ΔΗΜΗΤΡΙΟΣ Γ. ΚΕΤΙΚΙΔΗΣ

DNA ΥΠΟΛΟΓΙΣΤΕΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΤΜΗΜΑ ΜΑΘΗΜΑΤΙΚΩΝ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΣΤΗ
ΛΟΓΙΚΗ ΚΑΙ ΘΕΩΡΙΑ ΑΛΓΟΡΙΘΜΩΝ
(μΠΛΑ)
ΣΕΠΤΕΜΒΡΙΟΣ 2001

ΔΗΜΗΤΡΙΟΣ Γ. ΚΕΤΙΚΙΔΗΣ

DNA ΥΠΟΛΟΓΙΣΤΕΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΕΠΙΒΛΕΠΩΝ:**ΑΝΑΣΤΑΣΙΟΣ ΔΗΜΗΤΡΙΟΥ**
ΕΞΩΤΕΡΙΚΟΣ ΣΥΝΕΡΓΑΤΗΣ μΠΛΑ

ΠΡΟΛΟΓΟΣ

Μόλις το 1959 ο Richard Feynman[18] έκανε τη πρώτη νύξη για την κατασκευή υπολογιστών που θα είναι *μικρο-μικροσκοπικοί*(sub-microscopic). Το 1994 ο Adleman[1] έκανε τη πρώτη αξιόλογη προσπάθεια για την κατασκευή υπολογιστών που θα επεξεργάζονται πληροφορία σε *μοριακό επίπεδο*(*molecular computing*), λύνοντας ένα στιγμιότυπο του κατευθυνόμενου χαμιλτόνειου κυκλώματος (Directed Hamiltonian Path, DHP), χρησιμοποιώντας μέσα μοριακής βιολογίας. Η πρωτοποριακή εργασία του Adleman έδωσε μια ώθηση για περαιτέρω έρευνα προς την υλοποίηση του οράματος του Richard Feynman, διαμέσω της μοριακής βιολογίας και βιοτεχνολογίας.

Σκοπός της παρούσας εργασίας είναι η παρουσίαση μοντέλων υπολογιστών των οποίων οι στοιχειώδεις δομές αναπαράστασης πληροφορίας είναι τα μόρια DNA.Επιπλέον οι στοιχειώδεις λειτουργίες επεξεργασίας της πληροφορίας είναι οι διάφορες αντιδράσεις ανάμεσα σε αυτά τα μόρια. Για τους παραπάνω λόγους ονομάζονται DNA υπολογιστές. Το όλο σύστημα θεωρείται ότι υλοποιείται σε ένα άρτια εξοπλισμένο βιολογικό εργαστήριο. Γι' αυτό λέγονται και in-vitro μοντέλα, σε αντίθεση με τα in-vivo μοντέλα υπολογισμού όπου θεωρούνται ζωντανά κύτταρα. Στα in-vivo μοντέλα, που έχουν σχετικά πιο πρόσφατη ιστορία, γίνεται απλή αναφορά. Στην συνέχεια παρουσιάζονται τα θεωρητικά μοντέλα υπολογισμού, μέσω των οποίων αποδεικνύεται θεωρητικά η ύπαρξη universal DNA υπολογιστών.

Στο ΜΕΡΟΣ Ι, παρουσιάζεται αρχικά μια εισαγωγή, που περιγράφει γενικά τον ορισμό και τις δυνατότητες των DNA υπολογιστών. Στο δεύτερο κεφάλαιο δίνεται το βασικό in-vitro μοντέλο υπολογισμού, μαζί με δύο εκλεπτύνσεις του που είναι προσανατολισμένες στα προβλήματα της ικανοποιησιμότητας προτασιακών τύπων(SAT), και στο πρόβλημα της κρυπτανάλυσης του γνωστού κρυπτοσυστήματος DES(Data Encryption Standard).

Στο ΜΕΡΟΣ ΙΙ παρουσιάζονται τα θεωρητικά μοντέλα υπολογισμού. Αρχικά, στο κεφάλαιο 1, παρουσιάζονται οι μηχανές Turing, που είναι η βάση των θεωρητικών μοντέλων. Επίσης παρουσιάζεται η έννοια των NP-προβλημάτων. Στο κεφάλαιο 2 παρουσιάζονται οι γραμματικές Chomsky και αποδυκνείεται η ισοδυναμία τους με τις μηχανές Turing. Στο κεφάλαιο 3 που είναι η ουσία του θέματος, παρουσιάζεται το μοντέλο των συστημάτων splicing, που είναι η αφαίρεση του DNA-recombination(μέρος του in-vitro μοντέλου στο ΜΕΡΟΣ Ι). Τα splicing συστήματα αποδεικνύονται ισοδύναμα με τις γραμματικές Chomsky, άρα και με τις μηχανές Turing. Στη συνέχεια του κεφαλαίου αποδεικνύεται η ύπαρξη Universal splicing συστήματος, πολύ σημαντικό αποτέλεσμα, έστω και θεωρητικό, αφού δείχνει ότι είναι δυνατή η ύπαρξη «προγραμματιζόμενων» DNA-υπολογιστών. Το κεφάλαιο 3 κλείνει με μια παρουσίαση των ανοιχτών προβλημάτων και των μελλοντικών κατευθύνσεων στο χώρο των DNA-υπολογιστών.

Αθήνα, Σεπτέμβριος 2001
Κετικίδης Δημήτριος

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΜΕΡΟΣ Ι

ΥΠΟΛΟΓΙΣΤΙΚΑ ΜΟΝΤΕΛΑ IN-VITRO

1. Εισαγωγή

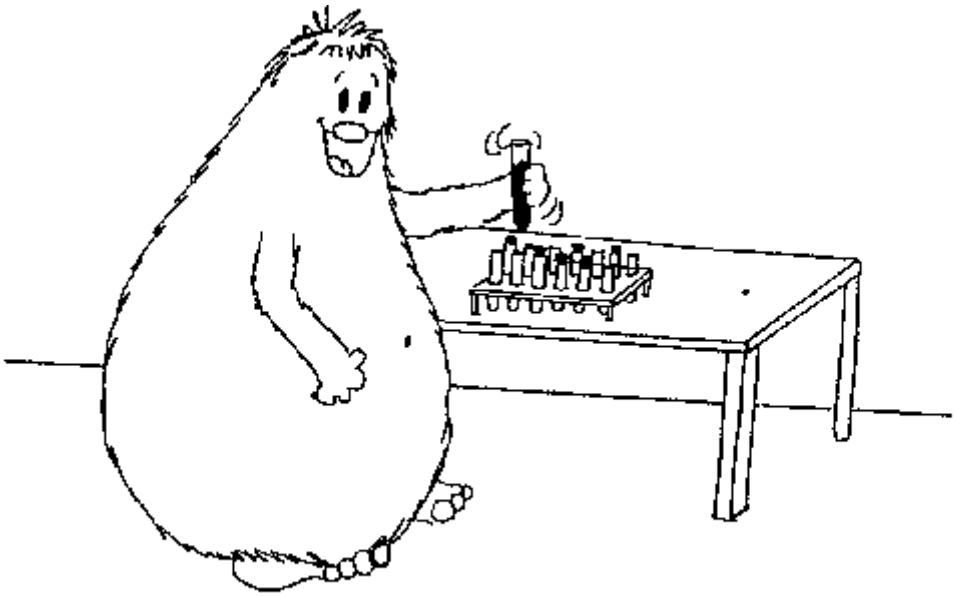
Η σημερινή εποχή είναι η εποχή της πληροφορικής και των ηλεκτρονικών υπολογιστών. Οι τελευταίοι χρησιμοποιούνται και επιδεικνύουν τις δυνατότητες τους σε ένα πλήθος δραστηριοτήτων στον οικονομικό, βιομηχανικό και επιστημονικό τομέα. Εντούτοις οι ηλεκτρονικοί υπολογιστές «δοκιμάζονται» πραγματικά όταν καλούνται να επιλύσουν προβλήματα που ανήκουν σε δύσκολες υπολογιστικά κλάσεις προβλημάτων(NP, NP-complete, NP-Hard). Τα προβλήματα που ανήκουν στις κλάσεις αυτές «ξεγυμνώνουν» τις αδυναμίες των ηλεκτρονικών υπολογιστών τόσο από άποψη χώρου(μνήμη για τα δεδομένα) όσο και από άποψη χρόνου(χρόνος για την εκτέλεση των λειτουργιών τους). Οι δυσκολίες που αντιμετωπίζουν οι ηλεκτρονικοί υπολογιστές σε τέτοιου είδους προβλήματα ώθησε την έρευνα για την δημιουργία νέων μοντέλων υπολογιστικών μηχανών(μηχανών που μπορούν να αποθηκεύουν, ανακτούν και να επεξεργάζονται πληροφορία). Ένα νέο μοντέλο υπολογιστικής μηχανής είναι οι DNA computers στους οποίους η επεξεργασία και αποθήκευση πληροφορίας γίνεται σε μοριακό επίπεδο σε αντίθεση με τους ηλεκτρονικούς υπολογιστές όπου το στοιχειώδες τμήμα τους μπορεί να θεωρηθεί ότι είναι το transistor(το οποίο βέβαια υλοποιείται σε αρκετά θεμελιώδες επίπεδο σε ένα μικροσκοπικό κομμάτι ημιαγωγού αλλά όχι σε μοριακό επίπεδο). Εκείνο που έχουν να αντιπαρατάξουν οι DNA υπολογιστές απέναντι στους ηλεκτρονικούς, είναι οι τεράστιες δυνατότητες αποθήκευσης και παράλληλης επεξεργασίας, από τις οποίες χαρακτηρίζονται.

Είναι γνωστό ότι οι ηλεκτρονικοί υπολογιστές έχουν χρησιμοποιηθεί για να λύσουν προβλήματα της βιολογίας, ένα κλάδος που είναι γνωστός σαν υπολογιστική βιολογία. Σε αυτή την εργασία θα μελετηθεί κατά μια έννοια το αντίστροφο: Το πώς τα μόρια DNA οι βιολογικές αντιδράσεις τους, και οι βιολογικές μέθοδοι για τα μόρια DNA, μπορούν να χρησιμοποιηθούν για να υλοποιηθεί ένα νέο είδος υπολογιστή. Τα παραπάνω συνιστούν ένα καινούριο κλάδο ανάμεσα στην βιολογία και στην επιστήμη των υπολογιστών(computer science) που είναι γνωστός ως biological computing ή DNA computing.

1.1 Γενικά περί DNA-υπολογιστών και των δυνατοτήτων τους

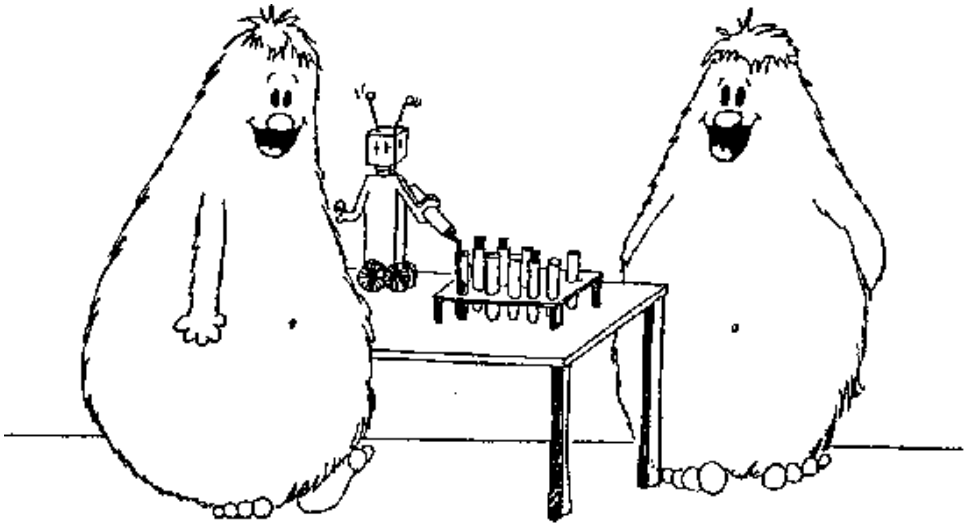
Η βασική ιδέα που οδήγησε στη δημιουργία των πρώτων DNA-υπολογιστών είναι ότι, οι δυνατότητες επεξεργασίας της πληροφορίας που υπάρχουν στα οργανικά μόρια (DNA, RNA, ένζυμα) μπορούν να χρησιμοποιηθούν για την επιτέλεση συγκεκριμένων έργων με συγκεκριμένο σκοπό. Εφόσον μιλάμε για επεξεργασία πληροφορίας σε μοριακό επίπεδο έχουμε να κάνουμε με μια μικροσκοπική κλίμακα ολοκλήρωσης, που ξεπερνάει κατά πολύ τις VLSI κλίμακες των ψηφιακών ηλεκτρονικών υπολογιστών. Στο νέο αυτό επίπεδο επεξεργασίας πληροφορίας, ανήκουν, εκτός από τους DNA υπολογιστές, και οι κβαντικοί υπολογιστές(Quantum Computers). Όμως τα προβλήματα υλοποίησης για τους τελευταίους, είναι αξεπέραστα από τις δυνατότητες της σημερινής τεχνολογίας, κάτι που κάνει την υλοποίηση τους σχεδόν ανέφικτη, για τα σημερινά δεδομένα. Αντιθέτως, περισσότερο εφικτή φαίνεται να είναι η δημιουργία DNA υπολογιστών, που λύνουν προβλήματα με «σοβαρό» μέγεθος .

Ένας DNA υπολογιστής θα μπορούσε, πολύ απλά, να είναι κάτι σαν αυτό που αναπαριστάται στο σχήμα 1.1.1. Ένα σύνολο δοκιμαστικών σωλήνων, κάποιοι από τους οποίους περιέχουν μόρια που κωδικοποιούν το στιγμιότυπο του προβλήματος, κάποιοι άλλοι ενδεχομένως να περιέχουν την κωδικοποίηση του προγράμματος που πρέπει να εκτελεστεί, ενώ άλλοι σωλήνες θα φυλάσσουν ενδιάμεσα αποτελέσματα στη διαδικασία του υπολογισμού. Ο υπολογισμός είναι στην ουσία μια σειρά από λειτουργίες που πρέπει να γίνουν (ανάμειξη, φιλτράρισμα)



Σχήμα 1.1.1

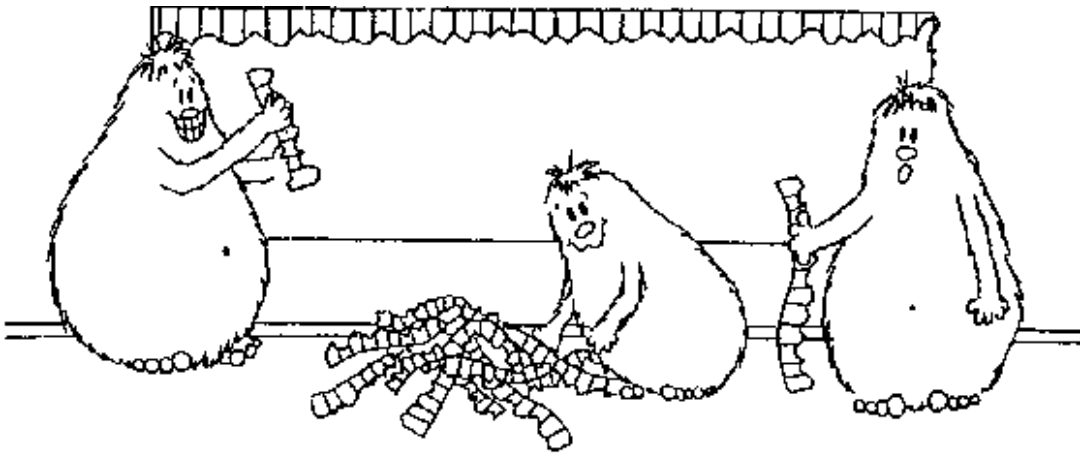
επί των σωλήνων, ενώ ένα πρόγραμμα, όπως αναφέρθηκε προηγουμένως, θα κωδικοποιεί τον τρόπο που θα αντιδρούν τα διάφορα μόρια στους σωλήνες, έτσι ώστε το αποτέλεσμα που θα προκύψει να είναι μία κωδικοποίηση της λύσης στο πρόβλημα που θέλουμε να επιλυθεί.



Σχήμα 1.1.2

Οι λειτουργίες που πρέπει να επιτελεστούν επί των δοκιμαστικών σωλήνων, μπορεί να γίνονται είτε από εξειδικευμένο προσωπικό, είτε να αυτοματοποιηθούν, χρησιμοποιώντας ρομποτικό εξοπλισμό, ο οποίος πιθανότατα θα ελέγχεται από κάποιο ψηφιακό ηλεκτρονικό υπολογιστή. Ένα τέτοιο μοντέλο φαίνεται στο σχήμα 1.1.2. Οι διάφορες μορφές τεχνολογίας, δεν είναι αναγκαίο να είναι ανταγωνιστικές, αλλά μπορούν να συμπληρώνουν η μια την άλλη.

Όπως φαίνεται από τις παραπάνω περιγραφές, η δημιουργία DNA υπολογιστών δεν είναι πολύ μακριά από τις δυνατότητες της σημερινής τεχνολογίας. Εντούτοις οι βιοχημικές τεχνικές, δεν είναι ακόμα επαρκώς εκλεπτυσμένες, ούτε ακριβείς.



Σχήμα 1.1.3

Ιδιαιτέρως, οι τεχνικές δεν έχουν ακόμα αναπτυχθεί προς τις συγκεκριμένες ανάγκες των DNA υπολογιστών. Εντούτοις, το τεχνολογικό χάσμα φαίνεται να είναι αρκετά μικρότερο από αυτό των κβαντικών υπολογιστών.

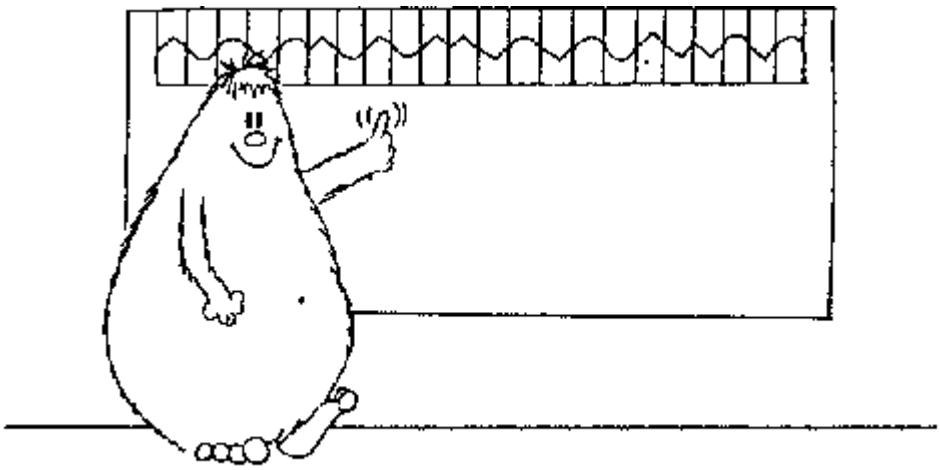
Οι ελπίδες για το μέλλον των DNA υπολογιστών, πηγάζουν από δύο σπουδαία χαρακτηριστικά:(1) Ο μαζικός παραλληλισμός των μορίων DNA και (2) Η Watson-Crick συμπληρωματικότητα . Παρακάτω περιγράφονται συνοπτικά, τα δύο αυτά χαρακτηριστικά.

(1) Τα περισσότερα από τα υπολογιστικά δύσκολα προβλήματα, μπορούν να λυθούν με μια εξαντλητική αναζήτηση ανάμεσα σε όλες τις πιθανές λύσεις. Εντούτοις, η αξεπέραστη δυσκολία, βρίσκεται στο γεγονός ότι μια τέτοια αναζήτηση είναι πολύ αχανής, για να εκτελεστεί από ψηφιακούς υπολογιστές . Αυτό γιατί ο χώρος των πιθανών λύσεων έχει μέγεθος εκθετικό ως προς το μέγεθος του προβλήματος. Από την άλλη μεριά, η πυκνότητα της πληροφορίας που αποθηκεύεται σε μόρια DNA και η ευκολία κατασκευής πολλών αντιγράφων αυτών, μπορεί να καταστήσει τέτοιες εξαντλητικές αναζητήσεις εφικτές. Ένα τυπικό παράδειγμα είναι η κρυπτανάλυση ενός κρυπτογραφήματος: όλα τα κλειδιά(κωδικοποιημένα σε μόρια DNA) θα μπορούσαν να δημιουργηθούν γρήγορα, και να δοκιμασθούν επίσης γρήγορα.

(2) Η συμπληρωματικότητα Watson-Crick(προς τιμή αυτών που την ανακάλυψαν) είναι ένα χαρακτηριστικό(feature), που προσφέρεται ελεύθερα από τη φύση. Όταν δημιουργείται δεσμός, ανάμεσα σε δύο απλά μόρια DNA, προς την δημιουργία μιας διπλής έλικας, ξέρουμε ότι η βάση συνδέεται με μια συμπληρωματική της. Έτσι αν ξέρουμε το ένα απλό μόριο DNA, ξέρουμε και το συμπληρωματικό του, και συνεπώς μπορούμε να το δεσμεύσουμε και να το επιλέξουμε γρήγορα μέσα από ένα μεγάλο «σωρό» με πιθανές λύσεις. Η δυνατότητα αυτή απεικονίζεται στο 1.1.4, όπου το ταίριασμα έχει γίνει αυτόματα(από τη φύση), ενώ στο σχήμα 1.1.3 φαίνεται η κατάσταση που θα επικρατούσε αν δεν υπήρχε η συμπληρωματικότητα: είμαστε καταδικασμένοι

να ψάξουμε να βρούμε το συγκεκριμένο μόριο, που μπορεί να κωδικοποιεί μια λύση σε πρόβλημα, και δεν υπάρχει τίποτα(υποθέτοντας ότι δεν υπάρχει συμπληρωματικότητα) για να μας βοηθήσει προς αυτό το σκοπό .

Αυτά τα δύο χαρακτηριστικά βοηθούν όπως φαίνεται στην επίλυση των δύσκολων προβλημάτων. Ο πίνακας στο σχήμα 1.1.5 δείχνει τα κυριότερα αποτελέσματα τα οποία έχουν επιτευχθεί μέχρι σήμερα, στο χώρο των DNA υπολογιστών. Το κάθε αποτέλεσμα, αξιολογείται με δύο τρόπους: τον αριθμό των *βιολογικών βημάτων(bio steps)* δηλαδή των λειτουργιών που επιτελούνται στο εργαστήριο, και των αριθμών των μορίων DNA(DNA strands) που χρησιμοποιεί.



Σχήμα 1.1.4

Οι διαφορετικές λειτουργίες που επιτελούνται στο εργαστήριο, μπορεί να έχουν και διαφορετικό χρόνο η κάθε μια, για να ολοκληρωθεί. Εντούτοις θεωρείται ότι όλες απαιτούν τον ίδιο χρόνο(ο οποίος μπορεί να μειωθεί χρησιμοποιώντας ρομποτικό εξοπλισμό του εργαστηρίου).

Πρόβλημα	Αριθμός βιο-λειτουργιών	Πλήθος μορίων DNA
(1) Hamilton Path	O(n)	$n!$
(2) SAT	O(s)	2^n
(3) Ικανοποιησιμότητα κυκλώματος	O(s)	2^n
(4) Το (3) ως πρόβλημα βελτιστοποίησης	O(s)	2^n
(5) Κανονική ικανοποιησιμότητα	O(s)	2^n
(6) 1-tape NTM	O(s)	2^N
(6a) Το (3) μέσω του (6)	$\Theta(s^2)$	2^n
(7) Cellular αυτόματα	O(1)	$t \cdot S$
8) PSPACE	O(S)	2^{2S}
(9) Πωλυωνυμική ιεραρχία	O(s)	2^n
(10) «Σπάσιμο» DES	O(1)	2^{56}

Πίνακας 1.1.5

Ο αριθμός των DNA μορίων, είναι απλά ο αριθμός των διακεκριμένων μορίων που μπορεί να εμφανιστούν στον ίδιο σωλήνα, κατά τη πορεία της εκτέλεσης των λειτουργιών. Για να είμαστε πιο ακριβείς θα έπρεπε να συμπεριληφθεί και μια παράμετρος για το μήκος των μορίων, αλλά το μήκος των μορίων είναι γραμμικό ως προς το μέγεθος του προβλήματος, ενώ το πλήθος είναι εκθετικό ως προς αυτό. Για

τον σκοπό της αξιολόγησης της εφικτότητας των DNA αλγορίθμων, υποθέτουμε ότι 10^{21} είναι ένα άνω φράγμα στον αριθμό των μορίων DNA που είναι διαθέσιμα στον αλγόριθμο. Είναι χρήσιμο να ειπωθεί ότι $10^{21} \approx 2^{70}$.

Παρακάτω σχολιάζονται συνοπτικά τα αποτελέσματα του πίνακα 1.1.5

(1) Αυτό είναι το διάσημο αποτέλεσμα του Adleman [1] που δείχνει ότι το Directed Hamiltonian Path πρόβλημα μπορεί να λυθεί από ένα υπολογιστή DNA, με αριθμό βιολογικών βημάτων γραμμικό στο μέγεθος του προβλήματος(n το μέγεθος του προβλήματος). Το πρόβλημα είναι ότι η προσέγγιση του Adleman χρησιμοποιεί $n!$ το πλήθος μόρια DNA.

(2) Αυτό είναι το αποτέλεσμα του Lipton [2] ότι το SAT(ικανοποιησιμότητα για προτασιακούς τύπους σε κανονική συζευκτική μορφή) μπορεί να επιλυθεί με αριθμό βημάτων γραμμικό στο μέγεθος του τύπου(s το μέγεθος του τύπου). Η σημαντική βελτίωση εδώ, είναι ότι αυτή η μέθοδος μπορεί να χρησιμοποιηθεί για μια γενικότερη κλάση προβλημάτων, ενώ ο αριθμός των μορίων είναι 2^n όπου n ο αριθμός των προτασιακών μεταβλητών του τύπου.

(3) Γενίκευση του (2) από τους Dunworth,Boneh,Sgall [3], για λογικά κυκλώματα γενικής μορφής, αποτελούμενα από δυαδικές πύλες(AND,OR,NOT κ.α). Το γεγονός ότι τέτοια κυκλώματα είναι πολύ αποτελεσματικά για μια ποικιλία προβλημάτων, και εύκολο να σχεδιαστούν, κάνει αυτό το αποτέλεσμα κατάλληλο για πρακτικά προβλήματα. Ο αριθμός των πυλών στο κύκλωμα είναι το s, ενώ ο αριθμός των μεταβλητών εισόδου είναι n.

(4) Επέκταση του (2) και του (3), πάλι από Dunworth,Boneh,Sgall [3],για την περίπτωση προβλημάτων *βελτιστοποίησης*: Το ζητούμενο του Max-Circuit-Satisfiability είναι να *βρεθεί μια ανάθεση* που να ικανοποιεί τον τύπο, ενώ συγχρόνως να μεγιστοποιεί τον αριθμό των μεταβλητών που είναι true.

(5) Τα αποτελέσματα από (3) και (4), μπορούν να γίνουν πιο αποτελεσματικά, αν συνδυαστούν με μηχανές πεπερασμένων καταστάσεων. Έστω $L = L_1 \cap L_2$ ένα σύνολο από ακολουθίες δυαδικών ψηφίων, μήκους n. Η L_1 μπορεί να αναγνωριστεί από ένα κύκλωμα με s πύλες, και η L_2 μπορεί να αναγνωριστεί από ένα αυτόματο με k καταστάσεις. Στους Dunworth,Boneh,Sgall [3] δείχνεται ότι μια λύση με DNA που αναπαριστά όλες τις ακολουθίες στην L μπορεί να κατασκευαστεί χρησιμοποιώντας $O(kn+s)$ μόρια DNA και $O(s)$ βιολογικές λειτουργίες. Αυτή η γενίκευση αυξάνει την αποτελεσματικότητα των (3) και (4), αφού μπορεί να μειωθεί το μέγεθος ενός κυκλώματος, υλοποιώντας μέρος αυτού ως πεπερασμένο αυτόματο.

(6) Και (6α). Το πρώτο αποτέλεσμα δείχνει ότι το DNA μπορεί να εξομοιώσει μια μη ντετερμινιστική μηχανή Turing(NTM) με μία ταινία. Εδώ t είναι ο χρόνος, και N είναι ο αριθμός των bits στην ταινία που χρησιμοποιεί η μηχανή Turing. Το δεύτερο αποτέλεσμα δείχνει γιατί οι 1-ταινίας NTM είναι κυρίως θεωρητικού ενδιαφέροντος. Ενώ στο (3) έχουμε $O(s)$ λειτουργίες, για να γίνει η ίδια δουλειά με 1-ταινίας NTM, απαιτούνται $\Theta(s^2)$ λειτουργίες. Ο συμβολισμός Θ σημαίνει ότι ο αριθμός των λειτουργίων είναι φραγμένος και από κάτω, κατά μια γραμμική συνάρτηση του s^2 (πράγμα που σημαίνει ότι δε μπορούμε να κάνουμε κάτι καλύτερο από $c \cdot s^2$), ενώ για το $O(s)$ σημαίνει ότι είναι φραγμένος μόνο από πάνω.

(7) Αυτή είναι η κατασκευή του Winfree[4] που δείχνει πως πολύπλοκα DNA σχέδια(patterns), μπορούν να χρησιμοποιηθούν για να προσομοιώσουν κυψελικά αυτόματα(cellular automata). Ο αριθμός των νουκλεοτιδίων που

χρησιμοποιούνται από το DNA pattern είναι ανάλογος με το γινόμενο του χώρου S που χρησιμοποιείται από τα αυτόματα, και του αριθμού των γενεών t, για τις οποίες το αυτόματο εκτελεί την λειτουργία του. Το ελκυστικό χαρακτηριστικό αυτής της κατασκευής, είναι ότι οι υπολογισμοί γίνονται αυτόματα. Δεν απαιτείται η παρέμβαση ενός τεχνικού βιολογικού εργαστηρίου.

(8) Αυτό είναι το αποτέλεσμα των Beaver[5], Reif[6] και Papadimitriou[7], ότι είναι δυνατό να προσομοιωθεί η κλάση PSPACE(βλέπε Μέρος ΙΙ) με λειτουργίες DNA. S είναι ο χώρος που απαιτείται. Αυτό επίσης είναι ένα θεωρητικό αποτέλεσμα:το πρόβλημα είναι ότι χρησιμοποιούνται βιολογικές λειτουργίες, που φαίνεται ότι είναι αδύνατο να υλοποιηθούν στην πράξη. Συνοπτικά, οι λειτουργίες περιλαμβάνουν την ανόπτηση όλων των μορίων DNA πάνω στα ακριβή συμπληρωματικά τους, παράλληλα για όλα τα διαφορετικά μόρια στο σωλήνα. Για να συμβεί κάτι τέτοιο σε λογικό χρόνο, θα πρέπει ο αριθμός των αντιγράφων του κάθε μορίου, να είναι ίσος με τον αριθμό των διακεκριμένων μορίων. Έχοντας το 2^{70} ως άνω φράγμα στον αριθμό των διαθέσιμων μορίων, έπεται ότι αυτές οι μέθοδοι μπορούν να χρησιμοποιηθούν μόνο για αλγόριθμους που απαιτούν 35 bits χώρο. Αλλά για τέτοιους αλγόριθμους, αρκούν οι δυνατότητες των ηλεκτρονικών υπολογιστών.

(9) Αυτό το αποτέλεσμα δείχνει πώς να προσομοιωθεί η πολυωνυμική ιεραρχία. n είναι ο αριθμός όλων των ποσοδεικτούμενων μεταβλητών. Όπως και στο (8) απαιτείται η επεξεργασία του DNA με λειτουργίες που είναι δύσκολο να υλοποιηθούν στη πράξη.

(10) Αυτό είναι το αποτέλεσμα για την κρυπταναλυτική επίθεση στο DES. Απαιτούνται $O(1)$ παράλληλες λειτουργίες και $O(2^n)$ μόρια DNA, όπου n το μήκος του κλειδιού του DES, σε bits.

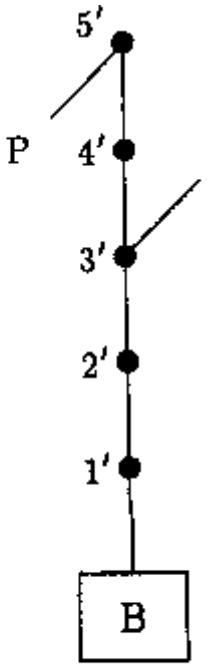
Εκτός από τα παραπάνω αποτελέσματα, υπάρχουν και άλλοι λόγοι για να εξερευνηθεί ο χώρος των DNA υπολογιστών. Από τη μια μεριά, είναι σημαντικό να προσπαθήσουμε να καταλάβουμε πως η μητέρα φύση «υπολογίζει»(το «θαύμα» της ζωής, από μια υλιστική άποψη, είναι απλώς αποτέλεσμα των λειτουργιών του DNA στα κύτταρα). Από την άλλη μεριά, οι DNA υπολογιστές, οδηγούν στην εύρεση *υπολογιστικών προτύπων*, (δηλαδή δομών δεδομένων και τύποι λειτουργιών που επιτελούνται επί αυτών των δομών, μοντέλα υπολογισιμότητας) που είναι διαφορετικά από αυτά που είναι καθιερωμένα στην σημερινή επιστήμη των υπολογιστών. Ακόμη και αν η δημιουργία DNA υπολογιστών αποδειχθεί μη ρεαλιστική(για παράδειγμα λόγω των λαθών που γίνονται στις βιολογικές λειτουργίες), μια "διαφυγή" θα ήταν η εφαρμογή των νέων υπολογιστικών προτύπων στο πλαίσιο των ψηφιακών ηλεκτρονικών υπολογιστών.

Η κλασική θεωρητική επιστήμη των υπολογιστών, είναι θεμελιωμένη σε λειτουργίες επανεγγραγής: αυτό είναι αληθές για τα περισσότερα αυτόματα και για τη θεωρία των γλωσσικών μοντέλων. Όπως θα δούμε (Μερος Ι, κεφάλαιο 2), η φύση επεξεργάζεται τα μόρια DNA με ένα υπολογιστικό τρόπο, χρησιμοποιώντας λειτουργίες αρκετά διαφορετικού τύπου: αποκοπή(cut), επικόλληση(paste), εισαγωγή(insertion),διαγραφή(deletion). Θα αποδειχθεί(Μερος ΙΙ κεφάλαιο 3) ότι χρησιμοποιώντας τέτοιες λειτουργίες μπορούμε να δομήσουμε υπολογιστικά μοντέλα, που είναι ισοδύναμα σε ισχύ με τις μηχανές Turing. Έτσι οι θεωρίες υπολογισιμότητας μπορούν να ανασκευαστούν σε αυτό το νέο πλαίσιο. Το αν αυτό έχει πρακτική σημασία για τις εφαρμογές της επιστήμης των υπολογιστών, είναι ένα πρόωρο ερώτημα.

2. Μοντέλο DNA υπολογιστή

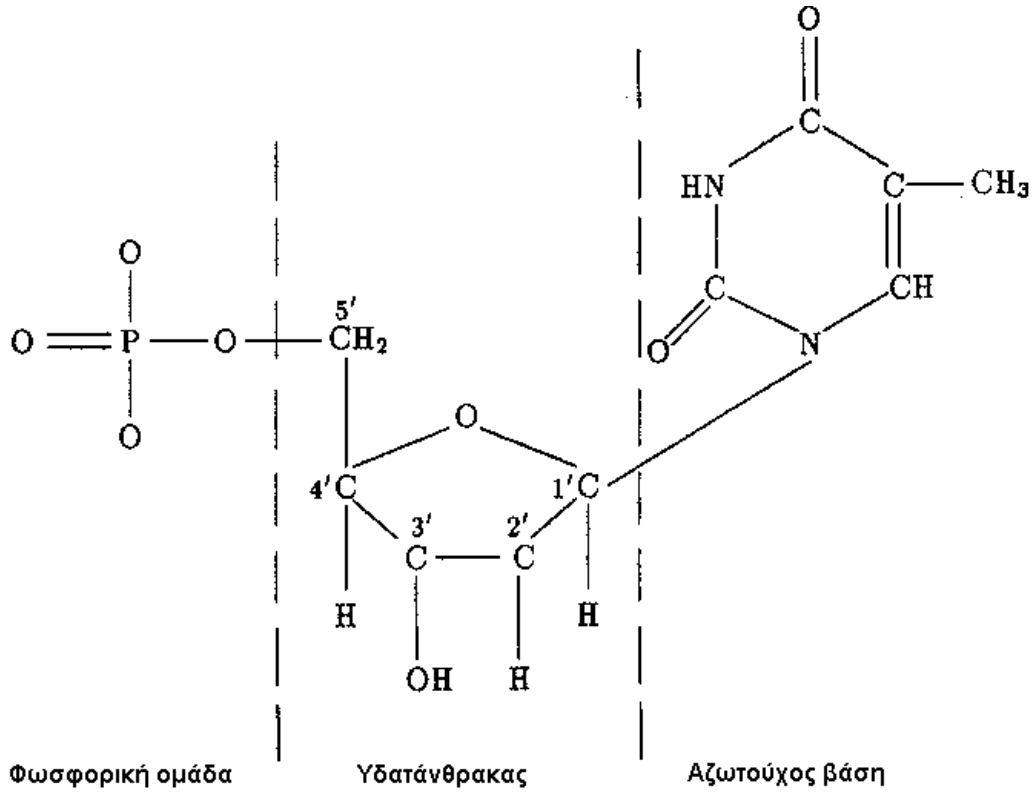
Ένα μόριο DNA μπορεί να θεωρηθεί ως ένας επεξεργαστής και οι βιοχημικές αντιδράσεις στις οποίες συμμετέχει αποτελούν τις λειτουργίες που υποστηρίζει. Αν και αυτή η αντιστοιχία δεν είναι απολύτως ακριβής, δίνει μια πρώτη γεύση των δυνατοτήτων του DNA υπολογιστή από άποψη ισχύος σε παράλληλη επεξεργασία(δισεκατομμύρια μόρια DNA σε ένα αντιδραστήριο, αντιδρώντας μεταξύ τους δηλαδή εκτελώντας λειτουργίες!). Δεν είναι 100% ακριβής γιατί ένας επεξεργαστής υλοποιεί κάποιο λογικό κύκλωμα με εισόδους και εξόδους, ενώ για ένα μόριο DNA δεν μπορεί να ειπωθεί το ίδιο. Πάντως το βασικό μοντέλο του DNA υπολογιστή μπορεί να μελετηθεί από τη σκοπιά ενός μαθηματικού μοντέλου γνωστού ως splicing system και με το οποίο μπορεί να αποδειχθεί ότι κάθε splicing system είναι υπολογιστικά ισοδύναμο με μια μηχανή Turing και αντιστρόφως. Η ισοδυναμία Turing μηχανών και λογικών κυκλωμάτων είναι γνωστή. Προτού επεκταθούμε στα splicing systems θα πρέπει να επεξηγηθεί το βασικό μοντέλο υπολογισμού για τον DNA υπολογιστή. Σκόπιμο είναι να επεξηγηθεί πρώτα η δομή του μορίου DNA.

2.1 Η δομή του DNA



Σχήμα 2.1.1 Μια σχηματική αναπαράσταση ενός νουκλεοτιδίου

Το DNA βρίσκεται σε κάθε κυτταρικό οργανισμό σαν το αποθηκευτικό μέσο της γενετικής πληροφορίας. Αποτελείται από μονάδες που ονομάζονται νουκλεοτίδια, που διακρίνονται από τη παρουσία μιας χημικής ομάδας ή βάσης που είναι προσκολλημένη σε αυτά. Οι τέσσερις βάσεις είναι *αδενίνη(A), γουανίνη(G)(πουρίνες) ,κυτοσίνη(C) και θυμίνη(T)(πυριμιδίνες).* Τα ονόματα των τεσσάρων αυτών βάσεων συνήθως χρησιμοποιούνται για να αναφερόμαστε στα νουκλεοτίδια που τις περιέχουν. Για την ακρίβεια κάθε νουκλεοτίδιο, εκτός από την *αζωτούχο βάση,* έχει και μια *ομάδα υδατάνθρακα*, και μια *φωσφορική ομάδα.* Ο υδατάνθρακας έχει 5

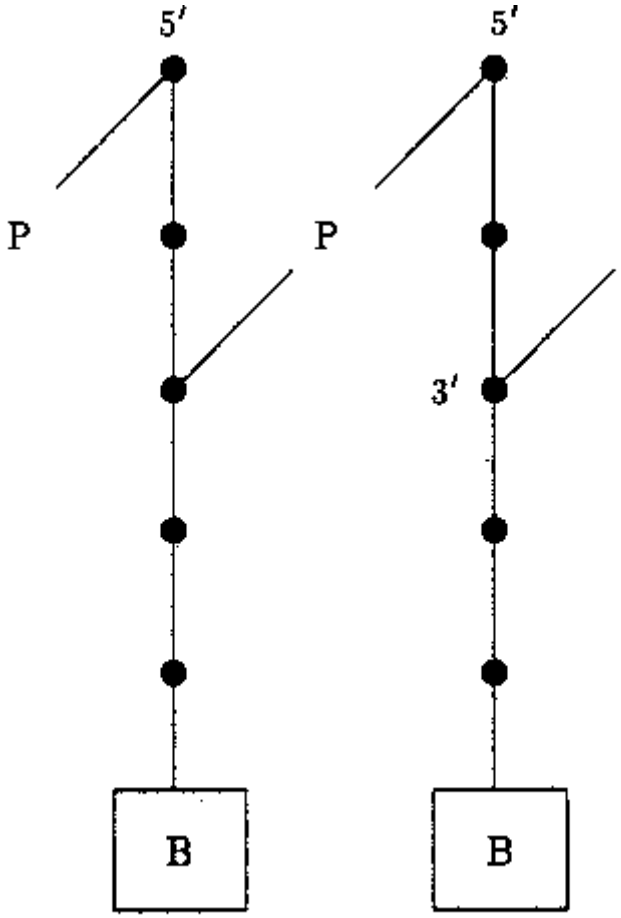


Σχήμα 2.1.2 Η χημική δομή ενός νουκλεοτιδίου με βάση θυμίνη(T)

άτομα άνθρακα(1′ ως 5′) και είναι ο συνδετικός κρίκος ανάμεσα στην φωσφορική ομάδα και στην αζωτούχο βάση. Η φωσφορική ομάδα συνδέεται στο άτομο 5′ ενώ η βάση στο 1′, σχηματίζοντας έτσι το νουκλεοτίδιο. Επίσης εντός της μορίου του υδατάνθρακα, υπάρχει ένα μόριο υδροξυλίου (*OH*) συνδεδεμένο στο άτομο 3′. Στα σχήματα (2.1.1) και (2.1.2) φαίνεται αντίστοιχα η γενική δομή, και μια συγκεκριμένη περίπτωση, ενός νουκλεοτιδίου. Η φωσφορική ομάδα και ο υδατάνθρακας είναι ακριβώς τα ίδια όπως στο σχήμα (2.1.2) για όλα τα νουκλεοτίδια, και διαφέρει μόνο η βάση.

Τα απλά νουκλεοτίδια συνδέονται μαζί άκρη με άκρη για να σχηματίσουν νήματα DNA. Αυτό γίνεται με το να συνδέεται η 5′-φωσφορική ομάδα του ενός, με την 3′-ομάδα υδροξυλίου, σχηματίζοντας έτσι ένα ισχυρό ομοιοπολικό δεσμό, τον *φωσφοδιεστερικό δεσμό(σχήμα 2.1.3)*. Έτσι προκύπτει ένα μόριο που έχει την 5′-φωσφορική ομάδα του ενός νουκλεοτιδίου, και την 3′-ομάδα υδροξυλίου του άλλου, διαθέσιμες για την δημιουργία νέων δεσμών. Έτσι, χρησιμοποιώντας φωσφοδιεστερικούς δεσμούς, μπορούν να σχηματιστούν απλές (μονονηματικές) αλυσίδες από νουκλεοτίδια (σχήμα 2.1.4).Μια μικρή μονονηματική αλυσίδα νουκλεοτιδίων, συνήθως μικρότερη από 30 νουκλεοτίδια σε μήκος, καλείται *ολιγονουκλεοτίδιο*. Η κάθε αλυσίδα DNA έχει μια *πολικότητα* : μια αλυσίδα (ή ακολουθία) DNA είναι διαφορετική από την αντίστροφη της(ως προς τον τρόπο γραφής). Τα δύο διακεκριμένα άκρα μιας ακολουθίας DNA είναι γνωστά με το όνομα 5′-άκρο και 3′ άκρο, από το ελεύθερο άκρο της φωσφορικής ομάδας και του υδροξυλίου αντίστοιχα(σχήμα 2.1.4). Τα νουκλεοτίδια A και T όπως και τα

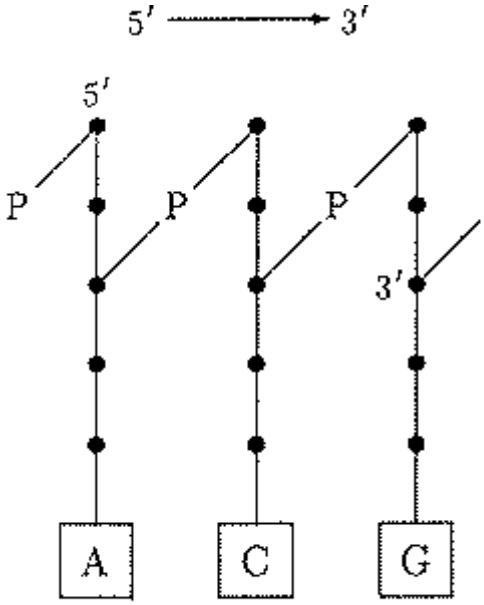
νουκλεοτίδια C και G είναι *συμπληρωματικά*. Ανάμεσα σε συμπληρωματικές βάσεις



Σχήμα 2.1.3 Φωσφοδιεστερικός δεσμός

μπορεί να σχηματιστεί ένα άλλο είδος δεσμού, που είναι ασθενής δεσμός, και λέγεται *δεσμός υδρογόνου(σχήμα 2.1.5)*. Για τη ακρίβεια ανάμεσα στα A και T σχηματίζονται 2 δεσμοί υδρογόνου, ενώ στα C και G σχηματίζονται τρεις. Δύο συμπληρωματικές μονονηματικές ακολουθίες DNA με αντίθετη πολικότητα μπορούν να ενωθούν μαζί σχηματίζοντας μια διπλή . Αυτή η διαδικασία λέγεται *ταίριασμα βάσεων* η *ανόπτηση(annealing).(σχήμα 2.1.6)* Η αντίστροφη διαδικασία – μια διπλή ακολουθία να σπάσει και να δώσει τα δύο συνιστώσα απλά νήματα- λέγεται *τήξη(melting)*. Πάλι, για την ακρίβεια, μια διπλή ακολουθία DNA έχει το σχήμα διπλής έλικας(σχήμα 2.1.7), αλλά για τους σκοπούς της εργασίας μπορεί να θεωρηθεί ότι μια διπλή ακολουθία DNA είναι απλά ένα διπλό string.

