Yönlü Çizgelerin Türetilmesi - Karşılık Gelen Düzenli Deyimler
İlk Yaklaşım ve Örnek Çalışma

Fevzi Belli[1] Mutlu Beyazıt[2]

[1,2]Elektrik Mühendisliği ve Bilgi Teknolojisi Enstitüsü, Paderborn Üniversitesi, Paderborn, Almanya

[1]e-posta: belli@adt.upb.de [2]e-posta: beyazit@adt.upb.de

Özetçe

Yönlü çizgeler (*directed graphs*), *durum geçiş çizgeleri* (*state transition diagrams*), ya da *sonlu durum özdevinirleri* (*finite state automata*) olarak yorumlanmakta sistemlerin modellenmesinde çok yaygın kullanılmaktadır. Fakat modellenecek sistem büyük ve karmaşık oldukça yönlü çizgeler *kolay işlenemez* (*intractable*) hale gelmektedir. Biçimsel diller bağlamında, sonlu durum özdevinirleri ve *düzenli deyimler* (*regular expressions*) tip 3 (düzenli) dilleri kabul etme ve türetmeleri dolayısıyla birbirlerine denktir. Bu durumlarda, düzenli deyimlerin modelleme için *sıkı* (*compact*) biçiminden ve analiz için cebirsel işlemlerinden yararlanmayı önermekteyiz.

Bu bildiri *çizge değiştirmeleri*nin (*graph manipulations*) karşılık gelen düzenli deyim üzerindeki etkilerini incelemektedir. Çizge değiştirmeleri düğüm ve kenarları ekleyip çıkaran *işleçler* (*operators*) ile gerçekleştirilmektedir. Bu işleçlerin karışık ve/veya birden çok kez uygulanması birçok farklı yönlü çizge (ve buna karşılık gelen düzenli deyim) dönüşümlerini yaratır. Bu nedenle, bu dönüşümleri *asıl* (*original*) yönlü çizgenin (ve karşılık gelen düzenli deyimin) *türevl*eri (*mutant*s*)* olarak görmekteyiz.

Çalışmamizin şu anki aşaması *döngü* (*cycle*) içeren yönlü çizgeleri incelememektedir. Buna rağmen çalışmanın sonuçlarının *türetme ile tarama ve sınama* (*mutation analysis and testing*) yöntemine katkıda bulunacağını beklemekteyiz.

1. Giriş

Model tabanlı sınama tekniklerinin çoğu (yönlü) çizgeleri kullanmaktadır. Bu gerçek yazılım sınamasının öncülerinden Beizer tarafından şöyle dile getirilmektedir: "Bir çizge bul ve onu kapsa!" [1, 2]. *Çizge kapsama* (*graph coverage*) arkasındaki temel fikir *sınama örnerkleri*nin (*test cases*) ve aralarından ek küçük sayıda örneği seçerek *sınama kümesi*nin (*test set*) oluşturulmasını içermektedir. Bunun sebebi *sınama altındaki sistem*deki (*system under test*) yapısal ya da işlevsel sorunları daha az maliyetle ortaya çıkarmaktır. İyi bir *sınama kapsamı* (*test coverage*) kullanıcının sistem üzerindeki güvenini artırmakta ve sistemin yapması gerek her şeyi yaptığını göstermektedir (*olumlu sınama* (*positive testing*) [3]).

Uygulama yönelimli (*ak kutu* (*white-box*)) sınama yöntemi yönlü çizgedeki *düğüm*ler (*node*) sınama altındaki sistemin tümcelerini ve *kenar*lar (*arc*) tümcelerden oluşan *ardışım*ları (*sequence*) temsil etmektedir [4]. Tanım yönelimli (*kara kutu* (*black-box*)) sınama yöntemi ise düğümler söz konusu sisteme ait *davranışsal olayları* (*behavioral events*) ve kenarlar bu *olayların ardışımlarını* (*event sequences*) göstermektedir [5].

*Sınama altındaki sistem*i (*system under test*) modellemek için çizge kullanırken Belli ve diğerleri sistemin sadece yönlü çizge modelini değil ek olarak *tümleyen*ini (*complement*) de kapsamayı önermiştir. Böylece sistemin yapmaması gereken bir şeyi yapmadığı da gösterilebilmektedir (*olumsuz sınama* (*negative testing*) [3, 5]). Bu sebeple adı geçen yazarlar *çizge değiştirme işleçleri* (*graph manipulation operators*) ortaya atmışlardır. Olumsuz sınama, aslen ak kutu sınama tekniklerinden olan *türetme ile sınama*yla (*mutation testing*) yakından ilişkilidir. Ayrıca, son yıllarda türetme ile sınamanın model tabanlı kara kutu sınama tekniği olarak kullanılabilineceği de gösterilmiştir [6].

Türetme ile sınama yöntemi, sisteme ait bir program, tanım veya model ile birlikte türetme işleçleri kümesine gereksinim duyar. Geleneksel uygulama yönelimli yaklaşımda, program veya tanım üzerinde *türetme* işleçleri kullanılarak bazı değiştirmeler meydana getirilir ve türevler üretilir. Daha önceden üretilmiş olan sınama kümeleri bu türevler üzerine uygulanır. Diğer taraftan, model yönelimli yaklaşımda, söz konusu sisteme ait model ele alınır ve bu model üzerinde türetme işlemleri gerçekleştirerek üretilen *türevler*den (*mutants*) sistemde denenmek üzere sınama örnekleri oluşturulur. Her iki yaklaşım hem üretilen sınama kümelerinin etkinliğini ve yeterliliğini ölçmek hem de sistemdeki hataları ortaya tespit etmek için kullanılır. Ayrıca, bahsedildiği üzere, yönlü çizge ve türevi yapılar her iki yaklaşımda da sınama altındaki sistemi temsil etmek için kullanılabilir.

Karmaşık sistemleri modellerken karşılaşılan önemli bir sorun temsili çizgelerin hızlı bir şekilde büyümesi ve çalışılması zor hale gelmesidir. Eğer temsili çizge bir sonlu durum devinirinin geçiş çizgesi olarak gösterilebiliyorsa, çizge modelini düzenli deyim karşılığına çevirmek ve böylece geniş ve büyük çizgeler kullanmaktansa sistemi cebirsel ve daha sıkı bir biçim aracılığıyla temsil etmek büyük bir kolaylık sağlayabilir. Bu süreçte iyi bilinen algoritmalar yönlü çizgeyi düzenli deyime ve düzenli deyimi yönlü çizgeye çevirmek için kullanılabilir [7, 8, 9].

Düzenli deyimler aynı zamanda özel bir sınıfa ait dillerin türetilmesinde kullanılmaktadır. Bu sınıfa ait dillere *düzenli diller* (*regular languages*) denilmektedir. Yönlü çizgeyi düzenli deyime ve düzenli deyimi yönlü çizgeye çevirmek için kullanılan algoritmalar sonlu durum devinirlerinin ve düzenli deyimlerin birbirlerine dönüştürülebilirliğini göstermektedir. Bu durum sonlu durum devinirlerinin düzenli dilleri düzenli deyimler kadar iyi bir şekilde temsil edebildiğini ve biçimsel diller açısından sonlu durum devinirleri ile düzenli deyimler arasındaki denkliği göstermektedir [10].

