



**ESKİŞEHİR OSMANGAZİ ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

BİÇİMSEL DİLLER VE OTOMATA

Verification of model in Octomap

Grup 6

152120181049 Efekan SARGIN

152120181012 Yusuf Kenan AKSAN

152120181068 Hüseyin Can ERGÜN

152120181071 Muhammed Talha ŞAHİN

Doç. Dr. Ahmet YAZICI

Mayıs 2021

İÇİNDEKİLER

A.GİRİŞ	4
A.1.MODEL CHECKING.....	4
A.1.1.TEMPORAL LOGIC.....	5
A.2.UPPAAL.....	6
A.2.1.UPPAAL MODELLEME.....	7
A.2.1.1.MANTIKSAL MODELLEME	7
A.2.1.1.1.LOCATION(STATE)'LERİN ÖZELLİKLERİ	7
A.2.1.1.2.GEÇİŞLERİN ÖZELLİKLERİ	7
A.2.1.2.VERİ MODELLEME	8
A.2.2.UPPAAL SİMÜLATOR	9
A.2.3.UPPAAL SORGULARI (QUERIES).....	10
A.2.3.1.SORGU KAVRAMI VE UPPAAL'IN SUNDUĞU SORGU SİSTEMİ.....	10
A.2.3.2. SORGULARIN MODEL DOĞRULAMA İÇERİSİNDEKİ YERİ	11
A.3.XML	12
A.3.1.XML'İN ÖZELLİKLERİ	12
A.3.2.XML'İN PROJEMİZDEKİ YERİ	12
A.4.OCTOMAP	12
A.4.1.BİR HARİTALAMA SİSTEMİ OLARAK OCTOMAP	12
A.4.2.OCTOMAP'İN VERİ SAKLAMA YÖNTEMİ.....	13
A.4.2.1.OCTREE.....	14
A.4.2.2.OCTOMAP VE HAREKETLİ SİSTEMLER	14
A.5 PROJEMİZİN DERS İLE İLİŞKİSİ	15
B.PROBLEM VE ÇÖZÜM	16
B.1.PROBLEM ÖZELLİKLERİ VE ÇÖZÜM YAKLAŞIMI.....	16
B.2.ÜZERİNDE ÇALIŞILACAK DURUM	16
B.3.UPPAAL ÜZERİNDE OLUŞTURULAN TEMPLATELER	16
B.3.1.IDLE	16
B.3.2.COORDINATES.....	17
B.3.3.OCTREEWALK	18
B.3.4.OCTREEWALKDYNAMIC	22
B.3.5.MOVEUPDATE	22
B.4.OCTREE VERİ YAPISININ UPPAAL ÜZERİNDE MODELLENMESİ	24
B.4.1.GEÇİRİLECEK ÖZELLİKLER	24
B.4.2.KULLANILABİLİR UPPAAL VERİ YAPILARI.....	24
B.4.3.MODELLEME YÖNTEMİMİZ.....	24
B.4.3.1.FORMÜL AYRINTILARI	26
B.4.3.2.FORMÜLÜN BÜTÜNLÜŞMESİ	29
B.4.4.SONUÇ.....	29
B.5.QUERIES	30
B.5.1.IDLE,COORDINATES,OCTREEWALK	30
B.5.2.OCTREEWALKDYNAMIC	32

B.5.2.1.DERİNLİK-İNDEX İLİŞKİSİ	32
B.5.2.2.AĞACIN DOĞRULUĞUNUN SINANMASI.....	33
B.5.3.MOVEUPDATE	34
B.5.4.SORGU SONUÇLARI HAKKINDA	34
C.SONUÇLAR	35
D.PROJE EKİBİ DEĞERLENDİRMESİ.....	35
E.KAYNAKÇA	38

A.GİRİŞ

Çalışmamızın temel hedeflerinden bahsetmemiz gerekirse; projemizin kapsayacağı konular:

- Otomatik sistemler üzerinde doğrulama kavramı
- Bir doğrulama aracı olan UPPAAL'ın özellikleri
- OctoMap haritalama sisteminin genel çalışma mantığı ve veri yapısıdır.

OctoMap projesinin tamamının modelini çıkartma imkanı bulunmasa da, temel olarak gördüğümüz yapıların, UPPAAL üzerinde modellemeleri ve doğrulamaları yapıldı.

A.1.MODEL CHECKING

Model kontrolü yani verification, bir sonlu durum modelinin belirli şartları sağlayıp sağlanmadığını kontrol etme yöntemidir.

Model kontrolü yöntemi öncesinde, hayatımızda gördüğümüz veya kullandığımız birçok bilgisayar tabanlı otomat sistemlerin her zaman doğru çalışacağının, etrafa ya da kullanıcıya zarar vermeyeceğinin garantisini yoktu. Bu makinelerin güven vermemesi maddi ve fiziksel olarak kötü sonuçlar doğuruyordu [1].