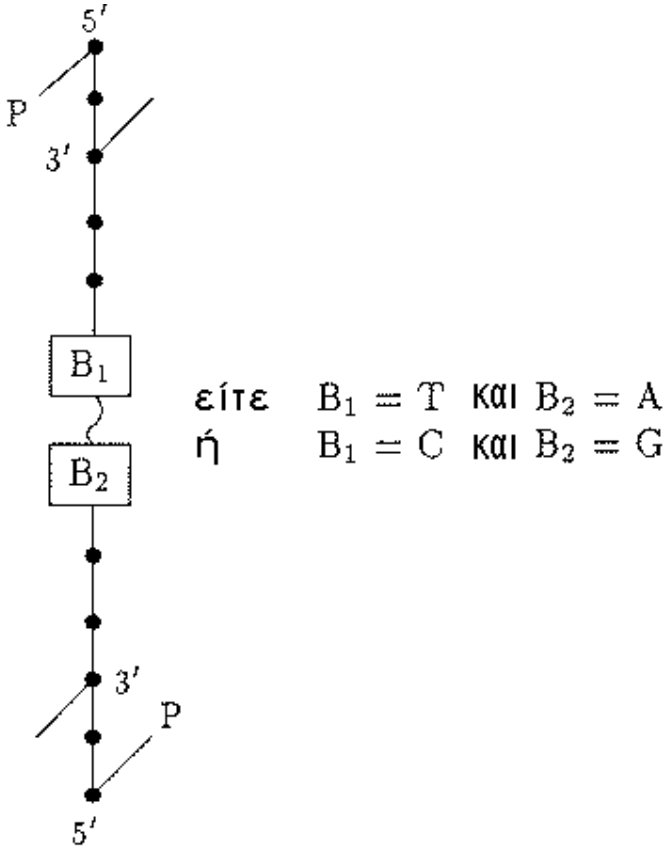
Όπως ειπώθηκε το DNA είναι ένα μέσο όπου κωδικοποιείται η γενετική πληροφορία. Με την ίδια λογική μπορούμε να χρησιμοποιήσουμε το DNA για να κωδικοποιηθεί πληροφορία που αντιστοιχεί στο στιγμιότυπο κάποιου προβλήματος. Το DNA μπορεί να θεωρηθεί ως ένα string (αλφαριθμητικό) που συνίσταται από συνδυασμό από 4 διαφορετικά σύμβολα, A,G,C,T. Μαθηματικά, αυτό σημαίνει ότι έχουμε στην διάθεση μας ένα αλφάβητο $\Sigma = \{A, G, C, T\}$ με 4 γράμματα για να κωδικοποιήσουμε πληροφορία, κάτι που είναι παραπάνω από αρκετό, θεωρώντας ότι σε ένα



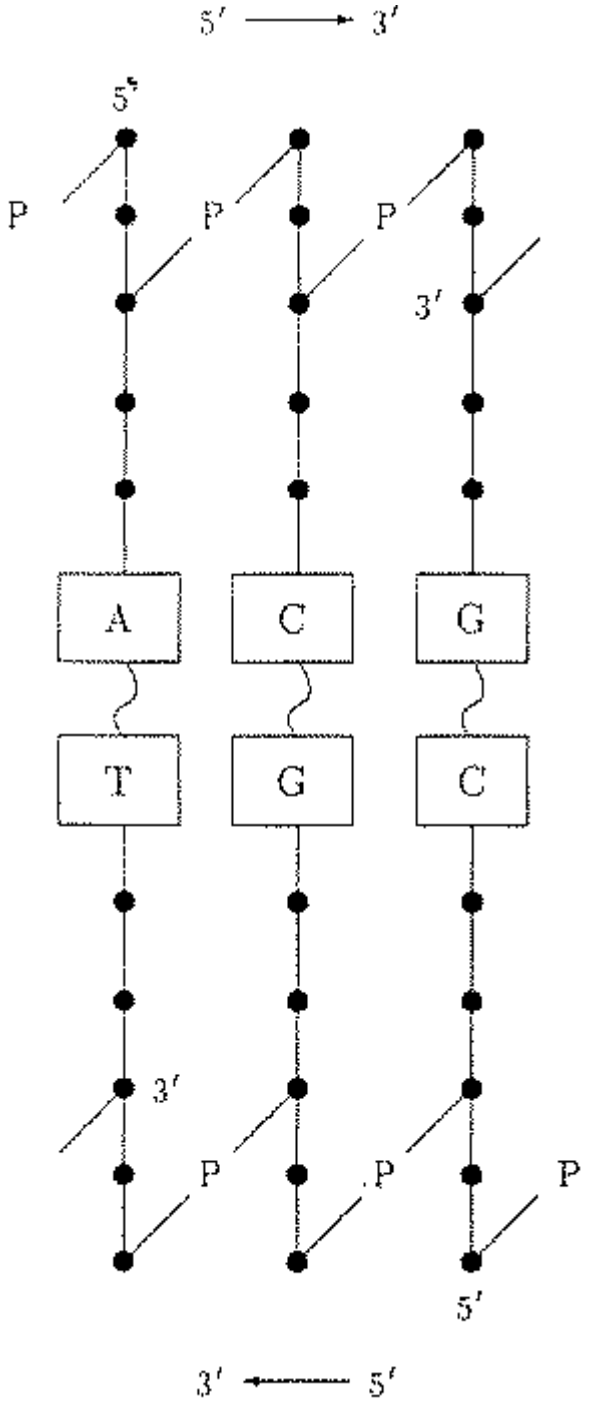
Σχήμα 2.1.4 DNA με απλή αλυσίδα

ηλεκτρονικό υπολογιστή χρειάζονται μόνο δύο ψηφία το 0 και το 1, για τον ίδιο σκοπό.

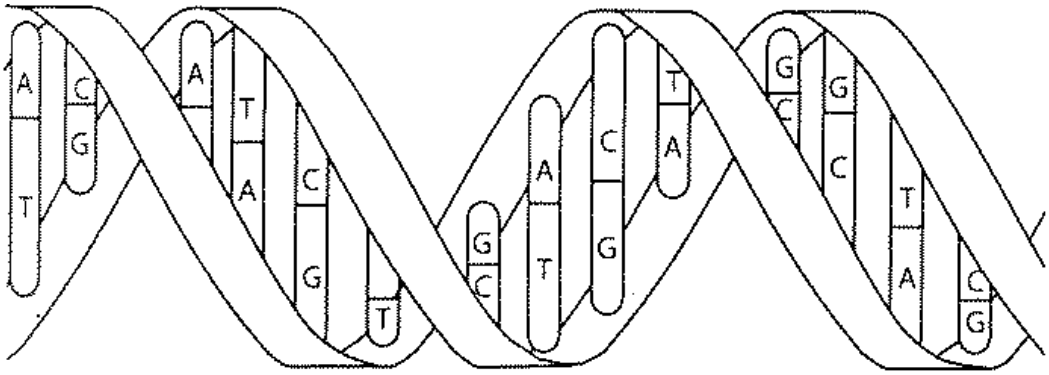
Θα χρησιμοποιηθεί ο παρακάτω συμβολισμός: Αν έχουμε ένα string x πάνω στο αλφάβητο $\Sigma = \{A, G, C, T\}$ τότε με $\uparrow x$ συμβολίζουμε το απλό μόριο DNA που αποτελείται από τα γράμματα του x και είναι προσανατολισμένο από το $5'$-άκρο προς το $3'$ -άκρο(το πρώτο γράμμα του x είναι στο $5'$-άκρο). Με $\downarrow x$ συμβολίζουμε το συμπληρωματικό του $\uparrow x$. Με $\updownarrow x$ συμβολίζεται η διπλή έλικα που περιέχει συνδεδεμένα το ένα κάτω από το άλλο το x και το συμπληρωματικό του. Για παράδειγμα το $\uparrow ACCTGC$ αναπαριστά το απλό μόριο DNA $5' - ACCTGC - 3'$. Το συμπληρωματικό αυτού συμβολίζεται με $\downarrow ACCTGC$ και είναι το απλό μόρια $3' - TGGACG - 5'$. Με $\updownarrow ACCTGC$ συμβολίζουμε τη διπλή έλικα $\begin{matrix} 5' - ACCTGC - 3' \\ 3' - TGGACG - 5' \end{matrix}$. Το $5'$ 3-μερές του $\uparrow ACCTGC$ είναι το $\uparrow ACC$ ενώ το $3'$ 3-μερές του είναι το $\uparrow TGC$. Περιστασιακά θα χρησιμοποιείται ο συμβολισμός $\overline{a}$ για το συμπληρωματικό του μορίου a και θα συνοδεύεται από τον προσδιορισμό της πολικότητας του, π.χ $3' - \overline{a}$.



Σχήμα 2.1.5 Δεσμός υδρογόνου



Σχήμα 2.1.6 Σχηματισμός διπλών αλυσίδων DNA



Σχήμα 2.1.7 Η διπλή έλικα

2.1 Οι βιολογικές λειτουργίες στο DNA

Μερικές από τις λειτουργίες που περιγράφονται παρακάτω επιτυγχάνονται από έναν αριθμό από ένζυμα που εκτελούν μερικές απλές εργασίες. Τα ένζυμα είναι *πρωτείνες* και είναι πολύ εξειδικευμένα(κάθε ένα είναι σχεδιασμένο για μια συγκεκριμένη αντίδραση) . Όλες η μερικές από αυτές τις λειτουργίες χρησιμοποιούνται για την δημιουργία προγραμμάτων που δέχονται ως είσοδο ένα δοκιμαστικό σωλήνα που περιέχει μόρια DNA και επιστρέφουν ως έξοδο «ναι» ή «όχι» ή ένα σύνολο από δοκιμαστικούς σωλήνες. Ένα υπολογισμός συνίσταται σε μια ακολουθία από σωλήνες που περιέχουν μόρια DNA.

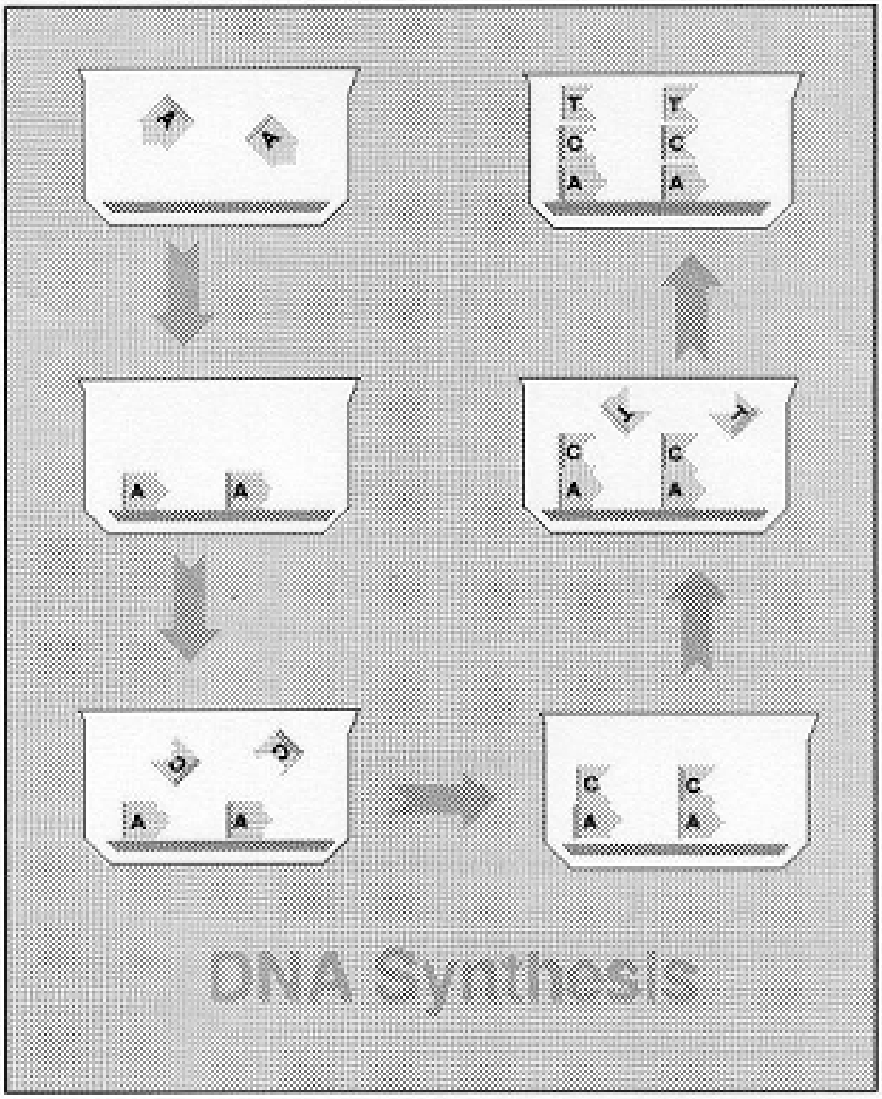
Οι βασικές βιολογικές λειτουργίες έχουν ως εξής:

-*Σύνθεση(Synthesizing)*:Γίνεται η σύνθεση ενός επιθυμητού μορίου DNA πολυωνυμικού μήκους(σχήμα 2.2.1). Ένα μόριο DNA δομείται νουκλεοτίδιο προς νουκλεοτίδιο πάνω σε ένα σώμα υποστήριξης, με διαδοχικά βήματα σύζευξης. Για παράδειγμα το πρώτο νουκλεοτίδιο έστω το A δεσμεύεται πάνω σε ένα γυάλινο σώμα. Ένα διάλυμα που περιέχει το δεύτερο νουκλεοτίδιο έστω το C χύνεται μέσα, και το A αντιδρά με το C για να σχηματίσει ένα δινουκλεοτίδιο AC. Μετά που ξεπλένεται το πλεόνασμα του C, μπορεί το C από την αλυσίδα AC να ενωθεί με ένα T για να σχηματιστεί μια 3-μερής αλυσίδα(που είναι ακόμα προσκολλημένη στην γυάλινη επιφάνεια) κ.ο.κ.

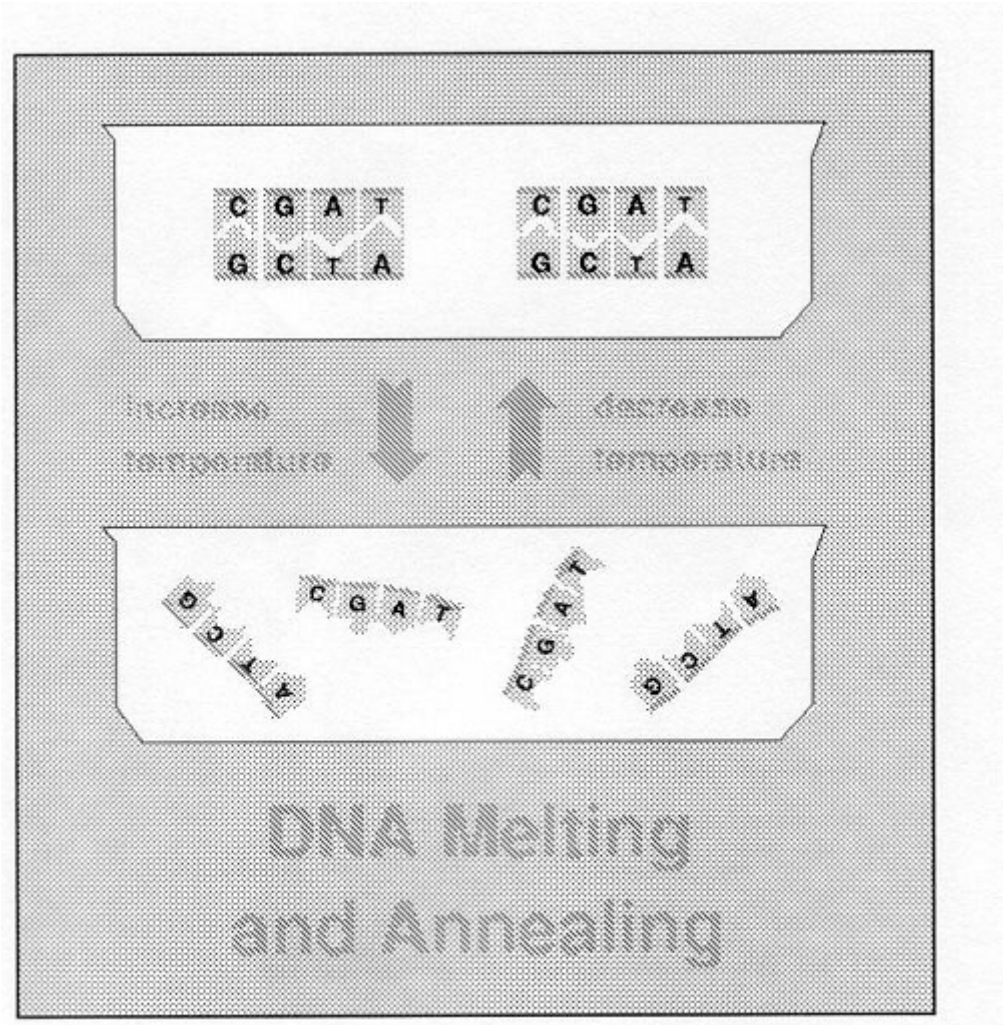
-*Ανάμειξη(mixing)*: ρίχνουμε τα περιεχόμενα δύο δοκιμαστικών σωλήνων σε ένα τρίτο για να επιτευχθεί ένωση. Η ανάμειξη μπορεί να επιτευχθεί με την ενυδάτωση των περιεχομένων των σωλήνων(αν δεν είναι ήδη σε διάλυμα) και μετά συνδυάζοντας τα διαλύματα μαζί σε ένα νέο σωλήνα. Τα περιεχόμενα των δύο σωλήνων θεωρείται ότι δεν αντιδρούν μεταξύ τους όταν γίνεται η ανάμειξη.

-*Ανόπτηση(annealing):* Η συνένωση(όχι ως προς το μήκος, αλλά η μια «κάτω» από την άλλη, με δεσμούς υδρογόνου) δύο μονών συμπληρωματικών αλυσίδων DNA. Αυτό γίνεται με την ψύξη του διαλύματος όπως φαίνεται στο σχήμα 2.2.2. Annealing in vitro(στο εργαστήριο) λέγεται υβριδοποίηση.

-*Τήξη(melting):* η διάσπαση μιας διπλής αλυσίδας DNA στις απλές συμπληρωματικές αλυσίδες από τις οποίες συνίσταται. Αυτό γίνεται με θέρμανση του διαλύματος όπως επίσης περιγράφεται στο σχήμα 2.2.2. Melting in vitro είναι επίσης γνωστό ως denaturation.



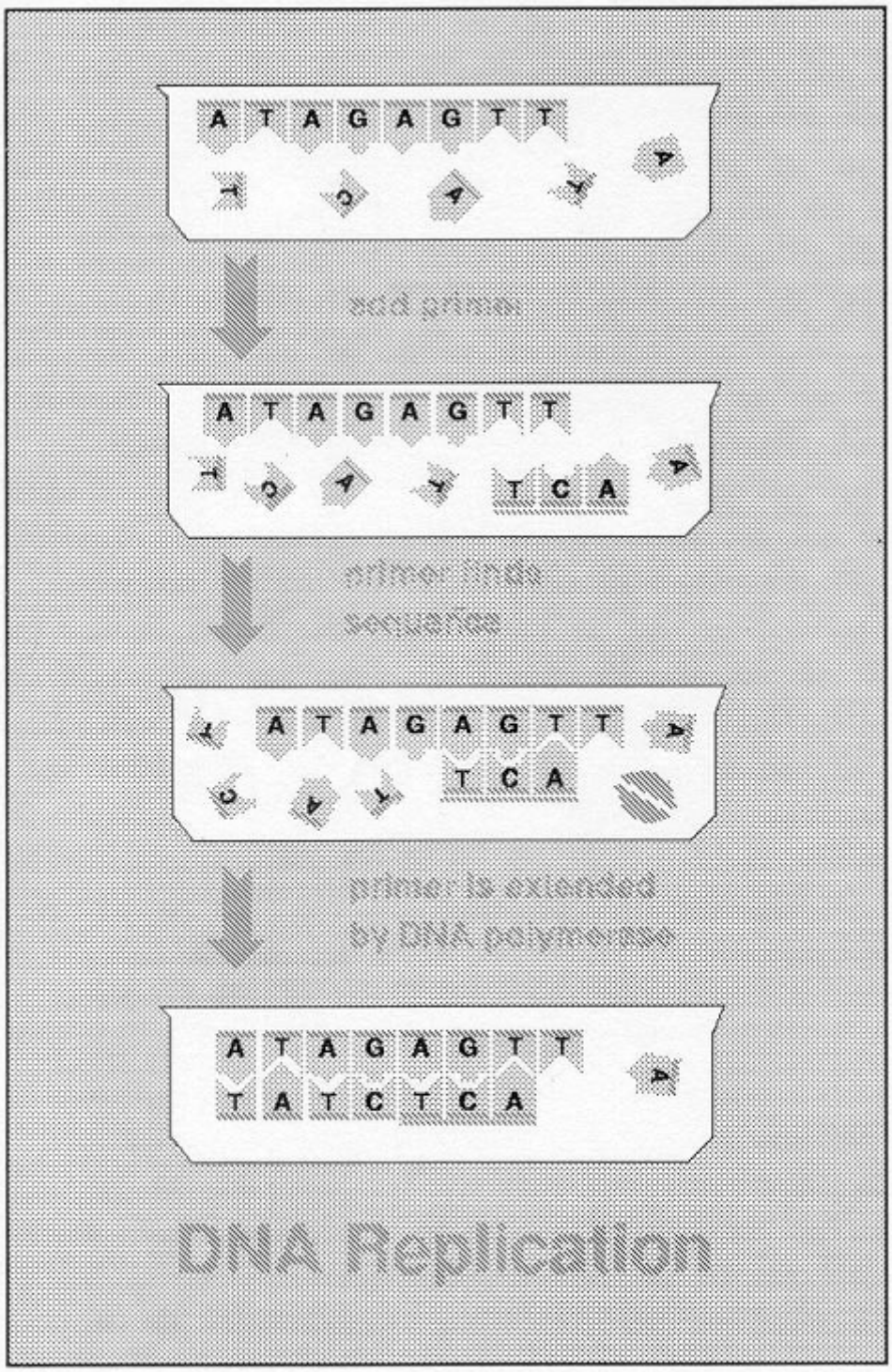
Σχήμα 2.2.1 Σύνθεση DNA



Σχήμα 2.2.2: Τήξη και ανόπτηση

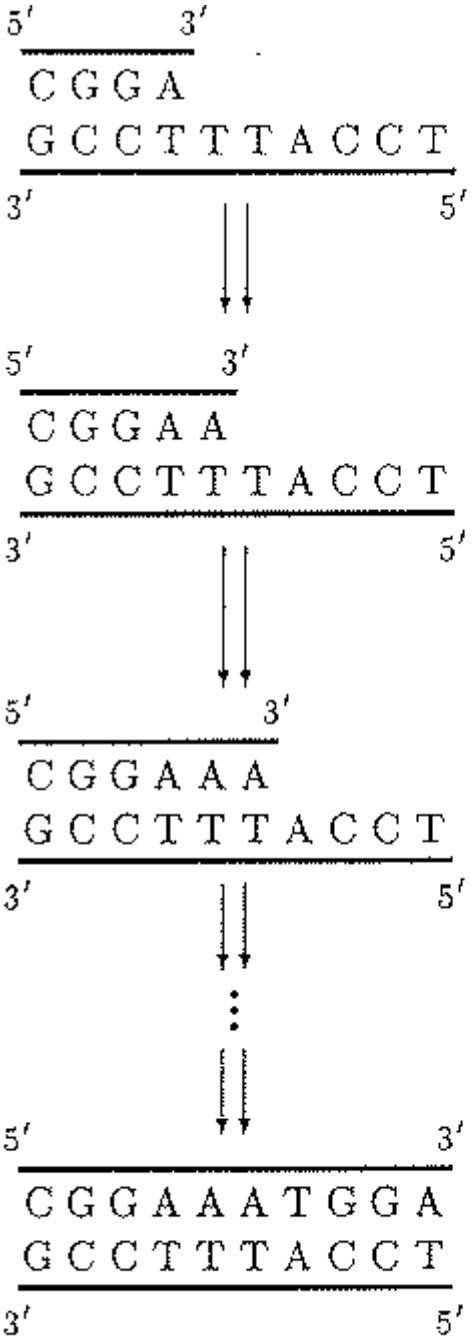
-*Ενίσχυση(Amplifying)*: Γίνονται αντίγραφα των μορίων DNA με την χρησιμοποίηση *αλυσιδωτής αντίδρασης πολυμεράσης(PCR, polymerase chain reaction)* όπως δείχνεται στο σχήμα 2.2.3. Η PCR είναι μια μέθοδος in-vitro που στηρίζεται σε DNA πολυμεράση για την γρήγορη ενίσχυση της παρουσίας συγκεκριμένων ακολουθιών DNA μέσα σε ένα διάλυμα. Με την μέθοδο PCR μπορούμε να κατασκευάσουμε το συμπληρωματικό μόριο(ή τμήμα αυτού) ενός δοσμένου απλού μορίου DNA. Το τελευταίο λέγεται και πρότυπο-οδηγός γιατί καθοδηγεί την διαδικασία. Για να προχωρήσει η διαδικασία πρέπει να χρησιμοποιηθεί ένα ολιγονουκλεοτίδιο που λέγεται primer και περιέχει την ακολουθία από την οποία θα αρχίσει η δημιουργία του συμπληρωματικού μορίου. Ο primer ανοπτάται(annealing) στο πρότυπο στην κατάλληλη θέση και στην συνέχεια το ένζυμο DNA πολυμεράση αναλαμβάνει την δημιουργία του υπόλοιπου συμπληρωματικού μορίου. Για παράδειγμα αν έχουμε το μόριο $\uparrow xyz$ όπου x, y, z ακολουθίες νουκλεοτιδίων(strings) και χρησιμοποιηθεί ως primer το $\downarrow z$ τότε μετά την ανόπτηση θα έχουμε το μόριο $\uparrow xy \updownarrow z$. Στην συνέχεια η DNA πολυμεράση θα προσθέσει νουκλεοτίδια στο ελεύθερο άκρο του z και θα δημιουργήσει την διπλή έλικα $\updownarrow xyz$. Αν είχε χρησιμοποιηθεί ως primer το $\downarrow y$ τότε

μετά την ανόπτηση θα είχαμε $\uparrow x \updownarrow y \uparrow z$ και επειδή η πολυμεράση εργάζεται προς



Σχήμα 2.2.3 Αντιγραφή DNA

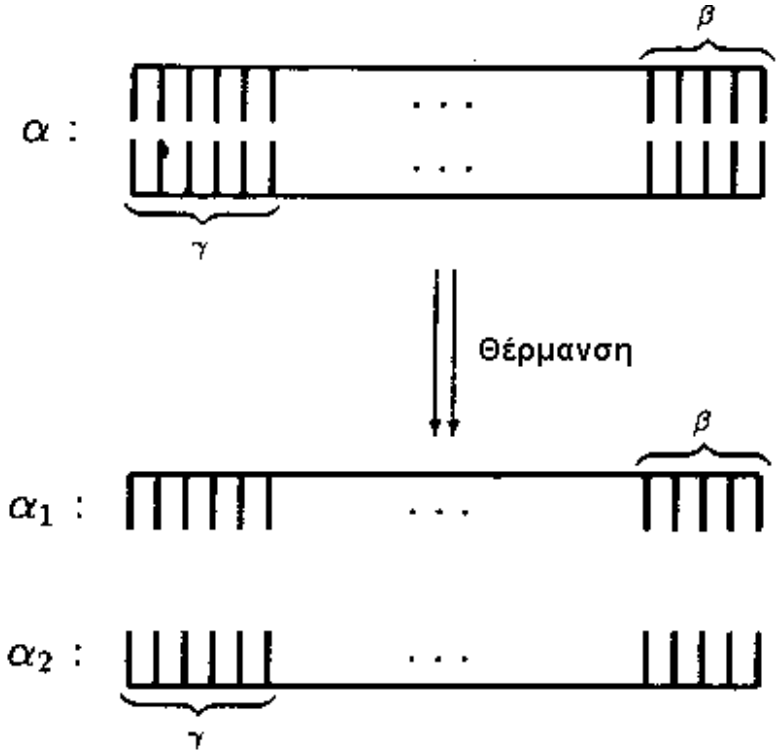
την μια κατεύθυνση($5'$ προς $3'$ του primer, δηλαδή επεκτείνει το $3'$-άκρο του primer) θα είχαμε στο τέλος το μόριο $\updownarrow xy \uparrow z$ (βλέπε και σχήμα 2.2.4). Παρατηρούμε ότι αν έχουμε να κάνουμε με πρότυπα που είναι διπλές έλικες και χρησιμοποιήσουμε 2 primers τότε μπορούμε να *πολλαπλασιάσουμε (multiplying)* τις διπλές έλικες και μάλιστα με εκθετικό ρυθμό. Η ενίσχυση μέσω PCR (για διπλά



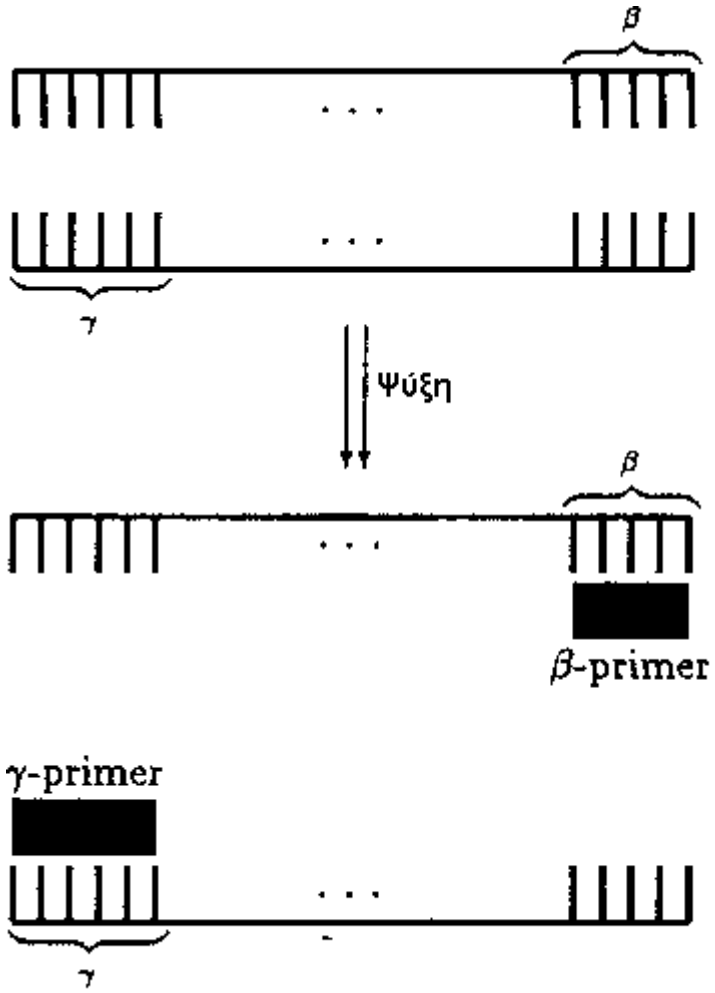
Σχήμα 2.2.4 Πολυμεράση σε δράση

μόρια DNA) περιλαμβάνει μια επαναληπτική διαδικασία από κύκλους, με το κάθε κύκλο να συνίσταται από τρία στάδια: *τήξη* του οδηγού-προτύπου DNA για να διαχωριστούν οι αλυσίδες του, μετά ψύξη για να επιτευχθεί η *ανόπτηση(annealing)* πάνω στα 2 πρότυπα, των 2 primer ολιγονουκλεοτιδίων, και τελικά επέκταση των primers προς μια κατεύθυνση από DNA πολυμεράση. Αν έχουμε ως πρότυπο το $\updownarrow \gamma\chi\beta$ μετά την θέρμανση θα έχουμε τα απλά μόρια $\uparrow \gamma\chi\beta$ και $\downarrow \gamma\chi\beta$ (σχήμα 2.2.5).

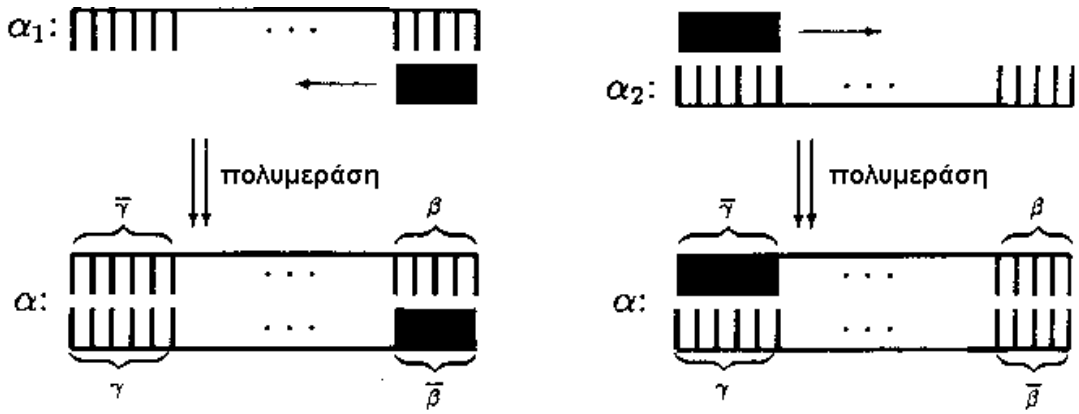
Χρησιμοποιώντας ως primers τα $\downarrow\beta$ και $\uparrow\gamma$ θα έχουμε μετά το στάδιο της ανόπτησης τα $\uparrow\gamma x\updownarrow\beta$ και $\updownarrow\gamma\downarrow x\beta$ (σχήμα 2.2.6). Η DNA πολυμεράση θα επεκτείνει το $\uparrow\gamma x\updownarrow\beta$ σε $\updownarrow\gamma x\beta$ καθώς επίσης και το $\updownarrow\gamma\downarrow x\beta$ σε $\updownarrow\gamma x\beta$ (σχήμα 2.2.7). Έτσι από ένα διπλό μόριο πήραμε δύο αντίγραφα του. Είναι εύκολο να διαπιστωθεί ότι κάθε κύκλος διπλασιάζει τον αριθμό των μορίων DNA που προέκυψαν από τον προηγούμενο κύκλο. Έτσι η αύξηση του πλήθους των μορίων DNA είναι εκθετική ως προς των αριθμό των κύκλων.



Σχήμα 2.2.5 Τήξη και διάσπαση

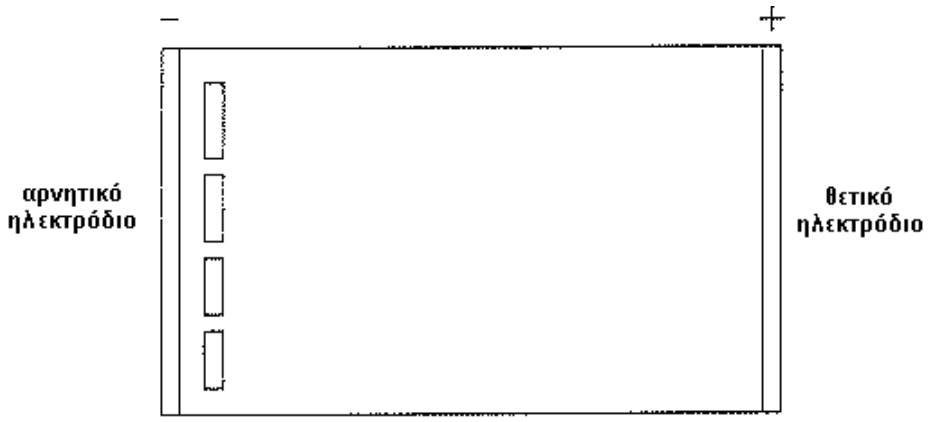


Σχήμα 2.2.6 Ανόπτηση του primer

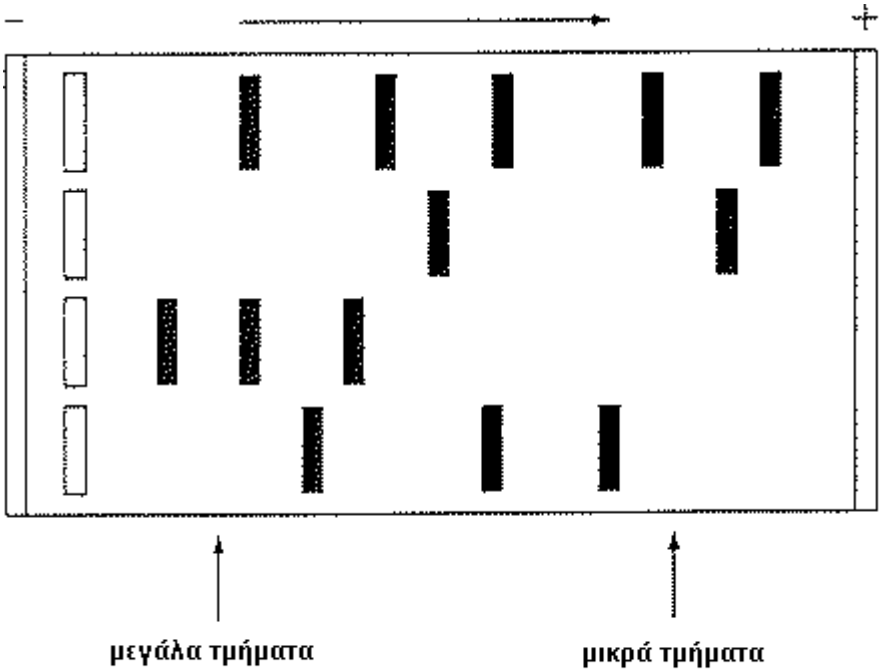


Σχήμα 2.2.7 Επέκταση των primers

- *Διαχωρισμός (Separating)* των μορίων DNA ανάλογα με το μήκος τους(τον αριθμό των νουκλεοτιδίων). Αυτό γίνεται με gel-electrophoresis. Τα μόρια τοποθετούνται πάνω σε ένα υγρό gel στο οποίο εφαρμόζεται ένα ηλεκτρικό πεδίο το οποίο τα τραβάει προ την άνοδο(θετικό ηλεκτρόδιο). Αυτό γιατί τα μόρια DNA έχουν αρνητικό φορτίο. Μάλιστα το φορτίο τους είναι ανάλογο με το μήκος τους. Αλλά και η μάζα τους το ίδιο. Έτσι αν δεν υπήρχε το gel όλα τα μόρια θα ταξίδευαν με την ίδια περίπου ταχύτητα, συνεπώς δεν θα μπορούσε να γίνει διαχωρισμός βάση της απόστασης που διανύουν, στον ίδιο χρόνο. Το gel δημιουργεί μεγαλύτερη αντίσταση (μεγαλύτερη από γραμμικό παράγοντα) στα μόρια με μεγαλύτερο μήκος. Έτσι τα μεγαλύτερα σε μήκος μόρια ταξιδεύουν πιο αργά. Έτσι μετά από την διέλευση συγκεκριμένης χρονικής περιόδου τα μόρια έχουν διαχωριστεί ανάλογα με την απόσταση που έχουν διανύσει η οποία εξαρτάται από την ταχύτητα τους και η τελευταία από το μήκος τους.(σχήματα 2.2.8,2.2.9).



Σχήμα 2.2.8 Το gel έτοιμο για ηλεκτροφόρηση



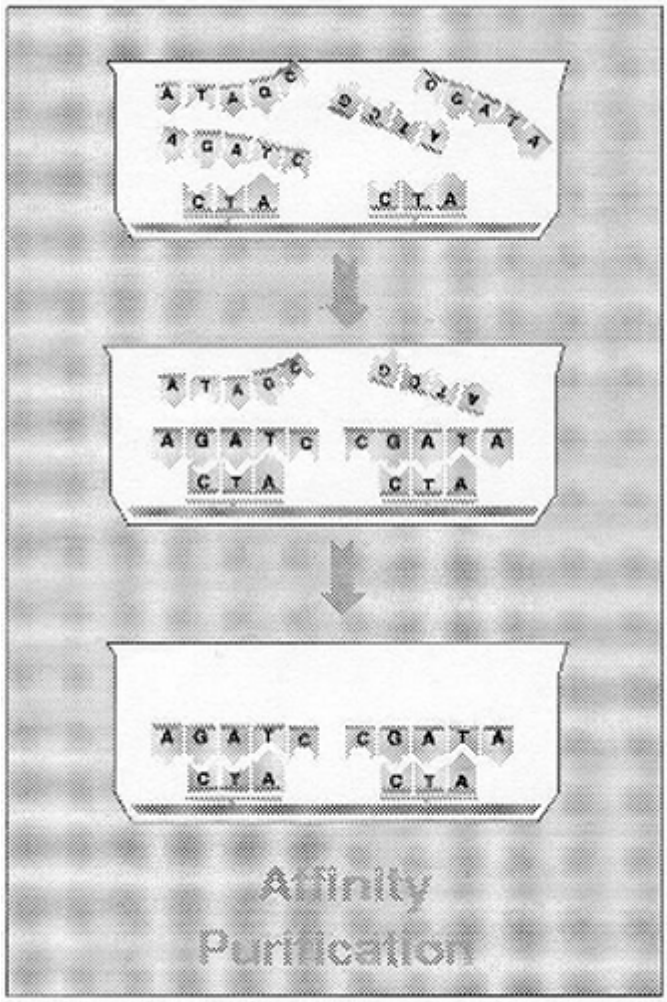
Σχήμα 2.2.9 Ηλεκτροφόρηση

Επειδή βέβαια τα μόρια DNA είναι άχρωμα, θα πρέπει πρώτα να χρωματιστούν, έτσι ώστε να είναι ορατές οι τελικές θέσεις τους μέσα στο gel. Ο χρωματισμός γίνεται είτε με βρώμιο, το οποίο όταν είναι συνδεδεμένο σε DNA, φωσφορίζει κάτω υπό υπεριώδες φώς, είτε με ραδιενεργές ουσίες που προσκολλούνται στα άκρα των μορίων, και όταν ένα film εκτεθεί στο gel, φαίνονται στο film οι διαχωρισμένες ομάδες. Ως gel μπορεί να χρησιμοποιηθεί είτε αγαρόση(agarose) ή πολυακρυλαμίδη(polyacrylamide). Η πρώτη χρησιμοποιείται για μόρια DNA με μήκος μεγαλύτερο του 500, ενώ η δεύτερη για μικρότερα σε μήκος μόρια. Τα μόρια μέσα στο gel μπορούν επίσης να ανακτηθούν με διάφορες τεχνικές.

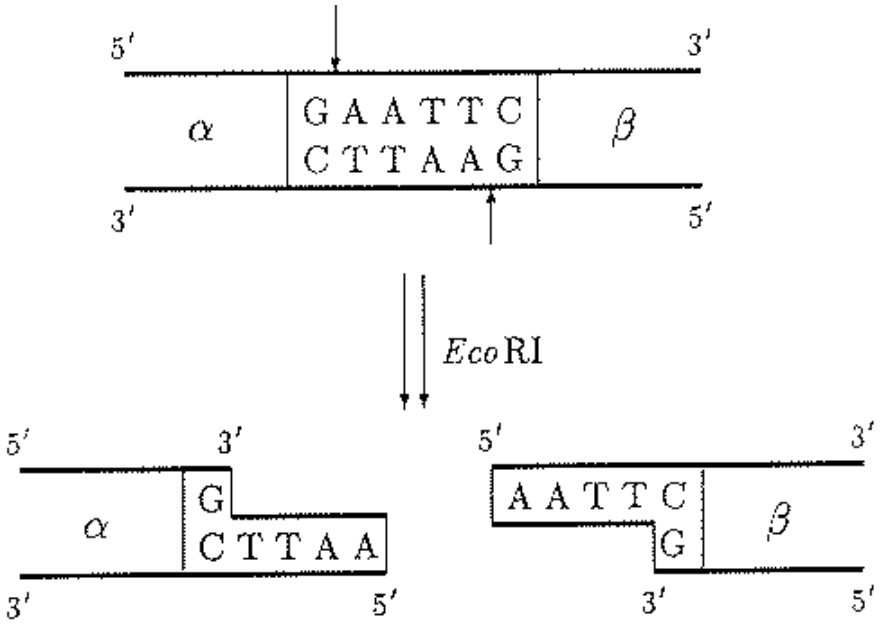
- *Εξαγωγή (Extraction)*: Εξάγονται τα μόρια DNA που περιέχουν μια συγκεκριμένη υποακολουθία νουκλεοτιδίων χρησιμοποιώντας εξαγνισμό συγγένειας(affinity purification) (σχήμα 2.2.10). Στην αρχή γίνεται *σύνθεση* μορίων $\bar{a}$ (*ανιχνευτές*, probes)που είναι συμπληρωματικά προς την ζητούμενη υποακολουθία a. Αυτά (τα $\bar{a}$)προσκολούνται σε μαγνητικά σώματα. Στην συνέχεια μέσα στο διάλυμα με τα αρχικά μόρια, ρίχνουμε τα μαγνητικά σώματα οπότε τα μόρια που έχουν την ζητούμενη υποακολουθία a με την λειτουργία της ανόπτησης προσκολούνται στα μόρια που έχουν την $\bar{a}$ και τα οποία είναι δεσμευμένα στα μαγνητικά σώματα. Στη συνέχεια τοποθετείται ένας μαγνήτης που τραβάει τα μαγνητικά σώματα προς ένα μέρος, από όπου μπορούμε και να τα πάρουμε. Άλλη μέθοδος είναι να προσκολήσουμε τους ανιχνευτές σε φίλτρα, και να ριχτεί το διάλυμα πάνω από το φίλτρο, οπότε θα μείνουν μόνο τα ζητούμενα μόρια (ανοπτημένα στους ανιχνευτές) πάνω στο φίλτρο.