Türetme ile sınama kapsamında, temel değiştirme işleçlerinin [6] bir yönlü çizgeye uygulanması o yönlü çizgeyi başka bir yönlü çizgeye dönüştürmektedir. Bu da karşılık

gelen düzenli deyimin büyük olasılıkla değişmesine sebebiyet vermektedir. Tersi düşünüldüğünde de değiştirilmiş bir düzenli deyime karşılık gelen yönlü çizge aslından farklı olmaktadır. Çalışmamızın ana hedeflerinden biri türetme ile sınama tekniğinin etkinliğini artırmak için yönlü çizge ve düzenli deyim değiştirmeleri arasında ilişkiler ortaya koymaktır. Çünkü bir çok çizge benzeri yapı (yönlü çizge, (genişletilmiş) sonlu durum özdevinirleri, olay ardışım çizgeleri, durum çizgeleri vb.) sınama altındaki sistemi modellemekte kullanılmakta ve bu yapıların ortak bir noktası olarak hepsine ait (genişletilmiş) düzenli deyim karşılıkları bulunmaktadır.

Bilgimiz dahilinde karşılık gelen düzenli deyimlerin yönlü çizgelerdeki türetme işleçleri sonucu oluşan değişimleri yansıtacak şekilde işlenilmesini (ve bunu tersini) ele alan bir çalışma bulunmamaktadır. Fakat düzenli deyimlerin cebirsel olarak dönüşümlerini işleyen ve söz edilmesinde fayda görülen bazı çalışmalar bulunmaktadır [11, 12, 13].

Kısaca tekrar vurgulamak gerekirse düzenli deyimlerin kullanılması farklı görsel gösterimlerin cebirsel ve sıkı bir çatı altında birleştirilmesini sağlayacaktır. Ayrıca, elde edilecek değiştirme, sınama örnekleri oluşturma vb. algoritmaların etkinliklerine bağlı olarak bu yaklaşım mevcut yaklaşımlara tamamlayıcı ve belki ek bir yöntem olarak ele alınabilecektir.

Bildirinin geri kalanı şu şekilde düzenlenmiştir: Kısım 2 bu çalışmada kullanılan kavramları tanıtmakta, Kısım 3 tanımlanmış soruna karşı şu anki yaklaşımımızı içermekte ve takiben Kısım 4'te bu yaklaşımı ele alarak üretilmiş algoritmaların örnek bir çalışma üzerinde kullanımlarını incelemektedir. Kısım 5 ise önemli bulunan bazı ilk aşamaya dair sonuçları iletmekte ve ilerisi için düşünülen araştırmadan bahsederek tartışmayı bitirmektedir.

2. Kavramlar

Bu bölüm izleyen tartışma için gerekli kavramları kısaca ve çoğu kez biçimsel olmayan bir şekilde tanıtmaktadır.

2.1. Yönlü Çizgeler ve Düzenli İfadeler

Tanım 1: Bir *yönlü çizge* (*YÇ*), sonlu *düğümler* kümesi ve sonlu *yönlü kenarlar* kümesinden oluşur ve (V, A) ile gösterilir. $V = \{v_1, \ldots, v_n\}$ düğümler kümesi için, her $a_i = (v_j, v_k)$ olmak üzere, $A = \{a_1, \ldots, a_m\} \subseteq V \times V$ yönlü kenarlar kümesini temsil eder.

Tanım 2: Bir *düzenli deyim* (*DD*) bir *alfabe*ye ait *simge*lerden türetilir ve belli bir *dizgiler* (*strings*) kümesini temsil etmek için kullanılır. İşlemsel bir bakış açısıyla, bir düzenli deyim bir simgeler dizisinin aşağıdaki işlemler kullanılarak birleştirilmesinden oluşturulabilir:

- *ardışım* (*sequence)* ("."), genelde işlem simgesi göz ardı edilebilir. Örnek: "*ab*", "*b, a*'yı takip eder demektir."),
- *seçim* (*selection*) ("+". Örnek: "*a+b*", "*a* veya *b*" demektir.),
- *yineleme* (*iteration*) ("*", *Kleene Yıldız İşlemi* (*Kleene's Star Operation*). Örnek:
  - -- "*a**", "*a* istenildiği kadar yinelenebilir" demektir.
  - -- "a^+", "*a* en azından bir defa yinelenmelidir" anlamına gelir).

Bu işlemler basit simgeler yanı sıra DD'lere de uygulanabilir. Beklenildiği üzere, *ayraçlama* (*parenthesization*) işlemi istenen öncelik ve birleşme sırasını belirlemek için kullanılır. Ayrıca DD'ye çevrilebilecek herhangi bir *P* yapısının DD karşılığını *RE(P)* ile gösterilmektedir.

Şekil 1 örnek bir YÇ ve onun DD karşılığını göstermektedir. Bir YÇ'nin DD gösterimini tanımlamak için, bazı düğümlerin *başlangıç* düğümleri ve *bitiş* düğümleri olarak nitelendirilmesine gerek duyulmaktadır. Bu bağlamda, düğümler kümesi alfabe olarak düşünebilir, ve DD tarafından temsil edilen dizgiler YÇ'deki başlangıç ve bitiş düğümlerini birleştiren yollar olmaktadır. Bu kullanım *olay ardışım çizgeleri*ni (*event sequence graphs*) tanımlamak için ortaya atılmıştır [5].

2.2. Yönlü Çizge Değiştirme İşleçleri

Bir YÇ'nin değiştirilmesi için tanımlanacak temel işleçler iki ana sınıfta toplanabilir: *Ekleme* (*insertion*) (*i*) ve *çıkarma* (*omission*) (*o*). Ayrıca bir YÇ basitçe düğüm ve kenarlar içerdiğinden, bu işleçler *düğüm ekleme* (*node insertion*) (i_n), *düğüm çıkarma* (*node omission*) (o_n), *kenar ekleme* (*arc insertion*) (i_a) ve *kenar çıkarma* (*arc omission*) (o_a) olarak düşünülebilir.

