την χρήση κατηγορημάτων επιλογής, αναδεικνύοντας μια νέα χρήση των μη-ντετερμινιστικών κατασκευών.

- Ορίστηκε μια αποδεικτική διαδικασία που μπορεί να εφαρμοστεί στα τελικά προγράμματα που προκύπτουν από τον μετασχηματισμό και αποδείχθηκε πως πάντοτε τερματίζει. Επιπλέον, προτάθηκαν διάφοροι κανόνες μετασχηματισμού του τελικού προγράμματος που αυξάνουν την αποδοτικότητά του.

- Προβλήθηκε η πρακτική σημασία του προτεινόμενου μετασχηματισμού καθώς υπερτερεί σε αρκετά παραδείγματα έναντι των κύριων ανταγωνιστών της.

Κλείνοντας, πιστεύω ότι οι μετασχηματισμοί που παρουσιάστηκαν σε αυτή την εργασία έχουν πολύ καλές προοπτικές για μελλοντικές επεκτάσεις, ώστε να εφαρμόζονται σε όλη την Datalog, αν αυτό είναι δυνατό. Μία πιθανή κατεύθυνση για την επέκταση της κλάσης εφαρμογής του μετασχηματισμού (με κατάλληλες αλλαγές ίσως) είναι να επιτρέπονται περισσότερες μεταβλητές εξόδου, όπως επίσης και διάφορα άλλα σχήματα καταναλώσεων.

Μια άλλη κατεύθυνση για μελλοντική επέκταση είναι να ερευνήσουμε αν τα αποτελέσματα μπορούν να επεκταθούν σε πιο γενικά λογικά προγράμματα (δηλαδή προγράμματα που χρησιμοποιούν συναρτησιακά σύμβολα). Ας θεωρήσουμε το παράδειγμα που ακολουθεί και εκτελεί τον πολλαπλασιασμό φυσικών αριθμών:

```
times(X,Y,Z) ← zero_product(X,Y,Z).
times(s(X),Y,Z) ← times(X,Y,W),plus(W,Y,Z).
plus(X,Y,Z) ← zero_sum(X,Y,Z).
plus(X,s(Y),s(Z)) ← plus(X,Y,Z).
zero_product(0,X,0).
zero_sum(0,X,X).
```

Επιπλέον, ας θεωρήσουμε ότι έχουμε έναν δοσμένο στόχο, έστω τον ← `times(s(0), s(s(0)),Z)`. Τότε, το παραπάνω πρόγραμμα μπορεί να μετασχηματιστεί κατά τον συνηθισμένο τρόπο. Η βασική ιδέα που μας επιτρέπει να κάνουμε αυτό τον μετασχηματισμό είναι ότι παρότι έχουμε συναρτησιακά σύμβολα, κάθε σύνθετος όρος περιλαμβάνει μόνο μια μεταβλητή. Η εφαρμογή του μετασχηματισμού σε τέτοια γενικά προγράμματα γεννά μια σειρά από ενδιαφέροντα νέα ερωτήματα. Οι απαντήσεις σε αυτά τα ερωτήματα, μπορούν να μας δώσουν μια καλύτερη εικόνα για τις σχέσεις μεταξύ του κλασικού λογικού προγραμματισμού και των γλωσσών του νοηματικού λογικού προγραμματισμού (intensional logic programming).

Ένα άλλο σημείο που αξίζει προσοχής και περαιτέρω έρευνας είναι η χρήση μη-ντετερμινιστικών κατασκευών που έχουν χρησιμοποιηθεί και σε άλλες περιπτώσεις στις επαγωγικές βάσεις δεδομένων, όπως αυτές που προτάθηκαν στα [18, 17].