-*Αποκοπή(Cutting):* Μόρια DNA διπλής έλικας κόβονται σε συγκεκριμένα σημεία όπως φαίνεται στο σχήμα 2.2.11. Αυτό επιτυγχάνεται με μια ομάδα ενζύμων που

λέγονται περιοριστικές ενδονουκλεάσες. Αυτά αναγνωρίζουν μια συγκεκριμένη



Σχήμα 2.2.10 Εξαγνισμός συγγένειας(Εξαγωγή)

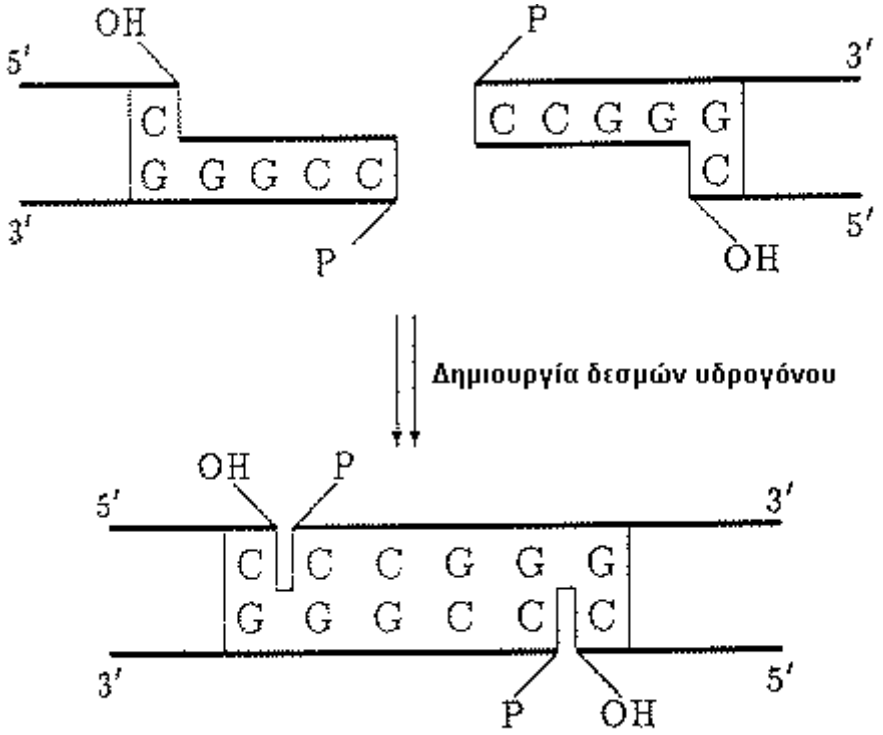


Σχήμα 2.2.11 Ένα ένζυμο αποκοπής σε δράση

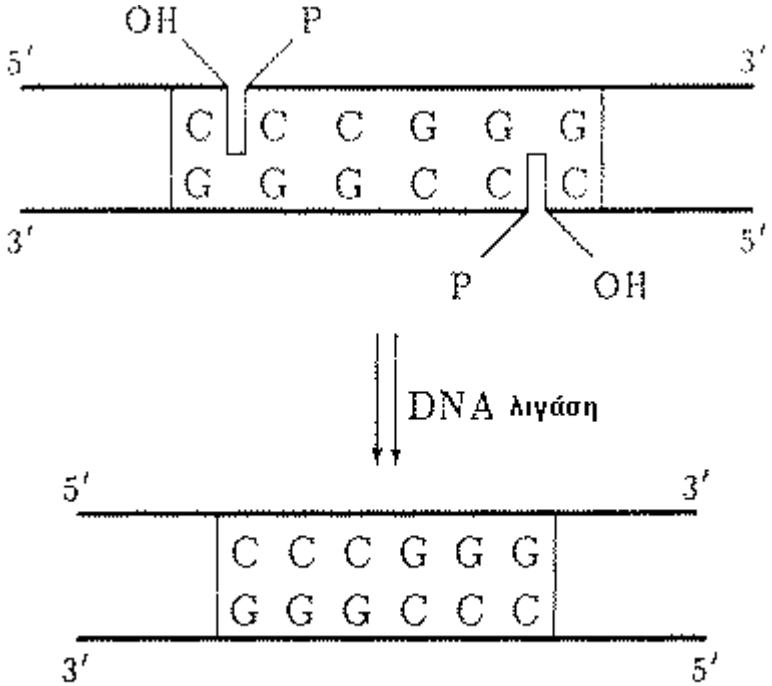
υποακολουθία μικρού μήκους που λέγεται *σημείο αποκοπής*. Οποιοδήποτε μόριο DNA με διπλή έλικα που περιέχει το *σημείο αποκοπής* μέσα στην ακολουθία του, κόβεται από το ένζυμο σε αυτή την τοποθεσία. Υπάρχουν επίσης ένζυμα, για την αποκοπή απλών αλυσίδων DNA.

-*Συνένωση(ligating):* Δύο μόρια DNA με *συμβατά κολλώδη άκρα*(συμπληρωματικά και αντίθετης πολικότητας) συνενώνεται κατά μήκος. Αυτό γίνεται χρησιμοποιώντας ένζυμο που λέγεται DNA-λιγάση. Ανάμεσα στα κολλώδη άκρα δημιουργούνται δεσμοί υδρογόνου, αν τα μόρια βρεθούν αρκετά κοντά. Για να δημιουργηθεί, όμως, και ο φωσφοδιεστερικός δεσμός είναι απαραίτητη η βοήθεια της λιγάσης. Το έργο της DNA-λιγάσης, διευκολύνεται από την παρουσία των νεοδημιουργηθέντων δεσμών υδρογόνου, γιατί έτσι κρατούνται αρκετά κοντά οι δύο απλές ακολουθίες(σχήματα 2.2.12, 2.2.13). Η DNA-λιγάση μπορεί να ενώσει και μόρια DNA ακόμη και όταν αυτά δεν έχουν κολλώδη άκρα. Το έργο της τότε είναι πιο δύσκολο γιατί δεν υπάρχουν οι δεσμοί υδρογόνου που κρατούν κοντά τα προς συνένωση μόρια. Έτσι η λιγάση δε μπορεί τόσο εύκολα, να αποκαταστήσει τους φωσφοδιεστερικούς δεσμούς. Αυτή η τελευταία περίπτωση είναι γνωστή και ως blunt-end ligation(σχήμα 2.2.14).

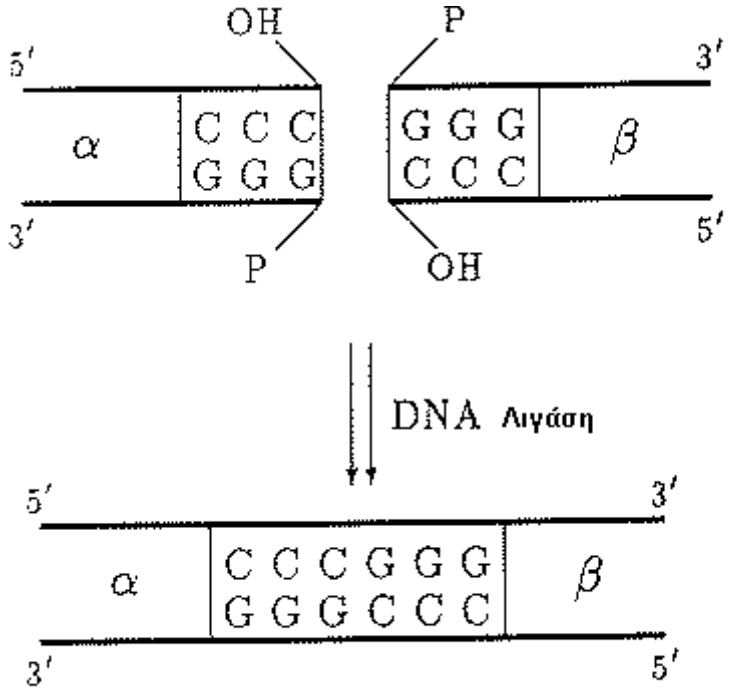
-*Επέκταση(Append):* Παρόμοια λειτουργία με τη *συνένωση* με τη διαφορά ότι χρησιμοποιείται πολυμεράση και παραλλαγή της μεθόδου PCR, καθότι επίσης απαιτείται να γνωρίζουμε τμήμα του διπλού μορίου που θέλουμε να επεκτείνουμε. Πιο αναλυτικά, αν έχουμε το μόριο $\updownarrow Xy$ όπου y γνωστό, τότε μετά την τήξη παίρνουμε $\uparrow Xy$ και $\downarrow Xy$. Κάνοντας *εξαγωγή* κρατάμε μόνο τα $\uparrow Xy$. Χρησιμοποιώντας ως primer $\downarrow yZ$, (όπου Z η ακολουθία κατά την οποία θέλουμε να επεκταθεί το $\updownarrow Xy$), έχουμε μετά την ανόπτηση του $\downarrow yZ$ στο $\uparrow Xy$, το $\uparrow X \updownarrow y \downarrow Z$. Η πολυμεράση θα επεκτείνει το $3'$ άκρο του $\downarrow y$ για να παράγει το $\downarrow X$, καθώς επίσης και το $3'$ άκρο του $\uparrow y$ για να παράγει το $\uparrow Z$. Όλα αυτά βέβαια γίνονται πάνω στο μόριο $\uparrow X \updownarrow y \downarrow Z$ και τελικά προκύπτει το $\updownarrow XyZ$.



Σχήμα 2.2.12 Ταίριασμα συμπληρωματικών αλυσίδων



Σχήμα 2.2.13 Σχηματισμός φωσφοδιεστερικών δεσμών με τη βοήθεια λιγάσης



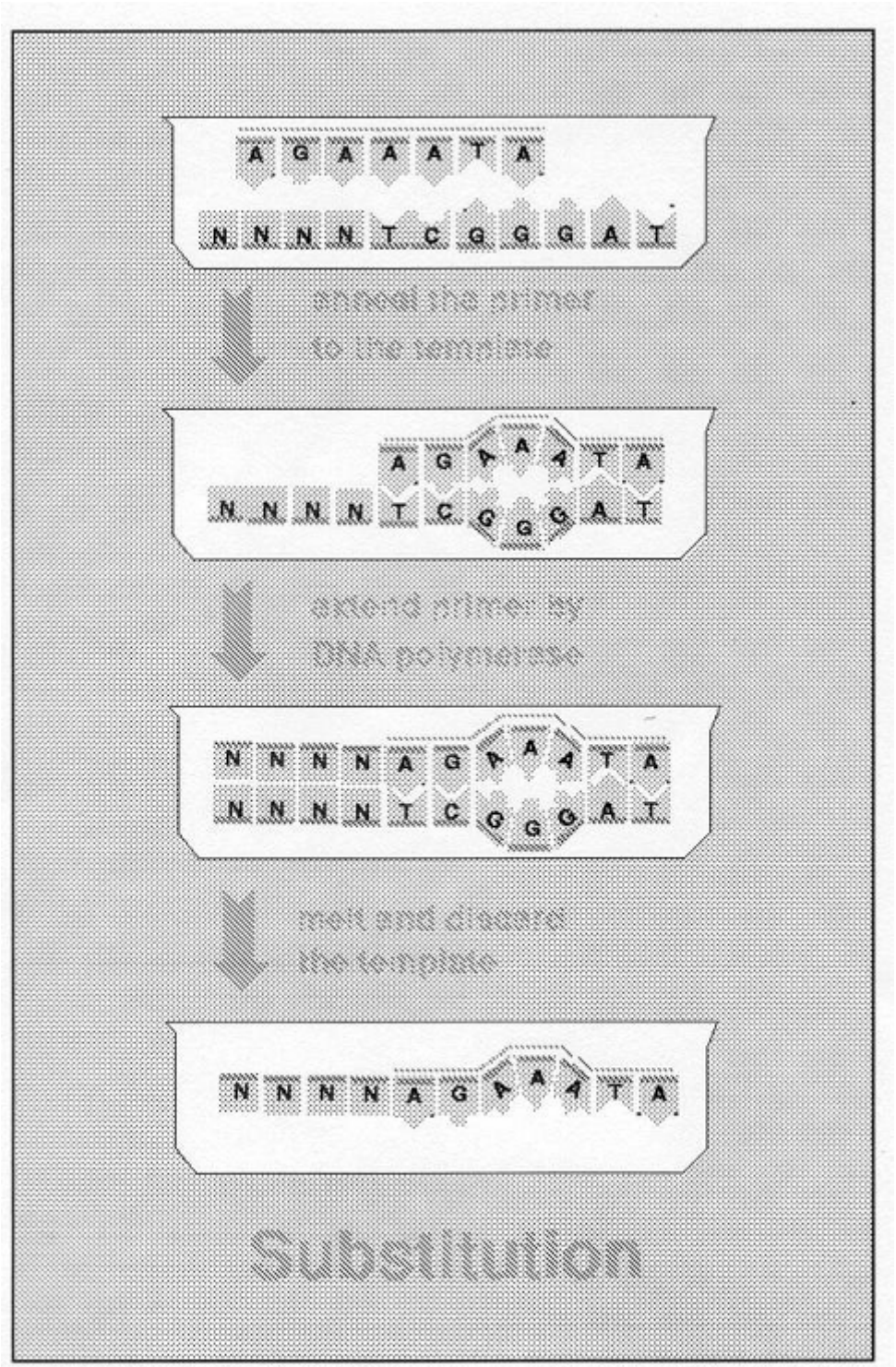
Σχήμα 2.2.14 Απευθείας αντίδραση λιγάσης

-*Αντικατάσταση(Substitution)*: Αντικατάσταση, εισαγωγή ή διαγραφή ακολουθιών DNA με τη χρησιμοποίηση μιας παραλλαγής της μεθόδου PCR (σχήμα 2.2.15). Εδώ χρησιμοποιούνται primers που είναι συμπληρωματικοί μόνο με ένα τμήμα του προτύπου μορίου. Ο primer θα πρέπει να περιέχει μια αρκετά μεγάλη σε μήκος συμπληρωματική ακολουθία προς αυτή του προτύπου για να μπορεί προσκολληθεί σε αυτό (anneal) παρά την ανομοιότητα του στο σύνολο της ακολουθίας. Μετά την επέκταση του primer από πολυμεράση, σύμφωνα με το πρότυπο, ο primer θα αποτελείται από τη συμπληρωματική ακολουθία του προτύπου πλην ορισμένων νουκλεοτιδίων που θα έχουν αντικατασταθεί από κάποια άλλα(αυτά που είχε αρχικά ο primer και δεν ήταν συμπληρωματικά με τμήμα του προτύπου).

- *Μαρκάρισμα(Marking)*: Απλές αλυσίδες μαρκάρονται με το να προσκολλούνται σε αυτές οι συμπληρωματικές τους. Δημιουργούνται έτσι διπλές αλυσίδες. Αυτό επιτυγχάνεται με υβριδοποίηση. Η αντίστροφη λειτουργία λέγεται *ξεμαρκάρισμα(unmarking)* και επιτυγχάνεται με *denaturation*. Τώρα οι διπλές αλυσίδες γίνονται μονές με την απόσπαση της συμπληρωματικής αλυσίδας. Οι μαρκαρισμένες ακολουθίες θα είναι διπλές, ενώ οι αμαρκάριστες μονές.

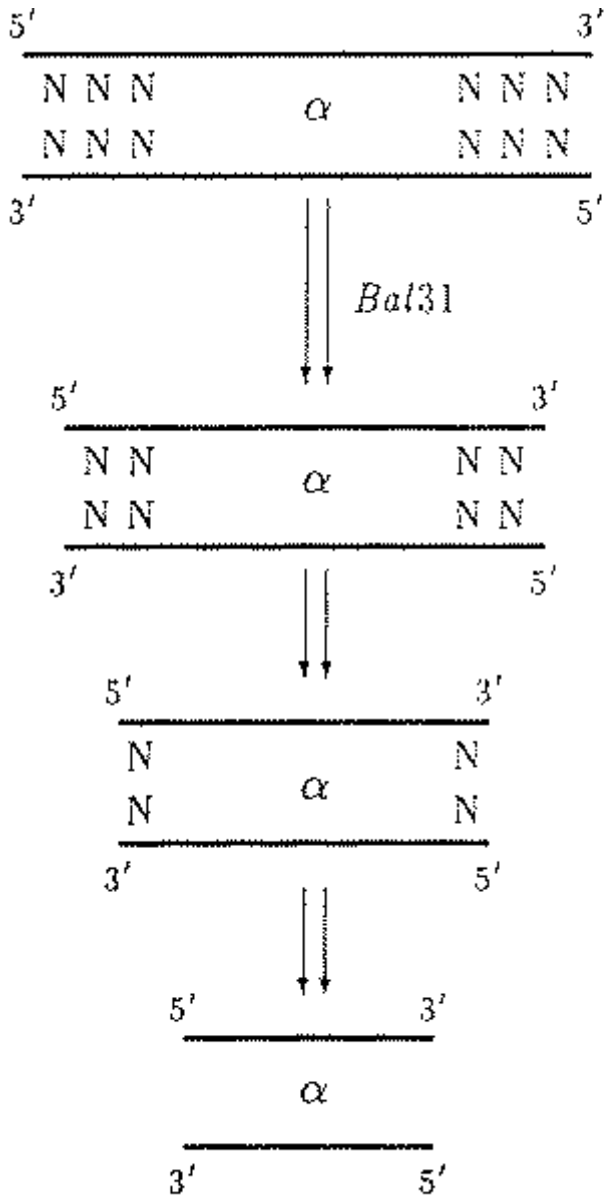
- Καταστροφή*(Destroying)*: Καταστρέφονται οι μαρκαρισμένες αλυσίδες χρησιμοποιώντας εξωνουκλεάσες που «τρώνε» το μόριο DNA από τα άκρα (σχήμα 2.2.16). Οι εξωνουκλεάσες είναι πιο ευέλικτες από τις πολυμεράσες(οι οποίες επίσης έχουν καταστροφική δραστηριότητα), υπό την έννοια ότι μπορούν να λειτουργούν και προς τις δύο κατευθύνσεις. ($5' \to 3'$ και $3' \to 5'$). Αυτή η λειτουργία μπορεί επίσης να επιτευχθεί με αποκοπή των μαρκαρισμένων μορίων με ένα περιοριστικό ένζυμο(restriction enzyme)(κάνοντας έτσι να έχουν μικρότερο μήκος) και με απομάκρυνση των αμαρκάριστων μορίων με gel- electrophoresis.

- *Ανίχνευση και ανάγνωση(Detecting and Reading):* με δεδομένα τα περιεχόμενα ενός δοκιμαστικού σωλήνα, δώσε απάντηση «ναι» αν περιέχει τουλάχιστον ένα μόριο DNA αλλιώς απάντησε «όχι». Η μέθοδος PCR μπορεί να χρησιμοποιηθεί για να ενισχυθεί το αποτέλεσμα και στην συνέχεια μια μέθοδος που λέγεται sequencing



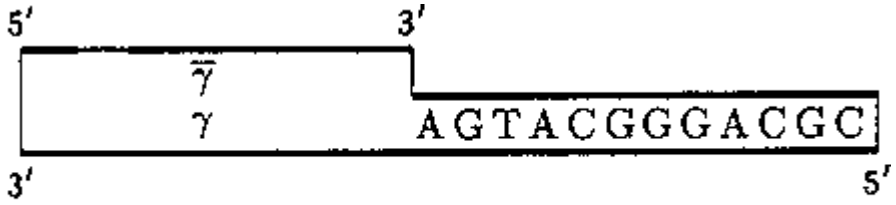
Σχήμα 2.2.15 Αντικατάσταση

χρησιμοποιείται για να γίνει το διάβασμα του διαλύματος. Η βασική ιδέα αυτής της ευρέως χρησιμοποιούμενης μεθόδου είναι να χρησιμοποιηθεί PCR και gel-electrophoresis. Υποθέτουμε ότι έχουμε ένα ομογενές διάλυμα, δηλαδή, ένα διάλυμα που περιέχει κυρίως αντίγραφα του μορίου DNA που θέλουμε να ανιχνεύσουμε, (να βρούμε ποια είναι η ακολουθία των βάσεων του) και πολύ λίγα μόρια από άλλα μόρια DNA. Έστω *a* το κυρίαρχο μόριο DNA. Για την ανίχνευση π.χ των θέσεων των



Σχήμα 2.2.16 Μία εξωνουκλεάση σε δράση

νουκλεοτιδίων Α στο μόριο DNA, χρησιμοποιείται ένας blocking agent ddA που εμποδίζει τους primer να επεκταθούν πέρα από τα ddA κατά την διάρκεια της μεθόδου PCR. Υπάρχουν επίσης ddT,ddC,ddG. Σε όλα αυτά έχει αντικατασταθεί το υδροξύλιο του υδατάνθρακα που έχουν, με ένα άτομο υδρογόνου. Έτσι χάνεται η δυνατότητα τους να δημιουργήσουν φωσφοδιεστερικούς δεσμούς. Η όλη διαδικασία βασίζεται στο ότι, κατά την διάρκεια του PCR η πολυμεράση μπορεί να επιλέξει π.χ ddA αντί για Α, και έτσι να σταματήσει η επέκταση στο ddA. Ως αποτέλεσμα αυτής της τροποποιημένης μεθόδου PCR, προκύπτει ένα πληθυσμός από υποακολουθίες του συμπληρωματικού του κύριου μορίου DNA που η κάθε μια αντιστοιχεί σε μια διαφορετική θέση του νουκλεοτιδίου Α στο συμπληρωματικό του κύριου μόριου DNA(αυτό που θέλουμε να ταυτοποιήσουμε). Στην συνέχεια γίνεται *διαχωρισμός*

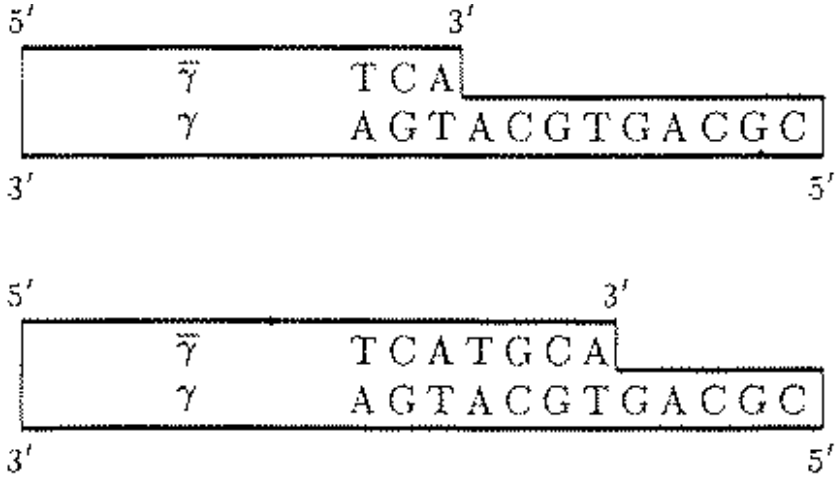


Σχήμα 2.2.17 β μόριο

αυτού του πληθυσμού κατά μήκος *με gel electrophoresis*. Έτσι προκύπτουν οι θέσεις των ddA στο συμπληρωματικό μόριο DNA. Η διαδικασία μπορεί να επαναληφθεί για να αποκαλυφθούν οι θέσεις των ddC,ddG,ddT. Έχοντας διαβάσει έτσι την ακολουθία στο συμπληρωματικό του, μπορούμε να συμπεράνουμε και την ακολουθία στο κύριο μόριο.

Σχήμα 2.2.18 Πλήρες διπλό μόριο

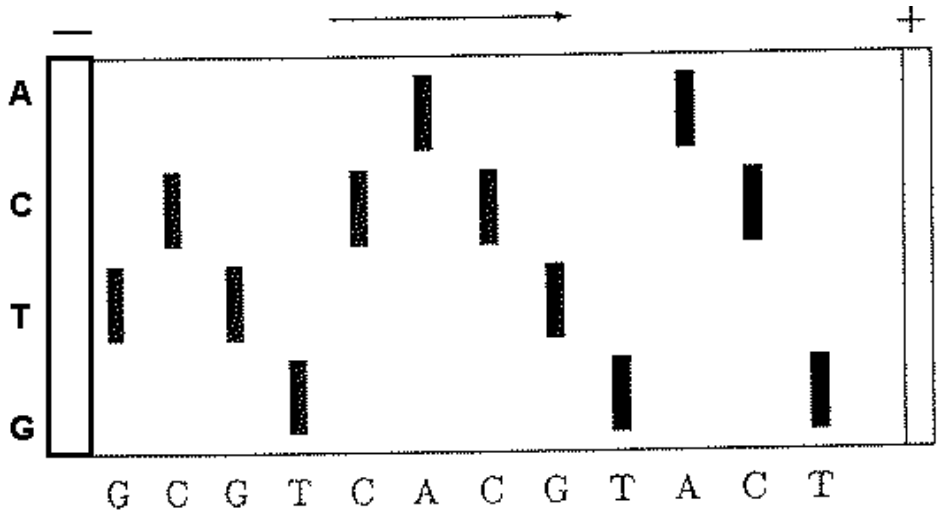
Πιο αναλυτικά η μέθοδος sequencing έχει ως εξής: Καταρχήν επεκτείνουμε το μόριο a, προσθέτωντας μια μικρή ακολουθία γ μπροστά από το $3'$-άκρο του. Έστω $\beta = 3'\gamma\alpha$ το νέο μόριο. Ο λόγος που μπαίνει το γ είναι για να μπορεί να λειτουργήσει η PCR, και να επεκτείνει το μόριο κατά το πρότυπο a, θεωρώντας ως primer $\bar{\gamma}$. Μάλιστα ο $\bar{\gamma}$ μαρκάρεται με κάποια φθορίζουσα ουσία, έτσι ώστε να μπορούμε εύκολα να βρούμε τα μόρια που αρχίζουν με $\bar{\gamma}$. Έστω β' το β με τον primer $\bar{\gamma}$ (σχήμα 2.2.17). Τώρα, ετοιμάζουμε 4 σωλήνες (σωλήνας A, σωλήνας T, σωλήνας C, σωλήνας G), έτσι ώστε ο καθένας από αυτούς να έχει μόρια β, primers $\bar{\gamma}$,πολυμεράση, και νουκλεοτίδια A,T,C,G. Επιπλέον ο σωλήνας A έχει κάποια μικρή ποσότητα από ddA, ο σωλήνας T κάποια ποσότητα από ddT, και αντίστοιχα για τους άλλους σωλήνες. Ας δούμε τι συμβαίνει στο σωλήνα A. Η πολυμεράση θα επεκτείνει τον primer $\bar{\gamma}$ του β', χρησιμοποιώντας τα νουκλεοτίδια στο σωλήνα A. Χρησιμοποιώντας μόνο νουκλεοτίδια A,T,C,G, το β' επεκτείνεται σε πλήρες διπλό μόριο. (σχήμα 2.2.18). Μερικές φορές, η πολυμεράση θα επιλέξει ddA, αντί για A. Όταν αυτό συμβεί, τότε η συμπλήρωση του primer θα σταματήσει στο ddA, γιατί δεν έχει το $3'-OH$, που είναι απαραίτητο για τον φωσφοδιεστερικό δεσμό. Έτσι εκτός από τα πλήρη διπλά μόρια θα έχουμε και τα μόρια στο σχήμα 2.2.19.



Σχήμα 2.2.19: Μη πλήρη μόρια στον δοκιμαστικό σωλήνα A

Στην συνέχεια μπορούμε εύκολο να πάρουμε τις απλές ακολουθίες που αρχίζουν από $\bar{\gamma}$, αφού πρώτα θερμάνουμε το διάλυμα, και στη συνέχεια παίρνοντας τα μόρια που αρχίζουν με $\bar{\gamma}$, αφού ο $\bar{\gamma}$ είναι μαρκαρισμένος με κάποια ουσία. Κάνοντας την ίδια διαδικασία, στους υπόλοιπους σωλήνες, παίρνουμε παρόμοια πληθυσμούς από απλά μόρια που αρχίζουν από $\bar{\gamma}$, και φανερώνουν τις θέσεις των T,C,G στο $\bar{a}$. Για να βρούμε αυτές τις θέσεις θα πρέπει να βρούμε το μήκος του κάθε μορίου. Συνεχίζουμε λοιπόν με gel electrophoresis, χρησιμοποιώντας polyacrylamide gel, και τέσσερα σημεία μέσα στο gel, όπου χύνουμε τα περιεχόμενα του κάθε σωλήνα. Έτσι προκύπτει το σχήμα 2.2.20 Η κατανομή των μορίων σχηματίζει μια σκάλα, όπου το κάθε σκαλί περιέχει μόρια με το ίδιο μήκος, και ένα σκαλί προηγείται κάποιου άλλου, αν έχει ακριβώς ένα νουκλεοτίδιο λιγότερο. Η σκάλα διαβάζεται από τα δεξιά προς τα αριστερά, γιατί τα μικρότερα σε μήκος μόρια βρίσκονται πιο δεξιά. Έτσι προκύπτει ότι το συμπληρωματικό του α είναι $\bar{a} = 5' - TCATGCACTGCG$, από προκύπτει εύκολα ότι το $\alpha = 3' - AGTACGTGACGC$.

Οι παραπάνω λειτουργίες, μαζί με τη δομή του μορίου DNA που μπορεί να χρησιμοποιηθεί για την κωδικοποίηση πληροφορίας, αποτελούν το βασικό μοντέλο υπολογισμού, μπορούν δε να χρησιμοποιηθούν, όπως ειπώθηκε στην αρχή, για την εγγραφή βιολογικών προγραμμάτων που λύνουν κάποια δύσκολα προβλήματα. Ένα τέτοιο παράδειγμα προγράμματος περιγράφεται στην ακόλουθη παράγραφο.



Σχήμα 2.2.20: Η "σκάλα" της διαδικασίας sequencing

2.3 Το πείραμα του Adleman

Ο Adleman το 1994 στην προσπάθεια του να λύσει ένα στιγμιότυπο του Directed Hamiltonian Path Problem, με μέσα μοριακής βιολογίας, δημιούργησε τον πρώτο υπολογιστή DNA και συγχρόνως το πρώτο "βιολογικό πρόγραμμα" που χρησιμοποιούσε κάποιες από τις λειτουργίες που περιγράφονται στην 2.2. Στο Directed hamiltonian path πρόβλημα δίνεται ένα προσανατολισμένο γράφημα ως είσοδος και ζητείται να βρεθούν σε αυτό(αν υπάρχουν) μονοπάτια που αρχίζουν από μια συγκεκριμένη κορυφή υ_{in} τερματίζουν σε μια άλλη επίσης δοσμένη κορυφή υ_{end} και περνάνε από κάθε κορυφή ακριβώς μια φορά. Μονοπάτι σε ένα προσανατολισμένο γράφημα είναι μια ακολουθία από τόξα (e_i) τέτοια ώστε το τέλος του e_i να είναι η αρχή του e_{i+1} για κάθε i. Το Directed Hamiltonian Path Problem(DHP) είναι γνωστό NP-complete πρόβλημα. Για NP-complete προβλήματα βλέπε δεύτερο μέρος.

Πρώτα θα δοθεί ο αλγόριθμος και στην συνέχεια θα περιγραφεί ο τρόπος που υλοποιήθηκε μέσα στα πλαίσια του μοντέλου της 2.2.

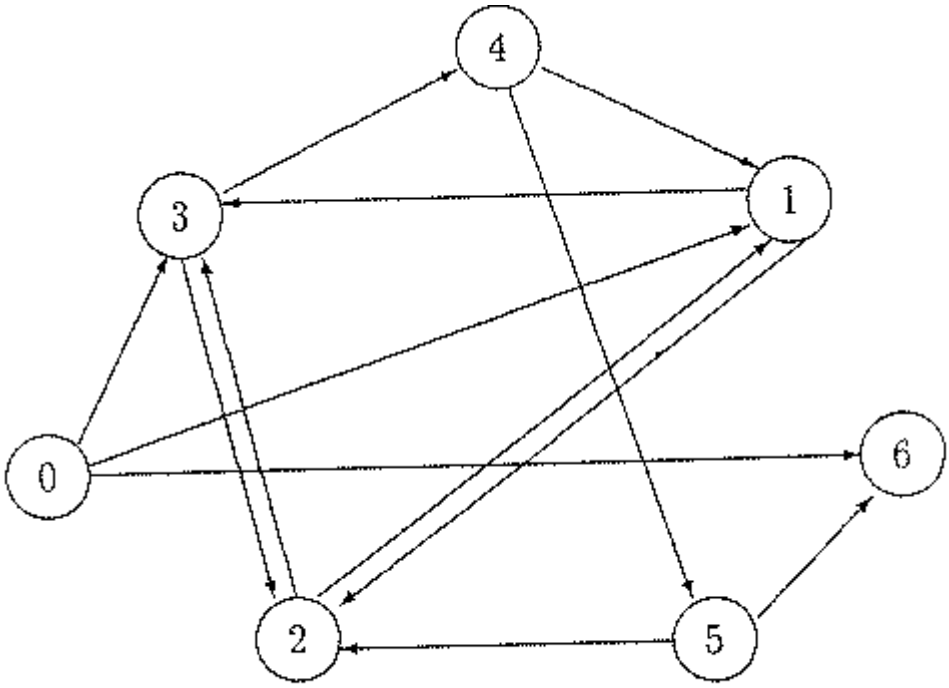
Ο ακόλουθος μη ντετερμινιστικός αλγόριθμος λύνει το πρόβλημα:

1. Δημιουργία τυχαίων μονοπατιών μέσα στο γράφημα.
2. Κράτησε τα μονοπάτια που αρχίζουν με υ_{in} και τελειώνουν σε υ_{end}
3. Έστω n το πλήθος των κορυφών του δοσμένου γραφήματος. Κράτησε τα μονοπάτια που περνάνε από ακριβώς n κορυφές.
4. Κράτησε τα μονοπάτια που περνάνε τουλάχιστον μια φορά από κάθε κορυφή του γραφήματος.
5. Αν παραμένουν μονοπάτια απάντησε «ΝΑΙ» αλλιώς δώσε απάντηση «ΟΧΙ»

Κάθε βήμα δέχεται ως είσοδο ένα σύνολο μονοπατιών και δίνει ως έξοδο ένα άλλο σύνολο μονοπατιών που πάει ως είσοδο στο επόμενο βήμα.

Τα βήματα του παραπάνω αλγορίθμου υλοποιούνται ως εξής στα πλαίσια του μοντέλου DNA.

Για την υλοποίηση του βήματος 1 κάθε κορυφή του γραφήματος(σχήμα 2.3.1) κωδικοποιήθηκε σε ένα απλό μόριο DNA με 20 νουκλεοτίδια(ουσιαστικά ακολουθία από 20 γράμματα με αλφάβητο 4 γραμμάτων). Για κάθε τόξο (u,v) του γραφήματος



Σχήμα 2.3.1 Το γράφημα στο πείραμα του Adleman

δημιουργήθηκε μια ακολουθία DNA που τα 10 πρώτα νουκλεοτίδια της είναι τα 10 τελευταία νουκλεοτίδια του μορίου DNA που αντιστοιχεί στην u (ξεκινώντας από τη μέση και καταλήγοντας στο $3'$ άκρο του μορίου της u, δηλαδή το $3'$ 10-μερές του ολιγονουκλεοτιδίου της u), ενώ τα 10 τελευταία νουκλεοτίδια της είναι τα 10 πρώτα του μορίου DNA που αντιστοιχεί στην v (ξεκινώντας από το $5'$ άκρο, το $5'$ 10-μερές του ολιγονουκλεοτιδίου της v). Δηλαδή αν $u = u'u''$ και $v = v'v''$, όπου u' τα 10 πρώτα νουκλεοτίδια του μορίου που αντιστοιχεί στην u, και u'' τα δέκα τελευταία, και αντίστοιχα για το v)τότε το τόξο (u, v) κωδικοποιείται ως $u''v'$. Όταν $u = \upsilon_{in}$ τότε χρησιμοποιείται όλο το μόριο που κωδικοποιεί την u (δηλαδή έχουμε uv'), ενώ αν $v = \upsilon_{out}$, τότε χρησιμοποιείται όλο το μόριο της v ($u''v$). Επίσης χρησιμοποιήθηκαν συμπληρωματικά μόρια προς αυτά που αντιστοιχούν σε κάθε κορυφή. Αυτά τα τελευταία παίζουν τον ρόλο συνδέσμου ανάμεσα σε δύο μόρια που αντιστοιχούν σε τόξα που το τέλος του ενός είναι η αρχή του άλλου, έτσι ώστε να διευκολυνθεί η λειτουργία της λιγάσης. Τοποθετώντας μια αρκετή ποσότητα από κάθε μόριο(δηλαδή από μόρια που κωδικοποιούν ακμές, και μόρια συμπληρωματικά ως προς αυτά που κωδικοποιούν κορυφές, εκτός από τις κορυφές 0 και 6) σε ένα δοκιμαστικό σωλήνα, και λόγω της λειτουργίας της συνένωσης με ένζυμο λιγάση, δημιουργούνται μόρια DNA που αντιστοιχούν σε τυχαία μονοπάτια μέσα στο γράφημα. Στο σχήμα 2.3.2 φαίνεται πώς δύο μόρια που αντιστοιχούν σε «γειτονικά» τόξα συνενώνονται και σχηματίζουν ένα μόριο που αντιστοιχεί σε ένα μονοπάτι μήκους 2. Επίσης στο σχήμα 2.3.3 φαίνονται μερικά από τα μόρια που σχηματίστηκαν μετά την αντίδραση με DNA-λιγάση, καθώς και τα αντίστοιχα μονοπάτια που κωδικοποιούν.

Σχήμα 2.3.2

Για την υλοποίηση του βήματος 2, το αποτέλεσμα του βήματος 1 υποβλήθηκε στη λειτουργία της ενίσχυσης μέσω PCR. Έτσι μόνο τα μόρια που κωδικοποιούν μονοπάτια που αρχίζουν με υ_{in} και τελειώνουν σε υ_{end} είναι αυτά που ενισχύονται σε ποσότητα. Ως πρότυπα στη διαδικασία ενίσχυσης με PCR είναι τα μόρια που κωδικοποιούν μονοπάτια που αρχίζουν με υ_{in} και τελειώνουν σε υ_{end} και ως primers χρησιμοποιούνται, το μόριο που αντιστοιχεί στην κορυφή υ_{in} και το μόριο συμπληρωματικό προς αυτό που αντιστοιχεί σε υ_{end}.

Για την υλοποίηση του βήματος 3 χρησιμοποιήθηκε η λειτουργία του διαχωρισμού ,ανάλογα με το μήκος των μορίων, χρησιμοποιώντας την μέθοδο gel electrophoresis.Κρατάμε τα μόρια DNA που έχουν μήκος 140=7x20 νουκλεοτίδια.

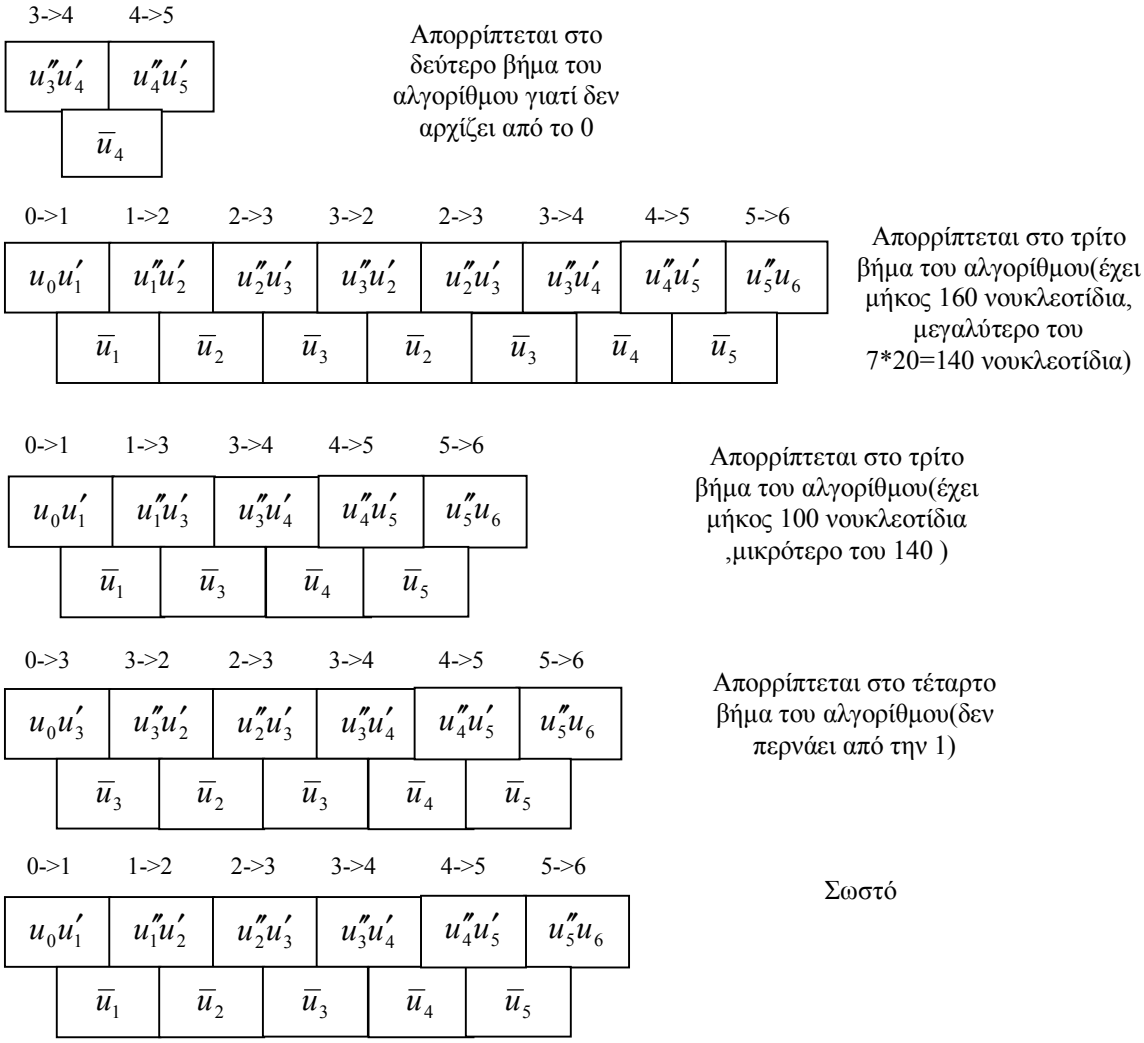
Για την υλοποίηση του βήματος 4 χρησιμοποιήθηκε η λειτουργία της εξαγωγής(extraction by affinity purification). Στην αρχή γίνεται σύνθεση μορίων που είναι συμπληρωματικά ως προς το μόριο που κωδικοποιεί την κορυφή υ_{in}. Αυτά δεσμεύονται πάνω σε μαγνητικά σώματα (magnetic beads). Έτσι τελικά συγκρατούνται τα μόρια που περιέχουν ως υποακολουθία το μόριο που κωδικοποιεί την υ_{in}. Στην συνέχεια τα προϊόντα της προηγούμενης διαδικασίας υποβάλλονται πάλι σε λειτουργία εξαγωγής αυτή τη φορά για την δεύτερη κορυφή. Η διαδικασία συνεχίζεται διαδοχικά για τις υπόλοιπες κορυφές. Δεν έχει σημασία η σειρά με την οποία γίνεται η εξαγωγή ως προς τις κορυφές.

Το βήμα 5 υλοποιήθηκε πρώτα με PCR ενίσχυση του προϊόντος του βήματος 4 και στην συνέχεια με την λειτουργία της *ανίχνευσης* της ακολουθίας DNA που υπάρχει στο τελικό διάλυμα για να εξακριβωθεί αν όντως είναι αυτή που κωδικοποιεί ένα hamilton path.

Όπως φαίνεται από την παρουσίαση του πειράματος του Adleman χρησιμοποιήθηκε τόσο μια κωδικοποίηση πληροφορίας που ταιριάζει στο πρόβλημα, αλλά και οι λειτουργίες προσαρμόστηκαν στα δεδομένα του προβλήματος. Αμέσως δημιουργούνται δύο ερωτήματα:

1) Αν είναι δυνατόν το παραπάνω μοντέλο DNA της 2.2, με τις δυνατότητες για κωδικοποίηση της πληροφορίας που προσδιορίζονται από τη δομή του DNA όσο και τις δυνατότητες για επεξεργασία της πληροφορίας αυτής που προσδιορίζονται από τις διαθέσιμες βιολογικές λειτουργίες, να υλοποιήσει κάθε αλγόριθμο
2) Αν υπάρχει Universal DNA υπολογιστής, δηλαδή υπολογιστής στον οποίο να μπορούν να κωδικοποιηθούν τόσο τα δεδομένα ενός προβλήματος όσο και το πρόγραμμα για την επίλυση του και στην συνέχεια να το εκτελέσει

Αποδεικνύεται ότι η απάντηση στα 2 παραπάνω ερωτήματα είναι θετική και μάλιστα οι λειτουργίες του μοντέλου είναι πολύ περισσότερες από αυτές που χρειαζόμαστε. Αναγκαίες είναι μόνο οι λειτουργίες που προσδίδουν στο μοντέλο DNA τον χαρακτήρα των splicing systems.