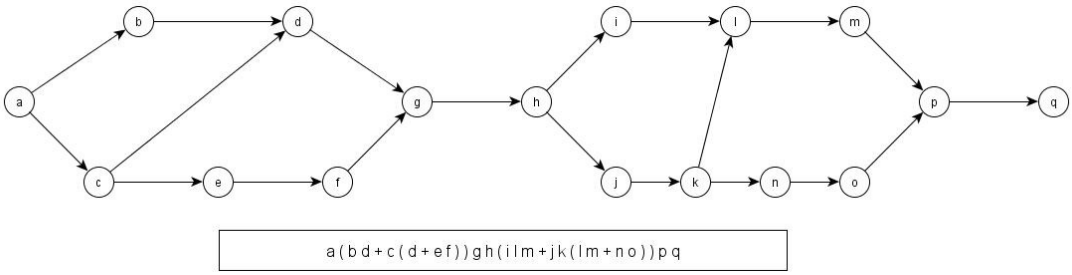
Tanım 3: YÇ değiştirme işleçleri bir YÇ'yi başka bir YÇ'ye dönüştürür ve izleyen şekilde tanımlanır:

- *Kenar ekleme* işleci YÇ'ye yeni bir (v_j, v_k) kenarı koyar:

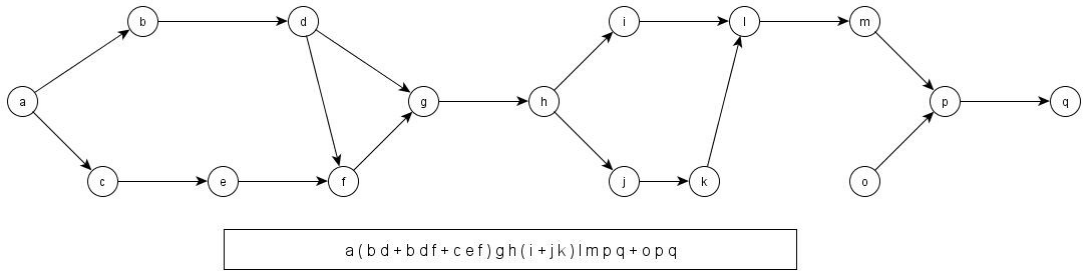
$$(v_j, v_k)\ i_a : (V, A) \rightarrow (V, A \cup \{(v_j, v_k)\}).$$

- *Kenar çıkarma* işleci YÇ'den mevcut bir (v_j, v_k) kenarını siler:

$$(v_j, v_k)\ o_a : (V, A) \rightarrow (V, A \setminus \{(v_j, v_k)\}).$$



Şekil 1: Örnek bir YÇ ve on karşılık gelen DD.



Şekil 2: Şekil 1'deki YÇ'ye $(cd)o_a(df)i_a(n)o_n$ İşlemlerinin Uygulanmasıyla Elde Edilmiş YÇ ve DD.

- *Düğüm ekleme* işleci yeni bir $v \notin V$ düğümünü YÇ'ye bu düğümü diğer düğümlere bağlayan kenarlarla birlikte koyar:

$$(v, \{a_1, \ldots, a_k\})\ i_n : (V, A) \rightarrow (V \cup \{v\}, A \cup \{a_1, \ldots, a_k\}).$$

- *Düğüm çıkarma* işleci var olan bir $v \in V$ düğümünü YÇ'den bu düğüme gelen ve bu düğümden giden kenarla birlikte siler:

$$(v)\ o_n : (V, A) \rightarrow (V \setminus \{v\}, A \setminus \{a_1, \ldots, a_k\}).$$

Şekil 2, Şekil 1'deki YÇ'ye basit bazı değiştirme işlemlerinin uygulanması sonucu elde edilmiştir.

2.3. Çarpımların Toplamı Biçimi ve Yardımcı İşlevler

DD'ler üzerinde dönüşümleri algoritmasal olarak gerçekleştirebilmek için, DD'ler için Boole Cebiri'ndekine benzer bir standart gösterim ve DD'ler üzerinde işleyen bazı yardımcı işlevler tanımlanması uygun bulunmuştur.

Tanım 4: Eğer bir DD sonlu sayıda *çarpım terimi*nin (*product term*) toplamı şeklinde gösterilmişse o DD *çarpımların toplamı biçimi*nde (*sum of products format*) (*ÇTB*) olmaktadır. Aşağıdaki biçimlerden herhangi birinde olan terime ise çarpım terimi denir:

- *r*
- *R**
- *r* ve/veya *R** terimlerinin sonlu sayıda ardışımlanması. Örnek: *rR**, *R*r*, *rR*rR** vb.

Burada, *r*, yalnızca simgelerin ardışımlanmasıyla oluşan sonlu bir dizgi ve *R*, ÇTB'de bir DD'dir. Dikkat edilecek olursa, ÇTB sıkılığı göz ardı eden gayet basit ve net bir biçimdir.

Tanım 5: R bir DD (ya da çarpım terimleri kümesi), *P* bir çarpım terimleri kümesi ve *s* bir dizgi olmak üzere aşağıda yardımcı işlevler tanımlanmaktadır:

- *pt(R, s)*, *s* dizgisini içeren çarpım terimlerinin kümesidir.
- *ht(P, s)*, *P* kümesinde verilen çarpım terimlerinden her birinin içinde *s* dizgisinin ilk konumuna kadar olan başlangıç alt terimini içeren terimler kümesidir.
- *tt(P, s)*, *P* kümesinde verilen çarpım terimlerinden her birinin içinde *s* dizgisinin son konumundan başlayan bitiş alt terimini içeren terimler kümesidir.

3. Yaklaşım

Elimizdeki sorun en genel halinde YÇ ve DD değiştirme işlemleri arasındaki karşılığı ortaya çıkarmaktır. Daha kesin olmak gerekirse, asıl amaç herhangi bir YÇ ve onun DD temsili verildiğinde YÇ'de tanımladığımız basit değiştirme işlemleri sonucunda oluşan dönüşümleri DD'ye yansıtmak ve aynı zamanda bunun tersini gerçekleştirmektir. Bu sebeple, ilk aşamada iki temel varsayımla soruna yaklaşılmaktadır: (1) Asıl ve dönüştürülmüş YÇ'ler döngü içermemektedir. (2) Tüm DD'ler ÇTB'dedir. Bunların yanında, DD değiştirme işlemleri ve bu işlemlerin karşılık gelen YÇ'deki etkilerinin araştırılması daha ileri aşamalara ertelenmiş olup bu aşamada sadece YÇ dönüşümlerinin DD üzerindeki etkileri ele alınmaktadır.

Yukarıda belirtilen varsayımlar doğrultusunda bu kısımda basit YÇ değiştirme işlemlerini karşılık gelen DD'lerdeki dönüşümleri gerçekleştirerek ele alan algoritmalar verilmektedir (İspatlara yer darlığından otürü girilmemiştir).

3.1. Kenar İşleçleri

İzleyen kısımlarda kenar ekleme ve çıkarma işleçlerini gerçekleştiren algoritmalar gösterilmektedir. Verilen algoritmaların *en kötü zaman karmaşıklığı* (*worst case time complexity*) değerlerinden söz edilmesine rağmen bu değerlerin kanıtlarına görece kapsam dışı olacağından girilmemiştir.

3.1.1. Kenar Ekleme

Algoritma 1, bir YÇ'ye kenar ekleme işlecinin uygulanması sonucu meydana gelen ve başlangıç ve bitiş düğümlerini birleştiren yeni yolların karşılık gelen DD'ye eklenmesini göstermektedir. Dikkat edilmesi gereken bir nokta DD'deki herhangi bir çarpım terimi simge v_j'yi v_i'den önce içermemelidir. Aksi takdirde, işlem bir döngü üretecektir.