Ακόμα, η υλοποίηση των αλγορίθμων θα μας επέτρεπε να πειραματιστούμε με ένα μεγάλο αριθμό προγραμμάτων και βάσεων δεδομένων και να συγκρίνουμε τους προτεινόμενους μετασχηματισμούς με άλλες γνωστές τεχνικές της περιοχής, αναδεικνύοντας τα πλεονεκτήματα και τα μειονεκτήματα της κάθε προσέγγισης. Προς το παρόν έχουμε μια υλοποίηση του αρχικού μετασχηματισμού διακλαδιζόμενου χρόνου [48]. Πιο συγκεκριμένα, η υλοποίηση του [48] παίρνει ως είσοδο ένα Chain Datalog πρόγραμμα, το μετατρέπει σε ένα απλό Chain Datalog, στη συνέχεια το μετατρέπει σε ένα Datalog$_{nS}$ πρόγραμμα και τελικά το αποτιμά με έναν bottom-up τρόπο. Η υλοποίηση είναι γραμμένη σε Prolog και έχει δώσει αρκετά ενθαρρυντικά αποτελέσματα για την περίπτωση των Chain Datalog προγραμμάτων. Πρόσφατα ξεκίνησα την επέκταση αυτής της υλοποίησης ώστε να μπορεί να χειρίζεται και fixed mode Datalog προγράμματα. Αυτό προφανώς απαιτεί περισσότερη προσπάθεια, καθώς η ύπαρξη των κατηγορημάτων επιλογής στο τελικό πρόγραμμα απαιτεί μια διαφορετική προσέγγιση στην υλοποίηση του μη-ντετερμινισμού στην εκτέλεση κατά bottom-up.

Τέλος, ζητήματα εκφραστικότητας των γλωσσών που χρησιμοποιούνται στους μετασχηματισμούς θα άξιζαν μελέτης, καθώς επίσης και γενικότερα θέματα πολυπλοκότητας. Ωστόσο, η μελέτη αυτών των θεμάτων θα μπορούσε να αποτελέσει το θέμα μιας ξεχωριστής εργασίας.

Βιβλιογραφία

[1] H. K. P. R. A. Colmerauer and R. Passero. Un systeme de communication homme-machine en francais. In *Technical Report, Groupe de Recherche en Intelligence Arificielle, Universite d' Aix-Mareseille II*, 1973.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*. Addison-Wesley, Reading, MA., 1995.

[3] F. Afrati, M. Gergatsoulis, and M. Katzouraki. On transformations into linear database logic programs. In *Proceedings of Perspectives of Systems Informatics (PSI'96)*, volume 1181, pages 433–444. Springer-Verlag, 1996.

[4] F. Afrati, M. Gergatsoulis, and F. Toni. Linearizability on datalog programs. *Theoretical Computer Science*, 308(1-3):199–226, November 2003.

[5] F. N. Afrati, S. S. Cosmadakis, and M. Yannakakis. On datalog vs. polynomial time. In *Proceedings of the Tenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 29-31, 1991, Denver, Colorado*, pages 13–25. ACM Press, May 29-31 1991.

[6] T. Andronikos, P. Potikas, and S. Zachos. A distributed temporal logic framework for the verification and synthesis of reactive systems. In *Proceedings of the 8th Hellenic Conference on Informatics*, volume 2, pages 1–9, 2001.

[7] F. Bancilhon, D. Maier, Y. Sagiv, and J. D. Ullman. Magic sets and other strange ways to implement logic programs (extended abstract). In *PODS '86: Proceedings of the fifth ACM SIGACT-SIGMOD symposium on Principles of database systems*, pages 1–15, New York, NY, USA, 1986. ACM Press.

[8] F. Bancilhon and R. Ramakrishnan. An amateur's introduction to recursive query processing strategies. In *SIGMOD*, volume 15, pages 16–52, 1986.

[9] C. Beeri and R. Ramakrishnan. On the power of magic. *The Journal of Logic Programming*, 10(1,2,3 & 4):255–299, 1991.

[10] I. Bratko. *Prolog Programming for Artificial Intelligence, second edition*. Addison-Wesley, Reading, Massachusetts, 1990.