3->4
4->5
$u_3''u_4'$
$u_4''u_5'$
$\overline{u}_4$
Απορρίπτεται στο δεύτερο βήμα του αλγορίθμου γιατί δεν αρχίζει από το 0
0->1
1->2
2->3
3->2
2->3
3->4
4->5
5->6
u_0u_1'
$u_1''u_2'$
$u_2''u_3'$
$u_3''u_2'$
$u_2''u_3'$
$u_3''u_4'$
$u_4''u_5'$
$u_5''u_6$
$\overline{u}_1$ $\overline{u}_2$ $\overline{u}_3$ $\overline{u}_2$ $\overline{u}_3$ $\overline{u}_4$ $\overline{u}_5$
Απορρίπτεται στο τρίτο βήμα του αλγορίθμου(έχει μήκος 160 νουκλεοτίδια, μεγαλύτερο του 7*20=140 νουκλεοτίδια)
0->1
1->3
3->4
4->5
5->6
u_0u_1'
$u_1''u_3'$
$u_3''u_4'$
$u_4''u_5'$
$u_5''u_6$
$\overline{u}_1$ $\overline{u}_3$ $\overline{u}_4$ $\overline{u}_5$
Απορρίπτεται στο τρίτο βήμα του αλγορίθμου(έχει μήκος 100 νουκλεοτίδια ,μικρότερο του 140)
0->3
3->2
2->3
3->4
4->5
5->6
u_0u_3'
$u_3''u_2'$
$u_2''u_3'$
$u_3''u_4'$
$u_4''u_5'$
$u_5''u_6$
$\overline{u}_3$ $\overline{u}_2$ $\overline{u}_3$ $\overline{u}_4$ $\overline{u}_5$
Απορρίπτεται στο τέταρτο βήμα του αλγορίθμου(δεν περνάει από την 1)
0->1
1->2
2->3
3->4
4->5
5->6
u_0u_1'
$u_1''u_2'$
$u_2''u_3'$
$u_3''u_4'$
$u_4''u_5'$
$u_5''u_6$
$\overline{u}_1$ $\overline{u}_2$ $\overline{u}_3$ $\overline{u}_4$ $\overline{u}_5$
Σωστό
$\overline{u}_i$
=20 νουκλεοτίδια για το συμπλήρωμα της πόλης i
$u_i''u_j'$
=20 νουκλεοτίδια για το δεσμό i->j

Σχήμα 2.3.3

2.4 DNA υπολογιστές και DNA προγράμματα.

Ένας *DNA υπολογιστής*, μπορεί να οριστεί πιο εύκολα από το τι κάνει, παρά από το τι είναι. Αυτό που θα γίνεται είναι να έχουμε ως *είσοδο* του DNA υπολογιστή, ένα σύνολο από δοκιμαστικούς σωλήνες(tubes) που περιέχουν μόρια DNA που κωδικοποιούν το στιγμιότυπο του εκάστοτε προβλήματος. Ο «υπολογισμός» του DNA-υπολογιστή συνίσταται σε μια ακολουθία από λειτουργίες(από αυτές που περιγράφονται στην 2.2), που μπορούν να γίνουν(είτε από ανθρώπους, είτε από ειδικά robot) χρησιμοποιώντας αυτούς τους σωλήνες. Κάθε λειτουργία αυτής της ακολουθίας, ενδεχομένως να παράγει ένα ή περισσότερους νέους σωλήνες, οι οποίοι μπορούν να χρησιμοποιηθούν μαζί με τους προηγούμενους, για την εκτέλεση των επόμενων λειτουργιών. Η *έξοδος* του DNA υπολογιστή θα συνίσταται πάλι από ένα σύνολο από δοκιμαστικούς σωλήνες, οι οποίοι, αυτή τη φορά, θα περιέχουν μόρια DNA που κωδικοποιούν λύσεις, στο πρόβλημα του οποίου το στιγμιότυπο, κωδικοποιούσαν τα μόρια DNA των δοκιμαστικών σωλήνων που συνιστούσαν την είσοδο.

Από τα παραπάνω προκύπτει ότι, για την ώρα, η έννοια του DNA υπολογιστή είναι συνυφασμένη με την έννοια του προβλήματος, και του *DNA-προγράμματος* που το επιλύει(η ακολουθία των λειτουργιών). Αν και η ύπαρξη Universal DNA υπολογιστών, (δηλαδή υπολογιστών που θα δέχονται ως είσοδο, εκτός των άλλων, και ένα σύνολο δοκιμαστικών σωλήνων που να κωδικοποιούν το πρόγραμμα) αποδεικνύεται θεωρητικά στο δεύτερο μέρος της παρούσας, εντούτοις περιορισμοί της βιοτεχνολογίας αποτρέπουν την πρακτική υλοποίηση των Universal DNA υπολογιστών.

Εφόσον μιλάμε για DNA προγράμματα, θα είναι χρήσιμο αυτά τα προγράμματα να προέρχονται από κάποια *γλώσσα προγραμματισμού.* Αυτή θα προκύψει από μια τυποποίηση των λειτουργιών της 2.2. Συγκεκριμένα θα τυποποιηθούν μόνο οι λειτουργίες, που θα χρησιμοποιηθούν για την επίλυση του SAT και του συστήματος κρυπτογράφησης DES.

Καταρχήν, ένας (δοκιμαστικός) *σωλήνας* θα είναι ένα *πολυσύνολο*(multiset) από *λέξεις*(πεπερασμένα το μήκος strings) του αλφαβήτου $\{A,C,G,T\}$. Για το τι είναι πολυσύνολο βλέπε δεύτερο μέρος, παράγραφο 3.3. Διαισθητικά, ένας σωλήνας είναι μια συλλογή από απλά μόρια DNA. Αυτά τα μόρια υπάρχουν στο σωλήνα με μια *πολλαπλότητα*, δηλαδή πολλά αντίγραφα του ίδιου μορίου, μπορεί να περιέχονται μέσα στο σωλήνα. Οι ακόλουθες απλές λειτουργίες, που συνιστούν την γλώσσα προγραμματισμού για τα DNA προγράμματα, αρχικά ορίζονται για σωλήνες(με τον ορισμό που δόθηκε, πιο πάνω). Εντούτοις, κατάλληλη τροποποίηση τους θα γίνει για την περίπτωση σωλήνων με μόρια DNA με διπλή αλυσίδα.

-*Merge:* Δοθέντος των σωλήνων N_1 και N_2, σχημάτισε την ένωση $N_1 \cup N_2$(ως πολυσύνολο). Αυτή η λειτουργία, αντιστοιχεί στη λειτουργία mixing της 2.2

-*Amplify*. Δοθέντος ενός σωλήνα N, κάνε δύο αντίγραφα αυτού.(PCR amplification στη 2.2)

-*Detect*. Δοθέντος του σωλήνα N, επέστρεψε *true* αν περιέχει τουλάχιστον ένα μόριο DNA, διαφορετικά επέστρεψε *false(*υλοποιείται με την βοήθεια της μεθόδου sequencing).

-*Separate(ή Extract)*. Δοθέντος του σωλήνα N και μια λέξης w του αλφαβήτου $\{A,C,G,T\}$, παράγει δύο σωλήνες $+(N,w)$ και $-(N,w)$, όπου $+(N,w)$ περιλαμβάνει όλα τα μόρια του N που περιέχουν το w ως substring, και παρόμοια, $-(N,w)$ συνίσταται από εκείνα τα μόρια του N, που δεν έχουν το w ως substring. Αυτή η λειτουργία μπορεί να υλοποιηθεί με *affinity purification*.

Αυτές οι τέσσερις λειτουργίες, ενδεχομένως μαζί με κάποιες τυπικές εντολές άλλων γλωσσών προγραμματισμού (For, do begin), μας επιτρέπουν να προγραμματίσουμε απλές ερωτήσεις που αφορούν την εμφάνιση ή μη κάποιων υπολέξεων. Για παράδειγμα το ακόλουθο πρόγραμμα:

(1) $input(N)$

(2) $N \leftarrow +(N, A)$

(3) $N \leftarrow +(N, G)$

(4) $\det ect(N)$

μας επιτρέπει να βρούμε το αν ένας σωλήνας έχει μόρια DNA που περιέχουν και τις δύο πουρίνες A και G. Το ακόλουθο πρόγραμμα εξάγει από ένα σωλήνα όλα τα μόρια που περιέχουν τουλάχιστον μία από τις πουρίνες A ή G, διατηρώντας την πολλαπλότητα αυτών των μορίων.

(1) $input(N)$

(2) $amplify(N)$ για να παράγεις N_1 και N_2

(3) $N_A \leftarrow +(N_1, A)$

(4) $N_G \leftarrow +(N_2, G)$

(5) $N'_G \leftarrow -(N_G, A)$

(6) $merge(N_A, N'_G)$

Οι επαναλήψεις της λειτουργίας amplify, παράγουν μια εκθετική (ως προς των αριθμό των επαναλήψεων) αντιγραφή του αριθμού των μορίων στο δοσμένο σωλήνα.

Εκτός από τις τέσσερις λειτουργίες, που περιγράφονται πιο πάνω, στο πείραμα του Adleman,χρησιμοποιούνται ειδικές περιπτώσεις της εντολής separate:

Length-separate. Δοθέντος του σωλήνα N και ενός θετικού ακέραιου n, παρήγαγε το σωλήνα $(N, \leq n)$ που περιέχει όλα τα μόρια του N, με μήκος μικρότερο η ίσο του n. Αυτό μπορεί να υλοποιηθεί με gel electrophoresis.

Position-separate. Δοθέντος του N και μιας λέξης w, παρήγαγε το σωλήνα $B(N, w)$ (αντίστοιχα $E(N, w)$) που περιέχει όλα τα μόρια του N που αρχίζουν(τελειώνουν) με τη λέξη w.

Η αντιστοιχία ανάμεσα στις παραπάνω έξι λειτουργίες, και τις λειτουργίες της 2.2 δεν είναι απαραίτηττα 1-1. Μπορεί κάποια από τις παραπάνω εντολές να υλοποιείται με περισσότερες της μιας λειτουργίας από αυτές της 2.2. Μπορεί επίσης δύο εντολές από τις παραπάνω, να υλοποιούνται με μια της 2.2, για παράδειγμα οι εντολές $B(N, w)$ και $E(N, w)$ έγιναν στο πείραμα του Adleman, με *ενίσχυση μέσω PCR*, μια λειτουργία που περιγράφεται στη 2.2.

Με τις παραπάνω έξι λειτουργίες, μπορούμε να γράψουμε το πρόγραμμα που αντιστοιχεί στο πείραμα του Adleman.

(1) $input(N)$

(2) $N \leftarrow B(N, \upsilon_{in})$

(3) $N \leftarrow E(N, \upsilon_{out})$

(4) $N \leftarrow (N, \leq 140)$

(5) $for \quad i = 1 \quad to \quad 5 \quad do \quad begin \quad N \leftarrow +(N, \upsilon_i) \quad end$

(6) $\det ect(N)$

Ο «βασικός υπολογισμός» γίνεται, όχι στις λειτουργίες 2-6, αλλά στην 1, όπου ξεκινάμε με τον βασικό σωλήνα που περιέχει όλα τα δυνατά μονοπάτια του γραφήματος. Αυτός προέκυψε από την αντίδραση συνένωσης με DNA-λιγάση. Μπορούμε να θεωρούμε ότι έχει απλά μόρια DNA και όχι διπλά, μια και τα τελευταία

μπορούν να γίνουν απλά με απλή θέρμανση. Από εκεί και πέρα, οι 2-6 μπορούν να υλοποιηθούν στο εργαστήριο με μια σειρά από λειτουργίες σαν αυτές της 2.2. Στον Adleman, πήρε περίπου 10 μέρες για την υλοποίηση τους στο εργαστήριο.

2.5 Επίλυση του SAT με DNA-πρόγραμμα

Εδώ θα παρουσιαστεί η επίλυση του προβλήματος της *ικανοποιησιμότητας για προτασιακούς τύπους*. Αυτό έχει επιτευχθεί, όχι από τον Adleman[1], αλλά από τον Lipton[2]. Το πρόβλημα της ικανοποιησιμότητας(SATisfiability problem) είναι NP-πλήρες πρόβλημα. Σε αυτό δίνεται ένας προτασιακός τύπος a, που δομείται από *προτασιακές μεταβλητές* $x_1,...,x_n$ και *λογικούς συνδέσμους(άρνηση(NOT)=$\neg$, διάζευξη(OR=$\vee$), σύζευξη(AND)=$\wedge$)* που μπορεί π.χ να έχει την μορφή :

$a = (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee x_3) \wedge \neg x_3$

Μια *ανάθεση τιμών αληθείας* είναι μια αντιστοιχία f,του συνόλου των μεταβλητών που εμφανίζονται στον a, μέσα στο σύνολο $\{0,1\}$. Για κάθε ανάθεση τιμών αληθείας στις μεταβλητές, η τιμή αληθείας για τον τύπο a, $f(a)$, μπορεί να υπολογιστεί χρησιμοποιώντας τους παρακάτω πίνακες αληθείας(σχήμα 2.5.1).

OR	0	1
0	0	1
1	1	0

AND	0	1
0	0	0
1	0	1

x	0	1
NOT(x)	1	0

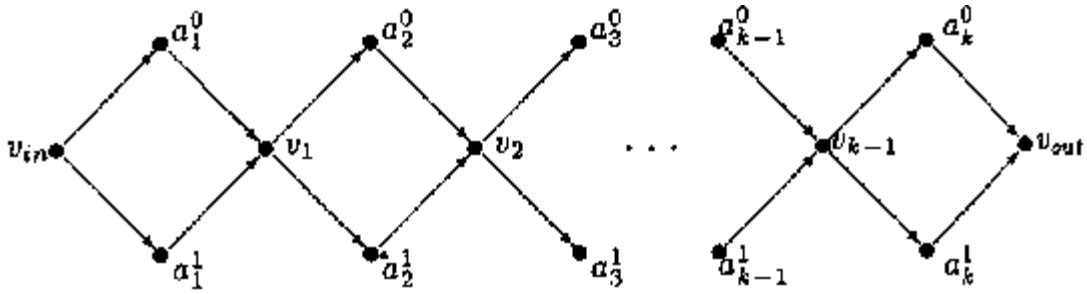
Σχήμα 2.5.1

Ο τύπος a είναι *ικανοποιήσιμος* αν υπάρχει τουλάχιστον μια ανάθεση τιμών αληθείας στις μεταβλητές, για την οποία ο a παίρνει τιμή αληθείας 1(true). Για τον συγκεκριμένο τύπο a, θα δειχθεί ότι δεν είναι ικανοποιήσιμος. Προς άτοπο, έστω ότι υπάρχει ανάθεση τιμών αληθείας f, για την οποία $f(a)=1$. Από τον πίνακα αληθείας του AND προκύπτει ότι $f(\neg x_3)=1$(γενικά αν $f(a)=1$, τότε όλα τα clauses(οι υπότυποι μέσα στις παρενθέσεις) γ, έχουν $f(\gamma)=1$). Άρα $f(x_3)=0$. Επειδή $f(\neg x_1 \vee x_3)=1$ και από τον πίνακα αληθείας του OR, προκύπτει ότι $f(\neg x_1)=1$, συνεπώς $f(x_1)=0$. Παρόμοια από το δεύτερο clause θα προκύψει ότι $f(x_2)=1$. Αλλά για αυτές τις αναθέσεις των x_1,x_2,x_3, δεν ικανοποιείται το πρώτο clause $x_1 \vee \neg x_2 \vee x_3$.

Φαίνεται λοιπόν, ότι σε ειδικές περιπτώσεις του SAT, μπορούν να χρησιμοποιηθούν adhoc μέθοδοι, όπως ο παραπάνω συλλογισμός,για να δοθεί απάντηση στο πρόβλημα. Εντούτοις, στη γενική περίπτωση, δεν υπάρχει μέθοδος καλύτερη από αυτή της *εξαντλητικής αναζήτησης*(exhaustive search, ή brute force μέθοδος). Σύμφωνα με αυτή, ψάχνουμε όλες τις δυνατές αναθέσεις τιμών αληθείας που είναι 2^n το πλήθος διαφορετικές, όπου n ο αριθμός των μεταβλητών που εμφανίζονται στον τύπο(για το παράδειγμα πιο πάνω είναι $n=3$). Αυτή η μέθοδος είναι υπολογιστικά μη ελκυστική. Για $n=200$ είναι υπολογιστικά ανέφικτη, γιατί το πλήθος των διαφορετικών αναθέσεων είναι 2^{200}. Φαίνεται λοιπόν, ότι το SAT είναι

πολύ δύσκολο πρόβλημα, και όπως ειπώθηκε στην αρχή είναι NP-πλήρες πρόβλημα, με την έννοια ότι ένα οποιοδήποτε άλλο πρόβλημα που είναι NP, ανάγεται σε αυτό. Για περισσότερες λεπτομέρειες, βλέπε Μέρος ΙΙ: Κεφάλαιο 1.

Η λύση του Lipton[2] για το SAT χρησιμοποιεί μερικές από τις βασικές εντολές που περιγράφηκαν στη παράγραφο 2.4. Συνίσταται στην εξαντλητική αναζήτηση, που όμως γίνεται υπολογιστικά εφικτή, λόγω του μαζικού παραλληλισμού των αντιδράσεων ανάμεσα στα μόρια DNA(όπως στο βήμα 1 του πειράματος Adleman, αντίδραση λιγάσης). Αρχίζουμε με μια γραφοθεωρητική περιγραφή των αναθέσεων τιμών αληθείας. Έστω ότι έχουμε ένα προτασιακό τύπο με n προτασιακές μεταβλητές. Έστω το γράφημα στο σχήμα 2.5.2.



Σχήμα 2.5.2 Το γράφημα του οποίου τα μονοπάτια αντιστοιχούν σε αναθέσεις

Υπάρχουν 2^n μονοπάτια από την u_{in} στη u_{out}, σε αυτό το γράφημα(κανένα από αυτά δεν είναι Hamiltonian). Πράγματι, υπάρχουν δύο επιλογές για κάθε κορυφή από τις $u_{in}, u_1, ..., u_{n-1}$, η κάθε επιλογή ανεξάρτητη από τις άλλες. Επιπλέον, τα μονοπάτια και οι αναθέσεις τιμών αληθείας για τις μεταβλητές $x_1, ..., x_n$, έχουν μια λογική 1-1 αντιστοιχία. Το μονοπάτι $u_{in} a_1^{i_1} u_1 a_2^{i_2} u_2 .. u_{n-1} a_n^{i_n} u_{out}$ αντιστοιχεί στην ανάθεση όπου η μεταβλητή x_j παίρνει την τιμή i_j για $j = 1,2,...,n$.

Στη συνέχεια, προχωρούμε ακριβώς όπως και στο πείραμα του Adleman. Κάθε κορυφή κωδικοποιείται από ένα τυχαίο ολιγονουκλεοτίδιο, έστω μήκους 20. Ανάλογα κωδικοποιούνται και τα τόξα ανάμεσα στις κορυφές. Πάλι στο πρώτο βήμα τοποθετούνται οι αναγκαίες ποσότητες από κάθε μόριο σε ένα σωλήνα, και γίνεται αντίδραση DNA-λιγάσης. Πάλι σχηματίζονται μόρια DNA που αντιστοιχούν σε μονοπάτια του γραφήματος 2.5.2, άρα και σε αναθέσεις τιμών αληθείας, για τα μόρια που κωδικοποιούν μονοπάτια που αρχίζουν από u_{in} και τελειώνουν σε u_{out}. Εφόσον τα ολιγονουκλεοτίδια που κωδικοποιούν τα στοιχεία του γραφήματος, επιλέγονται τυχαία, και έχουν αρκετό μήκος(για μεγάλο αριθμό μεταβλητών, το μήκος 20 μπορεί να μη είναι αρκετό), είναι μικρή η πιθανότητα να σχηματιστούν άσχετα μονοπάτια. Αυτό σημαίνει ότι μετά την αντίδραση λιγάσης, η «σούπα» δεν θα περιέχει ένα διπλό μόριο DNA που να κωδικοποιεί ένα αυθαίρετο μονοπάτι μέσα στο γράφημα. Επειδή το γράφημα είναι συμμετρικό, δεν υπάρχει λόγος κάποιο μονοπάτι να εμφανίζεται πιο συχνά από κάποιο άλλο, ως μόριο DNA μέσα στη σούπα.

Μια σημαντική διαφορά από αυτή στο πείραμα του Adleman, είναι ότι σε αυτή τη περίπτωση, το γράφημα είναι πάντα ίδιο και εξαρτάται μόνο από τον αριθμό των μεταβλητών. Έτσι μπορούμε να αρχίζουμε πάντα με τον ίδιο δοκιμαστικό σωλήνα, που περιέχει όλες τις δυνατές αναθέσεις για τις n μεταβλητές. Δημιουργώντας πολλά αντίγραφα αυτού, είναι δυνατό να χειριστούμε πολλούς διαφορετικούς προτασιακούς τύπους, ταυτόχρονα. Στο πείραμα του Adleman, το γράφημα είναι η είσοδος του

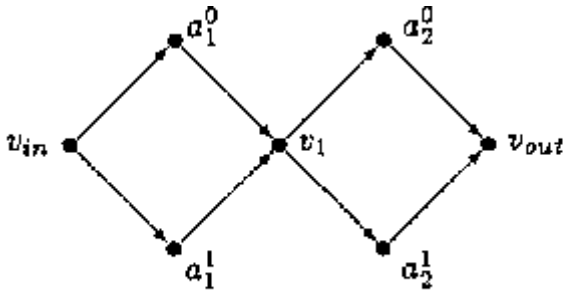
προβλήματος, και διαφέρει ανάμεσα στα διαφορετικά στιγμιότυπα του DHP. Άρα ο αρχικός δοκιμαστικός σωλήνας διαφέρει. Στο SAT η είσοδος είναι ο προτασιακός τύπος.

Στη συνέχεια θα μιλάμε για τον *αρχικό δοκιμαστικό σωλήνα*. Αυτός κατασκευάζεται όπως παραπάνω και περιέχει κωδικοποιήσεις για όλες τις δυνατές αναθέσεις τιμών αληθείας στις μεταβλητές. Εντολές όπως αυτές της 2.4 θα πραγματοποιηθούν. Μπορούμε να υποθέσουμε ότι τα μόρια DNA είναι πάντα απλά μόρια(μονής αλυσίδας). Είναι θέμα μοριακής βιολογίας το αν είναι καλύτερα να διαχωρίσουμε τα διπλά μόρια που προκύπτουν από το βασικό βήμα της αντίδρασης με λιγάση, ή αν να θεωρούμε ότι οι λειτουργίες επιτελούνται επί της μιας από τις δύο αλυσίδες των διπλών μορίων DNA.

Οι λειτουργίες separate, merge, detect θα χρησιμοποιηθούν. Το παράδειγμα που αναφέρεται στο [2] είναι για τον προτασιακό τύπο

$\beta = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$

Συνεπώς, έχουμε δύο μεταβλητές $(n = 2)$, και έτσι το αντίστοιχο γράφημα είναι αυτό στο σχήμα 2.5.3



Σχήμα 2.5.3 Το γράφημα για τον τύπο β

Κάθε ένα από τα 4 μονοπάτια που διασχίζει το γράφημα(δηλαδή που αρχίζει στην u_{in} και τερματίζει στην u_{out}) αντιστοιχεί σε μια από τις 4 δυνατές αναθέσεις στις μεταβλητές x_1 και x_2. Ο αρχικός δοκιμαστικός σωλήνας, έστω N_0, που κατασκευάστηκε όπως παραπάνω, περιέχει μόρια για κάθε ένα από τα μονοπάτια, και κατά συνέπεια, για κάθε μια από τις δυνατές αναθέσεις. Δοθέντος του μήκους των ολιγονουκλεοτιδίων που κωδικοποιούν κορυφές a_j^i, αυτά τα ολιγονουκλεοτίδια θα ξεχωρίζουν το ένα από το άλλο, ακόμα και στη περίπτωση μεγάλου αριθμού μεταβλητών(μεγάλο n). Αυτό σημαίνει ότι, για παράδειγμα, το ολιγονουκλεοτίδιο που κωδικοποιεί το a_1^1 δεν εμφανίζεται στα μονοπάτια πουθενά αλλού, εκτός από την προβλεπόμενη θέση. Συνεπώς αν εφαρμόσουμε την εντολή separate και πάρουμε τον σωλήνα $+(N_0, a_1^1)$, θα πάρουμε τα μόρια που κωδικοποιούν αναθέσεις στις οποίες $x_1 = 1$. (Ο $+(N_0, a_1^1)$ συνίσταται από εκείνα τα μόρια στον N_0, όπου το ολιγονουκλεοτίδιο a_1^1, εμφανίζεται ως συνεχόμενο substring). Αυτή η απλή παρατήρηση είναι η βάση για την όλη διαδικασία επίλυσης του προβλήματος.

Συμβολίζουμε τις αναθέσεις τιμών αληθείας ως πεπερασμένες ακολουθίες από 0 και 1, με μήκος n, όπου n ο αριθμός των μεταβλητών. Έτσι η πεπερασμένη ακολουθία 01 θα αναπαριστά την ανάθεση $x_1 = 0$ και $x_2 = 1 (n = 2)$. Αυτός ο συμβολισμός θα χρησιμοποιείται και για τα αντίστοιχα μόρια DNA. Έτσι το μόριο που κωδικοποιεί το μονοπάτι $u_{in} a_1^0 u_1 a_2^1 u_{out}$ θα συμβολίζεται με 01.Τέλος, δοθέντος

ενός σωλήνα N, που περιέχει μόρια DNA που συμβολίζονται όπως προηγουμένως, θα συμβολίζουμε με $S(N,i,j)$ τον σωλήνα που περιέχει εκείνα τα μόρια του N που συμβολίζονται με ακολουθίες στις οποίες το $i-bit$ (ι-οστό δυαδικό ψηφίο) ισούται με j. Σύμφωνα με την απλή παρατήρηση που αναφέρθηκε στη προηγούμενη παράγραφο, ο $S(N,i,j)$ προκύπτει από το N με τη λειτουργία separate:

$S(N,i,j) = +(N,a_i^j)$

Επίσης θεωρούμε τον σωλήνα που περιέχει τα μόρια του N όπου το $i-bit$ ισούται με το συμπλήρωμα του j:

$S^-(N,i,j) = -(N,a_i^j)$.

Το ακόλουθο πρόγραμμα είναι προσαρμοσμένο, όχι μόνο στο πρόβλημα, αλλά και στη συγκεκριμένη είσοδο(ο συγκεκριμένος προτασιακός τύπος β) :

(1) $input(N_0)$

(2) $N_1 = S(N_0,1,1)$

(3) $N_1' = S^-(N_0,1,1)$

(4) $N_2 = S(N_1',2,1)$

(5) $merge(N_1,N_2) = N_3$

(6) $N_4 = S(N_3,1,0)$

(7) $N_4' = S^-(N_3,1,0)$

(8) $N_5 = S(N_4',2,0)$

(9) $merge(N_4,N_5) = N_6$

(10) $\det ect(N_6)$

Ο ακόλουθος πίνακας δίνει τα περιεχόμενα των σωλήνων στα αντίστοιχα βήματα.

Βήμα	1	2	3	4	5	6	7	8	9
Σωλ.	00,01, 10,11	10,11	00,01	01	10,11, 01	01	10,11	10	01,10

Έτσι το αποτέλεσμα είναι true, από την εντολή στο βήμα 10.

Το πρόγραμμα βασίζεται σε εξαντλητική αναζήτηση. Ο αρχικός σωλήνας στο βήμα 1 περιέχει και τις 4 δυνατές αναθέσεις. Ο σωλήνας στο βήμα 5, περιέχει τις αναθέσεις που ικανοποιούν το πρώτο clause του τύπου. Αυτό γιατί για να συμβεί το τελευταίο, θα πρέπει μια εκ των x_1, x_2 να πάρει την τιμή 1. Στο βήμα (2) έχουμε τις αναθέσεις για τις οποίες $x_1 = 1$. Από τις υπόλοιπες(βήμα (3)) παίρνουμε αυτές που έχουν $x_2 = 1$ στο βήμα 4. Οι αναθέσεις που υπάρχουν στο σωλήνα N_3 του βήματος (5), φιλτράρονται επιπλέον για να δώσουν αναθέσεις που ικανοποιούν και το δεύτερο clause.

Η μέθοδος αυτού του απλού παραδείγματος μπορεί να εφαρμοσθεί απευθείας στη γενική περίπτωση. Σε αυτή, θεωρούμε προτασιακούς τύπους που βρίσκονται σε *κανονική συζευκτική μορφή(CNF, conjuctive normal form).* Αυτό σημαίνει ότι οι τύποι μοιάζουν ως εξής:

$(...) \wedge (...) \wedge ... \wedge (...)$

όπου κάθε ένα από τα clauses, είναι μια διάζευξη από *όρους*, κάθε όρος είναι μια μεταβλητή, ή άρνηση μεταβλητής. Γρήγοροι αλγόριθμοι είναι διαθέσιμοι για να μετασχηματίσουν ένα αυθαίρετο προτασιακό τύπο, σε τύπο CNF.

Έτσι θεωρούμε ένα προτασιακό τύπο $\gamma = C_1 \wedge C_2 \wedge ... \wedge C_m$ όπου C_i clause, δηλαδή διάζευξη όρων, και m ο αριθμός των clauses. Υποθέτουμε ότι συνολικά n μεταβλητές εμφανίζονται στον τύπο γ. Αυτές διαμοιράζονται μέσα στα clauses, αλλά είναι δυνατόν σε κάποια clauses να εμφανίζονται κάποιες μεταβλητές που είναι ίδιες. Εφόσον έχουμε n μεταβλητές, οδηγούμαστε στο γράφημα στο σχήμα 2.5.2 και στον αρχικό δοκιμαστικό σωλήνα N_0, που περιέχει όλες τις ακολουθίες μήκους n, θεωρώντας ότι τα μόρια στο N_0 κωδικοποιούν με τον προαναφερόμενο τρόπο το γράφημα στο σχήμα 2.5.2. Η διαδικασία γενικά, περιγράφεται ως εξής: αρχίζοντας με τον N_0, επεξεργαζόμαστε τα clauses του γ, εξάγοντας συνέχεια μόρια από τον N_0, μέχρι που όταν φτάσουμε στην C_m, μένουν μόνο τα μόρια που κωδικοποιούν αναθέσεις που ικανοποιούν κάθε $C_i, 1 \le i \le m$, συνεπώς και τον τύπο γ.

Πιο αναλυτικά, δείχνεται επαγωγικά πως προχωράει η διαδικασία. Υποθέτουμε ότι κάθε ανάθεση που κωδικοποιείται από μόρια στο N_i, $0 \le i \le m$, ικανοποιεί τον υποτύπο $\gamma_i = C_1 \wedge C_2 \wedge .... \wedge C_i$, και έστω $C_{i+1} = y_1 \vee y_2 \vee ... \vee y_l$, όπου κάθε y_j είναι όρος πάνω στις μεταβλητές $x_1,...,x_n$. Αρχικά έχουμε τον σωλήνα N_0, και τον κενό υποτύπο γ_0.

Χρησιμοποιώντας τις λειτουργίες separate και merge, μετασχηματίζουμε τον N_i στον N_{i+1}, με την ίδια διαδικασία όπως στο παράδειγμα. Θεωρούμε το y_1. Σχηματίζουμε τον (υπο)-σωλήνα $S(N_i, k, 1)$ αν $y_1 = x_k$, ή τον σωλήνα $S(N_i, k, 0)$ αν $y_1 = \neg x_k$. Έτσι εξάγουμε από τον N_i τον υποσωλήνα $T_1 = S(N_i, k, j)$ που ικανοποιεί και το y_1. Το υπόλοιπο του N_i, δηλαδή το $S^-(N_i, k, j)$, ερευνείται ως προς την ικανοποίηση του y_2, και παρόμοια θα πάρουμε έναν T_2. Το υπόλοιπο θα διερευνηθεί για την ικανοποίηση του y_3, από όπου θα πάρουμε ένα T_3 και ένα νέο υπόλοιπο, κ.ο.κ. Τελικά $N_{i+1} = merge(T_1, T_2, ..., T_l)$.

Προχωρώντας σύμφωνα με την παραπάνω διαδικασία και ξεκινώντας από N_0, κατασκευάζουμε τους $N_1, N_2, ..., N_m$. Οι αναθέσεις στο N_m, είναι αυτές που ικανοποιούν τον γ.Με μια απλή λειτουργία detect μπορούμε να απαντήσουμε αν ο τύπος είναι ικανοποιήσιμος ή όχι(βλέποντας το SAT σαν πρόβλημα απόφασης). Μπορεί να χρησιμοποιηθεί sequencing, για να διαβασθεί το διάλυμα του N_m και να βρούμε τις αναθέσεις που ικανοποιούν τον τύπο (το SAT ως πρόβλημα βελτιστοποίησης).

Η πολυπλοκότητα της παραπάνω διαδικασίας είναι εφικτή: απαιτούνται m βήματα , και στο κάθε βήμα έχουμε έναν αριθμό από λειτουργίες separate και merge. Αυτός ο αριθμός είναι ο αριθμός των μεταβλητών στο clause που εξετάζεται στο βήμα. Η πολυπλοκότητα φαίνεται καλύτερα αν θεωρηθεί το DNA-πρόγραμμα που λύνει το SAT:

(1) $input(N_0)$
(2) $for \quad i = 0 \quad to \quad m-1 \quad do \quad begin$
  (3) $M \leftarrow N_i$
  (4) $for \quad j = 1 \quad to \quad l[i] \quad do \quad begin$
    (5) $if \quad y_{ji+1} = x_k \quad then \quad T_j \leftarrow S(M, k, 1) \quad else \quad T_j \leftarrow S(M, k, 0)$

(6) $N_{i+1} \leftarrow merge(N_{i+1}, T_j)$

(7) $M \leftarrow T_j^-$ *end*

(8) *end*

Όπως φαίνεται από το πρόγραμμα, η χρονική πολυπλοκότητα είναι $\sum_{i=1}^{m} l[i]$ λειτουργίες separate, και άλλες τόσες λειτουργίες merge. Το $\sum_{i=1}^{m} l[i]$ είναι ο αριθμός των μεταβλητών που εμφανίζονται στα clauses, μετρώντας τις πολλαπλότητες(δηλαδή αν μια μεταβλητή εμφανίζεται σε πολλά clauses, τότε προσδίδει στο παραπάνω άθροισμα τον αριθμό των clauses στα οποία εμφανίζεται). Ένας άλλος τρόπος για να δούμε το $\sum_{i=1}^{m} l[i]$ είναι ως ο αριθμός των συνδέσμων που εμφανίζονται στον προτασιακό τύπο.

Με επαγωγή στο i, μπορεί να δειχθεί ότι η παραπάνω διαδικασία είναι σωστή. Πράγματι για $i = 0$ θα έχουμε N_0 το σύνολο των αναθέσεων που ικανοποιεί τον γ_0(τον κενό υπότυπο) κατά κενή έννοια. Αν υποθέσουμε ότι ο N_i περιέχει τις αναθέσεις που ικανοποιούν τον γ_i, τότε ο N_{i+1} είναι εκείνες οι αναθέσεις του N_i που ικανοποιούν το clause C_{i+1}. Όμως ο N_{i+1} προήλθε από τον N_i και άρα ικανοποιεί και τον γ_i. Επειδή $\gamma_{i+1} = \gamma_i \wedge C_{i+1}$ και από τον πίνακα αληθείας του AND, προκύπτει τελικά ότι ο N_{i+1} θα έχει αναθέσεις που ικανοποιούν τον γ_{i+1}.

Τέλος, να πούμε ότι η παραπάνω μέθοδος για την επίλυση του SAT, μπορεί σχετικά εύκολα να γενικευθεί για την περίπτωση προτασιακών τύπων σε οποιαδήποτε μορφή(και όχι αναγκαία σε κανονική συζευκτική μορφή). Περισσότερες λεπτομέρειες αναφέρονται στο Boneh,Dunworth,Lipton[3].

2.6 Επίθεση στο DES με DNA-πρόγραμμα

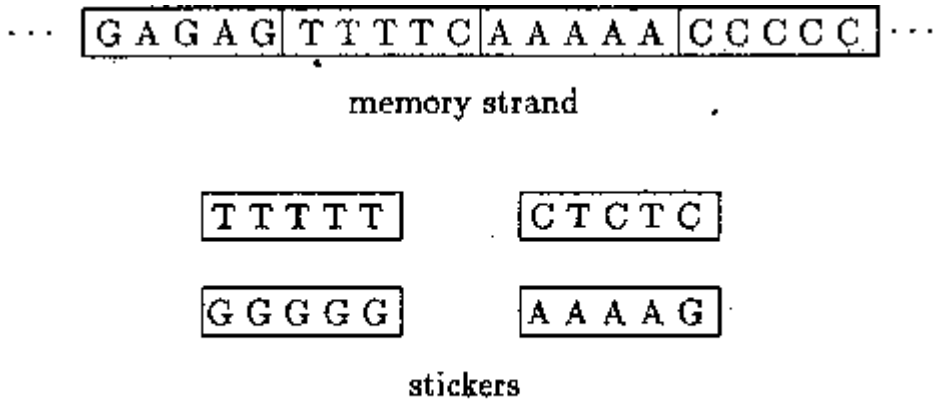
Αρχικά θα παρουσιαστεί ένα μοντέλο υπολογισμού, γνωστό ως sticker model, που παρουσιάστηκε για πρώτη φορά στο [8]. Αυτό το μοντέλο, αν και απέχει αρκετά από το να χαρακτηριστεί αυστηρό θεωρητικό μοντέλο, όπως εκείνο των συστημάτων splicing στο Μέρος ΙΙ: Κεφάλαιο 3, εντούτοις έχει χρησιμοποιηθεί για την επίλυση πολλών προβλημάτων. Το sticker μοντέλο βασίζεται στην Watson-Crick συμπληρωματικότητα των μορίων DNA. Χρησιμοποιεί μόρια DNA σαν το φυσικό μέσο αποθήκευσης της πληροφορίας. Βασικά, το sticker μοντέλο έχει μνήμη τυχαίας προσπέλασης(όχι βέβαια με την έννοια των ψηφιακών ηλεκτρονικών υπολογιστών), και δεν χρειάζεται να επεκτείνουμε τα μόρια DNA. Το κάθε μόριο DNA που αναπαριστά μνήμη, μπορεί να ξαναχρησιμοποιηθεί άπειρες φορές, τουλάχιστον θεωρητικά.

Πιο αναλυτικά, όπως κάθε μοντέλο, το sticker μοντέλο έχει μια δομή για την αναπαράσταση της πληροφορίας, και μια λειτουργικότητα που οφείλεται στις λειτουργίες που μπορούν να επιτελεστούν επί των δομών αναπαράστασης της πληροφορίας.

Η δομή για την αναπαράσταση της πληροφορίας, αποτελείται από *memory strands(μόρια μνήμης)* και *sticker strands* ή πιο απλά *stickers* . Ένα memory strand έχει μήκος n νουκλεοτίδια, και περιέχει k μη επικαλυπτόμενα(non overlapping) υπό-strand(substrand), με κάθε substrand να έχει μήκος m νουκλεοτίδια. Συνεπώς θα πρέπει $n \geq m \cdot k$. Υποθέτουμε ότι το κάθε substrand ακολουθεί το προηγούμενο του,

χωρίς να υπάρχουν νουκλεοτίδια ανάμεσα τους. Κατά την πορεία ενός υπολογισμού, κάθε substrand, αντιστοιχεί σε μία ακριβώς boolean μεταβλητή ή ισοδύναμα σε ένα ακριβώς bit. Τα substrand πρέπει να διαφέρουν το ένα από το άλλο: οποιαδήποτε δύο από αυτά, θα πρέπει να διαφέρουν σε αρκετά νουκλεοτίδια. Αυτό θέλουμε να ισχύει, έτσι ώστε η θέση κάθε bit να προσδιορίζεται μοναδικά. Ο κάθε sticker έχει μήκος m ίσο με αυτό του κάθε substrand, και είναι συμπληρωματικός με ένα ακριβώς substrand στο memory strand. Η αντιστοιχία substrand-sticker είναι 1-1 δηλαδή.

Ένα substrand ενός memory strand είναι σε θέση on είτε σε θέση off. Αν ένας sticker είναι κολλημένος(ανοπτημένος) στο μοναδικό συμπληρωματικό του substrand, τότε αυτό το substrand είναι σε θέση on. Αλλιώς, αν δεν υπάρχει sticker προσκολλημένος σε ένα substrand, τότε αυτό το substrand είναι σε θέση off. Ένα *σύμπλοκο μνήμης(memory complex)* είναι μια γενίκευση του memory strand, όπου τα substrand μπορεί να είναι on ή off. Τα memory complex είναι μόρια DNA που είναι μερικώς διπλά, και κωδικοποιούν δυαδικά νούμερα.



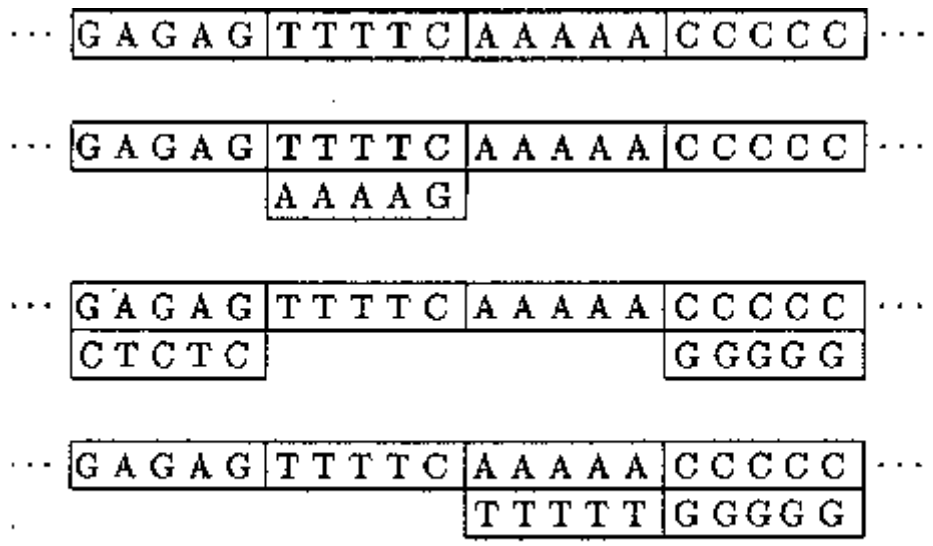
Σχήμα 2.6.1 Ένα memory strand

Στο σχήμα 2.6.1 δείχνεται ένα memory strand μήκους $n = 20$ που αποτελείται από $k = 4$ substrand το κάθε ένα με μήκος $m = 5$. Συνεπώς σε αυτή τη περίπτωση, το κάθε memory strand εμπεριέχει 4 bit πληροφορίας. Στο σχήμα 2.6.2 στο πρώτο memory complex, όλα τα substrand είναι off ενώ στο τελευταίο τα τελευταία δύο είναι σε θέση on. Οι αριθμοί που αναπαριστούν είναι 0000, 0100,1001,0011 αντίστοιχα. Στα memory strand στο σχήμα 2.6.1, κάθε substrand που αντιστοιχεί σε περιττή θέση(αντίστοιχα άρτια θέση) bit, συνίσταται εξολοκλήρου από πουρίνες(αντ. πυριμιδίνες). Αυτό γίνεται σκόπιμα για να δημιουργηθούν σύνορα ανάμεσα στα substrands. Έτσι δεν υπάρχει περίπτωση, ένας sticker να κολλήσει σε μια τέτοια περιοχή του memorystrand, ώστε να επικαλύπτει δύο substrand. Κάτι τέτοιο θα δημιουργούσε σύγχυση στο αποτέλεσμα των λειτουργιών που περιγράφονται παρακάτω. Γενικά στο sticker μοντέλο πρέπει να γίνει μια προσεκτική κωδικοποίηση των substrands και των stickers.

Η *πυκνότητα πληροφορίας* για μια κωδικοποίηση ορίζεται ως ο αριθμός των bits ανά νουκλεοτίδιο. Επειδή ένα bit κωδικοποιείται με m νουκλεοτίδια, έπεται ότι η πυκνότητα πληροφορίας είναι $1/m$ bits/base. Αν και η μέγιστη πυκνότητα πληροφορίας είναι $2 bits/base$, μια τέτοια κωδικοποίηση θα καθιστούσε επιρρεπή στο λάθος, οποιονδήποτε DNA υπολογιστή.

Οι *λειτουργίες* που συνιστούν την λειτουργικότητα του μοντέλου είναι *merge,separate,set* και *clear*. Όπως και πριν, ένας δοκιμαστικός σωλήνας, είναι ένα

πολυσύνολο τα στοιχεία του οποίου είναι τώρα σύμπλοκα μνήμης(memory complexes).



Σχήμα 2.6.2 Σύμπλοκα μνήμης

Η λειτουργία merge είναι όπως και πριν. Δύο δοκιμαστικοί σωλήνες, χύνονται σε έναν. Τα περιεχόμενα του τελευταίου, περιέχουν τα memory complexes και των δύο με τα προσκολλημένα σε αυτά stickers απείραχτα.

Η λειτουργία separate παράγει, δοθέντος ενός σωλήνα N, και ένα ακέραιο $i, 1 \le i \le k$, δύο νέους σωλήνες $+(N,i)$ και $-(N,i)$. Ο σωλήνας $+(N,i)$ (αντ. $-(N,i)$) περιέχει όλα τα σύμπλοκα μνήμης του πρωταρχικού σωλήνα N, όπου το i substrand είναι on(off).

Για ένα σωλήνα N, και ακέραιο $i, 1 \le i \le k$, η λειτουργία set παράγει ένα νέο σωλήνα $set(N,i)$ όπου το i substrand κάθε συμπλόκου μνήμης του N μπαίνει σε κατάσταση on.(Δηλαδή, ο κατάλληλος sticker προσκολλάται στο σύμπλοκο μνήμης αν το i substrand είναι off, διαφορετικά το i substrand παραμένει απείραχτο, αν είναι ήδη σε θέση on, δηλαδή έχει sticker προσκολλημένο.)

Τέλος, για ένα σωλήνα N, και ακέραιο $i, 1 \le i \le k$, η λειτουργία *clear* παράγει ένα νέο σωλήνα $clear(N,i)$ όπου το κάθε σύμπλοκο μνήμης του N έχει στη θέση του i substrand κατάσταση off, (δηλαδή ο sticker απομακρύνεται από το i substrand, αν υπάρχει προσκολλημένος, διαφορετικά παραμένει απείραχτο).

Οι *υπολογισμοί* σε ένα μοντέλο sticker, συνίστανται σε μια ακολουθία από τις παραπάνω λειτουργίες. *Είσοδοι* και *έξοδοι,* θα είναι δοκιμαστικοί σωλήνες. Για το *διάβασμα* της εξόδου, ένα σύμπλοκο μνήμης θα πρέπει να απομονωθεί από την έξοδο, και να καθοριστούν οι προσκολλημένοι σε αυτό stickers, διαφορετικά θα πρέπει να αναφερθεί, ότι η έξοδος δεν έχει σύμπλοκα μνήμης.

Η είσοδος, ή ο *αρχικός δοκιμαστικός σωλήνας* θα είναι μια *βιβλιοθήκη* από σύμπλοκα μνήμης. Συγκεκριμένα μια (k,l) *βιβλιοθήκη* αποτελείται από σύμπλοκα μνήμης με k substrand, από τα οποία τα τελευταία $k-l$ είναι σε θέση *off* ενώ τα πρώτα l είναι σε θέση on ή off με όλους τους δυνατούς τρόπους. Έτσι, αν το δούμε ως πολυσύνολο, μια (k,l) *βιβλιοθήκη περιέχει* 2^l διακεκριμένα σύμπλοκα μνήμης. Οι αναπαριστώμενες δυαδικές ακολουθίες είναι της μορφής $w0^{k-l}$, όπου w μια

οποιαδήποτε δυαδική ακολουθία μήκους l. Στον αρχικό δοκιμαστικό σωλήνα, τα πρώτα l substrand είναι η πραγματική είσοδος, ενώ τα υπόλοιπα $k-l$ χρησιμοποιούνται για τα ενδιάμεσα αποτελέσματα και για την έξοδο.