Algoritma 1: Kenar Ekleme

Girdi: *R* – ÇTB'de bir DD,
(v_i, v_j), $v_i, v_j \in V$ – eklenecek kenar
Çıktı: *R* – kenarın eklenmesiyle güncellenen ÇTB'de DD
$A = pt(R, v_i)$ //v_i'yi içeren çarpım terimleri kümesi
$B = pt(R, v_j)$ //v_j'yi içeren çarpım terimleri kümesi
$A' = ht(A, v_i)$ //*A* için başlangıç alt terimleri kümesi
$B' = tt(B, v_j)$ //*B* için bitiş alt terimleri kümesi
$C' = A'.B'$ //*A'* ve *B'* üzerinde küme ardışım işlemi
$R = R + RE(C')$ // *C'*'deki terimlerin eklenmesi

Algoritma 1'den anlaşılacağı gibi kenar ekleme işlemi sonucunda DD'ye yeni çarpım terimleri eklenmektedir. Eklenen terimlerin sayısı $|A'||B'|$'dir. Burada $|A'|$, başlangıç düğümlerinden düğüm v_i'ye götüren başlangıç alt terimlerinin sayısı ve $|B'|$ düğüm v_j'den bitiş düğümlerine götüren bitiş alt terimlerinin sayısıdır.

Algoritma 1'deki sonlu zamanda işletilebilmektedir ve $O(|P|^2\ |p|)$ en kötü zaman karmaşıklığına sahiptir. Burada $|P|$ DD'deki çarpım terimlerinin sayısı için bir üst sınır ve $|p|$ çarpım terimlerinin uzunluğu için bir üst sınırdır.

3.1.2. Kenar Çıkarma

Kenar çıkarma işlemi bazı düğümler için gelen/giden kenarları ortadan kaldırabilir. Bu düğümleri başlangıç/bitiş düğümleri olarak düşünmek gerekmektedir, çünkü izleyen kenar ekleme işlemleri bu düğümlerle bağlı yeni kenarlar meydana getirebilir. Bu doğrultuda, Algoritma 2, bir DD'yi temsil ettiği YÇ'ye uygulanan kenar ekleme işlecinin etkilerini yansıtacak şekilde güncellemektedir.

Algoritma 2: Kenar Çıkarma

Girdi: R – ÇTB'de bir DD,
(v_i, v_j), $v_i, v_j \in V$ – çıkarılacak kenar
Çıktı: R – kenarın çıkarılmasıyla güncellenen ÇTB'de DD
$A = pt(R, v_i)$ //v_i'yi içeren çarpım terimleri kümesi
$B = pt(R, v_j)$ // v_j'yi içeren çarpım terimleri kümesi
$C = pt(R, v_iv_j)$ //v_iv_j'yi içeren çarpım terimleri kümesi
$A' = \varnothing$
if $A = C$ **then**
$A' = ht(A, v_i)$ //A için başlangıç alt terimleri kümesi
endif
$B' = \varnothing$
if $B = C$ **then**
$B' = tt(B, v_j)$ //B için bitiş alt terimleri kümesi
endif
$C' = A' \cup B'$ //A' ve B' kümelerinin birleşimi
$R = R - RE(C)$ //C'deki terimlerin çıkarılması
$R = R + RE(C')$ // C''deki terimlerin eklenmesi

Algoritma 2'den yola çıkılarak eklenen çarpım terimlerinin sayısı $|A'|+|B'|$ ve çıkarılan çarpım terimlerinin sayısı $|C|$ olarak bulunabilir. $|A'|$ başlangıç düğümlerinden düğüm v_i'ye götüren başlangıç alt terimlerinin sayısı, $|B'|$ düğüm v_j'den bitiş düğümlerine götüren bitiş alt terimlerinin sayısı ve $|C|$ v_iv_j dizgisini içeren çarpım terimlerinin sayısıdır.

Algoritma 1'de olduğu gibi Algoritma 2'nin de en kötü zaman karmaşıklığı $O(|P|^2\ |p|)$ olarark gösterilebilir ve sonlanan bir algoritmadır.

3.2. Düğüm İşleçleri

Sonraki aşama olarak, ekleme ve çıkarma işleçleri düğümler üzerinde düşünülmektedir.

3.2.1. Düğüm Ekleme

Düğüm ekleme işleci kenar değişim işleçleriyle karşılaştırıldığında daha üst seviye bir işleçtir. Bunun nedeni, bu işlecin düğüm eklemenin yanı sıra eklenen düğümü diğer düğümlere bağlama işlevini de gerçekleştiriyor olmasıdır. Bunu yapmak için önce eklenen düğüm bir başlangıç ve bitiş düğümü olarak düşünülmekte ve daha sonra izleyen kenar ekleme işleçleri yürütülmektedir. Algoritma 3 verilen bir düğümle birlikte eklenen kenarların söz konusu DD'deki dönüşümünü gerçekleştirmektedir.

Algoritma 3: Düğüm Ekleme

Girdi: R – ÇTB'de bir DD,
$v_i \notin V$ – eklenecek düğüm
(v_i, x_j), $x_j \in V$, $j = 1, \ldots, s$ – eklenecek giden kenarlar
(y_k, v_i), $y_k \in V$, $k = 1, \ldots, t$ – eklenecek gelen kenarlar
Çıktı: R – düğümün eklenmesiyle güncellenen ÇTB'de DD
$R = R + v_i$ //v_i simgesini bir çarpım terimi olarak ekle
for each (v_i, x_j) **do**
(v_i, x_j) kenarını ekle ve R'yi güncelle //Algoritma 1
endfor
for each (y_k, v_i) **do**
(y_k, v_i) kenarını ekle ve R'yi güncelle //Algoritma 1
endfor
if $s \geq 1$ **or** $t \geq 1$ **then**
$R = R - v_i$ //v_i çarpım terimini çıkar
endif

Açıkça görülebileceği gibi, Algoritma 3 sonlu zamanda çalışmaktadır ve en kötü zaman karmaşıklığı $O((s+t)\ (|P|^2\ |p|))$'dir (s gelen kenarların sayısı ve t giden kenarların sayısıdır). Bunu yanında, döngü içermeyen bir YÇ'de $(s+t) \leq n$ eşitsizliği her zaman için sağlanmakta ve $|p|$ değerini n olarak seçmek mümkün olmaktadır.

3.2.2. Düğüm Çıkarma

Düğüm çıkarma işlemi söz konusu düğümle ilişkili kenarların çıkarılmasını da içerdiğinden (düğüm eklem işlemi gibi) kenar işleçlerine göre daha üst seviye bir işleç olarak düşünülebilir. Bir düğümü çıkarmak için önce o düğümün YÇ'nin geri kalanıyla olan bağını kesmek, daha sonra da onu YÇ'den atmak gereklidir. Algoritma 4 bu işlemi gerçekleştirmekte ve YÇ'ye karşılık gelen DD'yi düğüm çıkarma işlecinin etkilerini yansıtacak şekilde güncellemektedir.