[11] S. Ceri, G. Gottlob, and L. Tanca. *Logic programming and databases*. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[12] J. Chomicki. Depth-bounded bottom-up evaluation of logic programs. *Journal of Logic Programming*, 25(1):1–31, 1995.

[13] J. Chomicki and T. Imielinski. Finite representation of infinite query answers. *ACM Trans. Database Syst.*, 18(2):181–223, 1993.

[14] H. B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, Inc., 1972.

[15] M. Gergatsoulis and M. Katzouraki. Unfold/fold transformations for definite clause programs. In M. Hermenegildo and J. Penjam, editors, *Proceedings of the Sixth International Symposium on Programming Language Implementation and Logic Programming*, pages 340–354. Springer Verlag, 1994.

[16] M. Gergatsoulis and C. Spyropoulos. Transformation techniques for branching-time logic programs. In *Proc. of the 11th International Symposium on Languages for Intensional Programming (ISLIP'98), May 7-9, Palo Alto, California, USA*, pages 127–138, 1998.

[17] F. Giannotti, S. Greco, D. Sacca, and C. Zaniolo. Programming with nondeterminism in deductive databases. *Annals of Mathematics and Artificial Intelligence*, 19(1-2):97–125, 1997.

[18] F. Giannotti, D. Pedreschi, and C. Zaniolo. Semantics and expressive power of nondeterministic constructs in deductive databases. *Journal of Computer and System Sciences*, 62(1):15–42, 2001.

[19] S. Greco, D. Sacca, and C. Zaniolo. Grammars and automata to optimize chain logic queries. *International Journal of Foundations of Computer Science*, 10(3):349+, 1999.

[20] R. W. Haddad and J. F. Naughton. Counting methods for cyclic relations. In *Proceedings of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 21-23, 1988, Austin, Texas*, pages 333–340. ACM, 1988.

[21] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publishing Company, Reading, Massachusets, USA, 1979.

[22] R. Kowalski. Predicate logic as a programming language. *Information Processing*, 74:569–574, 1974.

[23] J. W. Lloyd. *Foundations of Logic Programming*, volume 2nd edition. Springer Verlag, 1987.

[24] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst.*, 4(2):258–282, 1982.

[25] J. Minker. Logic and databases: A 20 year retrospective. In *Logic in Databases*, pages 3–57, 1996.

[26] M. A. Orgun and W. W. Wadge. Towards a unified theory of intensional logic programming. *The Journal of Logic Programming*, 13(4):413–440, 1992.

[27] M. A. Orgun and W. W. Wadge. Extending temporal logic programming with choice predicates non-determinism. *Journal of Logic and Computation*, 4(6):877–903, 1994.

[28] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[29] A. Pettorossi and M. Proietti. *Transformation of logic programs*, volume 5. Oxford University Press, 1997.

[30] P. Potikas, M. Gergatsoulis, and P. Rondogiannis. Using branching-time logic to optimize an extended class of datalog queries. In *3rd Hellenic Conference on Artificial Intelligence,SETN 2004, Samos, Greece, May 5-8*, pages 197–207, 2004.

[31] P. Potikas, P. Rondogiannis, and M. Gergatsoulis. A transformation technique for datalog programs based on non-deterministic constructs. In *11th International Workshop on Logic-Based Program Synthesis and Transformation (LOPSTR 2001)*, volume 2372, pages 25–45, 2002.

[32] P. Potikas, P. Rondogiannis, and M. Gergatsoulis. A value-propagating transformation technique for datalog programs based on non-deterministic constructs. *Fundamenta Informaticae*, 72(4):485–527, 2006.

[33] R. Ramakrishnan, W. G. Roth, P. Seshadri, D. Srivastava, and S. Sudarshan. The coral deductive database system. In P. Buneman and S. Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993*, pages 544–545. ACM Press, 1993.

[34] R. Ramakrishnan and J. D. Ullman. A survey of deductive database systems. *The Journal of Logic Programming*, 23(2):125–149, 1995.