Το υπολογιστικό πρότυπο που συσχετίζεται με το sticker μοντέλο, είναι να λυθούν δύσκολα προβλήματα με εξαντλητική αναζήτηση πάνω σε είσοδο με μήκος l. Όλες οι πιθανές 2^l είσοδοι επεξεργάζονται παράλληλα. Μπορεί να ειπωθεί ότι αυτό το πρότυπο είναι η ουσία των DNA υπολογιστών.

Το μοντέλο sticker χρησιμοποιήθηκε στο [8], για να λυθεί ένα δύσκολο πρόβλημα, συγκεκριμένα το Minimal Set Cover πρόβλημα. Σε αυτό το πρόβλημα δίνεται ένα πεπερασμένο σύνολο $S=\{1,2,...,p\}$ και μια πεπερασμένη συλλογή $\{C_1,...,C_q\}$ με $C_i \subseteq S \quad i=1,...,q$, και ζητείται να βρεθεί το ελάχιστο υποσύνολο του $I=\{1,2,...,q\}$, έτσι ώστε $\bigcup_{i\in I} C_i = S$.

Φυσικά μια εξαντλητική αναζήτηση ανάμεσα σε όλα τα δυνατά υποσύνολα του I, που είναι $2^{|I|}=2^q$ το πλήθος θα λύσει το πρόβλημα. Η λύση με το sticker μοντέλο χρησιμοποιεί μια βιβλιοθήκη $(p+q,q)$. Περισσότερες λεπτομέρειες μπορούν να βρεθούν στο [8] και στο [9]. Παρακάτω θα εξεταστεί με λεπτομέρειες, η εφαρμογή του sticker μοντέλου(για την ακρίβεια μιας εξειδίκευσης του sticker model, του parallel sticker model) πάνω στην κρυπτανάλυση(«σπάσιμο») ενός κρυπτοσυστήματος πολύ γνωστού, του DES(Data Encryption Standard). Αυτό έγινε στο [10]. Είναι πιο ενδιαφέρον γιατί χρησιμοποιούνται *δύο επίπεδα* παραλληλίας: ένα στη παραλληλία του DNA(όπως π.χ στο βήμα 1 του πειράματος του Adleman), και ένα στη παραλληλία με την οποία εκτελούνται οι λειτουργίες του μοντέλου. Το δεύτερο επίπεδο παραλληλίας εξασφαλίζεται από την ύπαρξη ρομποτικού εξοπλισμού στο εργαστήριο.

Το κρυπτοσύστημα DES, μετασχηματίζει 64-bitα blocks από απλό ακρυπτογράφητο κείμενο(plaintext) σε 64-bitα κρυπτογραφημένα(ciphertext) blocks. Η κρυπτογράφηση γίνεται με τη βοήθεια ενός 56-bit κλειδιού. Το ίδιο κλειδί χρησιμοποιείται για την κρυπτογράφηση, και για την αποκρυπτογράφηση.(Το DES είναι ένα *συμμετρικό σύστημα* κρυπτογράφησης με μυστικό κλειδί τόσο για τη κρυπτογράφηση όσο και για την αποκρυπτογράφηση, σε αντίθεση με τα μη *συμμετρικά* κρυπτοσυστήματα που χρησιμοποιούν ένα δημόσιο κλειδί γνωστό σε όλους για την αποκρυπτογράφηση, και ένα μυστικό κλειδί για την κρυπτογράφηση). Θεωρούμε την «επίθεση γνωστού κειμένου». Σε αυτήν, έχουμε στη διάθεση μας ένα ζευγάρι από το plaintext και το αντίστοιχο του ciphertext, και θέλουμε να βρούμε το κλειδί. Προφανώς αν έχουμε περισσότερα του ενός από τέτοια ζευγάρια, διευκολύνεται κάπως η αναζήτησης μας.

Η πιο απλή μέθοδος είναι η μέθοδος της εξαντλητικής αναζήτησης: δοκιμάζουμε ένα-ένα κάθε κλειδί, κρυπτογραφούμε το plaintext, και βλέπουμε αν το αποτέλεσμα αυτής της κρυπτογράφησης ταιριάζει με το ciphertext που έχουμε στη διάθεση μας. Αν ταιριάζει τότε έχουμε βρει το κλειδί. Η διαδικασία της κρυπτογράφησης είναι δημόσια γνωστή . Ένδειξη της δύναμης του DES είναι,, ότι δεν υπάρχει καλύτερη μέθοδος από αυτή της εξαντλητικής αναζήτησης. Η μέθοδος της εξαντλητικής αναζήτησης, θα εφαρμοστεί στο sticker μοντέλο, μια και είναι κατάλληλο γι αυτή τη μέθοδο.

Η είσοδος στο sticker μοντέλο θα είναι μια βιβλιοθήκη (579,56) που θα περιέχει όπως ειπώθηκε 2^{56} σύμπλοκα μνήμης. Το κάθε σύμπλοκο μνήμης θα έχει 579 substrand που θα συμβολίζονται με $B_0, B_1, ..., B_{578}$. Άρα θα υπάρχουν και 579 stickers

που συμβολίζονται με $S_0, S_1, ..., S_{578}$. Το κάθε B_i υλοποιείται με 20 νουκλεοτίδια, άρα και ο κάθε S_i με 20 νουκλεοτίδια. Το σύμπλοκο μνήμης θα έχει μήκος $579 \times 20 = 11580$ νουκλεοτίδια. Το bit (substrand) B_0 χρησιμοποιείται, όπως θα δειχθεί αργότερα, για τη δημιουργία της βιβλιοθήκης. Τα *bit* $B_1, ..., B_{56}$ χρησιμοποιούνται για την αποθήκευση ενός διαφορετικού κλειδιού. Τα επόμενα *bit* $B_{57}, ..., B_{120}$ θα χρησιμοποιηθούν για την αποθήκευση του ciphertext(που προκύπτει μετά από τη κρυπτογράφηση, σύμφωνα με το κλειδί που είναι αποθηκευμένο στα πρώτα 56 bits). Τα bits $B_{121}, ..., B_{579}$ χρησιμοποιούνται για ενδιάμεσα αποτελέσματα, στη πορεία του υπολογισμού του ciphertext. Η μηχανή που υλοποιεί το parallel sticker μοντέλο(sticker machine), όπως θα δούμε αργότερα είναι ένας ρομποτικός σταθμός που μπορεί να εκτελεί παράλληλα κάποιες λειτουργίες του βιολογικού εργαστηρίου και ελέγχεται από ένα *μικροεπεξεργαστή*. Επειδή το plaintext είναι το ίδιο δε χρειάζεται να αποθηκευθεί στα σύμπλοκα μνήμης, αλλά αποθηκεύεται στη μνήμη του μικροεπεξεργαστή. Δοθέντος ενός ζευγαριού plaintext-ciphertext, ο αλγόριθμος, συνοπτικά, συνίσταται στα εξής τρία βήματα:

1. *Input step(βήμα εισόδου)*: Δημιουργία της (579,56) βιβλιοθήκης.
2. *Encryption step(βήμα κρυπτογράφησης):* Σε κάθε σύμπλοκο μνήμης, *υπολόγισε το ciphertext* που αντιστοιχεί στην κρυπτογράφηση του plaintext με το κλειδί που βρίσκεται αποθηκευμένο σε αυτό το σύμπλοκο μνήμης.
3. *Output step(βήμα εξόδου): Επιλογή* του συμπλόκου μνήμης του οποίου το ciphertext(που είναι αποθηκευμένο στις θέσεις $B_{57}, ..., B_{120}$ του συμπλόκου) ταιριάζει με το δοσμένο γνωστό ciphertext, και *διάβασμα* του κλειδιού που βρίσκεται στις θέσεις $B_1, ..., B_{56}$ του συμπλόκου αυτού.

Το βάρος της εργασίας, βρίσκεται στο δεύτερο βήμα, όπου γίνεται η κρυπτογράφηση δεδομένων σύμφωνα με το σύστημα DES. Θα εστιάσουμε στις στοιχειώδεις εντολές που εκτελούνται στο DES, και πώς αυτές υλοποιούνται από τη sticker machine. Περισσότερες λεπτομέρειες σχετικά με το DES, αναφέρονται στο [11].

Το DES είναι ένας αλγόριθμος κρυπτογράφησης με 16-*γύρους(rounds)*. Σε κάθε γύρο παράγεται ένα διαφορετικό 32-bit αποτέλεσμα. Έστω $R_1, ..., R_{16}$ αυτά τα αποτελέσματα. Τα R_{15} και R_{16} αποθηκεύονται στις θέσεις B_{57} ως B_{120} (δίπλα στο κλειδί για λόγους που θα αναφερθούν παρακάτω) και είναι δηλαδή το κρυπτογραφημένο κείμενο(ciphertext). Τα $R_1, ..., R_{14}$ αποθηκεύονται στα bit B_{121} ως B_{568}, είναι δηλαδή τα ενδιάμεσα αποτελέσματα. Τα 32 αριστερότερα bit και τα 32 δεξιότερα bit του plaintext, είναι αντίστοιχα τα R_{-1} και R_0 και αποθηκεύονται, όπως ειπώθηκε στη μνήμη του μικροεπεξεργαστή. Μένουν τα bits B_{569} ως B_{578} που χρησιμοποιούνται ως *workspace* και είναι τα μόνα bit που *γράφονται* και *σβήνονται* παραπάνω από μια φορά. Τα άλλα bit $B_{57}, ..., B_{568}$ γράφονται ακριβώς μια φορά(*write once bits).*

Βασικά, το $R_i \quad i = 1, ..., 16$ προέρχεται από τα R_{i-1} και R_{i-2} με τον ακόλουθο υπολογισμό:

$$R_i = R_{i-2} \oplus S(E(R_{i-1}) \oplus K_i) \quad (2.6.1)$$

όπου το $\oplus$ είναι το αποκλειστικό OR (eXclusive OR, XOR), K_i είναι μια (εξαρτώμενη από τον *γύρο*) επιλογή 48 bits από τα 56 του κλειδιού, E είναι η

συνάρτηση επέκτασης που παίρνει τα 32 bits του R_{i-1} και τα επαναλαμβάνει ή τα μεταθέτει για να δώσει 48 bits, και S είναι μια συνάρτηση που παίρνει 48 bits ως είσοδο και παράγει μια 32 bit έξοδο. Όλες οι παραπάνω συναρτήσεις(X-OR,E,S, K_i), είναι υλοποιημένες στο hardware του μικροεπεξεργαστή. Ο επεξεργαστής θα εκτελεί σε κάθε γύρο, μια φορά αυτές τις συναρτήσεις για κάθε δυνατή είσοδο, και στη συνέχεια εκμεταλλεύεται τον παραλληλισμό του DNA ώστε αυτές οι συναρτήσεις να εκτελεστούν παράλληλα σε όλα τα σύμπλοκα μνήμης. Περισσότερα παρακάτω, στο τμήμα που αναφέρεται στο τρόπο που υλοποιούνται οι συναρτήσεις στη parallel sticker μηχανή.

Η συνάρτηση X-OR έχει $2^2 = 4$ δυνατές εισόδους, ενώ οι συναρτήσεις K_i και E μπορεί να θεωρηθεί ότι επιστρέφουν 48 *θέσεις* πάνω στα 56 bit του κλειδιού, και στα 32 bit του R_{i-1} (επαναλαμβανόμενες θέσεις σε αυτή τη περίπτωση), αντίστοιχα. Οι θέσεις αυτές, στη περίπτωση της K_i, εξαρτώνται μόνο από τον γύρο i, ενώ στη περίπτωση της συνάρτησης E, δεν εξαρτώνται από τίποτα. *Δηλαδή και στις δύο περιπτώσεις, οι θέσεις που επιστρέφονται δεν εξαρτώνται από την είσοδο των συναρτήσεων*. Οι *τιμές* βέβαια αυτών των θέσεων, εξαρτώνται από το ποιο είναι το κλειδί, στη περίπτωση της K_i, και ποιο είναι το R_{i-1}, στη περίπτωση της E. Έτσι αυτές οι δύο συναρτήσεις δεν χρειάζεται να υλοποιούνται μέσω βιολογικών λειτουργιών από τη sticker μηχανή(αν και ο μικροεπεξεργαστής πρέπει να τις γνωρίζει, δηλαδή να γνωρίζει τις θέσεις για να μπορεί να καθοδηγήσει την sticker μηχανή ώστε να εκτελέσει το X-OR ανάμεσα στις σωστές θέσεις των συμπλόκων μνήμης, κατά την εκτέλεση της πράξης $E(R_{i-1}) \oplus K_i$). Έτσι μέχρι στιγμής ο επεξεργαστής κάνει 16*4(για X-OR)+16(για τη K_i)+16(για τη E)=96 υπολογισμούς. Η συνάρτηση S όμως έχει είσοδο 48 bit, οπότε ο επεξεργαστής θα έπρεπε να κάνει $16*2^{48}$ επιπλέον υπολογισμούς(και στους 16 γύρους). Ευτυχώς η συνάρτηση S μπορεί να σπάσει σε 8 διαφορετικές, ανεξάρτητες, 6bit προς 4bit, συναρτήσεις που ονομάζονται *S-boxes*. Έτσι ο επεξεργαστής χρειάζεται να κάνει μόνο $16*8*2^6$ υπολογισμούς. Είναι μάλιστα $S(A_1...A_8) = S_1(A_1)S_2(A_2)...S_8(A_8)$, όπου $A_1...A_8$ ένας 48bit αριθμός και $A_i, 1 \le i \le 8$ αποτελούνται από 6 bit. Άρα από τη σχέση 2.6.1, προκύπτει ότι τα 32bits του R_i, μπορούν να διαιρεθούν σε 8 ομάδες (chunks) των 4 bit. Κάθε chunk προκύπτει από 6 bit του $E(R_{i-1})$,άρα 6 κατάλληλα bit του R_{i-1} γιατί οι E επιστρέφει θέσεις πάνω στα bit της εισόδου της, 6 bit του K_i, άρα 6 κατάλληλα bit του κλειδιού, και 4 bit του R_{i-2} (Το αντίστοιχο chunk του R_{i-2}).

Πιο αναλυτικά ο υπολογισμός του j-οστού chunk έχει ως εξής:

1. 6 κατάλληλα bits του R_{i-1} (που βρίσκονται σε κάποιες θέσεις του εκάστοτε συμπλόκου μνήμης, ο επεξεργαστής ξέρει αυτές τις θέσεις γιατί ξέρει την E) και 6 κατάλληλα bits του K (του κλειδιού που βρίσκεται στο εκάστοτε σύμπλοκο μνήμης) γίνονται $X-OR$ για να παραχθεί ένα αποτέλεσμα των 6 bits, το οποίο αποθηκεύεται πάνω στο εκάστοτε σύμπλοκο μνήμης στις θέσεις $B_{569},...,B_{574}$. Αυτά τα XOR γίνονται παράλληλα σε όλα τα σύμπλοκα μνήμης, εκμεταλλευόμενοι τον παραλληλισμό των μορίων DNA. (Βλέπε τρόπο υλοποίησης της XOR πιο κάτω).
2. Υπολογίζεται η $S_j(B_{569}...B_{574})$ και το 4-bit αποτέλεσμα φυλάγεται στις θέσεις $B_{575},...,B_{578}$ του εκάστοτε συμπλόκου μνήμης. Πάλι αυτό το βήμα γίνεται

παράλληλα για όλα τα σύμπλοκα μνήμης.(Βλέπε τρόπο υλοποίησης S_j πιο κάτω)

3. Τα bits $B_{575},...,B_{578}$ του εκάστοτε συμπλόκου, γίνονται X-OR με τα 4-bit του j-οστού chunk του R_{i-2} (που βρίσκονται στις θέσεις $B_{120+(i-3)32+(j-1)\cdot 4},...,B_{120+(i-3)\cdot 32+(j-1)\cdot 4+3}$ για $15 > i-2 \geq 1$, ενώ για $-1 \leq i-2 < 1$ βρίσκονται μέσα στη μνήμη του επεξεργαστή(είναι bits του plaintext)) και έτσι παράγεται το j-οστό chunk του R_i. Το αποτέλεσμα φυλάγεται στις θέσεις $B_{120+(i-1)32+(j-1)\cdot 4},...,B_{120+(i-1)\cdot 32+(j-1)\cdot 4+3}$ του εκάστοτε συμπλόκου.
4. Αν ($i<16$ ή $j<8$) τότε όλο το workspace στις θέσεις $B_{569},...,B_{578}$ γίνεται clear για να καθαριστεί για τον υπολογισμού του επόμενου chunk.

Από τα βήματα 1,2,3 του παραπάνω αλγορίθμου, προκύπτει ότι οι συναρτήσεις XOR και S_j πρέπει να υλοποιηθούν και στην (parallel) sticker μηχανή, ενώ τόσο αυτές όσο και οι υπόλοιπες(δηλαδή όλες οι συναρτήσεις) πρέπει να υλοποιηθούν στο μικροεπεξεργαστή. Η υλοποίηση των συναρτήσεων στην (parallel) sticker μηχανή, γίνεται χρησιμοποιώντας τις απλούστερες λειτουργίες (parallel) separate, (parallel) merge και (parallel) set. Πρώτα θα γίνει μια ακριβέστερη περιγραφή της (parallel) sticker μηχανή, και των αντίστοιχων βασικών λειτουργιών, και στη συνέχεια θα περιγραφθεί πώς υλοποιούνται οι XOR και S_j.

Μια *parallel sticker μηχανή* μπορεί να θεωρηθεί ως ένας *παράλληλος ρομποτικός σταθμός εργασίας(parallel robotic workstation [8])*. Συνίσταται από μια *σειρά* από σωλήνες(*σωλήνες δεδομένων(data tubes), sticker σωλήνες, σωλήνες λειτουργίας(operator tubes))*, από *ρομποτικό εξοπλισμό(robotics)(*βραχίονες, αντλίες, συσκευές θέρμανσης/ψύξης, συνδέσμους κ.α) και ένα *μικροεπεξεργαστή* που ελέγχει το ρομποτικό εξοπλισμό. Στο Roweis et al.[8] γίνεται η υπόθεση ότι ο ρομποτικός εξοπλισμός και ένα σύνολο από 3 σωλήνες δεδομένων και 3 σωλήνες λειτουργίας, μπορούν να εκτελέσουν οποιαδήποτε από τις λειτουργίες *separate, merge, set και clear*.

Εδώ θα περιγραφθεί η προσέγγιση στο [10] όπου γίνεται η υπόθεση ότι ο ρομποτικός εξοπλισμός μπορεί να υλοποιήσει μια παράλληλη (parallel) έκδοση των παραπάνω 4 λειτουργιών(το γιατί επιλέγονται 32 ή 64 το πολύ σωλήνες δεδομένων για τις παράλληλες λειτουργίες θα φανεί παρακάτω, στην υλοποίηση των συναρτήσεων):

1. *Parallel separate* $(N_0,...,N_k)=(N_{00},N_{01},...,N_{k0},N_{k1})$. Ο ρομποτικός εξοπλισμός μπορεί να κάνει την λειτουργία separate παράλληλα, σε 32 το πολύ σωλήνες δεδομένων, χρησιμοποιώντας 32 separation operator tubes(που περιέχουν τα probes $\overline{B}_k$). Παράγονται έτσι 64 το πολύ σωλήνες δεδομένων($N_{i0}=+(N_i,B_k)$ και $N_{i1}=-(N_i,B_k)$ από κάθε $N_i, i=1,...,32$).
2. *Parallel merge* $(N_0,...,N_k)=N_{k+1}$. Ο ρομποτικός εξοπλισμός μπορεί να κάνει παράλληλα merge,64 το πολύ σωλήνες δεδομένων, σε ένα μόνο σωλήνα δεδομένων.
3. *Parallel set* $(N_1,..,N_m,B_k).(m\leq 64)$ Ο ρομποτικός εξοπλισμός μπορεί, χρησιμοποιώντας ένα σωλήνα sticker, με stickers S_k, να θέσει το bit που αναπαριστάται από το substrand B_k σε κατάσταση on, σε 64 το πολύ σωλήνες δεδομένων(που περιέχουν σύμπλοκα μνήμης) πάντα παράλληλα.

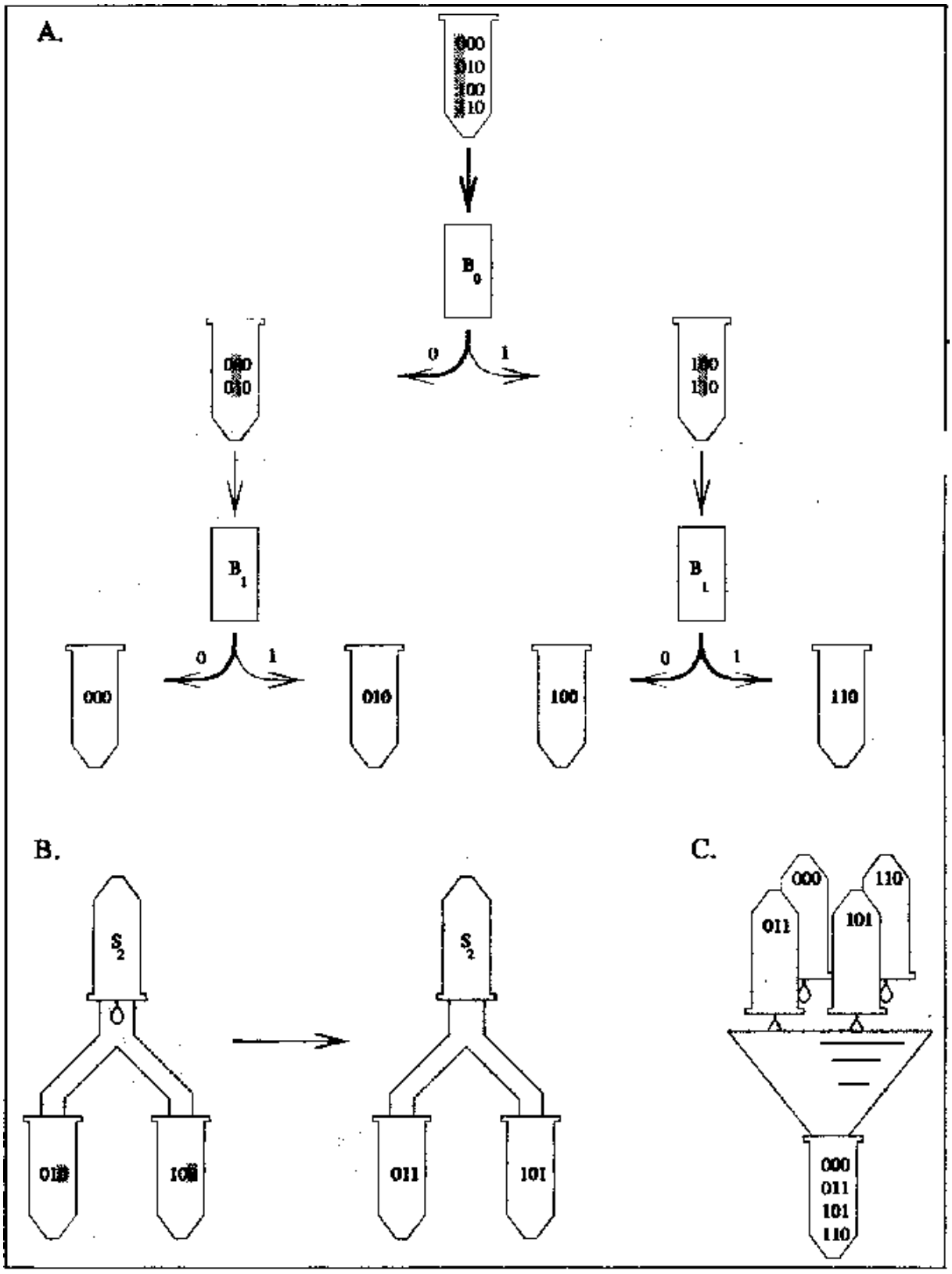
4. *Parallel Clear.* Ο ρομποτικός εξοπλισμός μπορεί να κάνει clear τα bits του workspace σε όλα τα σύμπλοκα μνήμης σε ένα σωλήνα δεδομένων. Εδώ ο παραλληλισμός βρίσκεται στο γεγονός ότι υποθέτουμε ότι όλα τα bits γίνονται σε θέση off παράλληλα(δηλαδή αφαιρούνται ταυτόχρονα όλοι οι stickers που ενδεχομένως είναι προσκολλημένοι στα bits του workspace).

Περνάμε τώρα στην υλοποίηση των συναρτήσεων XOR $(\oplus)$ και S_j για $j = 1,...,8$ στην parallel sticker μηχανή. (Αυτές οι συναρτήσεις υλοποιούνται και στον μικροεπεξεργαστή που ελέγχει το ρομποτικό εξοπλισμό(στον μικροεπεξεργαστή υλοποιούνται και οι άλλες δύο συναρτήσεις) ως κατάλληλο λογισμικό ή σαν απλοί πίνακες που για κάθε δυνατή είσοδο, βρίσκουμε την τιμή της συνάρτησης. Για παράδειγμα ο πίνακας της XOR είναι διαστάσεων $2^2 \times 2^2$ και περιέχει 16 στοιχεία, ενώ σύμφωνα με το [11] υπάρχουν οκτώ πίνακες ένας για κάθε S_j, διαστάσεων $4x16$ ο καθένας. Έστω ότι θέλουμε να υπολογίσουμε τη τιμή $S_j(A_1)$ όπου A_1 έχει 6 bits. Τα δύο πρώτα bits του A_1 μας δίνουν την γραμμή($2^2 = 4$ δυνατότητες), ενώ τα τέσσερα υπόλοιπα($2^4 = 16$ δυνατότητες) μας δίνουν την στήλη, για τον πίνακα της S_j. Επίσης υπάρχει ένας πίνακας για την E και 16 πίνακες ένα για κάθε K_i). Όμως, στην parallel sticker μηχανή υλοποιούνται σε τρία στάδια, το κάθε στάδιο χρησιμοποιεί τις λειτουργίες parallel separate, parallel set, parallel merge αντίστοιχα. Τα τρία στάδια (A,B,C) έχουν ως ακολούθως:

Στάδιο Α. Έστω ότι θέλουμε να υπολογίσουμε την $B_k = B_i \oplus B_j$, δηλαδή σε κάθε σύμπλοκο μνήμης που βρίσκεται σε ένα (αρχικό) σωλήνα N να θέσουμε την τιμή του bit B_k ως το XOR των τιμών στις θέσεις B_i και B_j του εκάστοτε συμπλόκου. Γίνεται parallel separate ο αρχικός σωλήνας δύο φορές, για να παραχθούν τέσσερεις σωλήνες δεδομένων, ένας για κάθε δυνατή τιμή στις θέσεις B_i και B_j. Στην πρώτη λειτουργία parallel separate(που μπορεί να θεωρηθεί ως απλό separate) γίνεται parallel separate$(\mathrm{N}, B_i) = (N_0, N_1)$ και χρησιμοποιείται ένας separation operator tube με probe $\overline{B}_i$. Στη δεύτερη λειτουργία έχουμε πραγματικό parallel separate $(N_0, N_1, B_j) = (N_{00}, N_{01}, N_{10}, N_{11})$ δηλαδή προκύπτουν 4 σωλήνες . Το δεύτερο παράλληλο separate γίνεται παράλληλα στους N_0 και N_1, χρησιμοποιώντας παράλληλα δύο όμοιους separation operator tubes με probe $\overline{B}_j$. Ο κάθε ένας N_{pl} $(p,l = 0,1)$ περιέχει τα σύμπλοκα μνήμης του N τα οποία στις θέσεις B_i και B_j έχουν τιμή αντίστοιχα p και l.

Για την συνάρτηση S_j έστω ότι θέλουμε να υπολογίσουμε το $S_j(B_{i_1}...B_{i_6})$. Γίνονται 6 parallel separate(το πρώτο είναι πάλι απλό) διαδοχικά, χρησιμοποιώντας separation operator tubes με probes $B_{i_1},...,B_{i_6}$ αντίστοιχα για κάθε parallel seperate, και τελικά παίρνουμε τους σωλήνες $N_{000000},...,N_{111111}$ δηλαδή 64 σωλήνες δεδομένων. Ανάλογα ισχύουν σχετικά με τη ταυτότητα των συμπλόκων που βρίσκονται σε κάθε ένα από αυτούς τους σωλήνες. Δηλαδή ο σωλήνας N_{010101} έχει σύμπλοκα των οποίων οι θέσεις $B_{i_1}, B_{i_3}, B_{i_5}$ έχουν την τιμή 0(είναι σε θέση off), ενώ οι θέσεις $B_{i_2}, B_{i_4}, B_{i_6}$ έχουν την τιμή 1.

Γενικά για μια συνάρτηση $f : \{0,1\}^n \to \{0,1\}^m$ δηλαδή μια n-bit προς m-bit συνάρτηση, αυτό το στάδιο έχει ως εξής:



Σχήμα 2.6.3 Υπολογισμός του X-OR B2=B0+B1

Parallel separate τον αρχικό σωλήνα n φορές για να πάρουμε 2^n σωλήνες δεδομένων, ο κάθε ένας με σύμπλοκα με έχουν στις ζητούμενες θέσεις την ανάθεση που αντιστοιχεί στο σωλήνα. Αυτό απαιτεί 2^{i-1} separation operator tubes για την i-οστή parallel separate, 2^i data tubes για να βάλουμε τα αποτελέσματα(από κάθε separation operator tube θα γεμίσουν δύο σωλήνες δεδομένων) και 2^{i-1} data tubes που είναι το αποτέλεσμα της $i-2$ parallel separate(και σε κάθε ένα από τους οποίους «εφαρμόζεται» και ένας separation operator tube).Άρα, συνολικά απαιτούνται $2^i + 2^{i-1} = 3 \cdot 2^{i-1}$ data tubes για την $i-1$ parallel separate. Άρα για την nοστή parallel separate απαιτούνται $3 \cdot 2^{n-1}$ data tubes και 2^{n-1} separation operator tubes. Επίσης ως *ενεργούς σωλήνες* (active tubes) θεωρούνται οι σωλήνες οι οποίοι επεξεργάζονται την ίδια χρονική στιγμή από τον ρομποτικό εξοπλισμό. Έτσι κατά την i-οστή parallel separate χρησιμοποιούνται 2^{i-1} data tubes (αυτοί που διαχωρίζονται), 2^{i-1} separation operator tubes(κανονίζουν ως προς τη θέση που θα γίνει ο διαχωρισμός) και 2^{i-1} data tubes προορισμού(όχι 2^i γιατί π.χ πρώτα γεμίζεται ο αριστερός και μετά ο δεξιός). Για την κατανόηση των παραπάνω θα πρέπει να μελετηθεί και το σχήμα 2.6.3(στάδιο Α) που είναι για την απλή περίπτωση της XOR ($n=2, m=1$). Τελικά έχουμε $3 \cdot 2^{n-1}$ active tubes για την n-οστή parallel separate.

Στάδιο Β. Για την περίπτωση της XOR $B_k = B_i \oplus B_j$ έχουμε ένα parallel set στη θέση B_k για εκείνους τους data tubes(που προέκυψαν από το στάδιο Α) για τους οποίους το B_k πρέπει να γίνει 1. Δηλαδή για τους N_{01} και N_{10}. Έχουμε λοιπόν $parallelset(N_{01}, N_{10}, B_k)$. Εδώ φαίνεται γιατί ο μικροεπεξεργαστής πρέπει να γνωρίσει την συνάρτηση XOR(βέβαια στη σημερινή εποχή δεν υπάρχει μικροεπεξεργαστής που να μη ξέρει να κάνει XOR, αλλά έτσι φαίνεται γιατί πρέπει να υλοποιηθούν και οι υπόλοιπες συναρτήσεις στο εσωτερικό του επεξεργαστή). Ο μικροεπεξεργαστής ξέρει με τη βοήθεια του ρομποτικού εξοπλισμού και λόγω του σταδίου Α, ότι οι σωλήνες N_{01} και N_{10} έχουν $B_i = 0, B_j = 1$ και $B_i = 1, B_j = 0$ και επειδή γνωρίζει ότι $1 \oplus 0 = 1$ και $0 \oplus 1 = 1$, αποφασίζει και δίνει εντολή στον ρομποτικό εξοπλισμό να κάνει το bit B_k on στα σύμπλοκα των σωλήνων N_{01} και N_{10}. Από τη στιγμή που θα μπει ο sticker S_k στους σωλήνες N_{01} και N_{10}, το bit B_k θα γίνει on σε όλα τα σύμπλοκα που περιέχουν αυτοί οι σωλήνες, σε μικρό χρόνο, *λόγω του παραλληλισμού των αντιδράσεων ανάμεσα στα μόρια DNA*. Βλέπε και σχήμα 2.6.3 (στάδιο Β) για καλύτερη κατανόηση.

Γενικά στη περίπτωση που έχουμε μια συνάρτηση n-bit σε m-bit θα έχουμε m φορές parallel set σε ένα κατάλληλο υποσύνολο των 2^n data tubes που προέκυψαν στο τέλος του Α σταδίου. Το υποσύνολο αυτό το βρίσκει ο επεξεργαστής εφόσον ξέρει την συνάρτηση. Αυτό το υποσύνολο αλλάζει σε κάθε μια από τις m parallel separate(στην περίπτωση του XOR έχουμε $m=1$, οπότε δε φαίνεται αυτό) γιατί π.χ μπορεί για $n=6, m=4$ να είναι $f(000000)=1010$ και $f(010101)=0010$, οπότε στο σωλήνα N_{000000} θα εφαρμοστεί parallel set την πρώτη και την τρίτη φορά, ενώ στο σωλήνα N_{010101} θα εφαρμοστεί parallel set μόνο την τρίτη φορά.(Το parallel set της τρίτης φοράς θεωρείται ως μια εντολή παρόλο που εφαρμόζεται και στον N_{000000} αλλά και στον N_{010101}. Αυτό είναι το *επίπεδο παραλληλίας του ρομποτικού εξοπλισμού)*. Επίσης όπως είναι ευνόητο αλλάζει και ο sticker σε κάθε ένα από τις m parallel separate γιατί γράφουμε σε διαφορετική θέση πάνω στο σύμπλοκο. Επίσης ο

αριθμός των active tubes είναι το πολύ 2^n+1(Σε κάθε βήμα ένας(1) sticker tube και 2^n το πολύ, data tubes, είναι active).

Στάδιο C. Parallel merge τα περιεχόμενα όλων των data tubes που προέκυψαν από το στάδιο B, σε ένα data tube. Εδώ πάλι φαίνεται το επίπεδο παραλληλίας του ρομποτικού εξοπλισμού αφού π.χ στη περίπτωση του XOR, 4 βραχίονες θα πιάσουν ταυτόχρονα τους 4 σωλήνες και θα τους χύσουν σε ένα σωλήνα.

Για μια συνάρτηση $n-bit$ σε $m-bit$ γενικεύεται ως Parallel merge τα περιεχόμενα των 2^n data tubes σε ένα data tube. Αυτό απαιτεί 2^n+1 data tube και 2^n+1 active tubes.

Αυτό που προκύπτει από τα παραπάνω 3 στάδια είναι ότι για μια συνάρτηση $n-bit$ σε $m-bit$, απαιτούνται n *Parallelseparate*, m *Parallelset* και 1 *Parallelmerge*. Άρα έχουμε $n+m+1$ εντολές συνολικά. Συνολικά απαιτούνται separation operator tubes που είναι $\sum_{i=1}^{n} 2^{i-1} = 2^n - 1$, οι sticker tubes είναι m (στάδιο B), ενώ θέλουμε το πολύ $3\cdot 2^{n-1}$ data tubes(το μέγιστο από τα 3 στάδια) και $3\cdot 2^{n-1}$ active tubes(πάλι ο μέγιστος αριθμός από τον απαιτούμενο σε κάθε στάδιο). Άρα για την περίπτωση της XOR θέλουμε $2+1+1=4$ βήματα, $2^2-1=3$ separation operator tubes(ένας τύπου B_0 και δύο τύπου B_1 όπως φαίνεται στο σχήμα 2.6.3), το πολύ $3\cdot 2=6$ data tubes(στο στάδιο A) και το πολύ 6 active tubes(πάλι στο στάδιο A).

Για τις S_j (6 bit προς 4bit) θέλουμε $6+4+1=11$ βήματα, ενώ οι data tubes είναι το πολύ $3\cdot 2^{6-1}=96$. Το ίδιο για τους active tubes. Ο αριθμός των active tubes καθορίζει το μέγιστο επίπεδο παραλληλισμού, και έτσι θα πρέπει ο ρομποτικός εξοπλισμός να είναι ικανός να χειρίζεται μέχρι και 96 σωλήνες ταυτόχρονα(από τους οποίους στη χειρότερη περίπτωση, κατά στο στάδιο A, οι 64 θα είναι data tubes και οι 32 separation operator tubes). Φαίνεται επίσης στο στάδιο C, ότι στη χειρότερη περίπτωση(εκτέλεση υπολογισμού για κάποια S_j) θα πρέπει να γίνει parallel merge σε 2^6 σωλήνες, στο στάδιο B, πάλι στη χειρότερη περίπτωση θα πρέπει να γίνει parallel merge σε 2^6=64 σωλήνες, και στο στάδιο A στη χειρότερη περίπτωση θα πρέπει να γίνει parallel separate σε $2^{6-1}=32$ σωλήνες. Γι αυτό λοιπόν επιλέχτηκαν αυτά τα νούμερα, στη περιγραφή της parallel sticker μηχανής.

Ο αριθμός των rack tubes είναι το άθροισμα του αριθμού των data,sticker, separation operator tubes που χρησιμοποιούνται. Αλλά το να υπολογίσουμε αυτό τον αριθμό έχοντας στο μυαλό μας κάποιο S_j ή και όλα μαζί δεν ανταποκρίνεται στην πραγματικότητα. Μπορεί οι data tubes να είναι επαναχρησιμοποιήσιμοι και να κάνουν την ίδια δουλειά, άρα η συνεισφορά των data tubes στον αριθμό των rack tubes θα είναι 96. Αλλά, οι sticker tubes και οι separation operator tubes αν και είναι επαναχρησιμοποιήσιμοι, εντούτοις δεν κάνουν όλοι την ίδια δουλειά, αλλά ο καθένας είναι εξειδικευμένος για μια συγκεκριμένη εργασία. Για παράδειγμα οι sticker tubes κάνουν 579 διαφορετικές εργασίες(το να προσκολλούνται σε συγκεκριμένο substrand) και παρόμοια για τους separation operator tubes. Έτσι στη χειρότερη περίπτωση θα θέλαμε $579\cdot 4=2316$ sticker tubes και $579\cdot(2^6-1)=36477$ separation operator tubes. Ευτυχώς μπορούμε να υλοποιήσουμε τις λειτουργίες που περιγράφηκαν πιο πάνω, με τέτοιο τρόπο ώστε να μειωθούν σημαντικά αυτοί οι αριθμοί. Απαιτείται προσοχή στη σειρά με την οποία γίνονται τα διαδοχικά parallel separate στο στάδιο A. Περισσότερες λεπτομέρειες ακολουθούν μετά το πρόγραμμα DNA.

Ας δούμε πως μπορούμε να κωδικοποιήσουμε τον αλγόριθμο υπολογισμού για το j-οστό chunk σε πρόγραμμα DNA, χρησιμοποιώντας τόσο τις λειτουργίες της parallel sticker machine, αλλά και τις συναρτήσεις στο μικροεπεξεργαστή.

Πρώτα το DNA πρόγραμμα για την υλοποίηση της XOR σε ένα σωλήνα N που περιέχει σύμπλοκα στα οποία θέλουμε να γίνει η πράξη $B_k = B_i \oplus B_j$

$XOR(N, B_k, B_i, B_j)$

{

$(N_0, N_1) \leftarrow Parallel\ separate(N, B_i)$

$(N_{00}, N_{01}, N_{10}, N_{11}) \leftarrow Parallel\ separate(N_0, N_1, B_j)$

$Parallel\ set(N_{01}, N_{10}, B_k)$

$N \leftarrow Parallel\ merge(N_{00}, N_{01}, N_{10}, N_{11})$

}

Ακολουθεί το πρόγραμμα για τις S_j σε ένα σωλήνα N που περιέχει σύμπλοκα στα οποία θέλουμε να γίνει η πράξη $D_1D_2D_3D_4 = S_j(C_1C_2C_3C_4C_5C_6)$ όπου D_i, C_k είναι κάποια από τα $B_0,...,B_{579}$. Με s_j είναι η ίδια συνάρτηση, αλλά μέσα στον μικροεπεξεργαστή. Η $projection(s_j(q), r)$ επιστρέφει το r bit της τιμής $s_j(q)$ και προφανώς μπορεί να την εκτελέσει ο μικροεπεξεργαστής.

$S_j(N, D_1D_2D_3D_4, C_1C_2C_3C_4C_5C_6)$

{

$N_0 \leftarrow N$

for r=1 to 6 do begin

$(T_0,...,T_{2^r-1}) \leftarrow Parallel\ separate(N_0,...,N_{2^{r-1}-1}, C_r)$

$(N_0,...,N_{2^r-1}) \leftarrow (T_0,...,T_{2^r-1})$

end

for $r = 1$ to 4 do begin

$l \leftarrow 0$

for $q = 0$ *to* 63 do begin

if $projection(s_j(q), r) = 1$ then $i_l = q; l \leftarrow l + 1$

end

$Parallel\ set(N_{i_1},...,N_{i_l}, D_r)$

end

$N \leftarrow Parallel\ merge(N_0,...,N_{2^6-1})$

}

Στο παραπάνω πρόγραμμα οι εντολές στο loop για το q, εκτελούνται εσωτερικά στον επεξεργαστή, και αποφασίζουν το υποσύνολο των σωλήνων (που προέκυψαν από το στάδιο Α) στο οποίο θα τεθεί το bit D_r, κατόπιν εντολής του μικροεπεξεργαστή στον ρομποτικό εξοπλισμό.

Εφόσον έχουμε στα χέρια μας τα DNA-υποπρογράμματα για τις XOR και S_j, $j = 1,...,8$, μπορούμε να γράψουμε το DNA πρόγραμμα που υπολογίζει το j-οστό chunk στον γύρο $i = 1,...,16$ για όλα τα σύμπλοκα μνήμης ενός σωλήνα N.

$chunk(N, j, i)$

{

for r=1 to 6 do begin

$p = 120 + 32 \cdot (i-2) + E(6 \cdot (j-1) + r)$

$q = K_i(6 \cdot (j-1) + r)$

if $p < 569$ then $C_r = B_p$;

if $i - 1 = 15$ then $p = 56 + E(6 \cdot (j-1) + r)$; $C_r = B_p$

if $i - 1 = 0$ then $p = 31 + E(6 \cdot (j-1) + r)$; $C_r = M_p$

$F_r = B_q$

$XOR(N, B_{569+r-1,} C_r, F_r)$

end

$S_j(N, B_{575}B_{576}B_{577}B_{578}, B_{569}B_{570}B_{571}B_{572}B_{573}B_{574})$

for r=1 to 4 do begin

$p = 120 + 32 \cdot (i-1) + (j-1) \cdot 4$

if $i = 15$ then $p = 56$

if $i = 16$ then $p = 88$

$q = 120 + 32 \cdot (i-3) + (j-1) \cdot 4$

$C_r = B_{q+r}$

if $i - 2 = -1$ then $q = (j-1) \cdot 4$; $C_r = M_{q+r}$

$XOR(N, B_{p+r}, B_{574+r}, C_r)$

end

}

Στο πιο πάνω πρόγραμμα τα M_l είναι τα bits του plaintext που είναι αποθηκευμένα στη μνήμη του μικροεπεξεργαστή στις θέσεις $0-63$. Το πρόγραμμα, που δοθέντος της αρχικής (579,56) βιβλιοθήκης που περιέχεται στο σωλήνα N, υπολογίζει τα ciphertext, για όλα τα memory strands του σωλήνα N είναι τώρα απλό.

$cipher(N)$

{

for $i = 1$ to 16 do begin

for $j = 1$ to 8 do begin

$chunk(N, i, j)$;

if $i < 16$ OR $j < 8$ then *Parallel clear(workspace);*

end

end;

}

Με τα παραπάνω DNA-προγράμματα, πραγματοποιείται το *βήμα 2(encryption step)* του αλγορίθμου για την επίθεση στο DES. Όπως ειπώθηκε, αυτό είναι και το πιο σημαντικό βήμα, διότι κουβαλάει το *βάρος* της εργασίας που πρέπει να γίνει. Παρακάτω θα περιγραφεί η διαδικασία για το input step, και στη συνέχεια για το output step.

Όπως περιγράφεται και στο Adleman, Rothermund,Roweis,Winfree[10], στο input step, θα πρέπει να δημιουργηθεί ο αρχικός δοκιμαστικός σωλήνας N (που στο δεύτερο βήμα δίνεται ως είσοδος στο πρόγραμμα cipher). Όπως ειπώθηκε επίσης, ο αρχικός σωλήνας N θα είναι η βιβλιοθήκη $(579,56)$, και θα περιέχει 2^{56} διακεκριμένα μόρια(χωρίς την μέτρηση των αντιγράφων του κάθε μορίου), το καθένα

από τα οποία θα αναπαριστά και ένα διαφορετικό κλειδί . Η δημιουργία της βιβλιοθήκης, μπορεί να γίνει με τον ακόλουθο τρόπο:

1. Ξεκινάμε με μια αρχική ποσότητα από memory strands που είναι πολλά αντίγραφα του μορίου $B_0 B_1 ... B_{579}$ (δεν υπάρχουν stickers προσκολλημένοι)
2. Διαιρούμε το σωλήνα του 1 σε δύο σωλήνες A και B
3. Τοποθετούμε μια αρκετή ποσότητα από τους S_1 ως και S_{56}, στο σωλήνα Α, και δημιουργούμε συνθήκες ανόπτησης(annealing) στο σωλήνα Α, δηλαδή τον υποβάλλουμε σε ψύξη.
4. Χρησιμοποιείται ο $S_0 = \overline{B}_0$ ως probe, για να διαχωριστούν (separate) τα σύμπλοκα μνήμης(memory complexes) που υπάρχουν τώρα στον Α, από τη περίσσεια των stickers $S_1,...,S_{56}$. Πράγματι επειδή δε τοποθετήθηκε ο sticker S_0 στο βήμα 3, όλα τα σύμπλοκα μνήμης θα έχουν ελεύθερο το block B_0.
5. A=Merge(A,B)
6. Θερμαίνουμε, και κατόπιν ψύχουμε τον Α ώστε η κατανομή των προσκολλημένων stickers πάνω στα memory strands, να γίνει τυχαία(Poisson).