Algoritma 4: Düğüm Çıkarma

Input: R – ÇTB'de bir DD,
$v_i \in V$ – çıkarılacak düğüm
(v_i, x_j), $x_j \in V$ – çıkarılacak tüm giden kenarlar
(y_k, v_i), $y_k \in V$ – çıkarılacak tüm gelen kenarlar
Çıktı: R – düğümün çıkarılmasıyla güncellenen ÇTB'de DD
for each (v_i, x_j) **do**
(v_i, x_j) kenarını çıkar ve R'yi güncelle //Algoritma 2
endfor
for each (y_k, v_i) **do**
(y_k, v_i) kenarını çıkar ve R'yi güncelle //Algoritma 2
endfor
$R = R - v_i$ //v_i çarpım terimini sil

Daha önce belirtildiği gibi kenar çıkarma işlemi ve son adımdaki terim silme işlemi sonlu zamanda gerçekleştirilebildiğinden, ve algoritmadaki döngüler döngü içermeyen bir YÇ'de en fazla n kez tekrar edildiğinden Algoritma 4 sonlanan bir algoritmadır. Ayrıca, en kötü zaman karmaşıklığı $O(k\ |P|^2\ |p|)$ ile gösterilebilir. Burada k çıkarılacak kenarların toplam sayısıdır ve $k < n$ eşitsizliğini sağlamaktadır. Son olarak, $|p| = n$ seçimi döngü içermeyen bir YÇ'de her zaman için geçerlidir.

4. Örnek Çalışma

Bu kısımda bahsedilen kavramların ve öne sürülen algoritmaların uygulanışları bir örnek çalışma üzerinde gösterilmektedir. Söz konusu sistem otomotiv endüstrisiyle ilgili olup bir aracın yönetim uç birimidir.

4.1. Kısa Bilgi

Söz konusu sistem RSM13 (Mercedenz-Benz'e ait bir "yol kenarı yüzey biçme makinesi") yönetim uçbirimi olup Şekil 3'te gösterilmektedir. Bu sistem birçok kere örnek çalışmalara konu olmuştur [14, 15, 16].

Şekil 3. RSM13: Yol Kenarı Yüzey Biçme Makinesi ve Yönetim Uçbirimi.

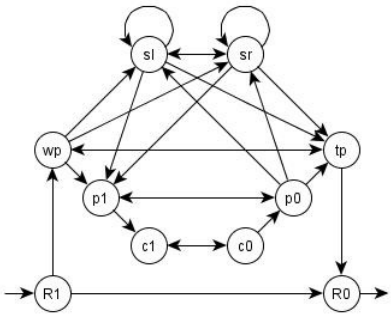
Yönetim uçbirimi aracılığıyla, biçme makinesi kılavuz direkleri, yol işaretleri, ağaçlar vb. etrafında biçme işlemi mümkün olduğunca iyi bir şekilde gerçekleştirilebilmekte, ve dönme noktasının eşsiz kinematiği ile özel tasarlanmış biçme kafası kılavuzu sayesinde çarpma bariyerleri arkasındaki büyük alanlara da rahatça ulaşılabilmektedir. Biçme biriminin kaydırılmasıyla çok dar alanlarda bile hassas ayarlama yapmak mümkün olmakta ve biçme kafası biçme sistemi ve iyi dizilmiş biçme birimleri sayesinde taşlardan korumaktadır. Araç yoğun trafik de bile kullanılabilmektedir.

Biçicinin çalışması taşıyıcının güç hidroliği ve ön tarafındaki güç biriminden etkilenmektedir.Yönetim uçbirimi biçicinin çalışma şeklini basitleştirmektedir. Örnek olarak, tek bir düğme basışıyla alet çalışma konumundan hareket konumuna geçirilebilmektedir. Ayrıca biçme kafasının konumu değişik şekillerde olabilmekte ve taşıyıcının sol ve sağ tarafları arasında çalışırken geçiş yapılabilmektedir. Konumlandırmanın yanı sıra, biçici birimin eğimi ve destek basıncı da yönetilebilmektedir.

Sistem temelde 4 farklı işleyiş moduna sahiptir:

- 1. mod devindiricilerin (actuators) yönetimi için kullanılmaktadır.
- 2. mod hareket (transport) ve çalışma (working) durumları arasında geçişi sağlamaktadır.
- 3. ve 4. modlar çalışma modları olmaktadır (Operation I, Operation II).

Şekil 4 RMS13'ün çalışmasını kısmen modelleyen bir YÇ göstermektedir [6]: Biçici çalışma konumundayken yönetim uçbirimi etkileşime izin vermekte ve biçme kafası sola veya sağa istenildiği kadar kaydırılabilmektedir. Basıncın açık veya kapalı olmasına bağlı olarak biçici aşağıda tutulabilmekte ve biçme işlemi başlatılabilmektedir. Biçicini çalıştırılabilmesi için basıncın açık olması gerekmektedir. Aksi takdirde, aracın etrafındaki nesneler zarar görebilir. Kesme işlemi tamamlandığında, biçici kapatılır, basınç kesilir ve araç hareket durumuna geçirilebilir.



Şekil 4: RSM13: Yönlü Çizge.

Şekil 4'te, düğümler ile olaylar arasında aşağıdaki gibi bir karşılık vardır:

- wp / tp: çalışma hali / hareket hali
- sl / sr: sola kaydırma / sağa kaydırma
- p1 / p0: basınç açık / basınç kapalı
- c1 / c0: biçici açık / biçici kapalı
- R1 / R0: mod etkin / mod etkin değil

Daha önce de bahsedildiği gibi YÇ'nin DD karşılığından söz edebilmek için bazı düğümlerin başlangıç ve bitiş düğümleri olarak belirlenmesi gerekmektedir. Bu örnekte, R1 başlangıç düğümü, R0 ise bitiş düğümüdür.

Şekil 4'teki YÇ'nin DD karşılığı aşağıdaki deyim ile gösterilebilir:

$$R1\ (wp\ (sl + sr + p1\ (c1\ c0)^*\ p0)^*\ tp)^*\ R0.$$

4.2. Döngülerin Atılması

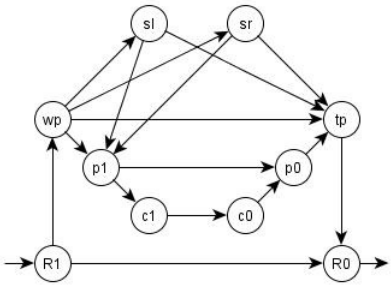
Önceki kısımda verilen algoritmaları örnek çalışma üzerinde işletebilmek için, modeldeki döngülerin ortadan kaldırılması gerekmektedir. Bu sebeple, (sl, sl), (sr, sr), (sl, sr), (sr, sl), (p0, sl), (p0, sr), (c0, c1), (p0, p1) ve (tp, wp) kenarları silinerek Şekil 5'teki YÇ elde edilmiştir. Yeni sistem eskisinin bir alt sistemidir, çünkü yeni sistemdeki tüm *düğüm ardışımları* (*node sequences*) eski sistemin içinde mevcuttur.

Diğer taraftan, silinen kenarların seçimleri keyfi görünmesine rağmen, bu seçimi yaparken sistemin tanımı gözetilmiştir: Modun etkin olduğunu ve yeni sistemin çalışma durumunda olduğunu varsayarsak, hareket haline geçmek için 3 temel yol mevcuttur:

1. Çalışma durumundan doğrudan hareket durumuna geçilebilir.