[35] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

[36] P. Rondogiannis and M. Gergatsoulis. The intensional implementation technique for chain datalog programs. In *Proc. of the 11th International Symposium on Languages for Intensional Programming (ISLIP'98), May 7-9, Palo Alto, California, USA*, pages 55–64, 1998.

[37] P. Rondogiannis and M. Gergatsoulis. The branching-time transformation technique for chain datalog programs. *Journal of Intelligent Information Systems*, 17(1):71–94, 2001.

[38] P. Rondogiannis, M. Gergatsoulis, and T. Panayiotopoulos. Branching-time logic programming: The language cactus and its applications. *Computer Languages*, 24(3):155–178, 1998.

[39] P. Rondogiannis and W. W. Wadge. First-order functional languages and intensional logic. *Journal of Functional Programming*, 7(1):73–101, 1997.

[40] P. Rondogiannis and W. W. Wadge. Higher-order functional languages and intensional logic. *Journal of Functional Programming*, 9(5):527–564, 1999.

[41] D. Saccà and C. Zaniolo. The generalized counting method for recursive logic queries. In G. Ausiello and P. Atzeni, editors, *ICDT'86, International Conference on Database Theory, Rome, Italy, September 8-10, 1986, Proceedings*, volume 243 of *Lecture Notes in Computer Science*, pages 31–53. Springer, 1986.

[42] D. Sacca and C. Zaniolo. Magic counting methods. In *SIGMOD '87: Proceedings of the 1987 ACM SIGMOD international conference on Management of data*, pages 49–59, New York, NY, USA, 1987. ACM Press.

[43] D. Sacca and C. Zaniolo. The generalized counting method for recursive logic queries. *Theoretical Computer Science*, 62(1-2):187–220, 1988.

[44] K. F. Sagonas, T. Swift, and D. S. Warren. The xsb programming system. In *Workshop on Programming with Logic Databases (Informal Proceedings), ILPS*, page 164, 1993.

[45] S. Sippu and E. Soisalon-Soininen. An analysis of magic sets and related optimization strategies for logic queries. *J. ACM*, 43(6):1046–1088, 1996.

[46] L. Sterling and E. Shapiro. *The Art of Prolog.* MIT Press, Cambridge, Massachusetts, 1990.

[47] H. Tamaki and T. Sato. Unfold/fold transformations of logic programs. In *Proc. of the Second International Conference on Logic Programming*, pages 127–138, 1984.

[48] K. Tsopanakis. *Implementation and evaluation of the branching-time transformation for chain datalog programs.* Diploma thesis, Department of Informatics and Telecommunications, University of Athens, Greece, 2003.

[49] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I & II. Computer Science Press, 1989.

[50] M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *J. ACM*, 23(4):733–742, 1976.

[51] W. W. Wadge. Tense logic programming: A respectable alternative. In *Proc. of the 1988 International Symposium on Lucid and Intensional Programming*, 1988.

[52] W. W. Wadge. Higher-order lucid. In *Proceedings of the Fourth International Symposium on Lucid and Intensional Programming*, 1991.

[53] D. Warren. An abstract prolog instruction set. In *SRI International, Technical Note 309, October*, 1983.

[54] A. Yaghi. *The intensional implementation technique for functional languages.* PhD thesis, Dept. of Computer Science, University of Warwick, Coventry, UK, 1984.

[55] M. Yannakakis. Graph-theoretic methods in database theory. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, April 2-4, 1990, Nashville, Tennessee*, pages 230–242. ACM Press, 1990.

[56] Ε. Ζάχος. *Αυτόματα και Τυπικές Γραμματικές.* Ε.Μ.Π., 2004.

[57] Ε. Ζάχος. *Υπολογισιμότητα και Πολυπλοκότητα.* Ε.Μ.Π., 2004.

[58] Π. Σταματόπουλος. *Σημειώσεις Λογικού Προγραμματισμού.* Ε.Κ.Π.Α., 1997.