Τα σύμπλοκα μνήμης που δημιουργούνται από αυτή τη διαδικασία ακολουθούν την κατανομή Poisson. Περίπου 63% των κλειδιών θα αναπαρασταθούν, και κατά μέσο όρο, υπάρχει ένα σύμπλοκο μνήμης για κάθε κλειδί. Έτσι, αν δεν γίνουν λάθη κατά την διάρκεια του υπολογισμού στο *encryption step*(λάθη κατά την εκτέλεση των βιολογικών λειτουργιών separate,merge,clear,set), έχουμε μια σημαντική πιθανότητα να βρούμε το ζητούμενο κλειδί. Αυτή η πιθανότητα μπορεί να αυξηθεί αν αρχίσουμε στο βήμα 1 παραπάνω, με μεγαλύτερη αρχική ποσότητα από memory strands. Για να γίνει βέβαιο, ότι περίπου 95% των κλειδιών θα αναπαρασταθούν, και ότι κατά μέσο όρο, 3 αντίγραφα του ίδιου κλειδιού θα υπάρχουν, θα πρέπει να χρησιμοποιηθεί τριπλάσια ποσότητα DNA.

Για το *output step*, θα πρέπει να επιλεχθούν τα σύμπλοκα μνήμης που στις θέσεις $B_{57} - B_{120}$ έχουν το γνωστό ciphertext. Αυτό μπορεί πολύ απλά να γίνει με 64 separation βήματα, διαδοχικά στα $B_{57}, B_{58}, ..., B_{120}$. Δεν πρέπει να γίνει parallel separate γιατί τότε στο τέλος θα προέκυπταν 2^{64} σωλήνες(θα γνωρίζαμε ποιος σωλήνας περιέχει τι σύμπλοκα, αλλά ο αριθμός αυτός είναι πολύ μεγάλος ώστε να έχουμε τόσους σωλήνες!!). Έτσι γίνεται ένα separate σε κάθε ένα από τα 64 βήματα, επιλέγοντας πάντα τον κατάλληλο σωλήνα από τους δύο που προκύπτουν ως έξοδο του κάθε βήματος. Για παράδειγμα αν τα δύο πρώτα bits του ciphertext είναι 01, τότε στο πρώτο separate, με probe το νουκλεοτίδιο $\overline{B}_{57}$,θα επιλεγεί ο αριστερός σωλήνας, πάνω στον οποίο θα γίνει το δεύτερο separate με probe $\overline{B}_{58}$, και θα επιλεγεί ο δεξιός σωλήνας, κ.ο.κ για τα επόμενα 62 βήματα. Στο σωλήνα που θα προκύψει στο τέλος της διαδικασίας, γνωρίζουμε το περιεχόμενο των συμπλόκων μνήμης(δηλαδή που είναι προσκολλημένοι οι stickers) στις θέσεις $B_{57} - B_{120}$(το ciphertext) αλλά δε γνωρίζουμε το περιεχόμενο στις θέσεις $B_1 - B_{56}$ (δηλαδή το κλειδί).

Για να διαβαστεί το κλειδί, η μέθοδος sequencing δε μπορεί να χρησιμοποιηθεί διότι τα σύμπλοκα μνήμης είναι μερικώς διπλά (partially double) μόρια DNA, και μάλιστα στην πληροφορία που θέλουμε να διαβάσουμε, δηλαδή σε ποιες θέσεις είναι προσκολλημένοι οι stickers(δηλαδή η αλυσίδα με τους stickers, δεν είναι απαραίτητα συνεχόμενη). Αυτό που μπορεί να γίνει, είναι να μιμηθούμε την διαδικασία του

separation, με 56 ακόμη βήματα για τις θέσεις $B_1 - B_{56}$, κάνοντας την λειτουργία detect στους δύο σωλήνες που προκύπτουν από κάθε βήμα, και επιλέγοντας αυτόν που έχει σύμπλοκα μνήμης, και συνεχίζοντας στο επόμενο βήμα με αυτόν που επιλέξαμε με τη βοήθεια της λειτουργίας detect. Η λειτουργία detect είναι επιρρεπής στο λάθος. Στο [10] περιγράφεται μια μέθοδος που δε βασίζεται στη λειτουργία detect, αλλά κάνει πλήρης την αλυσίδα των stickers που βρίσκεται κάτω από την αλυσίδα $B_0 ... B_{120}$. Αυτό γίνεται με το να δημιουργεί νέους stickers S_i' για κάθε $i = 0,...,120$. Οι S_i' διαφέρουν από τους S_i στα νουκλεοτίδια $9-12$. Άρα έχουν αρκετή μεγάλη ομοιότητα με τους S_i και έτσι μπορούν να προσκολληθούν στις κενές θέσεις των B_i, κάνοντας έτσι συνεχόμενη την αλυσίδα των stickers, που θα αποτελείται βέβαια από S_i αλλά και από S_i'. Αυτή η αλυσίδα μπορεί να γίνει ενίσχυση με PCR (χρησιμοποιώντας primers S_0' και $\overline{S_{120}'}$ ή $\overline{S}_{120}$ ανάλογα με το αν το τελευταίο bit του ciphertext είναι 0 ή 1. Αν είναι 0 σημαίνει ότι θα προσκολληθεί ο S_{120}', ενώ αν είναι 1 σημαίνει ότι είναι προσκολλημένος ο S_{120}. Στη συνέχεια μπορεί να εφαρμοσθεί η μέθοδος sequencing για να γίνει το διάβασμα του διαλύματος, με μικρή πιθανότητα λάθους, λόγω της ενίσχυσης με PCR.

ΜΕΡΟΣ ΙΙ

ΘΕΩΡΗΤΙΚΑ ΜΟΝΤΕΛΑ ΥΠΟΛΟΓΙΣΜΟΥ

1. Μηχανές Turing,Γλώσσες RE,κλάση NP

Εδώ θα αναφερθεί το τυπικό θεωρητικό μοντέλο υπολογισμού που είναι οι μηχανές Turing. Γενικά όταν αναφερόμαστε σε θεωρητικά μοντέλα υπολογισμού, με τον όρο μηχανή δεν εννοείται κάποια υλική μηχανική κατασκευή που εκτελεί λειτουργίες αλλά μια θεωρητική οντότητα η οποία έχει μαθηματική δομή και λειτουργικότητα και συνεπώς οι ιδιότητες αυτής μπορούν να μελετηθούν από αυστηρή μαθηματική σκοπιά.

Αν και οι ψηφιακοί ηλεκτρονικοί υπολογιστές έχουν ως θεωρητικό μοντέλο υπολογισμού τις μηχανές RAM(Random Access Machines), εδώ θα γίνει μια σύντομη αναφορά σε αυτές και θα περιγραφούν πιο διεξοδικά οι μηχανές Turing. Αυτές αν και διαφέρουν σαν θεωρητικό μοντέλο από τις RAM, εντούτοις αποδυκνείεται ότι είναι υπολογιστικά ισοδύναμα μοντέλα, δηλαδή με απλά λόγια, ότι πρόβλημα αντιμετωπίζεται από το ένα μοντέλο, αντιμετωπίζεται και από το άλλο, με "ελάχιστη" διαφορά στις απαιτήσεις χώρου και χρόνου. Οι μηχανές Turing είναι το standard θεωρητικό μοντέλο, υπό την έννοια ότι οποιοδήποτε άλλο υπολογιστικό μοντέλο, προκειμένου να θεωρηθεί ικανό(για την αντιμετώπιση προβλημάτων) θα πρέπει να αποδεικνύεται ότι είναι υπολογιστικά ισοδύναμο με μηχανές Turing. Ο αυστηρός ορισμός της έννοιας του "υπολογιστικά ισοδύναμου" θα δοθεί αφότου έχουμε μιλήσει για μηχανές Turing. Έτσι οι ψηφιακοί ηλεκτρονικοί υπολογιστές ως πρακτική εφαρμογή του μοντέλου RAM είναι το ίδιο ικανοί με μια μηχανή Turing(ως προς τις δυνατότητες επίλυσης προβλημάτων όχι ως προς τις multimedia δυνατότητες τους!). Στους DNA υπολογιστές θα θεωρηθούν άλλα θεωρητικά μοντέλα(Splicing Systems) τα οποία πάντα αποδεικνύεται ότι είναι υπολογιστικά ισοδύναμα με μηχανές Turing.

1.1 Μηχανές Turing και RE γλώσσες

Ορισμός 1.1.1

Μια *μηχανή RAM* αποτελείται γενικά από ένα *πρόγραμμα* που δρα σε μια *δομή δεδομένων*. Η *δομή δεδομένων* της μηχανής RAM είναι ένας πίνακας από καταχωρητές με τον κάθε καταχωρητή να είναι ικανός να κρατά ένα μεγάλο ακέραιο αριθμό (θετικό η αρνητικό). Το *πρόγραμμα* της RAM αποτελείται από στοιχειώδεις *εντολές* οι οποίες επιτελούν διάφορες λειτουργίες(LOAD,STORE,READ,ADD,SUB) επί των καταχωρητών, ή αλλάζουν την σειρά με την οποία εκτελείται το πρόγραμμα(εντολές JUMP,JZERO,JNEG,JPOS) ανάλογα με το αν ικανοποιείται μια συνθήκη(η τιμή ενός καταχωρητή να είναι 0 αρνητική η θετική).

Είναι κατανοητό ότι αυτό το μοντέλο είναι απολύτως «εφαρμοστό» στους ψηφιακούς ηλεκτρονικούς υπολογιστές. Οι μηχανές Turing, αν και υπολογιστικά ισοδύναμες με τις RAM έχουν τελείως διαφορετική δομή(ένα string από σύμβολα) και λειτουργικότητα: υπάρχει ένας δρομέας ο οποίος κινείται δεξιά η αριστερά πάνω στο string, μπορεί να διαβάσει το σύμβολο της συγκεκριμένης θέσης ή και να αντικαταστήσει(γράψει) αυτό με ένα άλλο σύμβολο. Παρόλο που η μηχανή Turing είναι τόσο απλή, είναι ικανή να εκτελέσει οποιοδήποτε αλγόριθμο, να προσομοιώσει μια γλώσσα προγραμματισμού και όπως ειπώθηκε μπορεί να κάνει ότι και το μοντέλο RAM.

Ορισμός 1.1.2

Μια *μηχανή Turing* είναι μια διατεταγμένη τετράδα $M = (K, \Sigma, \delta, s)$, όπου $:K$ είναι ένα πεπερασμένο σύνολο των *καταστάσεων*(μπορούν να ειδωθούν σαν τις εντολές που υποστηρίζει η μηχανή) και $s \in K$ είναι η αρχική κατάσταση. Σ είναι ένα πεπερασμένο σύνολο από *σύμβολα*(λέμε ότι το Σ είναι το αλφάβητο της M). Υποθέτουμε ότι $K \cap \Sigma = \varnothing$. Το Σ πάντα περιέχει τα σύμβολα $\triangleright$ και $\perp$. Το πιο σημαντικό ίσως είναι η συνάρτηση μετάβασης δ, που αντιστοιχίζει το καρτεσιανό

γινόμενο $K \times \Sigma$ στο σύνολο $(K \cup \{h, "yes", "no"\}) \times \Sigma \times \{\rightarrow, \leftarrow, -\}$. Υποθέτουμε ότι το h (η κατάσταση Halt), το "yes" (η κατάσταση αποδοχής) το "no" (κατάσταση απόρριψης), καθώς και οι κατευθύνσεις κίνησης του δρομέα (δεξιά, αριστερά και παραμονή στο ίδιο σημείο) δεν ανήκουν στο $K \cup \Sigma$.

Όπως φαίνεται η συνάρτηση δ είναι κατά μια έννοια το πρόγραμμα της μηχανής Turing.Καθορίζει, για κάθε συνδυασμό τρέχουσας κατάστασης $q \in K$ και συμβόλου του δρομέα $\sigma \in \Sigma$, μια τριάδα $\delta(q, \sigma) = (p, \rho, D)$ όπου $p \in K$ είναι η επόμενη κατάσταση, ρ το σύμβολο που θα αντικαταστήσει το σ και $D \in \{\rightarrow, \leftarrow, -\}$ είναι η κατεύθυνση προς την οποία θα κινηθεί ο δρομέας. Επίσης απαιτούμε $\delta(q, \triangleright) = (p, \triangleright, \rightarrow)$ δηλαδή το $\triangleright$ κατευθύνει τον δρομέα προς τα δεξιά και δεν αλλάζει.

Η μηχανή λειτουργεί ως εξής: αρχικά η κατάσταση της είναι s. Το string αρχικοποιείται σε κάποιο πεπερασμένου μήκους $x \in (\Sigma - \perp)^*$ όπου Σ^* είναι το σύνολο των strings με αλφάβητο το Σ.Μπροστά από το x τοποθετείται το $\triangleright$ και ο δρομέας τοποθετείται στο $\triangleright$. Λέμε ότι το x είναι η είσοδος στη μηχανή Turing. Στην συνέχεια η μηχανή αλλάζει σε μια κατάσταση p και κινείται δεξιά, σύμφωνα με την πιο πάνω απαίτηση. Κατόπιν διαβάζει το πρώτο σύμβολο σ_1 του x και ανάλογα με την τιμή $\delta(p, \sigma_1) = (p_1, \rho_1, D_1)$ αλλάζει σε μια κατάσταση p_1, αντικαθιστά το σ_1 με το ρ_1 και στην συνέχεια ο δρομέας κινείται κατά την κατεύθυνση D_1. Στην συνέχεια γίνεται ξανά το ίδιο, δηλαδή διαβάζεται το σύμβολο στη νέα θέση(που μπορεί να είναι η ίδια αν π.χ $D_1 = -$) και η μηχανή πράττει ανάλογα με την τρέχουσα κατάσταση της και την δ.Η μηχανή Turing μπορεί να επεκτείνει το αρχικό string αλλά όχι να το μικρύνει, δηλαδή να σβήσει κάποια σύμβολα του. Αυτό είναι ένα βασικό χαρακτηριστικό της μηχανής Turing. Η επέκταση του string γίνεται όταν ο δρομέας φτάσει στο τέλος του string και η δ αποφασίζει να κινηθεί δεξιά οπότε θεωρούμε ότι ο δρομέας πάει σε ένα σύμβολο $\perp$. Αυτό το σύμβολο μπορεί να αντικατασταθεί από κάποιο άλλο ανάλογα με την τιμή $\delta(p_i, \perp)$ όπου $p_i \in K$ η κατάσταση της μηχανής όταν ο δρομέας βρίσκεται στο $\perp$.

Η μηχανή σταματάει όταν η τρέχουσα κατάσταση της γίνει μια εκ των h, yes, no. Αν συμβεί αυτό λέμε ότι η μηχανή έχει τερματίσει. Αν η τερματική κατάσταση είναι yes τότε λέμε ότι η μηχανή «δέχεται» την είσοδο. Αν είναι no τότε την «απορρίπτει».Όταν η μηχανή τερματίζει τότε μπορούμε να ορίσουμε την έξοδο της μηχανής ως $M(x) = yes$ η $M(x) = no$ ανάλογα με την τερματική κατάσταση. Αν η τερματική κατάσταση είναι h τότε $M(x) =$το string της μηχανής . Αν η μηχανή δεν φτάσει μια τερματική κατάσταση τότε δεν θα σταματήσει πότε οπότε γράφουμε $M(x) = \uparrow$.

Μια *περιγραφή* της μηχανής είναι μια τριάδα (p, w, u) όπου p η τρέχουσα κατάσταση, w το string αριστερά του δρομέα(μαζί με το σύμβολο που αυτός δείχνει), u το string δεξιά του δρομέα. Λέμε ότι η *περιγραφή* (p, w, u) δίδει την *περιγραφή* (q, w', u') σε ένα βήμα και γράφουμε $(p, w, u) \overset{1}{\rightarrow} (q, w', u')$ όταν: (ι) $\delta(p, \sigma) = (q, \rho, D)$ όπου σ το σύμβολο στην τρέχουσα θέση του δρομέα, και (ιι) Αν $D = \rightarrow$ τότε το w' θα πρέπει να είναι το w με το τελευταίο σύμβολο του αλλαγμένο σε ρ και στην συνέχεια το πρώτο σύμβολο του u. Επίσης το u' θα πρέπει να είναι το u μείον το αρχικό του σύμβολο. Αν $D = \leftarrow$ τότε το w' θα πρέπει να είναι το w

μείον το τελευταίο του σύμβολο και το u' θα πρέπει να είναι το ρ ακολουθούμενο από το u.

Αυτός ο ορισμός μπορεί να επεκταθεί ώστε να μιλάμε για πολλά βήματα(η μεταβατική κλειστότητα του $\overset{1}{\rightarrow}$) δηλαδή, $(p,w,u)\overset{k}{\rightarrow}(q,w',u')$ όταν έχουμε k ακριβώς βήματα ή να λέμε απλώς ότι, η ότι η *περιγραφή* (p,w,u) δίδει την *περιγραφή* (q,w',u'), συμβολικά $(p,w,u)\overset{*}{\rightarrow}(q,w',u')$ όταν υπάρχει ένας αριθμός βημάτων k έτσι ώστε $(p,w,u)\overset{k}{\rightarrow}(q,w',u')$.

Όπως φαίνεται οι μηχανές Turing είναι ιδανικές για να αντιμετωπίζουν προβλήματα που σχετίζονται με strings, όπως το να υπολογίζουν συναρτήσεις από strings σε strings, και να *δέχονται* ή και να *αποφασίζουν* γλώσσες.

Ορισμός 1.1.3

Έστω $L\subset(\Sigma-\{\perp\})^*$ μια γλώσσα δηλαδή ένα σύνολο από strings με σύμβολα από το Σ. Έστω M μια μηχανή Turing τέτοια ώστε, για κάθε string $x\in(\Sigma-\{\perp\})^*$ αν το $x\in L$ τότε $M(x)=yes$ (δηλαδή η M με είσοδο x τερματίζει στην κατάσταση yes), και αν $x\notin L$ τότε $M(x)=no$. Τότε λέμε ότι η M *αποφασίζει* την L. Επίσης τότε η L θα λέγεται *αναδρομική γλώσσα*. Θα λέμε ότι η M *δέχεται* την L αν για κάθε $x\in(\Sigma-\{\perp\})^*$ αν $x\in L$ τότε $M(x)=yes$ ενώ αν $x\notin L$ τότε $M(x)=\uparrow$. Σε αυτή την περίπτωση η L θα λέγεται *αναδρομικά απαριθμητή* γλώσσα. Επίσης η μια μηχανή Turing M υπολογίζει μια συνάρτηση $f:(\Sigma-\{\perp\})^*\rightarrow\Sigma^*$ αν για κάθε $x\in(\Sigma-\{\perp\})^*$ είναι $M(x)=f(x)$. Αν για μια συνάρτηση f υπάρχει μια τέτοια μηχανή τότε η f λέγεται *αναδρομική* συνάρτηση.

Οι αναδρομικά απαριθμητές(Recursively Enumerable ή απλά RE) γλώσσες είναι κατά κάποιο τρόπο αυτές που εκφράζουν την υπολογιστική ικανότητα στις μηχανές Turing. Ένα άλλο θεωρητικό μοντέλο υπολογισμού για να αποδειχθεί ισοδύναμο με τις μηχανές Turing θα πρέπει να δειχθεί ότι δέχεται ή παράγει τις RE γλώσσες. Οι recursive γλώσσες είναι και αυτές RE όπως διατυπώνεται στην επόμενη πρόταση:

Πρόταση 1.1.1. Αν μια γλώσσα L είναι αναδρομική τότε είναι και αναδρομικά απαριθμητή.

Απόδειξη. Έστω ότι υπάρχει μια μηχανή M που αποφασίζει την L(αφού L αναδρομική) . Θα κατασκευαστεί μια μηχανή M' που να δέχεται την L. Η M' λειτουργεί ακριβώς όπως η M εκτός από το όταν η M πάει να περάσει στην κατάσταση no τότε η M' κινεί τον δρομέα προς τα δεξιά και δεν τερματίζει πότε. Είναι εύκολο να τροποποιηθεί η συνάρτηση μετάβασης της M ώστε να επιτευχθεί κάτι τέτοιο.

Όπως φαίνεται οι μηχανές Turing μπορούν να αντιμετωπίζουν προβλήματα που αφορούν strings. Αλλά πολλά ενδιαφέροντα προβλήματα αναφέρονται σε γραφήματα, δίκτυα, αριθμούς κ.τ.λ. Για να λύσει ένα τέτοιο πρόβλημα μια μηχανή Turing θα πρέπει να βρεθεί ένας τρόπος ώστε να αναπαρασταθεί(κωδικοποιηθεί) ως string, οποιοδήποτε στιγμιότυπο ενός τέτοιου προβλήματος. Από την στιγμή που έχει βρεθεί η αναπαράσταση, ένας αλγόριθμος για ένα πρόβλημα απόφασης(δηλαδή πρόβλημα στο οποίο η απάντηση θα είναι είτε yes είτε no, για παράδειγμα αν ένα γράφημα είναι συνεκτικό) είναι απλά μια μηχανή Turing που αποφασίζει την γλώσσα που ορίζουν τα string που αναπαριστούν στιγμιότυπα του προβλήματος . Παρόμοια, προβλήματα που απαιτούν πιο πολύπλοκη έξοδο όπως για παράδειγμα το πρόβλημα εύρεσης της μέγιστης ροής σε ένα γράφημα, λύνονται από μια μηχανή Turing που υπολογίζει την κατάλληλη συνάρτηση από strings σε strings όπου για παράδειγμα το string της

εξόδου θα αναπαριστά την μέγιστη ροή του γραφήματος που αναπαριστά το string της εισόδου.
Είναι δυνατόν να υπάρχουν πολλές αναπαραστάσεις για στιγμιότυπα του ίδιου προβλήματος. Θα πρέπει να επιλεχθούν «λογικές» αναπαραστάσεις για να αναπαραστήσουν το στιγμιότυπο του προβλήματος. Δύο αναπαραστάσεις Α και Β είναι λογικές, αν η αναπαράσταση ενός στιγμιότυπου με την Α είναι ένα string με n σύμβολα τότε χρησιμοποιώντας την Β θα πρέπει να έχουμε ένα string με το πολύ $p(n)$ μήκος για κάποιο πολυώνυμο p. Έτσι για παράδειγμα η αναπαράσταση του αριθμού n, χρησιμοποιώντας n το πλήθος 1 δεν είναι αποδεκτή, γιατί χρησιμοποιώντας 0 και 1 μπορούμε να αναπαραστήσουμε το n με $\log_2 n$ ψηφία. Ο λόγος που απαιτούμε «λογικές» αναπαραστάσεις είναι για να έχουμε πολυωνυμική διατήρηση της χρονικής η χωρικής πολυπλοκότητας του αλγορίθμου ή της μηχανής Turing που λύνει το πρόβλημα.

Για να μιλήσουμε για χώρο και χρόνο που ξοδεύουν οι μηχανές Turing κρίνεται σκόπιμο να οριστούν μηχανές Turing με περισσότερα του ενός string, δηλαδή k-string Turing machines.

Ορισμός 1.1.4 Μια $k-string\ k \geq 1, k \in N$ μηχανή Turing είναι μια τετράδα $M = (K, \Sigma, \delta, s)$. Τα $K, \Sigma, s \in K$ ορίζονται ακριβώς όπως και στον ορισμό 1.1.1. Τώρα όμως η συνάρτηση μετάβασης δ πρέπει να αντανακλά ένα πρόγραμμα που λαμβάνει υπόψη τα πολλά strings. Διαισθητικά η δ αποφασίζει την επόμενη κατάσταση, όπως και πριν, αλλά το κάνει αυτό κοιτάζοντας τα σύμβολα που υπάρχουν σε κάθε δρομέα του κάθε ενός string και στην συνέχεια αποφασίζει για κάθε δρομέα τι θα γράψει και προς τα πού θα κινηθεί. Έτσι η δ είναι μια συνάρτηση από το $K \times \Sigma^k$ στο $(K \cup \{yes, no, h\}) \times (\Sigma \times \{\rightarrow, \leftarrow, -\}^k)$. Δηλαδή $\delta(q, \sigma_1, ..., \sigma k) = (p, \rho_1, D_1, ..., \rho_k, D_k)$ σημαίνει ότι αν η Μ βρίσκεται στην κατάσταση q και ο δρομέας στο πρώτο string βρίσκεται στο σύμβολο σ_1, αυτός του δεύτερου string βρίσκεται στο σ_2, κ.ο.κ, τότε η επόμενη κατάσταση θα είναι p και ο πρώτος δρομέας θα γράψει ρ_1 και θα κινηθεί κατά την D_1, κ.ο.κ για τους υπόλοιπους δρομείς. Το αποτέλεσμα $M(x)$ ορίζεται όπως και στις απλές μηχανές με την διαφορά ότι αν η Μ υπολογίζει συνάρτηση, τότε ως $M(x)$ είναι η συμβολοσειρά που βρίσκεται στο $k-$οστό string.

Μια *περιγραφή* για μια μηχανή με $k-string$ ορίζεται ως $(q, w_1, u_1, ..., w_k, u_k)$ όπου τα w_i, u_i αναφέρονται σε κάθε ένα string . Ανάλογα ορίζονται και οι σχέσεις της μετάβασης ανάμεσα σε περιγραφές $(q, w_1, u_1, ..., w_k, u_k) \overset{1}{\rightarrow} (q', w_1', u_1', ..., w_k', u_k')$ και για πολλά βήματα θα έχουμε την μετάβαση $(q, w_1, u_1, ..., w_k, u_k) \overset{k}{\rightarrow} (q', w_1', u_1', ..., w_k', u_k')$ και $(q, w_1, u_1, ..., w_k, u_k) \overset{*}{\rightarrow} (q', w_1', u_1', ..., w_k', u_k')$ αν υπάρχει ένας συγκεκριμένος αριθμός k έτσι ώστε $(q, w_1, u_1, ..., w_k, u_k) \overset{k}{\rightarrow} (q', w_1', u_1', ..., w_k', u_k')$.

Μια $k-string$ μηχανή ξεκινάει από την περιγραφή $(s, \triangleright, x, \triangleright, e, ..., \triangleright, e)$ όπου e το κενό string. Αν $(s, \triangleright, x, \triangleright, e, ..., \triangleright, e) \overset{*}{\rightarrow} (yes, w_1, u_1, ..., w_k, u_k)$ τότε $M(x) = yes$. Αν δε ισχύει $(s, \triangleright, x, \triangleright, e, ..., \triangleright, e) \overset{*}{\rightarrow} (no, w_1, u_1, ..., w_k, u_k)$ τότε $M(x) = no$. Αν ισχύει

$(s,\triangleright,x,\triangleright,e,...,\triangleright,e) \overset{*}{\rightarrow} (h,w_1,u_1,...,w_k,u_k)$ τότε $M(x)=w_k u_k$ δηλαδή η συνένωση των w_k,u_k.

Ορισμός 1.1.5 Αν για μια k-string μηχανή Turing M με είσοδο x ισχύει $(s,\triangleright,x,\triangleright,e,...,\triangleright,e) \overset{t}{\rightarrow} (H,w_1,u_1,...,w_k,u_k)$ για $H \in \{h,yes,no\}$ τότε ο *χρόνος που απαιτείται* από την M στην είσοδο x, είναι t. Δηλαδή ο χρόνος που απαιτείται είναι απλά ο αριθμός των βημάτων για να φτάσει η μηχανή σε μια τερματική κατάσταση. Αν $M(x)=\uparrow$ τότε ο χρόνος που απαιτείται για το x είναι ∞. Έστω τώρα $f: N \rightarrow N$. Θα λέμε ότι η μηχανή λειτουργεί σε χρόνο $f(n)$ αν για κάθε είσοδο x ο χρόνος που απαιτείται από την M στο x είναι το πολύ $f(|x|)$ όπου $|x| \in N$ είναι το μήκος του string x.

Έστω τώρα ότι μια γλώσσα L είναι αναδρομική και αποφασίζεται από μια μηχανή M που λειτουργεί σε χρόνο $f(n)$. Θα λέμε ότι $L \in TIME(f(n))$. Άρα $TIME(f(n))$ είναι ένα σύνολο από γλώσσες. Περιέχει ακριβώς εκείνες τις γλώσσες οι οποίες αποφασίζονται από μηχανές Turing με πολλά string που λειτουργούν σε χρόνο $f(n)$.

$TIME(f(n))$ είναι αυτό που λέμε μια *κλάση πολυπλοκότητας*. Είναι ένα σύνολο από γλώσσες, πιθανότατα πολλές από αυτές να αναπαριστούν σημαντικά προβλήματα απόφασης. Η κοινή ιδιότητα σε αυτές τις γλώσσες, είναι ότι μπορούν να αποφασιστούν εντός ενός συγκεκριμένου ορίου που αφορά κάποια πλευρά απόδοσης(συγκεκριμένα χρόνος, αλλά θα δούμε και κλάσεις πολυπλοκότητας για χώρο).

Το παρακάτω θεώρημα που δίνεται χωρίς απόδειξη μας λεει δύο πράγματα: α) ότι οι multistring μηχανές Turing δεν έχουν επιπλέον δυνατότητες, όσο αφορά τις γλώσσες που αποφασίζουν ή δέχονται, από τις απλές μηχανές Turing. β) Όσο αφορά την απόδοση από πλευράς χρόνου οι multistring μηχανές Turing υπερτερούν μόνο κατά ένα τετραγωνικό παράγοντα.

Θεώρημα 1.1.1 Δοθείσας μια k-string μηχανής Turing M που λειτουργεί σε χρόνο $f(n)$, μπορεί να κατασκευαστεί μια απλή μηχανή Turing M' που λειτουργεί σε χρόνο $O(f(n)^2)$ και τέτοια ώστε για κάθε είσοδο $x, M(x)=M'(x)$.

$O(f(n)^2)$ σημαίνει ότι λειτουργεί σε χρόνο μικρότερο η ίσο του $c \cdot f(n)^2$ για κάποια σταθερά c η οποία εξαρτάται από την M. Αυτή η σταθερά δεν έχει καμία απολύτως σημασία λόγω του παρακάτω θεωρήματος (LINEAR SPEEDUP):

Θεώρημα 1.1.2 Έστω $L \in TIME(f(n))$. Τότε για κάθε $\varepsilon > 0$, $L \in TIME(f'(n))$ όπου $f(n)=\varepsilon \cdot f(n)+n+2$.

Αν τώρα μια γλώσσα $L \in TIME(p(n))$ όπου $p(n)$ πολυώνυμο βαθμού k είναι το ίδιο με το να πούμε ότι η γλώσσα ανήκει στη κλάση $TIME(n^k)$, λόγω του θεωρήματος 1.1.2 και επειδή $c \cdot n^k \geq p(n)$ για κατάλληλη σταθερά c. Άρα οι γλώσσες που είναι *πολυωνυμικά αποφασίσιμες* ανήκουν στην κλάση $P=\bigcup_k TIME(n^k)$. Τα προβλήματα που αναπαριστούν αυτές οι γλώσσες αντιμετωπίζονται αποτελεσματικά από τους ψηφιακούς ηλεκτρονικούς υπολογιστές. Εντούτοις τα προβλήματα που ανήκουν στη κλάση NP (οι γλώσσες που αποφασίζονται από *μη ντετερμινιστικές μηχανές Turing* που λειτουργούν σε πολυωνυμικό χρόνο) είναι αυτά που προβληματίζουν τους ηλεκτρονικούς

υπολογιστές, και ακόμη και ένας ταχύτατος Η/Υ μπορεί να απαιτήσει χιλιάδες χρόνια για να λύσει ένα μέτριο(από πλευράς μεγέθους) στιγμιότυπο ενός προβλήματος που αναπαριστάται από γλώσσα που ανήκει στη κλάση NP. Παρακάτω συνεχίζουμε με ορισμό για κλάσεις πολυπλοκότητας χώρου και στην 1.2 παρουσιάζεται η κλάση NP.

Ορισμός 1.1.6 Μια k-string μηχανή Turing *με είσοδο και έξοδο* είναι μια συνήθης μηχανή Turing με k-string με έναν σημαντικό περιορισμό στο πρόγραμμα της δ: αν $\delta(q,\sigma_1,...,\sigma_k)=(p,\rho_1,D_1,...,\rho_k,D_k)$ τότε α) $\sigma_1=\rho_1$ και β) $D_k\neq\leftarrow$. Το α) απλά λεει ότι το πρώτο string δεν αλλάζει και άρα μπορεί να θεωρηθεί ως read only και input string. Το b) λεει ότι το τελευταίο string ο δρομέας δεν κινείται ποτέ προς τα δεξιά και άρα είναι write-only και μπορεί να θεωρηθεί ως output string.

Πρόταση 1.1.2 Για κάθε k-string μηχανή Turing M που λειτουργεί σε χρόνο $f(n)$ υπάρχει μια k+2-string μηχανή Turing M' με είσοδο και έξοδο, που λειτουργεί σε χρόνο $O(f(n))$ και $M(x)=M'(x)$.

Απόδειξη. Η M' αντιγράφει το input string της στο δεύτερο της string και στη συνέχεια εξομοιώνει την M χρησιμοποιώντας τα k-string της από το 2 ως το k+1. Στη συνέχεια αντιγράφει στο k+2 string(output string) το k+1 string της, και τερματίζει. Ο χρόνος που απαιτεί η M' για είσοδο μήκους n, είναι $n+f(n)+f(n)=O(f(n))$. Αυτό γιατί το μήκος του τελευταίου string της M (και άρα του k+1 της M') δεν μπορεί να ξεπεράσει το $f(n)$ αφού η μηχανή M κάνει το πολύ $f(n)$ βήματα και σε κάθε βήμα μπορεί να επεκτείνει το string το πολύ κατά 1. Αυτό μας ωθεί σε μια πιο γενική παρατήρηση: μια μηχανή δε μπορεί να απαιτήσει περισσότερο χώρο από ότι χρόνο.

Ορισμός 1.1.7 Έστω ότι για μια k-string μηχανή Turing με είσοδο x έχουμε ότι $(s,\triangleright,x,\triangleright,e,...,\triangleright,e)\overset{*}{\rightarrow}(H,w_1,u_1,...,w_k,u_k)$ με $H\in\{h,yes,no\}$ δηλαδή η μηχανή τερματίζει. Τότε ο *χώρος που απαιτείται από τη M με είσοδο* x είναι $\sum_{i=1}^{k}|w_iu_i|$. Αν η *M είναι μηχανή με είσοδο και έξοδο*, τότε ο χρόνος που απαιτείται από την M είναι $\sum_{i=2}^{k-1}|w_iu_i|$. Έστω τώρα $f:N\rightarrow N$. Λέμε ότι η M λειτουργεί σε χώρο $f(n)$ *αν για κάθε είσοδο* x απαιτεί χώρο το πολύ $f(|x|)$.

Έστω L μια γλώσσα. Τότε L είναι στη κλάση πολυπλοκότητα $SPACE(f(n))$ αν υπάρχει μια μηχανή Turing με είσοδο και έξοδο, που αποφασίζει την L, και λειτουργεί σε χώρο $f(n)$.

Ένα ανάλογο θεώρημα με το 1.1.2 λεει ότι και στη περίπτωση του χώρου οι σταθερές δεν παίζουν σημαντικό ρόλο:

Θεώρημα 1.1.3 Έστω $L\in SPACE(f(n))$. Τότε $L\in SPACE(\varepsilon f(n)+2)$ για $\varepsilon>0$.

Τελειώνοντας αυτή τη παράγραφο αναφερόμαστε στην υπολογιστική ισοδυναμία των μηχανών RAM με τις μηχανές Turing. Αυτή η ισοδυναμία δίνεται από τα ακόλουθα δύο θεωρήματα:

Θεώρημα 1.1.4 Έστω $L\in TIME(f(n))$. Τότε υπάρχει μια μηχανή RAM που υπολογίζει την χαρακτηριστική συνάρτηση φ_L της L, σε χρόνο $O(f(n))$.

Θεώρημα 1.1.5 Έστω ότι μια μηχανή RAM, Π ,υπολογίζει μια συνάρτηση φ σε χρόνο $f(n)$. Τότε υπάρχει μια 7-string μηχανή Turing που υπολογίζει την φ σε χρόνο $O(f(n)^3)$.

1.2 Οι κλάσεις NP,NP-complete και NP-hard.

Για να μιλήσουμε για τη κλάση NP καταρχήν, θα πρέπει να οριστούν οι *μή ντετερμινιστικές μηχανές Turing.*

Ορισμός 1.2.1 *Μια μή ντεντερμινιστική μηχανή Turing N είναι* μια τετράδα $N=(K,\Sigma,\Delta,s)$. Τα $K,\Sigma,s\in K$ είναι όπως και στις απλές(ντετερμινιστικές) μηχανές Turing. Η Δ δεν είναι τώρα συνάρτηση, αλλά σχέση δηλαδή είναι $\Delta\subseteq(K\times\Sigma)\times[(K\cup\{h,"yes","no"\})\times\Sigma\times\{\rightarrow,\leftarrow,-\}]$. Αυτό σημαίνει ότι για κάθε συνδυασμό κατάστασης-συμβόλου δρομέα, υπάρχουν μία, περισσότερες από μια επιλογές ή και καμία.

Η *περιγραφή* μια μη ντετερμινιστικής μηχανής είναι πάλι μια τριάδα (q,w,u) με την διαφορά ότι η μετάβαση από την μια περιγραφή στην άλλη δεν είναι πια συνάρτηση αλλά σχέση. Δηλαδή μπορεί $(q,w,u)\overset{1}{\rightarrow}(q',w',u')$ αλλά και $(q,w,u)\overset{1}{\rightarrow}(q'',w'',u'')$ καθότι δύναται να είναι $((q,\sigma),(q',\rho,D))\in\Delta$ και $((q,\sigma),(q'',\rho'',D''))\in\Delta$. Ανάλογα ισχύουν και για τις $\overset{k}{\rightarrow}$ και $\overset{*}{\rightarrow}$.

Ορισμός 1.2.2 Μια μη-ντετερμινιστική μηχανή Turing N αποφασίζει μια γλώσσα L αν για κάθε $x\in\Sigma^*$ ισχύει : $x\in L\Leftrightarrow(s,\triangleright,x)\overset{*}{\rightarrow}(yes,w,u)$ για κάποια w,u.

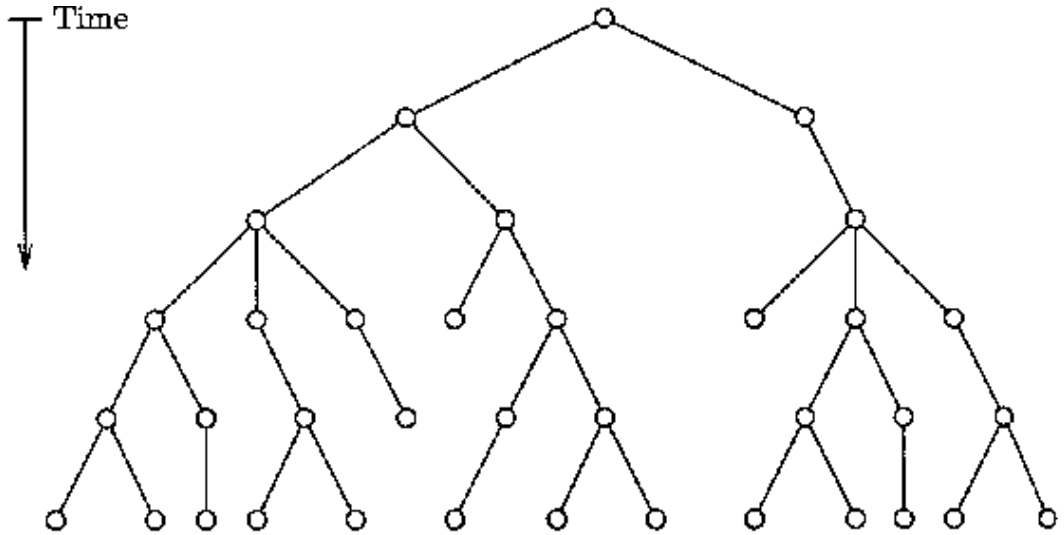
Αυτός ο ορισμός σημαίνει ότι η N αποφασίζει την L αν υπάρχει τουλάχιστον μια ακολουθία από μη ντετερμινιστικές επιλογές(όπως στον ορισμό 1.2.1) που να οδηγεί στη τερματική κατάσταση yes. Άλλες επιλογές μπορεί να οδηγούν σε no, φτάνει να υπάρχει μια ακολουθία επιλογών που να οδηγεί σε yes.

Ορισμός 1.2.3 Μια μη-ντετερμινιστική μηχανή Turing N αποφασίζει μια γλώσσα L σε χρόνο $f(n)$, αν η N αποφασίζει την L και για κάθε $x\in\Sigma^*$, αν $(s,\triangleright,x)\overset{k}{\rightarrow}(q,u,w)$ τότε $k\le f(|x|)$. Δηλαδή η N δεν έχει μονοπάτια υπολογισμού(κάθε μονοπάτι αντιστοιχεί σε μια ακολουθία επιλογών, όπως φαίνεται στο σχήμα 1) με μήκος μεγαλύτερο από $f(n)$ όπου n το μήκος της εισόδου. Ο χρόνος που χρεώνεται είναι το ύψος του δέντρου που αναπαριστά όλα τα δυνατά μονοπάτια υπολογισμού(σχήμα 1). Προφανώς η συνολική υπολογιστική δραστηριότητα είναι το άθροισμα των μηκών των μονοπατιών, από την ρίζα προς τα φύλλα και είναι εκθετικά μεγάλος ως προς το μήκος της εισόδου.

Το σύνολο των γλωσσών που αποφασίζονται από μη ντετερμινιστικές μηχανές Turing σε χρόνο $f(n)$ είναι η κλάση $NTIME(f(n))$.

Η κλάση NP είναι $NP=\bigcup_k NTIME(n^k)$, δηλαδή οι γλώσσες που αποφασίζονται από μη ντετερμινιστικές μηχανές πολυωνυμικού χρόνου. Παρατηρούμε ότι $P\subseteq NP$ διότι κάθε ντετερμινιστική μηχανή είναι ειδική περίπτωση μη ντετερμινιστικής μηχανής: Η σχέση Δ είναι συνάρτηση. Άρα TIME$(f(n))\subseteq NTIME(f(n))$ για κάθε $f(n)$ και πιο ειδικά για $f(n)=n^k$.

Το παρακάτω θεώρημα μας λεει ότι αν θέλουμε να αποφασίζουμε ντετερμινιστικά, μια γλώσσα $L \in NP$, τότε είμαστε υποχρεωμένοι να εξερευνήσουμε όλο το δέντρο υπολογισμού στο σχήμα 1, και συνεπώς να δαπανήσουμε εκθετικό χρόνο(ως προς το μήκος της εισόδου).



Σχήμα 1 Μή ντεντερμινιστικός υπολογισμός

Θεώρημα 1.2.1. Έστω μια γλώσσα L που αποφασίζεται από μια μη ντετερμινιστική μηχανή Turing N σε χρόνο $f(n)$. Τότε η L αποφασίζεται από μια (ντετερμινιστική) 3-string μηχανή Turing M σε χρόνο $O(c^{f(n)})$ όπου $c > 1$ μια σταθερά που εξαρτάται από τη μηχανή N.
Από αυτό το θεώρημα προκύπτει αμέσως ότι $NTIME(f(n)) \subseteq \bigcup_{c>1} TIME(c^{f(n)})$.

Προβλήματα ενδεικτικά της κλάσης NP είναι το SAT, το TSP(στο οποίο μας δίνεται ένα σύνολο πόλεων και οι μεταξύ τους αποστάσεις σε μορφή κόστους, και θέλουμε να βρούμε την διαδρομή ελαχίστου κόστους, που περνάει μόνο μια φορά από κάθε πόλη). Επίσης στο NP ανήκουν όλα τα προβλήματα που ανήκουν στο P, όπως για παράδειγμα το CIRCUIT-VALUE(μας δίνεται ένα λογικό κύκλωμα, και η είσοδος του, και πρέπει να υπολογιστεί η τιμή της εξόδου του κυκλώματος για τη δοσμένη είσοδο). Τα προβλήματα του P δεν είναι ενδεικτικά της δυσκολία της κλάσης NP, γιατί λύνονται σε πολυωνυμικό χρόνο, σε αντίθεση με το SAT και το TSP που αιχμαλωτίζουν την δυσκολία της κλάσης NP. Αυτά τα προβλήματα ανήκουν στη κλάση NP-complete, που είναι εκείνο το υποσύνολο της κλάσης NP, το οποίο έχει τα προβλήματα εκείνα, σε κάθε ένα από τα οποία ανάγονται όλα τα προβλήματα της κλάσης. Επίσης ανήκουν στη κλάση NP-hard, η οποία έχει όλα τα προβλήματα(ανεξαρτήτου κλάσης) σε κάθε ένα από τα οποία ανάγονται όλα τα προβλήματα της κλάσης NP. Όταν μιλάμε για αναγωγή εννοούμε το εξής:

Ορισμός 1.2.4. Μια γλώσσα L_1 *ανάγεται* L_2 αν υπάρχει μια συνάρτηση R, από strings σε strings, που υπολογίζεται από μια ντετερμινιστική μηχανή Turing σε χώρο $O(\log n)$, έτσι ώστε για όλες τις εισόδους x να ισχύει: $x \in L_1 \Leftrightarrow R(x) \in L_2$. Η R είναι η *αναγωγή* από την L_1 στην L_2.

Πρόταση 1.2.1.Αν R μια αναγωγή που υπολογίζεται από μια μηχανή Turing M, τότε για όλα τα input x η M τερματίζει μετά από πολυωνυμικό αριθμό βημάτων.