2. Basınç açıldıktan sonra, ya bir defa biçme işlemi gerçekleştirilip basınç kesilir ya da doğrudan basınç kapatılır ve hareket durumuna geçilir.
3. Bir defa sol veya sağ kaydırma işlemi gerçekleştirilir ve 1 ya da 2'de belirtilen şekilde devam edilir ve hareket durumuna geçilir.

Çalışma durumuna geri dönmek için, mod etkin olmayan duruma getirilmeli ve bir üst birimden tekrar etkinleştirilmelidir. Tabi şekilde de görüldüğü gibi mod etkinleştirildikten sonra çalışma durumuna geçilmeden etkinliği sonlandırılabilir.



Şekil 5: RSM13: Döngüsüz Yönlü Çizge.

Şekil 5'te verilen YÇ'nin DD gösterimi

$$R1\ (\varepsilon + wp\ (\varepsilon + sl + sr)\ (\varepsilon + p1\ (\varepsilon + c1\ c0)\ p0)\ tp)\ R0$$

şeklindedir. Buna denk olarak, DD'nin ÇTB'deki hali

$$\begin{aligned} R = {} & R1\ R0 + R1\ wp\ tp\ R0 + R1\ wp\ sl\ tp\ R0 + \\ & R1\ wp\ sr\ tp\ R0 + R1\ wp\ p1\ p0\ tp\ R0 + \\ & R1\ wp\ p1\ c1\ c0\ p0\ tp\ R0 + R1\ wp\ sl\ p1\ p0\ tp\ R0 + \\ & R1\ wp\ sl\ p1\ c1\ c0\ p0\ tp\ R0 + R1\ wp\ sr\ p1\ p0\ tp\ R0 + \\ & R1\ wp\ sr\ p1\ c1\ c0\ p0\ tp\ R0 \end{aligned}$$

ile verilir.

Seçilen örnek çalışmaya ait DD için aşağıdaki değerler seçilebilir:

- $n = 10$, düğümlerin sayısı.
- $|p| = 9$, çarpım terimlerinin uzunluğu için üst sınır (en küçük).
- $|P| = 10$, ÇTB'deki DD'deki çarpımın terimlerinin sayısına ait üst sınır (en küçük).

Ayrıca, Döngü içermeyen bir YÇ'de $n = 10$ için en kötü durumda değerler aşağıdaki gibi olmaktadır:

- $|p| = n = 10$.
- $|P| = 1 + n + n(n-1) + \ldots + n(n-1)\ldots 2 + n! = 9864101$.

4.3. Örnek Türetmeler

Bu başlık altında, YÇ değiştirme işleçlerinin örnek sistem üzerinde değişik türevler yaratmak için uygulanması ele alınmıştır. YÇ'lerde modellenen sistemin *anlamı* (*semantics*) göz önünde bulundurulmasa da burada türetme şekilleri sistem tanımı gözetilerek ciddi hataları yansıtacak düğüm ardışımları içeren türevler üretecek şekilde seçilmiştir.

4.3.1. Kenar Eklemeler

Belirtildiği gibi biçiciyi basıncı açmadan çalıştırmak tehlikeli bir durum oluşturmaktadır. Bu durumu modele yansıtmak için Şekil 5'teki YÇ'de farklı kenar ekleme işlemleri uygulanabilir. Bu doğrultuda *(R1, c1)*, *(wp, c1)*, *(sl, c1)* veya *(sr, c1)* kenarlarının eklenmesi değişik türevler oluşturmakta ve sistemde varsayımlar doğrultusunda herhangi bir döngüye sebebiyet vermemektedir.

İlk olarak, *(R1, c1)* kenarının eklenmesi sonucu sistemi temsil eden DD aşağıdaki şekilde değişmektedir:

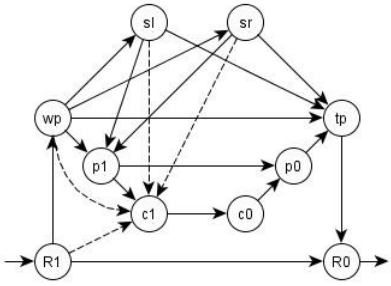
$$R' = R + \mathbf{R1\ c1\ c0\ p0\ tp\ R0}.$$

Burada eklenen çarpım terimi "kalın" olarak gösterilmiştir, ve üst sınır değerleri $|p| = 9$ ve $|P| = 11$ olmaktadır.

Ek olarak, *(wp, c1)*, *(sl, c1)* ve *(sr, c1)* kenarlarının da YÇ'ye eklenmesi sonucunda temsili DD (YÇ için Şekil 6'ya bakınız)

$$\begin{aligned} R'' = {} & R' + \mathbf{R1\ wp\ c1\ c0\ p0\ tp\ R0} + \\ & \mathbf{R1\ wp\ sl\ c1\ c0\ p0\ tp\ R0} + \mathbf{R1\ wp\ sr\ c1\ c0\ p0\ tp\ R0} \end{aligned}$$

olmakta ve üst sınır değerleri $|p| = 9$ ve $|P| = 14$ ile verilmektedir.



Şekil 6: Eklenen (Kesikli) Kenarlar Sonucu YÇ.

Belirtilmesi gereken önemli bir nokta sistemi temsil eden DD'de çarpım terimleri sayısı ($|P|$) bakımından en fazla genişlemeye sebep olan kenar ekleme işleminin *(p0, R0)* kenarının eklenmesi olduğudur. Bu işlem asıl DD'ye 6 yeni çarpım terimi eklemektedir.

4.3.2. Kenar Çıkarma

Kenar çıkarma işlemi için, YÇ'deki diğer düğümlerin hiç birini ziyaret etmeden biçicinin çalıştırılmasına olanak sağlayan iki türev türetilmektedir. İlk türetmede basıncı açtıktan sonra biçicinin işletilmesine olanak sağlayan kenar çıkarılmakta ve ikinci türetmede de birinciye ek olarak biçiciyi durdurduktan sonra basıncın kapatılmasını sağlayan kenar çıkarılmaktadır (Şekil 7).

(p1, c1) kenarının çıkarılması ilk türevi oluşturmakta ve aşağıdaki DD'yi türetmektedir:

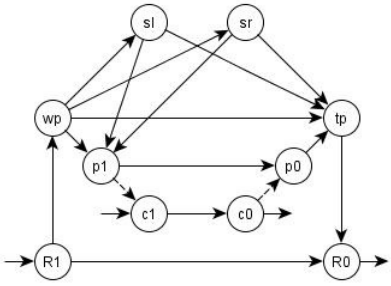
$$\begin{aligned} R' = {} & R + \mathbf{c1\ c0\ p0\ tp\ R0} - \\ & (R1\ wp\ p1\ c1\ c0\ p0\ tp\ R0 + \\ & R1\ wp\ sl\ p1\ c1\ c0\ p0\ tp\ R0 + \\ & R1\ wp\ sr\ p1\ c1\ c0\ p0\ tp\ R0). \end{aligned}$$

Burada "–" işleci çıkarılan çarpım terimlerini ve "kalın" kullanım da eklenen çarpım terimlerini vurgulamak için

kullanılmaktadır. Türetme sonunda $|p| = 7$ ve $|P| = 8$ olmaktadır.