Απόδειξη. Έστω $n = |x|$. Υποθέτοντας ότι η M έχει τρία string(ένα εισόδου ένα εξόδου και ένα work tape), οι δυνατές περιγραφές της(configuration) υπολογίζοντας μόνο το work tape που δεν μπορεί να έχει μήκος μεγαλύτερο από $\log n$, είναι $c_1 \cdot n \cdot c^{\log n}$ γιατί n είναι οι δυνατές θέσεις του δρομέα πάνω στο input string, και $c^{\log n}$ είναι όλες οι δυνατές λέξεις που σχηματίζονται σε ένα tape μήκους $\log n$ όπου c είναι το πλήθος των γραμμάτων του αλφαβήτου της μηχανής και c_1 ο αριθμός των καταστάσεων της μηχανής. Επειδή η μηχανή είναι ντετερμινιστική, κατά την πορεία ενός υπολογισμού δεν μπορεί να επαναληφτεί μια περιγραφή, άρα το μήκος ενός υπολογισμού είναι το πολύ $c_1 \cdot n \cdot c^{\log n}$ το οποίο είναι μικρότερο η ίσο του n^k για κατάλληλο k (ανεξάρτητο του n).

Παραδείγματα αναγωγών είναι του Hamilton Path προς το SAT, και τα δύο από τα οποία τα συναντήσαμε στο μέρος I. Επίσης υπάρχει αναγωγή του SAT προς το Hamilton Path. Το SAT όμως είναι η μάνα όλων των NP-complete προβλημάτων, με την έννοια ότι αν θέλουμε να αποδείξουμε ότι ένα πρόβλημα είναι NP-complete, ανάγουμε το SAT στο πρόβλημα. Άρα και το Hamilton Path είναι NP-complete.

Κλείνοντας αυτό το κεφάλαιο θα πρέπει να αναφερθεί συνοπτικά η έννοια της *Universal (καθολικής) Μηχανής Turing*. Είναι μια μηχανή Turing $U(w,x)$, η οποία δέχεται ως είσοδο ένα string $w(M)$, που κωδικοποιεί μια οποιαδήποτε μηχανή Turing M, και ένα string x, και στη συνέχει εξομοιώνει τον υπολογισμό της μηχανής M με είσοδο x. Είναι δηλαδή $U(w(M),x) = M(x)$ για κάθε x. Αποδεικνύεται στο [12] η ύπαρξη της καθολικής μηχανής Turing.

2. Γραμματικές Chomsky

Οι γραμματικές Chomsky είναι μηχανισμοί που παράγουν strings, συνεπώς παράγουν γλώσσες. Οι γλώσσες που παράγουν οι γραμματικές Chomsky είναι ακριβώς οι RE γλώσσες, δηλαδή οι γλώσσες που χαρακτηρίζουν την υπολογιστική ισχύ των μηχανών Turing.Οι γραμματικές Chomsky θα παίξουν ένα ενδιάμεσο ρόλο στην ισοδυναμία μεταξύ των μηχανών Turing και των Splicing συστημάτων, που αποτελούν αφαιρετικό μοντέλο των DNA υπολογιστών. Δηλαδή πρώτα θα αποδειχθεί ότι οι γραμματικές Chomsky είναι ισοδύναμες με τις μηχανές Turing, και στη συνέχεια ότι τα splicing συστήματα είναι ισοδύναμα με τις γραμματικές Chomsky. Το συμπέρασμα αυτών των ισοδυναμιών είναι ότι τα splicing συστήματα είναι ισοδύναμα με τις μηχανές Turing.

2.1 Ορισμός Γραμματικών Chomsky

Ορισμός 2.1.1 Μια *γραμματική Chomsky* είναι μια τετράδα $G=(N,T,S,P)$ όπου N,T είναι ξένα αλφάβητα(δηλαδή $N\cap T=\varnothing$), $S\in N$ και $P\subseteq (N\cup T)^* N(N\cup T)\times(N\cup T)^*$, με P πεπερασμένο. Το αλφάβητο N είναι το μη τερματικό αλφάβητο(non-terminal), T είναι το *τερματικό αλφάβητο*, S είναι το αξίωμα, και P είναι το σύνολο των κανόνων παραγωγής της G. Οι κανόνες $(u,v)\in P$, γράφονται ως $u\to v$. Είναι $|u|_N\geq 1$ δηλαδή το μήκος του u ως προς τον αριθμό των συμβόλων από το N είναι τουλάχιστον ένα, ή πιο απλά το αριστερό μέλος του κάθε κανόνα περιέχει τουλάχιστον ένα σύμβολο από το μη τερματικό αλφάβητο.

Με L_1L_2 συμβολίζεται η γλώσσα $\{uv : u\in L_1, v\in L_2\}$.

Για $x,y\in (N\cup T)^*$ είναι $x\Rightarrow_G y$ ανν $x=x_1ux_2$, $y=x_1vx_2$ για κάποια $x_1,x_2\in (N\cup T)^*$ και $u\to v\in P$. Τότε θα λέμε ότι x *παράγει άμεσα* το y. Με $\Rightarrow_G^*$ θα συμβολίζεται η μεταβατική κλειστότητα της σχέσης $\Rightarrow_G$. Δηλαδή είναι $x\Rightarrow_G^* y$ ανν υπάρχουν $k\in N$ και $y_1,...,y_k$ με $x\Rightarrow_G y_1\Rightarrow_G y_2\Rightarrow_G ...y_k\Rightarrow_G y$. Κάθε $w\in (N\cup T)^*$ με $S\Rightarrow_G^* w$ λέγεται *προτασιακός τύπος*.

Η *γλώσσα που παράγεται*, $L(G)$, από την γραμματική G είναι εκείνο το σύνολο των προτασιακών τύπων που περιέχουν μόνο γράμματα του τερματικού αλφαβήτου:

$L(G)=\{x\in T^* \mid S\Rightarrow_G^* x\}$.

Δύο γραμματικές G_1,G_2 λέγονται *ισοδύναμες* αν $L(G_1)-\{\lambda\}=L(G_2)-\{\lambda\}$ όπου λ η κενή λέξη.

Οι γραμματικές, ανάλογα με την μορφή των κανόνων τους κατατάσσονται ως ακολούθως. Μια γραμματική $G=(N,T,S,P)$ καλείται:

- *μονότονη(αυξάνοντος μήκους)*, αν για όλα τα $u\to v\in P$ είναι $|u|\leq|v|$
- *εξαρτημένη από συμφραζόμενα(context sensitive)* αν κάθε $u\to v\in P$ έχει $u=u_1Au_2$ και $v=u_1xu_2$, για $u_1,u_2\in (N\cup T)^*$, $A\in N$ και $x\in (N\cup T)^+=(N\cup T)^*-\{\lambda\}$
- *ελεύθερη από συμφραζόμενα(context free)* αν κάθε $u\to v\in P$ έχει $u\in N$
- *γραμμική(linear)* αν κάθε $u\to v\in P$ έχει $u\in N$ και $v\in T^*NT^*$
- *κανονική(regular)* αν κάθε $u\to v\in P$ έχει $u\in N$ και $v\in T\cup TN\cup\{\lambda\}$.

Μια γραμματική Chomsky, της οποίας οι κανόνες δεν υπόκεινται, κατά ανάγκη, σε μια «φόρμα» περιορισμών, λέγεται *αυθαίρετη(arbitrary) γραμματική*. Οι αυθαίρετες γραμματικές λέγονται και *τύπου-0 γραμματικές.(type-0 grammars)*.Οι μονότονες

λέγονται και τύπου-1, οι ελεύθερες από συμφραζόμενα τύπου-2 και οι κανονικές λέγονται και τύπου-3 γραμματικές. Από τους πιο πάνω ορισμούς προκύπτει η εξής ιεραρχία: $type-3 \subset type-2 \subset type-1 \subset type-0$, δηλαδή μια γραμματική τύπου-3 είναι και τύπου 2 κ.ο.κ. Αυτό σημαίνει ότι αν συμβολίσουμε με TYPE-i τις οικογένειες των γλωσσών που παράγονται από τις αντίστοιχες γραμματικές, τότε θα είναι $TYPE-3 \subset TYPE-2 \subset TYPE-1 \subset TYPE-0$.
Επίσης αν συμβολίσουμε με FIN την οικογένεια των πεπερασμένων γλωσσών, και με LIN,CS τις οικογένειες των γλωσσών που παράγονται από γραμμικές και context-sensitive γραμματικές αντίστοιχα, θα είναι $FIN \subset TYPE-3 \subset LIN \subset TYPE-2 \subset CS \subseteq TYPE-1 \subset TYPE-0$.

Ένα σημαντικό θεώρημα κανονικής μορφής για τις γραμματικές type-0, που θα χρησιμοποιηθεί στο σημαντικό Θεώρημα 3.2.3 του κεφαλαίου 3, είναι το παρακάτω θεώρημα:

Θεώρημα 2.1 (Kuroda normal form) Για κάθε type-0 γραμματική G, μια ισοδύναμη γραμματική G' ($L(G) = L(G')$), μπορεί να κατασκευαστεί αποτελεσματικά, με $G' = (N, T, S, P)$, και τους κανόνες στο P να έχουν μια από τις ακόλουθες μορφές: $A \to BC, A \to a, A \to \lambda, AB \to CD$ για $A, B, C, D \in N$ και $a \in T$.

2.2 Ισοδυναμία γραμματικών Chomsky και μηχανών Turing..

Δοθέντος μιας μηχανής Turing M, μπορούμε να κατασκευάσουμε μια γραμματική Chomsky G,τύπου-0 τέτοια ώστε $L(M) = L(G)$, όπου $L(M)$ η γλώσσα που δέχεται η μηχανή Turing.

Έστω η μηχανή Turing $M = (K, V, s, \delta)$. Αυτή μπορεί να επεκταθεί σε μια μηχανή Turing $M'(K', V, T, s, \delta, F)$ με τερματικό αλφάβητο $T = V$ και τερματικές καταστάσεις $F = \{"yes", "no", h\}$, και $K' = K \cup F$. Αντιστρόφως, για κάθε μηχανή Turing $M(K, V, T, s, \delta, F)$ με $T \subseteq V$ (τερματικό αλφάβητο) και $F \subseteq K$ (F το σύνολο των τερματικών καταστάσεων), υπάρχει μια μηχανή Turing $M' = (K, V, s, \delta')$ με $L(M) = L(M')$. Αυτό που κάνει η M' είναι αρχικά να ελέγχει το input string να δει αν περιέχει μόνο γράμματα του τερματικού αλφαβήτου Τ της Μ. Στην συνέχεια, η M' κάνει ακριβώς ότι και η Μ, με την διαφορά ότι όταν η Μ πρόκειται να μπει σε μια τερματική (για την Μ) κατάσταση $q \in F$, η M' θα περάσει στην κατάσταση "yes". Άρα οι μηχανές Turing με τερματικό αλφάβητο είναι ισοδύναμες με τις απλές μηχανές Turing.

Η ισοδυναμία μεταξύ γραμματικών chomsky, και μηχανών Turing με τερματικό αλφάβητο έχει ως εξής: Καταρχήν, από τη θέση Church-Turing προκύπτει ότι για μια οποιαδήποτε γραμματική Chomsky, υπάρχει μια μηχανή Turing με που να δέχεται την ίδια γλώσσα, διότι η οποιαδήποτε γραμματική Chomsky παρέχει άμεσα μια αλγοριθμική διαδικασία(όχι κατά ανάγκην που τερματίζει).

Επίσης για κάθε μηχανή Turing M μπορεί να κατασκευαστεί μια γραμματική Chomsky G έτσι ώστε $L(G) = L(M)$. Θα παρουσιαστεί πρώτα η βασική ιδέα και κατόπιν θα δοθεί λεπτομερώς η περιγραφή της γραμματικής. Η ιδέα έχει ως εξής: η G θα δημιουργεί (με τυχαίο τρόπο, από τυχαία επιλογή κανόνων) ένα string w από το τερματικό αλφάβητο Τ καθώς και ένα αντίγραφο του w. Στην συνέχεια η G εξομοιώνει τους υπολογισμούς της Μ πάνω στο αντίγραφο του w. Αν φτάσει σε μια τερματική κατάσταση(που σημαίνει ότι $w \in L(M)$), τότε διαγράφεται το τροποποιημένο αντίγραφο του w, και μένει το αρχικό w.

Αναλυτικά, έστω $M=(K,V,T,s,\delta,F)$. Θεωρούμε την γραμματική $G=(N,T,S,P)$ με $N=\{[a,b] \quad a\in T, b\in V\}\cup\{S,X,Y\}\cup K$ και το P περιέχει τους ακόλουθους κανόνες:

1) $S\to s_0X$
2) $X\to[a,a]X \quad a\in T$
3) $X\to Y$
4) $Y\to[\lambda,B]Y$
5) $Y\to\lambda$
6) $s[a,c]\to[a,b]s'$ για $a\in T\cup\{\lambda\}$, $s\in K$, $c,b\in V$ έτσι ώστε $\delta(s,c)=(s',b,\to)$
7) $[e,d]s[a,c]\to s'[e,d][a,b]$ για $c,b,d\in V$, $a,e\in T\cup\{\lambda\}$ $s,s'\in K$, έτσι ώστε $\delta(s,c)=(s',b,\leftarrow)$
8) $[a,c]s\to sas$, $s[a,c]\to sas$, $s\to\lambda$, για $s\in F, a\in T\cup\{\lambda\}, c\in V$.

Οι κανόνες 1-5 παράγουν το string w(οι πρώτες συνιστώσες από τα $[a,a]$) και το αντίγραφο του(για παράδειγμα μπορεί να προκύψει $[\lambda,B][a,a][b,b][c,c]$ οπότε $w=abc$ $B=\triangleright$). Οι κανόνες 6 και 7 εξομοιώνουν τη συνάρτηση μετάβασης δ, ενώ οι κανόνες στο 8, σβήνουν το τροποποιημένο αντίγραφο του w, όταν φτάσουμε σε μια τερματική κατάσταση.

Μετά από όλα αυτά, είναι εύκολο να επαληθευτεί ότι $L(G)=L(M)$. Επίσης εύκολο είναι να διαπιστωθεί, ότι η γραμματική G είναι γενικά τύπου-0. Συνεπώς οι γραμματικές τύπου-0 είναι υπολογιστικά ισοδύναμες των μηχανών Turing.

Αν εφαρμόσουμε την παραπάνω κατασκευή σε μια Universal μηχανή Turing, θα πάρουμε μια γραμματική η οποία θα είναι κατά μια έννοια Universal: θα εξομοιώνει οποιαδήποτε άλλη μηχανή Turing(για την οποία θα υπάρχει μια αντίστοιχη γραμματική) με την έννοια ότι η γλώσσα της universal γραμματικής θα περιέχει strings της μορφής $w\#code(M)$ έτσι ώστε $w\in: L(M)$. Εντούτοις, ενδιαφερόμαστε για έναν ορισμός της καθολικότητας που να είναι πιο «γραμματικός», δηλαδή να έχει πιο άμεση σχέση με τις γραμματικές.

Μια τριάδα $G=(N,T,P)$, όπου τα N,T,P ορίζονται όπως στις συνήθεις γραμματικές chomsky, είναι ένα *σχήμα γραμματικής*. Για ένα string $w\in(N\cup T)^*$, ορίζουμε τη γλώσσα $L(G,w)=\{x\in T^* \mid w\Rightarrow^* x\}$, όπου οι παραγωγές γίνονται σύμφωνα με τους κανόνες στο P.

Μια *καθολική τύπου-0 γραμματική* είναι ένα σχήμα γραμματικής $G_u=(N_u,T_u,P_u)$ όπου N_u,T_u είναι ξένα αλφάβητα, και P_u είναι ένα πεπερασμένο σύνολο κανόνων πάνω στο $N_u\cup T_u$, με την ιδιότητα ότι για κάθε τύπου-0 γραμματική $G=(N,T_u,S,P)$, υπάρχει ένα string $w(G)$, τέτοιο ώστε $L(G_u,w(G))=L(G)$.

Συνεπώς, μια καθολική γραμματική εξομοιώνει οποιαδήποτε δοθείσα γραμματική, δοσμένου ενός κώδικα $w(G)$ της δοθείσας γραμματικής G, που χρησιμεύει σαν το αρχικό string στην καθολική γραμματική.

Αποδεικνύεται ότι υπάρχουν καθολικές τύπου-0 γραμματικές. Η απόδειξη ξεφεύγει κάπως από τους σκοπούς αυτής της εργασίας. Οι λεπτομέρειες της απόδειξης αναφέρονται στο [9].

3. Splicing systems

Τα splicing systems είναι ουσιαστικά η φορμαλιστική μελέτη συστημάτων που στηρίζονται σε κανόνες αποκοπής και συνένωσης για να παράγουν νέα δεδομένα(που στην ουσία είναι συμβολοσειρές) από παλιά(αξιώματα η αυτά που έχουν ήδη παραχθεί). Πριν οριστούν αυστηρά θα δοθεί ένα παράδειγμα το πώς οι λειτουργίες της αποκοπής από ένζυμα αποκοπής και της συνένωσης με την βοήθεια DNA-λιγάσης δίνουν στο μοντέλο της 2.2 έναν χαρακτήρα που με κάποιες λογικές υποθέσεις μπορεί να θεωρηθεί ως splicing system(με την απλή περιγραφική έννοια που δόθηκε στην αρχή της παραγράφου).

3.1 DNA μοντέλο ως splicing system

Έστω ότι έχουμε τα ακόλουθα διπλά μόρια DNA

$$\begin{matrix} 5'CCCCCTCGACCCCC3' \\ 3'GGGGGAGCTGGGGG5' \end{matrix} \quad \text{και} \quad \begin{matrix} 5'AAAAAGCGCAAAAA3' \\ 3'TTTTTCGCGTTTTT5' \end{matrix}$$

και δύο ένζυμα αποκοπής (TaqI και SciNI), των οποίων οι υποακολουθίες αναγνώρισης είναι

$$\begin{matrix} 5'TCGA3' \\ 3'AGCT5' \end{matrix} \quad \text{και} \quad \begin{matrix} 5'GCGC3' \\ 3'CGCG5' \end{matrix} \quad \text{αντίστοιχα.}$$

Η επίδραση των ενζύμων στα δύο δοσμένα μόρια DNA θα έχει ως αποτέλεσμα το σπάσιμο των μορίων στο σημείο που βρίσκεται η υποακολουθία αναγνώρισης του κάθε ενζύμου. Έτσι το πρώτο ένζυμο επιδρά μόνο στο πρώτο μόριο και δίνει τα εξής δύο μόρια DNA:

$$\begin{matrix} 5'CCCCCT \\ 3'GGGGGAGC \end{matrix} \ (\alpha) \ \text{και} \ \begin{matrix} CGACCCCC3' \\ TGGGGG5' \end{matrix} (\beta)$$

Ομοίως το δεύτερο ένζυμο επιδρά στο δεύτερο μόριο DNA(από τα αρχικά στην αρχή της παραγράφου) το οποίο και περιέχει την υποακολουθία αναγνώρισης και δίνει:

$$\begin{matrix} 5'AAAAAG \\ 3'TTTTTCGC \end{matrix} \ (\gamma) \ \text{και} \ \begin{matrix} CGCAAAAA3' \\ GTTTTT5' \end{matrix} (\delta)$$

Παρατηρούμε ότι το μόριο (α) και το μόριο (δ) έχουν συμβατά άκρα(δηλαδή συμπληρωματικά και αντίθετης πολικότητας) και μπορούν να συνενωθούν σε ένα καινούριο μόριο DNA με τη βοήθεια DNA-λιγάσης και να δώσουν το εξής διαφορετικό μόριο:

$$\begin{matrix} 5'CCCCCTCGCAAAAA3' \\ 3'GGGGGAGCGTTTTT5' \end{matrix}$$

Παρόμοια τα μόρια (γ) και (β) μπορούν να ενωθούν και να δώσουν το μόριο

$$\begin{matrix} 5'AAAAAGCGACCCCC3' \\ 3'TTTTTCGCTGGGGG5' \end{matrix}$$

Παρατηρούμε λοιπόν πώς από δύο αρχικά μόρια DNA και με μόνο τις λειτουργίες της αποκοπής(με την βοήθεια περιοριστικών ενζύμων) και συνένωσης(με τη βοήθεια DNA-λιγάσης) παράχθηκαν δύο καινούρια μόρια DNA διαφορετικά από τα αρχικά. Αυτός ο συνδυασμός των παραπάνω λειτουργιών είναι γνωστός ως DNA-επανασυνδυασμός (DNA-recombination). Είναι το σχεδόν το ίδιο με την μαθηματική

αφαίρεση του που λέγεται splicing operation και είναι ο βασικός κανόνας στα splicing systems. Για να περάσουμε από το DNA-recombination στο splicing operation είναι απαραίτητες οι παρακάτω υποθέσεις:

1. Θεωρούμε σειρές από σύμβολα (strings) και όχι δομές με διπλή έλικα (όπως τα διπλά μόρια DNA). Αυτή η υπόθεση είναι ασφαλής και λογική διότι λόγω της συμπληρωματικότητας A-T και C-G δεν έχουμε απώλεια πληροφορίας χρησιμοποιώντας απλά strings.
2. Το μέγεθος του αλφάβητου δεν περιορίζεται σε 4 γράμματα. Αυτή η γενίκευση φαίνεται λογική αφού κάθε αλφάβητο μπορεί να κωδικοποιηδεί σε ένα δυαδικό αλφάβητο.
3. Το μήκος των υποακολουθιών αναγνώρισης των ενζύμων όσο και ο αριθμός των τελευταίων που μπορούν να εργάζονται ταυτόχρονα δεν είναι περιορισμένος. Αυτή είναι μια γενναιόδωρη υπόθεση που η αλήθεια της εξαρτάται από την πρόοδο της βιοτεχνολογίας.

Λαμβάνοντας υπόψη τις παραπάνω υποθέσεις μπορούμε να προχωρήσουμε στον μαθηματικό ορισμό της λειτουργίας splicing όσο και των splicing συστημάτων.

3.2 Ισοδυναμία συστημάτων Splicing και γραμματικών Chomsky

Υπάρχουν πολλοί τύποι συστημάτων splicing. Μπορούμε να έχουμε συστήματα Splicing μονής η διπλής κατεύθυνσης(one-way ή two-way), με επαναλαμβανόμενο η μή splicing(iterated ή non-iterated), απλά ή επεκταμένα(extended), πάνω σε απλά σύνολα ή σε πολυσύνολα. Ανάλογα με το τύπο του συστήματος, πετυχαίνεται με διαφορετικό τρόπο η ισοδυναμία με τις type-0 γραμματικές Chomsky, και συνεπώς με τις μηχανές Turing. Παρακάτω θα εξεταστούν απλά μονής κατεύθυνσης με επαναλαμβανόμενο splicing, επεκταμένα μονής κατεύθυνσης με επαναλαμβανόμενο splicing, και επεκταμένα διπλής κατεύθυνσης με επαναλαμβανόμενο splicing πάνω σε πολυσύνολα.

3.2.1 Συστήματα απλά μονής κατεύθυνσης με επαναλαμβανόμενο splicing(one-way iterated splicing systems)

Έστω αλφάβητο V και δύο σύμβολα $\#,\$$ που δεν είναι στο V. Ένας *κανόνας splicing(σύνδεσης)* είναι ένα string της μορφής $r = u_1\#u_2\$u_3\#u_4$ με $u_1,...,u_4 \in V^*$. Για ένα τέτοιο κανόνα r και strings $x, y, z \in V^*$ θα είναι:

$(x, y) \mid -_r z \Leftrightarrow x = x_1u_1u_2x_2, y = y_1u_3u_4y_2, z = x_1u_1u_4y_2$ για $x_1, x_2, y_1, y_2 \in V^*$.

Το πέρασμα από τα x, y στο z μέσω της $\mid -_r$ φαίνεται στο σχήμα 3.2.1.1.

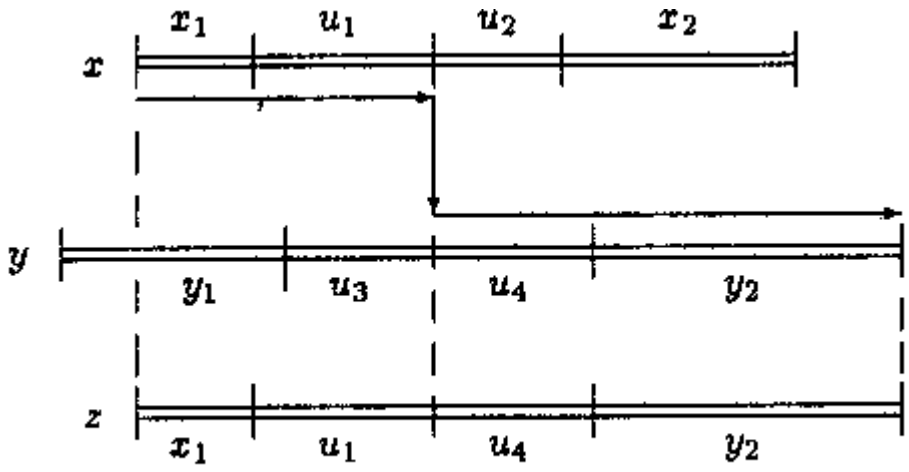
Ένα $H-\sigma\chi\acute{\eta}\mu\alpha$ είναι μια δυάδα $\sigma = (V, R)$ όπου V αλφάβητο και $R \subseteq V^*\#V^*\$V^*\#V^*$ είναι ένα σύνολο κανόνων splicing. Το R μπορεί να είναι απειροσύνολο, και μπορούμε να το δούμε μέσα στην ιεραρχία Chomsky ή σε κάποια άλλη κατάταξη γλωσσών. Γενικά αν $R \in FL$ για μια οικογένεια γλωσσών FL, τότε λέμε ότι το $H-\sigma\chi\acute{\eta}\mu\alpha$ είναι τύπου FL.

Για ένα $H-\sigma\chi\acute{\eta}\mu\alpha$ $\sigma = (V, R)$ και μια γλώσσα $L \subseteq V^*$, ορίζουμε $\sigma_1(L) = \{z \in V^* \mid (x, y) \mid -_r z \quad \gamma\iota\alpha\ x, y \in L, r \in R\}$. Παρατηρούμε ότι $\sigma_1(L) \subseteq V^*$, άρα μπορεί να οριστεί το $\sigma_1(\sigma_1(L))$ κ.ο.κ. Αυτή είναι η ιδέα για το επαναλαμβανόμενο splicing. Γενικά ορίζουμε:

$\sigma_1^0(L) = \sigma_1(L)$

$\sigma_1^{i+1}(L) = \sigma_1(\sigma_1^i(L)) \cup \sigma_1^i(L)$, $i \geq 0$.

$\sigma_1^*(L) = \bigcup_{i \geq 0} \sigma_1^i(L)$.



Σχήμα 3.2.1.1 Η πράξη του splicing

Τελικά ένα απλό σύστημα splicing μονής κατεύθυνσης με επαναλαμβανόμενο splicing, είναι μια τριάδα $\gamma = (V, L, R)$ και η γλώσσα που παράγεται από αυτό είναι $L(\gamma) = \sigma_1^*(L)$,όπου $\sigma = (V, R)$ το υποκείμενο $H - \sigma\chi\acute{\eta}\mu\alpha$ του γ. Το απλό σημαίνει ότι δεν υπάρχει τερματικό αλφάβητο όπως στα επεκταμένα. Η γλώσσα L λέγεται και σύνολο αξιωμάτων του γ.

Με $H_1(FL_1, FL_2)$ συμβολίζουμε την οικογένεια των γλωσσών που παράγονται από one-way (γι αυτό και ο δείκτης 1 στο H_1) iterated splicing συστήματα με σύνολο αξιωμάτων της οικογένειας FL_1 και υποκείμενο H-σχήμα τύπου FL_2. Είναι δηλαδή:

$H_1(FL_1, FL_2) = \{\sigma_1^*(L) \mid L \in FL_1, \sigma = (V, R), R \in FL_2\} =$

$= \{L(\gamma) \mid \gamma = (V, L, R), L \in FL_1, R \in FL_2\}$

Βέβαια οι γλώσσες των FL_1, FL_2 είναι πάνω στο V^*.

Έτσι $H_1(FIN, FIN)$ είναι οι γλώσσες που παράγονται από one-way splicing συστήματα με πεπερασμένο σύνολο αξιωμάτων και πεπερασμένο σύνολο κανόνων, ενώ $H_1(FIN, REG)$ είναι οι γλώσσες που παράγονται από πεπερασμένο σύνολο αξιωμάτων και με σύνολο κανόνων κάποια κανονική γλώσσα(που παράγεται από κανονική γραμματική(type-3 Chomsky grammar).

Ο παρακάτω πίνακας μας δίνει τις οικογένειες $H_1(FL_1, FL_2)$ για διάφορες FL_1, FL_2. Το στοιχείο του πίνακα στη γραμμή FL_1 και στη στήλη FL_2 μας δίνει είτε τη οικογένεια $H_1(FL_1, FL_2)$, είτε δύο οικογένειες FL_3, FL_4 τέτοιες που $FL_3 \subset H_1(FL_1, FL_2) \subset FL_4$.

	FIN	REG	LIN	CF	CS	RE
FIN	FIN,REG	FIN,RE	FIN,RE	FIN,RE	FIN,RE	FIN,RE
REG	REG	REG,RE	REG,RE	REG,RE	REG,RE	REG,RE
LIN	LIN,CF	LIN,RE	LIN,RE	LIN,RE	LIN,RE	LIN,RE
CF	CF	CF,RE	CF,RE	CF,RE	CF,RE	CF,RE
CS	CS,RE	CS,RE	CS,RE	CS,RE	CS,RE	CS,RE
RE	RE	RE	RE	RE	RE	RE

Όπως φαίνεται από τον πίνακα, χρησιμοποιώντας πεπερασμένο σύνολο αξιωμάτων(που αντιστοιχούν στα αρχικά μόρια DNA μέσα στο σωλήνα) και πεπερασμένο σύνολο κανόνων(που αντιστοιχούν στα ένζυμα), δε γίνεται να περάσουμε το φράγμα των κανονικών γλωσσών. Εμείς βέβαια θέλουμε να φτάσουμε όχι απλώς τις κανονικές γλώσσες, αλλά τις γλώσσες RE. Όπως φαίνεται από τον πίνακα είναι $FIN \subset H_1(FIN, REG) \subset RE$. Το ακόλουθο βασικό λήμμα μας δίνει κάτι καλύτερο(και ταυτόχρονα μια ιδέα για γενίκευση του συστήματος splicing σε επεκταμένο χρησιμοποιώντας τερματικό αλφάβητο):

Για τα one-way iterated splicing συστήματα ισχύει το ακόλουθο βασικό λήμμα

Λήμμα 3.2.1 (Βασικό λήμμα καθολικότητας)(Basic Universality Lemma). Κάθε γλώσσα $L \in RE, L \subseteq T^*$, μπορεί να γραφεί στη μορφή $L = L' \cap T^*$ για κάποια $L' \in H_1(FIN, REG)$.

Απόδειξη. Από την ισοδυναμία γραμματικών Chomsky και μηχανών Turing έπεται ότι θα υπάρχει μια γραμματική $G = \{N, T, S, P)$ type-0, έτσι ώστε $L(G) = L$. Θα κατασκευαστεί ένα one-way iterated splicing σύστημα γ έτσι ώστε $L(G) = L(\gamma) \cap T^*$.

Έστω $U = N \cup T \cup \{B\}$ όπου B ένα καινούριο σύμβολο, και θεωρούμε το $H-\sigma\chi\acute{\eta}\mu\alpha$ $\sigma = (V, R)$ όπου $V = N \cup T \cup \{X, X', B, Y, Z\} \cup \{Y_a \mid a \in U\}$ και το σύνολο R περιέχει τους ακόλουθους κανόνες:

Simulate: 1. $Xw\#uY\$Z\#vY$ $\gamma\iota\alpha\ u \to v \in P, w \in U^*$

Rotate: 2. $Xw\#aY\$Z\#Y_a$ $\gamma\iota\alpha\ a \in U, w \in U^*$

3. $X'a\#Z\$X\#wY_a$ για $a \in U, w \in U^*$

4. $X'w\#Y_a\$Z\#Y$ για $a \in U, w \in U^*$

5. $X\#Z\$X'\#wY$ για $w \in U^*$

Terminate: 6. $\#ZY\$XB\#wY$ για $w \in T^*$

7. $\#Y\$XZ\#$

Είναι εύκολο να δούμε ότι το σύνολο R παράγεται από μια *κανονική(regular) γραμματική* G'(π.χ οι κανόνες της ομάδας simulate είναι παράθεση των γλωσσών $\{X\}, U^*, \{\#uY\$Z\#vY \mid u \to v \in P\}$,όπου η πρώτη και η τελευταία είναι πεπερασμένες, άρα κανονικές, ενώ η μεσαία είναι κανονική. Η παράθεση κανονικών γλωσσών είναι κανονική γλώσσα, όπως επίσης και η ένωση, παίρνοντας δηλαδή την ένωση των κανονικών γλωσσών που ορίζει κάθε κανόνας, έχουμε το R) και συνεπώς $R \in REG$. Επίσης θεωρούμε τη γλώσσα

$L_0 = \{XBSY, ZY, XZ\} \cup \{ZvY \mid u \to v \in P\} \cup \{ZY_a, X'aZ \mid a \in U\}$

Έστω $\gamma = (V, L_0, R)$ τότε θα είναι $L = L(\gamma) \cap T^* = \sigma_1^*(L_0) \cap T^*$. Πράγματι:

Καταρχήν, κανένα string στην L_0 δεν είναι στο T^*. Όλοι οι κανόνες του συνόλου R περιλαμβάνουν το σύμβολο Z, αλλά αυτό το σύμβολο δεν εμφανίζεται στο string που παράγεται από το splicing. Συνεπώς, σε κάθε βήμα θα πρέπει να χρησιμοποιήσουμε τουλάχιστον ένα string από την L_0, και μπορεί ένα string από γλώσσα που παράγεται σε κάποιο προηγούμενο βήμα.

Το σύμβολο B είναι για να μαρκάρεται η αρχή των προτασιακών τύπων της G, καθώς παράγονται string από το $H-\sigma\chi\acute{\eta}\mu\alpha$.

Με τους κανόνες στην ομάδα simulate(ομάδα 1) προσομοιώνουμε τους κανόνες της γραμματικής που περιέχονται στο σύνολο P. Οι κανόνες στις ομάδες 2-5(που

συνιστούν την ευρύτερη ομάδα rotate), μεταφέρουν σύμβολα από το δεξί άκρο του string, στο αριστερό άκρο, θυμίζοντας την λειτουργία του δυαδικού rotate. Τα rotate στους 2-5 χρησιμεύουν ώστε να μπορούμε να εφαρμόζουμε τους κανόνες σε όποιο σημείο θέλουμε: Πρώτα κάνουμε τα κατάλληλα rotates ώστε να εμφανιστεί το u του κανόνα $u \to v$, στο δεξιό άκρο του string, στην συνέχεια εφαρμόζεται ο κατάλληλος κανόνας της ομάδας 1, και στην συνέχεια με κατάλληλα rotates, επαναφέρουμε το string στην αρχική του κατάσταση. Επειδή το B είναι πάντα παρόν και μαρκάρει την αρχή του string της G, ξέρουμε σε κάθε στιγμή ποιο είναι αυτό το string. Για παράδειγμα, αν το τρέχον string είναι της μορφής $\beta_1 w_1 B w_2 \beta_2$ για $\beta_1, \beta_2 \in \{X, Y, X', Y_a \mid a \in U\}$ και $w_1, w_2 \in (N \cup T)^*$ τότε το $w_2 w_1$ είναι ένας προτασιακός τύπος της G.

Αρχίζουμε με το αξίωμα της G δηλαδή το $XBSY$, που μαρκάρεται από τα αριστερά με το B, και εντός των X, Y.

Ας δούμε πως δουλεύουν οι κανόνες 2-5. Παίρνουμε ένα string $XwaY$ για κάποιο $a \in U$, και $w \in U^*$. Από ένα κανόνα του 2, παίρνουμε:

$(Xw \mid aY, Z \mid Y_a \mid -XwY_a$.

(Οι μπάρες μέσα στη παρένθεση είναι βοηθητικές για να αντιληφθούμε που γίνεται το slicing). Το σύμβολο Y_a, κρατά στη μνήμη το γεγονός ότι το a έχει διαγραφεί από το δεξί άκρο του wa. Κανένας κανόνας στο R δε μπορεί να εφαρμοστεί στο XwY_a εκτός από κανόνα του τύπου 3:

$(X'a \mid Z, X \mid wY_a) \mid\!= X'awY_a$.

Άρα το σύμβολο a έχει τώρα προστεθεί στην αρχή του w. Πάλι υπάρχει ένας τρόπος για να συνεχίσουμε, χρησιμοποιώντας ένα κανόνα τύπου 4:

$(X'aw \mid Y_a, Z \mid Y) \mid -X'awY$.

Αν τώρα χρησιμοποιήσουμε ένα κανόνας από το 5 θα έχουμε:

$(X \mid Z, X' \mid awY) \mid -XawY$.

Αρχίσαμε λοιπόν από το $XwaY$ και πήραμε το $XawY$, ένα string με τους ίδια σύμβολα μαρκαρίσματος στα δύο άκρα. Αυτά τα βήματα μπορούν να επαναληφθούν όσες φορές θέλουμε, οπότε μπορεί να παραχθεί οποιαδήποτε κυκλική μετάθεση του string μεταξύ των X και Y.

Σε κάθε string XwY μπορεί να εφαρμοστεί ένας κανόνας της ομάδας 1, αρκεί το w να τελειώνει με το u ($w = w'u$) ενός κανόνα $u \to v \in P$. Άρα κάθε κανόνας του P μπορεί να εφαρμοστεί, σε οποιαδήποτε θέση, προετοιμάζοντας κατάλληλα το string w, με χειρισμούς όπως αυτούς παραπάνω και χρησιμοποιώντας κανόνες 2-5.

Συνεπώς, για κάθε προτασιακό τύπο w της G υπάρχει ένα string $XBwY$, που παράγεται από το σ, και, αντιστρόφως, αν $Xw_1 B w_2 Y$ έχει παραχθεί από το σ, τότε $w_2 w_1$ είναι ένας προτασιακός τύπος της G. Όταν λέμε ότι παράγεται από το σ, εννοούμε ότι παράγεται σε κάποιο βήμα της διαδικασίας σ_1^i.

Το w ενδέχεται να περιέχει μη τερματικά σύμβολα(που δεν ανήκουν στο T). Τότε δεν γίνεται να χρησιμοποιηθούν οι κανόνες 6-7 για να απαλλαχθεί το w από τα σύμβολα που το μαρκάρουν. Φυσικά το π.χ $XBwY \in \sigma_1^*(L_0)$ αλλά όχι στο $\sigma_1^*(L_0) \cap T^*$. Για να απαλλαχθούμε από τα X, B, Y θα πρέπει να χρησιμοποιήσουμε πρώτα τον κανόνα 6 και το ZY της L_0, για να πάρουμε το wY και στη συνέχεια το κανόνα 7 και το $XZ \in L_0$ για να απαλλαγούμε από το Y. Παρατηρούμε ότι ο κανόνας 6 μπορεί να εφαρμοστεί μόνο όταν το B είναι στην πιο αριστερή θέση(όπως στο

XBwY) και όταν $w \in T^*$. Αυτό εγγυάται ότι τα w με τερματικά σύμβολα μόνο, που παράγονται από το σ, είναι προτασιακοί τύποι της G και εφόσον έχουν μόνο τερματικά σύμβολα είναι μέσα στο $L(G)$. Άρα $\sigma_1^*(L_0) \cap T^* \subseteq L(G)$. Επίσης είναι κατανοητό ότι κάθε string της $L(G)$ μπορεί να παραχθεί από το σ και συνεπώς $\sigma_1^*(L_0) \cap T^* \supseteq L(G)$. Άρα $L(G) = \sigma_1^*(L_0) \cap T^*$ και η απόδειξη έχει ολοκληρωθεί.

1.2.2 Συστήματα επεκταμένα μονής κατεύθυνσης με επαναλαμβανόμενο splicing(extended one-way iterated splicing systems)

Το λήμμα 3.2.1 δίνει την πιο άμεση ιδέα για το τι πρέπει να γίνει σε μια προσπάθεια προσέγγισης των γλωσσών RE με όσο το δυνατόν ελαφρύτερα μέσα(σύνολο κανόνων και σύνολο αξιωμάτων, χαμηλότερα στην ιεραρχία chomsky). Πρέπει να προστεθεί ένα τερματικό αλφάβητο, και να οριστεί ανάλογα η γλώσσα που παράγεται.

Πιο συγκεκριμένα ένα *επεκταμένο* σύστημα splicing μόνης κατεύθυνσης με επαναλαμβανόμενο splicing, είναι μια τετράδα $\gamma = (V, T, L, R)$, όπου V αλφάβητο, $T \subseteq V, L \subseteq V^*$ και $R \subseteq V^* \# V^* \$ V^* \# V^*$, όπου $\#,\$$ ειδικά σύμβολα που δεν εμφανίζονται στο V. Η γλώσσα που παράγεται από το γ είναι:

$L(\gamma) = \sigma_1^*(L) \cap T^*$, με $\sigma = (V, R)$ το υποκείμενο $H - \sigma\chi\acute{\eta}\mu\alpha$ του γ.

Εδώ T είναι το τερματικό αλφάβητο. Ένα τέτοιο σύστημα splicing λέγεται και επεκταμένο $H - \sigma\acute{\upsilon}\sigma\tau\eta\mu\alpha$. Όταν $T = V$ τότε το σύστημα θα λέγεται *μη επεκταμένο* και είναι στην ουσία το σύστημα της προηγούμενης παραγράφου. Η δυνατότητα του να έχουμε τερματικό αλφάβητο, οποιοδήποτε υποσύνολο του V, είναι αυτή που μας επεκτείνει τις δυνατότητες του συστήματος.

Με $EH_1(FL_1, FL_2)$ συμβολίζουμε την οικογένεια των γλωσσών από επεκταμένα συστήματα splicing $\gamma = (V, T, L, R)$ με $L \in FL_1$ και $R \in FL_2$. Το T είναι ελεύθερο να παίρνει οποιοδήποτε υποσύνολο του V (για τα ίδια L, R).

Λήμμα 3.2.2. $REG = EH_1(FIN, FIN)$**.**

Απόδειξη. Παρόμοια με του λήμματος 3.2.1. Η ιδέα είναι να θεωρήσουμε μια γλώσσα $L \in REG$ και μια γραμματική $G = (N, T, S, P)$ τέτοια ώστε $L(G) = L$, και να κατασκευαστεί ένα επεκταμένο σύστημα πεπερασμένων κανόνων και αξιωμάτων, με τερματικό αλφάβητο T (ίδιο με της G), που να εξομοιώνει τους κανόνες της G (που βρίσκονται στο σύνολο P). Έτσι αποδεικνύεται ότι $REG \subseteq EH_1(FIN, FIN)$. Το $EH_1(FIN, FIN) \subseteq REG$ αποδεικνύεται κατά την αντίστροφη ιδέα: κατασκευάζουμε μια γραμματική που να μιμείται το επεκταμένο σύστημα splicing(που έχει πεπερασμένους κανόνες και αξιώματα), και δείχνουμε ότι είναι κανονική.

Από το λήμμα 3.2.1 προκύπτει εύκολα ότι $RE \subseteq EH_1(FIN, REG)$. Από τη θέση Church-Turing μπορούμε εύκολα να πάρουμε ότι $EH_1(FIN, REG) \subseteq RE$. Συνεπώς έχουμε ότι

$1. RE = EH_1(FIN, REG)$

$2. REG = EH_1(FIN, FIN)$

Αν συνδυάσουμε με μια ψυχρή μαθηματική λογική τα δύο αυτά αποτελέσματα προκύπτει ότι $RE = EH_1(FIN, EH_1(FIN, FIN))$. Άρα από μια πρώτη σκοπιά, φαίνεται να έχουμε φτάσει τον σκοπό μας, που ειπώθηκε στην αρχή της 3.2.2, δηλαδή να φτάσουμε τις γλώσσες RE, και συνεπώς την υπολογιστική δυνατότητα των μηχανών Turing, με συστήματα splicing που να έχουν τα χαμηλότερα δυνατά μέσα

στην ιεραρχία Chomsky:πεπερασμένα σύνολα(για κανόνες και αξιώματα). Όμως το $REG = EH_1(FIN, FIN)$ από σκοπιά DNA υπολογιστών είναι μόρια DNA, ενώ το REG στο $RE = EH_1(FIN, REG)$ από σκοπιά DNA υπολογιστών είναι ένζυμα(που αντιστοιχούν στους κανόνες).

Τελικά, αυτό που έχουμε πετύχει είναι να φτάσουμε τις γλώσσες RE με πεπερασμένο αρχικό αριθμό μορίων DNA, και ενδεχομένως άπειρο αριθμό ενζύμων(αφού είναι κανονική γλώσσα, όχι κατανάγκην πεπερασμένη). Για να ξεπεραστεί αυτή η δυσκολία, υπάρχουν πολλοί τρόποι([9]):συστήματα με επιτρεπόμενα πλαίσια, απαγορευτικά πλαίσια, τοπικές ή σφαιρικές γλώσσες στόχου, αντιστοιχίες ταιριάσματος, αντιστοιχίες επόμενου κανόνα, μεταλλαγές σημείων που αλλάζουν τους κανόνες splicing, διπλό splicing(χρησιμοποιώντας δύο κανόνες ταυτόχρονα), συστήματα με πολυσύνολα. Εδώ θα ακολουθηθεί η τελευταία προσέγγιση: θα θεωρηθούν splicing συστήματα διπλής κατεύθυνσης και πάνω σε πολυσύνολα.

1.2.3 Συστήματα επεκταμένα διπλής κατεύθυνσης με επαναλαμβανόμενο splicing πάνω σε πολυσύνολα(extended two-way iterated splicing systems on multisets)

Καταρχήν το splicing διπλής κατεύθυνσης χρησιμοποιεί και τα δύο κομμάτια στα οποία σπάει το κάθε ένα από τα δύο αρχικά strings, δίνοντας έτσι ως αποτέλεσμα δύο καινούρια string και όχι ένα. Το splicing διπλής κατεύθυνσης(ή 2-splicing) συμβολίζεται με $|=_r$ όπου r ο κανόνας. Πιο συγκεκριμένα είναι:

$(x, y) |=_r (z, w) \quad \Leftrightarrow x = x_1u_1u_2x_2, y = y_1u_3u_4y_2, \quad z = x_1u_1u_4y_2, w = y_1u_3u_2x_2$ για κάποια $x_1, x_2, y_1, y_2 \in V^*$ και $r = u_1\#u_2\$u_3\#u_4$.

Για ένα H-σχήμα $\sigma = (V, R)$ και μια γλώσσα $L \subseteq V^*$ ορίζουμε τώρα ως $\sigma_2(L) = \{z \in V^* \mid (x, y) |=_r (z, w)$ ή $(x, y) |=_r (w, z)$ για κάποια $x, y \in L$ και $r \in R\}$. Ανάλογα μπορούμε να ορίσουμε το $\sigma_2^i(L)$ και το $\sigma_2^*(L)$. Ο δείκτης 2 στο σ_2 δηλώνει ότι έχουμε 2-splicing. Ανάλογα ορίζεται και η οικογένεια $EH_2(FL_1, FL_2) = \{\sigma_2^*(L) : L \in FL_1, \sigma = (V, R), R \in FL_2\}$.