İkinci türevi üretmek için yukarıdakine ek olarak *(c0, p0)* kenarı modelden çıkarılmaktadır. Bu işlem sonucunda $|p| = 7$ ile $|P| = 8$ değerleri değişmemekte ve DD aşağıdaki şekilde güncellenmektedir:

R'' = R' + ***c1 c0*** *–*
(c1 c0 p0 tp R0).



Şekil 7: Çıkarılan (Kesikli) Kenarlar Sonucu YÇ.

4.3.3. Düğüm Ekleme

Sistemde tehlikeye sebebiyet verecek diğer bir durum basınç kapatıldıktan veya hareket durumuna geçildikten sonra biçicinin çalıştırılabiliyor olmasıdır. Bu durumları modelleyen türevler üretebilmek için *(p0, c1)* ve/veya *(tp, c1)* kenarlarının modele eklenmesi düşünülebilir. Fakat bu kenar ekleme işlemleri YÇ'de döngü oluşmasına sebep olmaktadır ve şimdilik kapsamımız dışındadır. Bu sebeple, söz edilen durumları modellemek için kenar ekleme işlemi yerine düğüm ekleme işlemi *dizinleme* (*indexing*) ile birlikte kullanılmaktadır.

c1-2 düğümünün *(p0, c1-2)* ve *(c1-2, tp)* kenarları ile birlikte YÇ'ye eklenmesi basıncın kapatılmasından sonra biçicinin çalıştırılabilmesine izin vermekte ve türev aşağıdaki DD ile temsil edilmektedir:

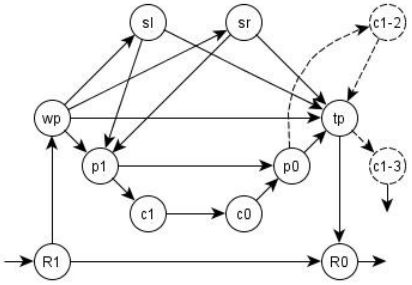
R' = R + ***R1 wp p1 p0 c1-2 tp R0 +***
R1 wp p1 c1 c0 p0 c1-2 tp R0 +
R1 wp sl p1 p0 c1-2 tp R0 +
R1 wp sl p1 c1 c0 p0 c1-2 tp R0 +
R1 wp sr p1 p0 c1-2 tp R0 +
R1 wp sr p1 c1 c0 p0 c1-2 tp R0*.*

Daha önce belirtildiği gibi "kalın" çarpım terimleri yeni eklenen terimleri göstermektedir. Düğüm ekleme işlemi sonucunda sahip olduğumuz üst sınır değerleri $|p| = 9$ ve $|P| = 16$ olmaktadır. Ekleme işlemi sonucunda düğümlerin sayısı artığından en kötü durum üst sınır değerleri $|p|$ için $n = 11$'e ve $|P|$ için $O(n!)$'lik formül uyarınca *108505112*'e yükselmektedir.

c1-3 düğümünün *(tp, c1-3)* kenarı ile takibi eklenmesi YÇ'yi Şekil 8'de gösterilen hale dönüştürmektedir. Temsili DD ise

R'' = R' + ***R1 wp tp c1-3 + R1 wp sl tp c1-3 +***
R1 wp sr tp c1-3 + R1 wp p1 p0 tp c1-3 +
R1 wp p1 c1 c0 p0 tp c1-3 + R1 wp sl p1 p0 tp c1-3 +
R1 wp sl p1 c1 c0 p0 tp c1-3 +
R1 wp sr p1 p0 tp c1-3 +
R1 wp sr p1 c1 c0 p0 tp c1-3 +
R1 wp p1 p0 c1-2 tp c1-3 +
R1 wp p1 c1 c0 p0 c1-2 tp c1-3 +
R1 wp sl p1 p0 c1-2 tp c1-3 +
R1 wp sl p1 c1 c0 p0 c1-2 tp c1-3 +
R1 wp sr p1 p0 c1-2 tp c1-3 +
R1 wp sr p1 c1 c0 p0 c1-2 tp c1-3

ile verilmektedir. Üst sınır değerleri $|p| = 10$ ile $|P| = 31$ ve bunlara ait en kötü durum değerleri $|p| = n = 12$ ile $|P| = 1302061345$ olmaktadır.



Şekil 8: Eklenen (Kesikli) Düğümler Sonucu YÇ.

4.3.4. Düğüm Çıkarma

Bu kısımda düğüm çıkarma işlecini kullanarak türevler üretmek için *c0* ve *p0* düğümleri ayrı ayrı YÇ'den çıkarılmaktadır. Oluşan türevlerden ilki biçicinin kapatılamadığı bir sistemi diğeri ise basıncın kapatılamadığı bir sistemi temsil göstermektedir.

c0 düğümünün çıkarılması *(c1, c0)* ve *(c0, p0)* kenarlarının çıkarılmasını da gerektirmektedir. Fakat *c1* düğümü bir bitiş düğümü olarak işaretleneceğinden Şekil 9'da *(c1, c0)* kenarı çıkarılmıyor gibi görünmektedir (*c0* düğümünün çıkarılmasıyla oluşan değişiklikler kesikli çizgilerle gösterilmektedir). Çıkarma işlemi sonucu oluşan sistemi temsil eden DD aşağıdaki şekildedir:

R' = R + ***R1 wp p1 c1 + R1 wp sl p1 c1 +***
R1 wp sr p1 c1 *– (R1 wp p1 c1 c0 p0 tp R0 +*
R1 wp sl p1 c1 c0 p0 tp R0 +
R1 wp sr p1 c1 c0 p0 tp R0).

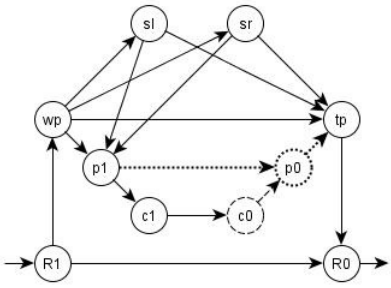
"Kalın" terimler yeni eklenen ve "–" işleci türevde olmayan çarpım terimlerini göstermektedir. Çıkarma işlemi sonucu $|p|$ değeri azalmakta ($|p| = 7$) ve $|P|$ değerinde bir değişiklik olmamaktadır. Bunun yanında, en kötü durum değerleri $|p| = 9$ ve $|P| = 986410$ olmak üzere azalma göstermektedir.

Diğer taraftan, *p0* düğümünün çıkarılması izleyen DD'nin oluşmasına sebep olmaktadır:

R'' = R + ***R1 wp p1 + R1 wp p1 c1 c0 +***
R1 wp sl p1 + R1 wp sl p1 c1 c0 + R1 wp sr p1 +
R1 wp sr p1 c1 c0 *– (R1 wp p1 p0 tp R0 +*
R1 wp p1 c1 c0 p0 tp R0 + R1 wp sl p1 p0 tp R0 +
R1 wp sl p1 c1 c0 p0 tp R0 + R1 wp sr p1 p0 tp R0 +
R1 wp sr p1 c1 c0 p0 tp R0).

$|p| = 6$ ve $|P| = 10$ olan üst sınırlar için en kötü durum değerleri $|p| = 9$ ve $|P| = 986410$ olarak değişmektedir. *p0*

düğümünün çıkarılmasıyla oluşan değişiklikler Şekil 9'da noktalı çizgilerle gösterilmektedir.



Şekil 9: Çıkarılan (Kesikli / Noktalı) Düğümler Sonucu YÇ.

5. Sonuç

Bu çalışma basit yönlü çizge türetme işleçlerinin karşılık gelen düzenli deyim üzerindeki etkilerini incelemekte olup bunu gerçekleştiren bazı algoritmalar vermekte ve bunları örnek çalışma üzerinde uygulamalarıyla ele almaktadır. Belirtildiği üzere, ele alınan yönlü çizgeler döngü içermediğinden bu çalışma henüz daha geniş sorun için bir ilk yaklaşım olmaktadır. Şimdiye kadar gerçekleştirilen araştırmanın temel vurguları şöyle özetlenebilir:

(i) Düzenli deyimin biçimi: Düzenli deyimin genişliği onun uzunluğu (içerisindeki toplam simge ve işleçlerin sayısı) olarak düşünülebilir. Açıkça görülebileceği gibi, düzenli deyimin biçimi onun genişliğine doğrudan bir etki yapmakta ve bu etki de ele alınan algoritma ve işlemlerin karmaşıklığında karşımıza çıkmaktadır. Dolayısıyla düzenli değimin biçimi o biçim için tanılanan algoritmaların başarımını veya etkinliğini etkilemektedir. Ne yazık ki çarpımların toplamı biçimi bir anlamda en kötü durum biçimidir, çünkü düzenli deyimin sıkılığını göz ardı edilmektedir. Diğer taraftan, bu algoritmaların açık ve basit olmasına yardımcı olmakta ve de biçimin korunması için ek bir dönüşüm veya işleme gerek duymadığından bir anlamda doğal bir biçim olmaktadır. Yine de, öne sürülen karmaşıklık değerleri en kötü durumun da en kötüsü olarak düşünülmelidir (Fakat şu da unutulmamalıdır ki bu her zaman için uygulama çerçevesinde kötü bir sonuç doğuracağı anlamına gelmemektedir).

(ii) Yaklaşımın boyutu: Çalışma boyunca söz konusu yönlü çizgelerin döngü içermediği varsayılmıştır. Bu döngü içeren yönlü çizge modellerinin tamamiyle dışlandığı anlamına gelmemektedir. Çeşitli döngü atma stratejileri (döngüleri en fazla tanımlanmış bir sayı kadar dolanmak, dizinleme tekniğini kullanmak, sistemin anlamını hesaba katarak bazı indirgemeler gerçekleştirmek vb.) kullanılarak yönlü çizge modeli kapsam içine alınacak hale çevrilebilir. Tabi ki, bu yeni hal sistemi bütünüyle temsil edemeyecek ve sistemin bir alt sistemi olacaktır. Fakat uygulamada bunu yeterli görülebileceği hatta tercih edilebileceği durumlar olabilmektedir. Ayrıca, örnek çalışmada belirtildiği üzere normalde döngü oluşmasına sebebiyet veren bazı türetme işlemleri gerekirse sistemin tanımı da hesaba katılarak döngü oluşmaya sebebiyet vermeyen türetme işlemleriyle gerçeklenebilmektedir.

Diğer taraftan, gelecek çalışmalar için döngü içeren yönlü çizgelerin ele alınması ve düzenli deyimin biçiminin etkinlikten (doğabilecek ek dönüşümlere bağlı) ödünler vermeden geliştirilmesi düşünülmektedir. Düzenli deyimin sıkılığını onu başka bir biçimde (mesela biraz daha iyi görünen toplamların çarpımı biçimi vb. gibi) düzenlenerek iyileştirilmesi düşünülmektedir. Fakat bu konudaki en iyi yaklaşımlardan biri (belki de teki) türetme işleçlerini düzenli deyimlerin formuna bağlı kalmadan etkin ve net cebirsel dönüşümler olarak gösterebilmek olacaktır. Bu sayede hem matematiksel olarak daha genel hem de uygulamada (türetme ile sınamada) kullanılabilecek verimi yüksek bir sonuç ortaya çıkacaktır.

6. Kaynakça

[1] Beizer, B., *Software Testing Techniques*, Van Nostrand Reinhold, 1990.

[2] Binder, R. V., *Testing Object-Oriented Systems*, Addison-Wesley, Boston, 2000.

[3] Belli, F., Linschulte, M., "On 'Negative' Tests of Web Applications", *Annals of Mathematics, Computing & Teleinformatics*, Vol. 1, No. 5, 44-56, 2008.

[4] Gossens, S., Belli, F., Beydeda, S., Dal CIN, M., "View Graphs for Analysis and Testing of Programs at Different Abstraction Levels", *Proc. of High-Assurance Systems Eng. Symp. (HASE)*, 121-13, 2005.

[5] Belli, F., "Finite-State Testing and Analysis of Graphical User Interfaces", *Proc. of 12th International Symposium on Software Reliability Engineering*, 34-43, 2001.

[6] Belli, F., Budnik, Ch.J., Wong, E., "Basic Operations for Generating Behavioral Mutants", *Proc. 2nd Workshop on Mutation Analysis* in conjunction with ISSRE'06, 2006.

[7] Salomaa, A., *Theory of Automata*, Pergamon Press, Oxford, etc., 1969.

[8] Myhill, J., "Finite automata and the representation of events", *WADD TR-57-624*, Dayton, OH: Wright Patterson Air Force Base, 1957.

[9] Gill, A., *Introduction to the Theory of Finite-State Machines*, McGraw-Hill, 1962.

[10] Hopcroft, J. E., Motwani. R., Ullman, J. D., *Introduction to Automata Theory, Languages and Computation*, 2nd Edition, Addison-Wesley, Massachusetts, 2001.

[11] Stearns, R. E., Hartmanis, J., "Regularity Preserving Modifications of Regular Expressions", *Information and Control*, Vol. 6, 55-69, 1963.

[12] Brzozowski, J. A., "Derivatives of Regular Expressions", *J. ACM*, 11, 4 , 481-494, 1964.

[13] Salomaa, A. 1966. "Two Complete Axiom Systems for the Algebra of Regular Events", *J. ACM*, 13, 1, 158-169, 1966.

[14] Belli, F., Budnik, Ch. J., "Test Cost Reduction for Interactive Systems", *LNI*, 62, 149-160, 2005.

[15] Belli, F., Budnik, Ch. J., "Test Minimization for Human-Computer Interaction", *International Journal of Artificial Intelligence, Neural Networks, and Complex P.*, Vol. 26, 2, 161-174, 2007.

[16] Belli, F., Hollmann, A., "Test Generation and Minimization with 'Basic' Statecharts", *23rd Annual ACM Symposium on Applied Computing (SAC 2008)*, Fortaleza, Ceará, Brazil, Mar., 2008.