Μέχρι τώρα, είτε θεωρήσουμε 1-splicing(splicing μονής κατεύθυνσης) είτε 2-splicing, θεωρούμε ότι τα strings x, y βρίσκονται σε απεριόριστο αριθμό και το ότι δεν καταναλώνονται από την λειτουργία του splicing. Κάτι τέτοιο μπορεί να φαίνεται ρεαλιστικό, με την έννοια ότι σε ένα δοκιμαστικό σωλήνα υπάρχουν πάρα πολλά αντίγραφα ενός δοσμένου string x, συνεπώς μπορούμε να τα θεωρούμε άπειρα. Εντούτοις η ύπαρξη πολλών αντιγράφων του κάθε string, είναι μια δυσκολία στο να ελέγξουμε την λειτουργία του splicing. Αν θεωρήσουμε ότι τα strings έχουν ένα πεπερασμένο(και κάπως περιορισμένο) αριθμό από αντίγραφα, έχουμε ένα τρόπο να ελέγξουμε την λειτουργία του splicing, με άλλα λόγια βάζουμε μια επιπλέον δυνατότητα στο σύστημα splicing. Αυτή η επιπλέον δυνατότητα μας επιτρέπει να «πιάσουμε» τις γλώσσες RE με πεπερασμένο αριθμό στα μέσα.

Για να εισάγουμε αυτή τη δυνατότητα θα πρέπει αρχικά να θεωρηθεί ή έννοια του πολυσυνόλου(multiset), δηλαδή σύνολα όπου έχουμε και τις πολλαπλότητες με τις οποίες εμφανίζεται το κάθε στοιχείο του συνόλου.

Πιο φορμαλιστικά, ένα πολυσύνολο πάνω σε ένα σύνολο X είναι μια αντιστοιχία $M : X \to N \cup \{\infty\}$, με $M(x)$ να είναι ο αριθμός των αντιγράφων του στοιχείου $x \in X$. Όταν $M(x) = \infty$ τότε ο αριθμός των αντιγράφων του x είναι απεριόριστος. Το σύνολο $Supp(M)\{x \in X \mid M(x) > 0\}$ λέγεται το *στήριγμα* του M.

Για δύο πολυσύνολα M_1, M_2 πάνω στο X, ορίζεται η ένωση τους $(M_1 \cup M_2)(x) = M_1(x) + M_2(x)$, και η διαφορά τους $(M_1 - M_2)(x) = M_1(x) - M_2(x)$ για $M_1(x) \geq M_2(x)$ και $M_1(x), M_2(x) < \infty$ (πεπερασμένα). Αν $M_1(x) = \infty$ τότε $(M_1 - M_2)(x) = \infty$. Ένα πολυσύνολο με πεπερασμένο στήριγμα, αναπαριστάται ως ένα σύνολο ζευγών $(x, M(x))$ για $x \in Supp(M)$.

Για παράδειγμα, $M_1 = \{(ab,3),(abb,1),(aa,\infty)\}$ είναι ένα πολυσύνολο πάνω στο $\{a,b\}^*$ με το στήριγμα να αποτελείται από τρεις λέξεις ab, abb, aa. Η πρώτη εμφανίζεται σε 3 αντίγραφα, η δεύτερη σε 1, και η τρίτη σε απεριόριστο αριθμό.

Ένα επεκταμένο σύστημα διπλής κατεύθυνσης με επαναλαμβανόμενο splicing, πάνω σε πολυσύνολα, ή πιο απλά ένα *επεκταμένο μH σύστημα,* είναι μια τετράδα $\gamma = (V, T, L, R)$ όπου V αλφάβητο, $T \subseteq V$ (τερματικό αλφάβητο), L είναι ένα πολυσύνολο πάνω στο $V^+ = V^* - \{\lambda\}$ με πεπερασμένο $Supp(L)$, και R είναι ένα πεπερασμένο σύνολο κανόνων πάνω στο V.

Για ένα *μH σύστημα* και δύο πολυσύνολα M_1, M_2 πάνω στο V^* ορίζουμε την σχέση $M_1 \Rightarrow_\gamma M_2$ αν και μόνο αν υπάρχουν $x, y, z, w \in V^*$ ώστε να ισχύουν τα εξής:

(1) $M_1(x) \geq 1, (M_1 - \{(x,1)\})(y) \geq 1$

(2) $(x, y) \models_r (z, w)$ για κάποιο $r \in R$.

(3) $M_2 = (((M_1 - \{(x,1)\}) - \{(y,1)\}) \cup \{(z,1)\}) \cup \{(w,1)\}$.

Στο (1) έχουμε τη συνθήκη έτσι γραμμένη ώστε να καλύπτει και τη περίπτωση που $x = y$ οπότε θα πρέπει $M_1(x) \geq 2$. Επίσης η (3) είναι γραμμένη έτσι ώστε να καλύπτει τη περίπτωση που $z = w$ οπότε θα πρέπει να προσθέσουμε 2 στο $M_2(z)$.

Με απλά λόγια όταν περνάμε με την $\Rightarrow_\gamma$ από ένα πολυσύνολο M_1 σε ένα πολυσύνολο M_2, η πολλαπλότητα δύο στοιχείων του M_1, x, y, μειώνεται κατά ένα, ενώ η πολλαπλότητα των αποτελεσμάτων της λειτουργίας 2-splicing, z, w, αυξάνεται κατά 1. Η πολλαπλότητα όλων των άλλων στοιχείων στο $Supp(M_1)$ δεν αλλάζει.

Η γλώσσα που παράγεται από ένα επεκταμένο *μH σύστημα* γ συνίσταται από όλες τις λέξεις που περιέχουν μόνο τερματικά σύμβολα, και των οποίων η πολλαπλότητα είναι τουλάχιστον μια φορά, μεγαλύτερη η ίση του 1, κατά την διάρκεια της εργασίας του γ. Φορμαλιστικά, η γλώσσα ορίζεται ως:

$L(\gamma) = \{w \in T^* \mid w \in Supp(M)$ για M τέτοιο ώστε $L \Rightarrow_\gamma^* M\}$, όπου $\Rightarrow_\gamma^*$ το μεταβατικό κλείσιμο της $\Rightarrow_\gamma$, δηλαδή $M_1 \Rightarrow_\gamma^* M_2$, αν υπάρχει $k \in N$ και πολυσύνολα $X_1, ..., X_k$ ώστε $M_1 \Rightarrow_\gamma X_1 \Rightarrow_\gamma ... X_k \Rightarrow_\gamma M_2$.

Η οικογένεια των γλωσσών που παράγονται από *επεκταμένα μH συστήματα* $\gamma = (V, T, L, R)$ με $|Supp(A)| \leq n$ και $rad(R) \leq m$ για $n, m \geq 1$, συμβολίζεται με $EH_2(\mu[n],[m])$. Όταν τα n και m δεν είναι φραγμένα(αλλά πάντα είναι πεπερασμένα) αντικαθιστούμε τα $[n],[m]$ με FIN. Το $|Supp(A)|$ είναι απλά ο αριθμός των στοιχείων του $Supp(A)$, ενώ $rad(R)$ είναι η *ακτίνα* του συνόλου των κανόνων και ορίζεται ως $rad(R) = \max\{|x| \mid x = u_i, 1 \leq i \leq 4, u_1 \# u_2 \$ u_3 \# u_4 \in R\}$, όπου $|x|$ το μήκος του string x. Δηλαδή μας ενδιαφέρει το μήκος των κανόνων του R, μια και το πλήθος τους είναι ήδη πεπερασμένο, από τον ορισμό των *μH συστημάτων.*

Το ότι $M(x)=\infty$ δεν σημαίνει ότι έχουμε στη κατοχή μας άπειρα αντίγραφα του x. Απλά σημαίνει ότι όποτε χρειαζόμαστε επιπλέον αντίγραφα του x μπορούμε να τα παράγουμε(π.χ με μέθοδος PCR).

Όπως ειπώθηκε, τα πολυσύνολα μας δίνουν μια επιπλέον δυνατότητα για να πιάσουμε τις γλώσσες RE: εξομοιώνοντας πλήρως τις type-0 γραμματικές, χρησιμοποιώντας πεπερασμένα μέσα για το splicing: πεπερασμένο αρχικό σύνολο αξιωμάτων, και πεπερασμένο σύνολο κανόνων με μήκος το πολύ 20.(Από σκοπιά DNA computing, το μήκος του κανόνα, αντιστοιχεί στο μήκος της ακολουθίας αναγνώρισης του ενζύμου που αντιστοιχεί στον κανόνα). Αυτό δίνεται στο παρακάτω θεώρημα

Θεώρημα 3.2.3 $RE = EH_2(\mu FIN,[5])$.

Απόδειξη. Θεωρούμε μια γραμματική τύπου-0 $G=(N,T,S,P)$ η οποία σύμφωνα με το θεώρημα 2.1 μπορούμε να θεωρούμε ότι ήδη βρίσκεται σε Kuroda normal form. Αυτό σημαίνει ότι για κάθε $u\to v\in P$ είναι $1\le|u|\le 2$ και $0\le|v|\le 2$ και $u\neq v$. Έστω $U=N\cup T$. Κατασκευάζουμε το επεκταμένο μH σύστημα $\gamma=(V,T,A,R)$ με το σύνολο $V=U\cup\{X_1,X_2,Y,Z_1,Z_2\}\cup\{(r),[r]\,|\,r\in P\}$ και το πολυσύνολο A περιέχει την λέξη $w_0=X_1X_1YSX_2X_2=X_1^2YSX_2^2$ με πολλαπλότητα $A(w_0)=1$, καθώς επίσης και τις ακόλουθες λέξεις με άπειρη πολλαπλότητα:
$w_r=(r)v[r]$ για $r:u\to v\in P$, $w_a=Z_1aYZ_2, w_a'=Z_1YaZ_2$ για $a\in U$ και $w_t=YY$.
Τέλος το σύνολο R περιέχει τους ακόλουθους κανόνες splicing:

1. $\delta_1\delta_2Yu\#\beta_1\beta_2\$(r)v\#[r]$ για $r:u\to v\in P$, $\beta_1,\beta_2\in U\cup\{X_2\}$, $\delta_1,\delta_2\in U\cup\{X_1\}$
2. $Y\#u[r]\$(r)\#va$ για $r:u\to v\in P, a\in U\cup\{X_2\}$
3. $\delta_1\delta_2Ya\#\beta_1\beta_2\$Z_1aY\#Z_2$ για $a\in U,\beta_1,\beta_2\in U\cup\{X_2\},\delta_1,\delta_2\in U\cup\{X_1\}$
4. $\delta\#YaZ_2\$Z_1\#aY\beta$ για $a\in U,\delta\in U\cup\{X_1\}$, $\beta\in U\cup\{X_2\}$
5. $\delta aY\#\beta_1\beta_2\beta_3\$Z_1Ya\#Z_2$ για $a\in U$, $\beta_1\in U,\beta_2,\beta_3\in U\cup\{X_2\},\delta\in U\cup\{X_1\}$
6. $\delta\#aYZ_2\$Z_1\#Ya\beta$ για $a\in U,\delta\in U\cup\{X_1\},\beta\in U\cup\{X_2\}$
7. $\#YY\$X_1^2Y\#w$ για $w\in\{X_2^2\}\cup T\{X_2^2\}\cup T^2\{X_2\}\cup T^3$
8. $\#X_2^2\$Y^3\#$.

Σκοπός είναι να δειχθεί ότι $L(\gamma)=L(G)$ δηλαδή ότι $L(G)\subseteq L(\gamma)$ και $L(G)\supseteq L(\gamma)$. Για να δείξουμε ότι $L(G)\subseteq L(\gamma)$, με αυστηρό τρόπο, θα πρέπει να χρησιμοποιηθεί επαγωγή στο μήκος της παραγωγής της G. Θα ήταν όμως καλύτερα προς χάρη της διαίσθησης να δούμε πως δουλεύουν οι κανόνες 1-8. Οι κανόνες στις ομάδες 1 και 2 εξομοιώνουν τους κανόνες του P με την βοήθεια του w_r, και δοσμένης της παρουσία του Y στο άλλο αλφαριθμητικό. Οι κανόνες στις ομάδες 3 και 4 μετακινούν το σύμβολο Y στα δεξιά, ενώ αυτοί στις 5 και 6 το μετακινούν στα αριστερά. Το «κύριο αξίωμα»(του μH συστήματος) είναι το w_0. Όλοι οι κανόνες στις ομάδες 1 ως 6 αφορούν ένα string που παράγεται από το w_0, και που περιέχει το σύμβολο Y που μπαίνει από αυτό το αξίωμα, με τέτοιο τρόπο, ώστε να μπορούν να χρησιμοποιήσουν μόνο ένα αξίωμα διαφορετικό από το w_0. Σε κάθε στιγμή, έχουμε 2 εμφανίσεις του X_1 στην αρχή της λέξης, και δύο εμφανίσεις του X_2 στο τέλος της λέξης. Οι κανόνες στις ομάδες 1, 3,5 σπάνε λέξεις της μορφής $X_1^2zX_2^2$ σε δύο λέξεις

$X_1^2 z_1$, $z_2 X_2^2$, η κάθε μια με πολλαπλότητα 1. Οι κανόνες στις ομάδες 2,4,6 ενώνουν αυτές τις λέξεις φτιάχνοντας μια λέξη του τύπου $X_1^2 z' X_2^2$. Οι κανόνες στις ομάδες 7,8 βγάζουν τα σύμβολα X_1, X_2, Y. Αν η λέξη που απομένει είναι τερματική, τότε είναι στοιχείο της $L(G)$. Τα σύμβολα $(r),[r]$ σχετίζονται με κανόνες του P, ενώ τα Z_1, Z_2 σχετίζονται με λειτουργίες μετακίνησης. Με όλες τις παραπάνω παρατηρήσεις μπορεί να διαπιστωθεί ότι $L(G) \subseteq L(\gamma)$.

Για να αποδειχθεί ότι $L(\gamma) \subseteq L(G)$ η ισοδύναμα ότι αν $A \Rightarrow_\gamma^* M$ και $w \in T^*$ και $M(w) > 0$ τότε $w \in L(G)$ σκεφτόμαστε ως εξής: Καταρχήν, όπως εύκολα μπορεί να διαπιστωθεί, δεν γίνεται να γίνει splicing ανάμεσα σε δύο από τα w_r, w_a, w_a', w_t. (Για παράδειγμα τα σύμβολα δ, β στους κανόνες 4 και 6, απαγορεύουν το splicing των $w_a, w_a', a \in U$. Στο πρώτο βήμα θα πρέπει να ξεκινήσουμε με το w_0 και να χρησιμοποιήσουμε κάποιο από τα w_r, w_a, w_a', w_t. Αν υποθέσουμε ότι είμαστε σε κάποιο βήμα και έχουμε τη λέξη $X_1^2 w_1 Y w_2 X_2^2$ με πολλαπλότητα 1. Αν το w_2 αρχίζει με το αριστερό μέρος ενός κανόνα του P τότε μπορούμε να εφαρμόσουμε ένα κανόνα της ομάδας 1. Πράγματί έστω ότι η λέξη είναι $X_1^2 w_1 Y u w_3 X_2^2$ και έστω ο κανόνας $r : u \to v \in P$. Χρησιμοποιώντας το αξίωμα $(r)v[r]$ παίρνουμε:

$(X_1^2 w_1 Y u w_3 X_2^2, (r)v[r]) \models_1 (X_1^2 w_1 Y u[r], (r)v w_3 X_2^2)$.

Όπως μπορεί να διαπιστωθεί, μπορεί να εφαρμοστεί μόνο κανόνας από την ομάδα 2 στις δύο λέξεις που προέκυψαν και έχουμε:

$(X_1^2 w_1 Y u[r], (r)v w_3 X_2^2) \models_2 (X_1^2 w_1 Y v w_3 X_2^2, (r)v[r])$.

Βλέπουμε λοιπόν ότι είναι σαν να έχουμε χρησιμοποιήσει τον κανόνα της γραμματικής $u \to v$.

Αν τώρα το w_2 δεν αρχίζει με το αριστερό μέρος κάποιο κανόνα της γραμματικής τότε έστω ότι έχουμε $X_1^2 w_1 Y a w_3 X_2^2$ στην οποία μπορούμε να εφαρμόσουμε κανόνα της ομάδας 3 οπότε παίρνουμε:

$(X_1^2 w_1 Y a w_3 X_2^2, Z_1 a Y Z_2) \models_3 (X_1^2 w_1 Y a Z_2, Z_1 a Y w_3 X_2^2)$.

Τώρα μπορούμε να εφαρμόσουμε μόνο κανόνα της ομάδας 4 οπότε θα έχουμε

$(X_1^2 w_1 Y a Z_2, Z_1 a Y w_3 X_2^2) \models (X_1^2 w_1 a Y w_3 X_2^2, Z_1 Y a Z_2)$

Βλέπουμε λοιπόν ότι έχουμε μετακινήσει το Y στα δεξιά, ανταλλάσσοντας θέση με το a. Έτσι διαδοχικά προετοιμάζεται το w_3 για να εφαρμοστεί κάποιος κανόνας. Φυσικά χρησιμοποιώντας ανάλογα τους κανόνες 4,5 μπορούμε να μετακινήσουμε το Y προς τα αριστερά.

Όπως φαίνεται από τα παραπάνω, σε κάθε στάδιο έχουμε ένα πολυσύνολο είτε με μια λέξη $X_1^2 w_1 Y w_2 X_2^2$, είτε με δύο λέξεις της μορφής $X_1^2 z_1, z_2 X_2^2$ σε κάθε περίπτωση όλες με πολλαπλότητα 1. Μόνο στην πρώτη περίπτωση όταν $w_1 = \lambda$ μπορούμε να χρησιμοποιήσουμε κανόνα της ομάδας 7 και να αφαιρέσουμε τη λέξη $X_1^2 Y$, και στη συνέχεια με το κανόνα 8 να αφαιρέσουμε και τη λέξη X_2^2. Αν η λέξη που θα προκύψει, δεν είναι από τερματικά σύμβολα μόνο, δεν θα μπορεί να επεξεργαστεί περαιτέρω διότι δεν θα έχει το σύμβολο Y. Συνεπώς μπορούμε να εξομοιώσουμε μόνο τις παραγωγές της γραμματικής G, άρα ισχύει τελικά $L(G) \subseteq L(\gamma)$.

Άρα αποδείχθηκε ότι $RE \subseteq EH_2(\mu FIN, [5])$. Με επίκληση της θέσης Church-Turing δείχνουμε ότι $RE \supseteq EH_2(\mu FIN, [5])$. Συνεπώς ισχύει το θεώρημα. Σημαντική

η παρατήρηση ότι $Supp(A)$ είναι πεπερασμένο και $Rad(R)=5$, όπως προκύπτει από τους κανόνες της ομάδας 1 όπου $|\delta_1\delta_2 Yu| \le 5$ γιατί $|u| \le 2$. Επίσης εύκολα ελέγχεται ότι το σύνολο R είναι πεπερασμένο.

1.3 Universal σύστημα splicing- Universal DNA υπολογιστές

Στην προηγούμενη παράγραφο δείχθηκε το θεώρημα 3.2.3 από το οποίο προκύπτει ότι τα επεκταμένα συστήματα splicing πάνω σε πολυσύνολα, έχουν την ίδια δύναμη με τις γραμματικές Chomsky και συνεπώς με τις μηχανές Turing. Αυτό σημαίνει ότι το υπολογιστικό μοντέλο που συνιστούν τα splicing συστήματα πάνω σε πολυσύνολα, είναι υπολογιστικά πλήρες(computationally complete). Συνεπώς ήδη έχει απαντηθεί το πρώτο ερώτημα που τέθηκε στο τέλος της παραγράφου 2.3 στο Μέρος Ι.

Η απάντηση στο δεύτερο ερώτημα της 2.3 προκύπτει άμεσα από την θετική απάντηση του πρώτου ερωτήματος με τον εξής συλλογισμό: Εφόσον τα μH συστήματα είναι υπολογιστικά πλήρη, σημαίνει ότι υπάρχει ένα τέτοιο σύστημα που μπορεί να αντικαταστήσει την Universal μηχανή Turing. Άρα υπάρχει μH σύστημα που η γλώσσα που παράγει είναι $(w(M),x)$, έτσι ώστε $x \in L(M)$ όπου M μια μηχανή Turing και $w(M)$ η κωδικοποίηση της μηχανής Μ. Εντούτοις απαιτούμε ένα πιο άμεσο ορισμό καθολικότητας για την περίπτωση των μH συστημάτων: θέλουμε το καθολικό σύστημα να εξομοιώνει οποιοδήποτε splicing σύστημα, και όχι οποιαδήποτε μηχανή Turing.

Ο ακόλουθος ορισμός ανταποκρίνεται σε αυτήν την απαίτηση μας:

Ορισμός 3.3.1 Έστω αλφάβητο T και η κλάση H των επεκταμένων μH συστημάτων. Ένα στοιχείο της H της μορφής $\gamma_u = (V_u, T, A_u, R_u)$ όπου V_u αλφάβητο τέτοιο ώστε $T \subseteq V_u$, $A_u \subseteq V_u^*$ και R_u είναι ένα σύνολο splicing κανόνων πάνω στο V_u, λέγεται *καθολικό (universal)* για την κλάση H αν για κάθε $\gamma \in \mathsf{H}$ υπάρχει ένα string $w_\gamma \in V_u^*$, έτσι ώστε $L(\gamma) = L(\gamma'_u)$ όπου $\gamma'_u = (V_u, T, A_u \cup \{w_\gamma\}, R_u)$.

Το w_γ είναι ο κώδικας του τυχαίου γ, είναι δηλαδή ένα «πρόγραμμα» που μπορεί να εκτελεστεί από το γ_u, έτσι ώστε το τελευταίο να παράγει την ίδια γλώσσα με το γ. Τα αξιώματα στο A_u μπορούν να θεωρηθούν ως το «λειτουργικό σύστημα» του γ_u.

Το επόμενο θεώρημα αποδεικνύει την ύπαρξη universal μH συστημάτων.

Θεώρημα 3.3 Για κάθε αλφάβητο T υπάρχει ένα επεκταμένο μH σύστημα, τύπου $(\mu[1], FIN)$, με μόνο δύο βοηθητικά σύμβολα, που είναι καθολικό για την κλάση των επεκταμένων μH συστημάτων του τύπου $(\mu FIN, FIN)$ με τερματικό αλφάβητο T.

Απόδειξη. Έστω ένα αλφάβητο T και δύο σύμβολα c_1, c_2 που δεν ανήκουν στο T. Στο κεφάλαιο 2 του Μέρους ΙΙ ειπώθηκε ότι υπάρχουν καθολικές γραμματικές Chomsky για δοσμένο τερματικό αλφάβητο. Έστω $G_u = (N_u, T, P_u)$ μια καθολική τύπου-0 γραμματική.

Ακολουθώντας την απόδειξη του θεωρήματος 3.2.3, κατασκευάζουμε ένα επεκταμένο μH σύστημα $\gamma_1 = (V_1, T, A_1, R_1)$ που να εξομοιώνει την G_u. Το αξίωμα

w_0 δεν το λαμβάνουμε υπόψη στη κατασκευή του γ_1. Αυτό σημαίνει ότι το A_1 έχει λέξεις με άπειρη πολλαπλότητα.

Ένα λήμμα που χρειάζεται για να προχωρήσουμε είναι το παρακάτω:

Λήμμα: $EH_2(\mu FIN(\infty),[m]) \subseteq EH_2(\mu[1],[m])$

Το λήμμα λεει ότι κάθε μH σύστημα που έχει πεπερασμένο σύνολο αξιωμάτων και το κάθε αξίωμα με άπειρη πολλαπλότητα,(και σύνολο κανόνων με ακτίνα m) τότε υπάρχει ένα μH σύστημα με ένα μόνο αξίωμα (και άπειρη πολλαπλότητα) και με την ίδια ακτίνα του συνόλου κανόνων, που να παράγει την ίδια γλώσσα.

Άρα μπορούμε να πάρουμε ένα σύστημα $\gamma_2 = (V_2, T, A_2, R_2)$ με $|Supp(A_2)| = 1$ και $Rad(R_2) = m$ και τέτοιο ώστε $L(\gamma_2) = L(\gamma_1)$. Αποδεικνύεται ότι είναι δυνατόν να κωδικοποιηθούν όλα τα σύμβολα $V_2 - T$ με λέξεις πάνω στο αλφάβητο $\{c_1, c_2\}$, με ένα τέτοιο τρόπο ώστε το σύστημα

$\gamma_u = (\{c_1, c_2\} \cup T, T, A_u, R_u)$ να είναι το καθολικό μH σύστημα που θέλουμε.

Τα A_u, R_u προκύπτουν από τα A_2, R_2 αντίστοιχα, αντικαθιστώντας τα σύμβολα του $V_2 - T$ με τις λέξεις από το $\{c_1, c_2\}$.

Πράγματι έστω ένα τυχαίο επεκταμένο μH σύστημα $\gamma_0 = (V, T, A, R)$. Εφόσον $L(\gamma_0) \in RE$ υπάρχει μια γραμματική τύπου-0 έστω η $G_0 = (N_0, T, S_0, P_0)$ και τέτοια ώστε $L(\gamma_0) = L(G_0)$. Έστω ο κώδικας $w(G_0)$ που χρησιμοποιείται από την G_u για να εξομοιώσει την G_0. Έστω $w_0' = X_1^2 Y w(G_0) X_2^2$ το αξίωμα που αντιστοιχεί στο w_0 της απόδειξης του θεωρήματος 3.2.3. Έστω $w(\gamma_0)$ η κωδικοποίηση του w_0' χρησιμοποιώντας τις κατάλληλες λέξεις από το $\{c_1, c_2\}$. Τότε $L(\gamma_0) = L(G_0) = L(w(G_0), G_u) = L(\gamma_u')$ για

$$\gamma_u' = (\{c_1, c_2\} \cup T, T, A_u \cup \{(w(\gamma_0),1)\}, R_u).$$

Η τελευταία ισότητα μπορεί να εξεταστεί εύκολα, αν δούμε την δουλειά που κάνει το γ_u', που εξομοιώνει την G_u.

Σύμφωνα με αυτό το τελευταίο θεώρημα, η ύπαρξη Universal DNA υπολογιστών έχει τουλάχιστον θεμελιώδης θεωρητική υπόσταση. Στην πράξη βέβαια υπάρχουν προβλήματα σχετικά με το πλήθος των ενζύμων και το μήκος της ακολουθίας αναγνώρισης για το κάθε ένζυμο, στην προσπάθεια να φτιάξουμε ένα DNA υπολογιστή που να ανταποκρίνεται στο θεωρητικό μοντέλο γ_u. Ας ελπίσουμε ότι η πρόοδος της βιοτεχνολογίας θα κάνει το έδαφος πιο γόνιμο για την υλοποίηση του Universal DNA υπολογιστή.

4. Ανοιχτά προβλήματα και μελλοντικές κατευθύνσεις στον χώρο των DNA-υπολογιστών

Τα ανοιχτά προβλήματα που ξεπήδησαν από την πρωτοποριακή εργασία του Adleman[1] ήταν πολλά . Εντούτοις αρκετά από αυτά έχουν αντιμετωπιστεί επιτυχώς, ενώ άλλα εξακολουθούν να βρίσκονται υπό μελέτη. Μια κατηγοριοποίηση των ανοιχτών προβλημάτων, μπορεί να γίνει ως προς την σχετική συνάφεια που έχουν με την θεωρητική ή την εφαρμοσμένη πλευρά των DNA-υπολογιστών. Μπορούμε δηλαδή να μιλάμε για ανοιχτά προβλήματα που άπτονται περισσότερο της θεωρητικής επιστήμης των υπολογιστών, και για ανοιχτά προβλήματα που άπτονται περισσότερο επί της βιοτεχνολογίας. Όπως μπορεί κάποιος εύκολα να συμπεράνει, αυτή η κατηγοριοποίηση δεν είναι απόλυτη, με την έννοια ότι μια εξέλιξη στον τομέα της επιστήμης των υπολογιστών, μπορεί να λύσει με έμμεσο τρόπο κάποιο πρόβλημα της βιοτεχνολογίας(πάντα σχετικό με την πρακτική εφαρμογή των DNA-υπολογιστών). Το παράδειγμα για το μήκος της ακολουθίας αναγνώρισης των ενζύμων, που δίνεται πιο κάτω, είναι σαφής ένδειξη της μη απόλυτης υπόστασης της πιο πάνω κατηγοριοποίησης.

Τα πιο σπουδαία ανοιχτά προβλήματα που άπτονται περισσότερο επί της βιοτεχνολογίας είναι τα ακόλουθα:

α) Τα λάθη που γίνονται στις βιολογικές λειτουργίες όπως αυτές που περιγράφονται στην 2.2 του μέρους Ι. Αυτές οι λειτουργίες, αν και περιγράφονται ως επί των πλείστων για την in-vitro μορφή τους(δηλαδή πως επιτελούνται στο εργαστήριο), έχουν εντούτοις και την in-vivo μορφή τους(η πραγματοποίηση τους μέσα σε ζωντανά κύτταρα). Ενώ στην in-vitro πραγματοποίηση τους τα λάθη που γίνονται είναι πολλά και δυσκόλως ελεγχόμενα, στην in-vivo μορφή τους όποια λάθη γίνονται, ανιχνεύονται και διορθώνονται από διορθωτικούς μηχανισμούς του ζωντανού κυττάρου. Το πρόβλημα λοιπόν είναι πώς θα ελέγξουμε τα λάθη των βιολογικών λειτουργιών(μελετώντας ίσως τους διορθωτικούς μηχανισμούς των ζωντανών κυττάρων) ή πως θα χρησιμοποιήσουμε τα ζωντανά κύτταρα για να επιτελέσουν αυτά τις λειτουργίες. Αυτό το πρόβλημα είναι και το πιο σημαντικό ίσως και στις δύο κατηγορίες που αναφέρθηκαν στην αρχή αυτής της ενότητας. Τα λάθη στις βιολογικές λειτουργίες(in-vitro) καθιστούν σχεδόν αδύνατη την υλοποίηση των θεωρητικών μοντέλων Universal DNA υπολογιστών, και δημιουργούν αρκετά προβλήματα στα πρακτικά μοντέλα όπου ενδιαφερόμαστε για λύσεις σε πραγματικά προβλήματα(όπως το να βρούμε το κλειδί ενός κρυπτοσυστήματος, ή την ανάθεση που ικανοποιεί ένα προτασιακό τύπο).

Μιλώντας για λάθη στις βιολογικές λειτουργίες ίσως θα έπρεπε να γίνουμε πιο συγκεκριμένοι και αναφερθούμε σε συγκεκριμένες λειτουργίες. Για παράδειγμα:

- Στην λειτουργία της *εξαγωγής(affinity purification)* είναι δυνατόν κάποιο μόριο με την ζητούμενη υποακολουθία να μην *ανοπτηθεί(annealing)* στον αντίστοιχο ανιχνευτή του. Είναι δε πολύ πιθανό αυτό το μόριο να κωδικοποιεί μια λύση του προβλήματος. Για παράδειγμα στο βήμα 4 του αλγορίθμου του Adleman[1] όπου θέλουμε να βρούμε το Hamilton path, είναι δυνατόν να χαθεί κάποιο μόριο το οποίο να κωδικοποιούσε το Hamilton path. Βέβαια η ύπαρξη πλήθους μορίων που κωδικοποιούν την λύση είναι η εύκολη απάντηση σε αυτό το πρόβλημα(δεν γίνεται όλα να ξεφύγουν από τους ανιχνευτές). Αναγκαζόμαστε όμως έτσι να μιλάμε για πιθανότητα να υπάρχει η σωστή λύση μέσα στον τελικό δοκιμαστικό σωλήνα.
- Στην λειτουργία της ενίσχυσης μέσω αλυσιδωτής αντίδρασης πολυμεράσης(PCR amplification) μπορεί να ενισχυθεί η παρουσία μορίων που είναι αρκετά διαφορετικά από το μόριο που θέλουμε πραγματικά να ενισχυθεί.

Αυτό φαίνεται στη λειτουργία της *αντικατάστασης* όπου είναι αρκετή η μερική(όχι πλήρης) ομοιότητα με μια υποακολουθία του προς ενίσχυση μορίου, ώστε να γίνει η ανόπτηση του primer, και τελικά να γίνει αντιγραφή του μη επιθυμητού μορίου.

- Στη λειτουργία του διαχωρισμού κατά μήκος με gel electrophoresis, είναι δυνατόν κάποια μόρια με παραπλήσιο μήκος να έχουν την μη αναμενόμενη πορεία μέσα στο gel(δηλαδή αυτό με το μεγαλύτερο μήκος να αποκτήσει και τη μεγαλύτερη ταχύτητα, κάτι που δεν είναι επιθυμητό) .
- Στη λειτουργία της *συνένωσης με λιγάση(*με δημιουργία φωσφοδιεστερικού δεσμού) μπορεί είτε να μην πραγματοποιηθεί, είτε να συμβεί *blunt-end ligation* εκεί που δεν είναι επιθυμητό, ή να μην συμβεί εκεί που είναι επιθυμητό.

β) Το μέγιστο μήκος των μορίων DNA είναι ένα άλλο πρόβλημα. Για μήκος μορίων DNA μεγαλύτερο από 15000 νουκλεοτίδια, υπάρχει σημαντικό πρόβλημα ακόμα και στις απλές βιολογικές λειτουργίες όπως αυτή της ανάμειξης(merge, mixing). Τόσο μεγάλα σε μήκος μόρια μπορούν να σπάσουν(δίνοντας δύο μικρότερα σε μήκος μόρια) από τις μικρές μηχανικές δονήσεις των δοκιμαστικών σωλήνων, κατά την ανάμειξη. Επίσης είναι εξαιρετικά ασταθής η συμπεριφορά τους και στις υπόλοιπες λειτουργίες: τήξη, ανόπτηση, συνένωση κ.τ.λ.

Η σύνθεση των μορίων DNA νουκλεοτίδιο προς νουκλεοτίδιο είναι επίσης ένα δύσκολο πρόβλημα, που άπτεται βέβαια καθαρά επί της βιοτεχνολογίας. Αυτή η σύνθεση είναι ένα αναπόφευκτό στάδιο, από τη στιγμή που είτε πρέπει να κωδικοποιήσουμε το στιγμιότυπο του προβλήματος σε μόρια DNA, είτε να παράγουμε τα αξιώματα και τους κανόνες, ως μόρια DNA, στην περίπτωση των θεωρητικών μοντέλων.

γ) Το μήκος της ακολουθίας αναγνώρισης των ενζύμων είναι ένα ακόμη πρόβλημα. Όπως είδαμε ένα ένζυμο αντιστοιχεί σε ένα κανόνα splicing(για την ακρίβεια ένας κανόνας splicing θέλει δύο ένζυμα με διαφορετικές ακολουθίες αναγνώρισης, για να υλοποιηθεί). Η δημιουργία ενζύμων είναι εξαιρετικά δύσκολη, και το μήκος της ακολουθίας αναγνώρισης είναι σχετικά μικρό(6-7 νουκλεοτίδια). Όπως είδαμε αυτό το πρόβλημα τείνει να αντιμετωπιστεί με τη βοήθεια της επιστήμης των υπολογιστών, δημιουργώντας μοντέλα που απαιτούν μικρή σε μήκος ακολουθία αναγνώρισης. Είδαμε στη περίπτωση των Universal συστημάτων splicing, ότι οι κανόνες έχουν ακτίνα το πολύ 5, κάτι που σημαίνει ότι μπορούμε να απαιτούμε από την ακολουθία αναγνώρισης να έχει μήκος μικρότερο του 10.

δ) Ο αριθμός των ενζύμων που μπορούν να δρουν παράλληλα. Ένα άλλο δύσκολο πρόβλημα της βιοτεχνολογίας, αφού η παρουσία πολλών ενζύμων μέσα σε ένα δοκιμαστικό σωλήνα κάνει απρόβλεπτη τη συμπεριφορά των ενζύμων.

Τα ανοιχτά προβλήματα που άπτονται περισσότερο επί της επιστήμης των υπολογιστών είναι τα ακόλουθα:

α) Το μήκος των μορίων DNA, η σύνθεση τους και το πλήθος τους. Αυτό το πρόβλημα είναι σε συσχέτιση με το πρόβλημα β) της πρώτης κατηγορίας. Εδώ θέλουμε να βρούμε ποιο είναι το βέλτιστο μήκος, ποια είναι η βέλτιστη σύνθεση τους (δηλαδή πως θα γίνει η κωδικοποίηση, δεδομένου ότι η σύνθεση περιορίζεται στα 4 νουκλεοτίδια), και πόσα αντίγραφα πρέπει να έχει το κάθε μόριο, ώστε να αντιμετωπιστούν λάθη των βιολογικών λειτουργιών, ή λάθη του αλγορίθμου. Για παράδειγμα στο πείραμα του, ο Adleman[1] αποφάσισε να κωδικοποιήσει με 20 νουκλεοτίδια τη κάθε πόλη, και επίσης καθόρισε και μια ακολουθία νουκλεοτιδίων για αυτή τη πόλη. Δηλαδή π.χ για τη πόλη C χρησιμοποίησε το μόριο GCTATTCGAGCTTAAAGCTA. Η ακολουθία με την οποία κωδικοποιείται η κάθε πόλη δε πρέπει να είναι εντελώς τυχαία. Πρέπει να είναι τέτοια ώστε να μειώνεται η

πιθανότητα δημιουργίας άσχετου μονοπατιού, λόγω μερικής ομοιότητας με το μόριο κάποιας άλλης πόλης, κατά την διάρκεια της αντίδρασης λιγάσης στο βασικό βήμα 1. Γενικά μόρια που χρησιμοποιούμε τα οποία έχουν μεγάλη ανομοιότητα μεταξύ τους, είναι και τα καλύτερα. Αλλά αν είμαστε αναγκασμένοι να χρησιμοποιήσουμε μεγάλο μήκος για να πετύχουμε την ανομοιότητα ερχόμαστε σε σύγκρουση με το πρόβλημα β) της πρώτης κατηγορίας. Μια αντιμετώπιση αυτού του προβλήματος συναντάται από τον E.Baum στο [13].

β) Ένα καθαρά θεωρητικό πρόβλημα, είναι αυτό που αφορά την δημιουργία του θεωρητικού μοντέλου του Universal DNA υπολογιστή. Όπως φάνηκε στο 3.3 του δεύτερου μέρους, έχουμε να κάνουμε με ένα splicing σύστημα με ένα αρχικό αξίωμα, αλλά με πεπερασμένους το πλήθος κανόνες. Το πλήθος των κανόνων δεν είναι φραγμένο και εξαρτάται από το πληθάριθμο του τερματικού αλφαβήτου. Το πιο καλό αποτέλεσμα είναι $O(n^4)$ πλήθος κανόνων για πληθάριθμο τερματικού αλφαβήτου n. Βέβαια γνωρίζουμε ότι η ακτίνα των κανόνων είναι το πολύ 5. Εναλλακτικά μπορεί να γίνει trade-off μεταξύ των κανόνων και των αξιωμάτων, οπότε μπορεί να κατασκευαστεί Universal splicing system με $O(n^2)$ πλήθος αξιωμάτων και επίσης $O(n^2)$ πλήθος κανόνων. Αυτή η περίπτωση μελετάται στο [15]. Για να επιτευχθεί κάτι καλύτερο ίσως θα πρέπει να κατασκευαστεί ένα διαφορετικό μοντέλο που να μη σχετίζεται με γραμματικές Chomsky και μηχανές Turing.

Το μέλλον των DNA υπολογιστών εξαρτάται πολύ από την πρόοδο της βιοτεχνολογίας, τουλάχιστον για την περίπτωση των in-vitro υπολογιστών. Ας μην ξεχνάμε ότι οι ψηφιακοί ηλεκτρονικοί υπολογιστές χρωστάνε πολλά στις εξελίξεις της ηλεκτρονικής/μικροηλεκτρονικής. Σε περασμένες δεκαετίες ήταν άκομψοι, καταλαμβάνοντας τεράστιο χώρο, και εξαιρετικά ευαίσθητοι σε βλάβες. Η πρόοδος της ηλεκτρονικής βελτίωσε κατά πολύ τη κατάσταση και έφερε την επανάσταση στις multimedia δυνατότητες. Οι DNA υπολογιστές δε προσφέρονται για ικανοποίηση multimedia αναγκών, αλλά για επίλυση προβλημάτων με αχανή χώρο λύσης. Τα παραπάνω προβλήματα είναι ενδεικτικά των δυσκολιών των DNA υπολογιστών. Οι μελλοντικές κατευθύνσεις στο χώρο του DNA-computing ακολουθούν την απαιτούμενη πορεία για την αντιμετώπιση των πιο πάνω προβλημάτων. Πιο συγκεκριμένα:

α) Η δημιουργία υβριδικών συστημάτων(DNA υπολογιστές+ρομποτική+ψηφιακοί υπολογιστές για έλεγχο). Οι ψηφιακοί υπολογιστές, ελέγχοντας τον ρομποτικό εξοπλισμό, μπορούν να μειώσουν τον χρόνο που απαιτείται για τις βιολογικές λειτουργίες, καθώς και την πιθανότητα λάθους.

β) Η εξερεύνηση μοντέλων in-vivo. Αυτά προσπαθούν είτε να μιμηθούν, είτε να χρησιμοποιήσουν ζωντανά κύτταρα, για την επιτέλεση των επιμέρους λειτουργιών στις οποίες σπάει ο υπολογισμός. Είναι γνωστό ότι τα κύτταρα έχουν μηχανισμούς ανίχνευσης και διόρθωσης των λαθών, όποτε αυτά παρουσιάζονται. Τέτοια μοντέλα είναι τα P-συστήματα(P-systems)[17].

γ) Η δημιουργία καλύτερων θεωρητικών μοντέλων, που είναι πιο ανεκτικά στα λάθη των βιολογικών λειτουργιών. Αυτό μπορεί να γίνει είτε ρυθμίζοντας τις παραμέτρους των υπάρχοντων μοντέλων(όπως πλήθος κανόνων στα splicing systems) είτε να δομηθούν εξαρχής καινούρια μοντέλα(όπως τα P-systems) που δε περνάνε κατά ανάγκη από τις γραμματικές Chomsky.

δ) Οι προσεγγίσεις στα περισσότερα πρακτικά μοντέλα βασίζονται στη μέθοδο της εξαντλητικής αναζήτησης. Είναι δυνατό να γίνει κάτι πιο έξυπνο, όπως να εφαρμοστούν μέθοδοι δυναμικού προγραμματισμού ή divide-conquer. Μια τέτοια προσέγγιση ακολουθείται στο [16].

Ασχέτως από το ποια θα είναι η τελική κατάληξη των DNA υπολογιστών, η δημιουργία τους σηματοδότησε τις πρώτες σημαντικές εξελίξεις στο τομέα του υπολογισμού σε μοριακό επίπεδο. Οι ψηφιακοί ηλεκτρονικοί υπολογιστές κινούνται προς την ίδια κατεύθυνση. Πολύ πρόσφατα αποτελέσματα στο τομέα των ψηφιακών ηλεκτρονικών υπολογιστών αναφέρουν την δημιουργία λογικών πυλών σε μοριακό επίπεδο(χρησιμοποιώντας άτομα άνθρακα(όπως στα οργανικά μόρια DNA) και όχι άτομα πυριτίου από τα οποία είναι κατασκευασμένα τα υπάρχοντα ολοκληρωμένα κυκλώματα). Ακόμα λοιπόν και αν οι ψηφιακοί ηλεκτρονικοί υπολογιστές κυριαρχήσουν και σε αυτό το τομέα, οι DNA υπολογιστές θα είναι οι πρώτοι που σηματοδότησαν τις εξελίξεις.

Αναφορές

[1] L.Adleman. Molecular computation of solution to combinatorial problems. *Science*, 266:1021-1024, Nov.11,1994.

[2] R.J.Lipton. Using DNA to solve NP-complete problems. *Science* ,268:542-545, Apr.28,1995.

[3] Dan Boneh, Christopher Dunworth, Richard Lipton,Jiri Sgall. On the computational power of DNA.

[4] E.Winfree. On the Computational Power of DNA Annealing and Ligation. Available at http://dope.caltech.edu/winfree/DNA.html.

[5] D.Beaver. A Universal molecular computer. Technical report CSE-95-001, Penn State University,1995.

[6] J.Reif Parallel Molecular Computation. In proceedings of 7th Annual ACM Symposium on Parallel Algorithms and Architectures SPAA'95, 1995, pp.213-223.

[7] C.Papadimitriou. Private communications.

[8] S.Roweis, E.Winfree, R.Burgoyne, N.Chelyapov, M.Goodman,P.Rothermund,L.Adleman: A sticker based architecture for DNA computation. Proc. Of the Second Annual Meeting, Princeton, 1996,1-27.

[9] Gheorghe Paun, Grzegorz Rozenberg, Arto Salomaa:*DNA computing-New computing paradigms*. ISBN 3-540-64196-3 Springer – Verlag Berlin Heidelberg New York.

[10] L.M. Adleman, P.W.K. Rothermund, S.Roweis, E.Winfree: On applying molecular computation to the Data Encryption Standard.[8] 28-48.

[11] Douglas R. Stinson: Cryptography: Theory and Practice. ISBN 0-8493-8521-0, CRC Press, Boca Raton.

[12] Christos H. Papadimitrioy. Computational Complexity. ISBN 0-201-53082-1, Addison – Wesley Publishing Company.

[13] Eric B. Baum . DNA sequences usefull for computation. Available at http://www.wi.leidenuniv.nl\~pier\dna.html.

[14] Lila Kari. DNA computing:Arrival of biological mathematics. Available at http://www.wi.leidenuniv.nl\~pier\dna.html.

[15] C. Ferreti, G.Mauri, S.Kobayashi, T.Yokomori:On the universality of Post and splicing systems. Manuscript 1997.

[16] Eric B. Baum, Dan Boneh. Running dynamic programming algorithms on DNA computers. Available at http://www.wi.leidenuniv.nl\~pier\dna.html.

[17] Gh.Paun. Computing with Membranes. Journal of computer and system sciences,61,1(2000),108-143.

[18] R.P.Feynman. in Minaturization, D.H.Gilbert, Ed.(Reinhold Publishing Corporation, New York,1961),pp.282-296.