Bir yiyecek otomatının (Şekil A.1) hata aldığı durumlarda oluşturabileceği kötü sonuçlar az ve büyütülmeyecek derecededir ve bu otomatın doğrulanması test etme yöntemiyle sağlanabilir. Olası hataların az olması, herhangi bir zarara yol açabilecek durumun olmaması nedeniyle bu tarz otomatlar test etme yöntemiyle doğrulanabilir. Fakat bu otomat sisteminin uzaya gönderilen bir roket olduğunu düşündüğümüzde, bunun hata sonucunda oluşturabileceği zarar hem ekonomik hem de fiziksel olarak çok büyük olabilir. Bu tarz otomat sistemlerinin doğrulanması test etme yöntemi ile imkansızdır. Test yönteminin yanında simülle etme yöntemi de kullanılmıştır. Bu yöntemin kullanılması yaygındır fakat bir otomatın her olasılığını ve durumunu simülle etmek çok zordur. Simülle etmediğimiz durumların doğru olmadığından emin olmadığımız ve bu durumların hata verebilme ihtimali olduğundan bu yöntem de büyük sistemler için güvenilir değildir., bu güvenilirliği sağlamak için 1980'lerin başında Joseph Sifakis - J.P. Queille ve Clarke – Emerson, temporal logic model checking yöntemini tanımlamıştır[2].



Şekil A.1: Basit otomata örneği
[www.alibaba.com]



Şekil A.2: Başarısız model checking yöntemi
[https://www.lefigaro.fr/sciences]

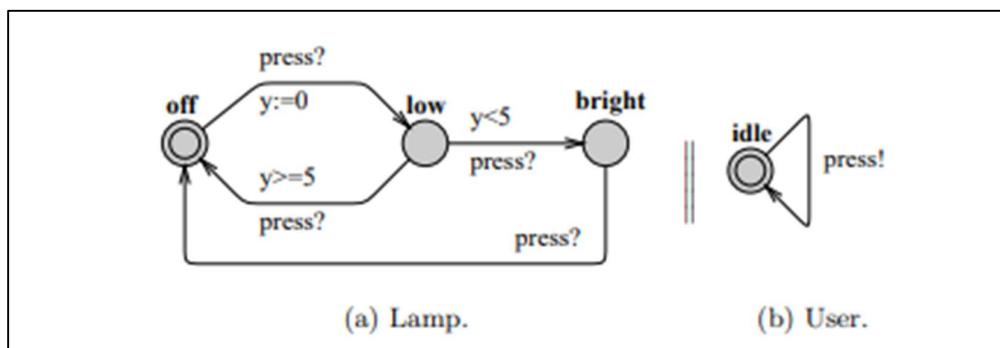
A.1.1.TEMPORAL LOGIC

Matematiksel mantık ile durumun zamana bağlı olmaksızın doğruluğu kontrol edilir. Bu işlem “ve (\wedge), veya (\vee), ise (\rightarrow), değil (\neg)” operatörleri ile yapılır. Zamansal mantığa ise şu durumlarda ihtiyaç duyulur; örneğin “Ben üzüyorum.” cümlesinden kişinin şu an üzüldüğünü anlayabiliriz. Fakat gelecekte bu kişinin üzümeye devam edeceğini anlayamaz ve bu konuda yorum yapamayız. Bu yüzden 1957 yılında Arthur Prior zamanla ilgili felsefik soruları analiz etmeye ihtiyaç duyulduğunu anlatan bir makale yazmış ve zamansal mantığın ortaya çıkışına öncayak olmuştur. 1977 yılında ise Amir Pnueli, bilgisayar bilimlerinde zamansal mantığın kullanılmasını önermiştir. Günümüzde zamansal mantık “Gelecek, Genel, Sonra, Her zaman, Mevcut,-e kadar,-den sonra” ifadeleri ile kullanılır. Model doğrulanması yapılırken de matematiksel mantık ve zamansal mantık birlikte kullanılır[3].

Günümüzde Model doğrulanması bazı araçlar (UPPAAL, PRISM, ECLAIR, BLAST vb.) yardımı ile gerçekleştirilir.

Textual	Symbolic	Definition	Explanation	Diagram
Binary operators				
$\phi \cup \psi$	$\phi \cup \psi$	$(B \cup C)(\phi) = (\exists i : C(\phi_i) \wedge (\forall j < i : B(\phi_j)))$	Until: ψ holds at the current or a future position, and ϕ has to hold until that position. At that position ϕ does not have to hold any more.	
$\phi \mathcal{R} \psi$	$\phi \mathcal{R} \psi$	$(B \mathcal{R} C)(\phi) = (\forall i : C(\phi_i) \vee (\exists j < i : B(\phi_j)))$	Release: ϕ releases ψ if ψ is true up until and including the first position in which ϕ is true (or forever if such a position does not exist).	
Unary operators				
$\mathbf{N} \phi$	$\square \phi$	$\mathbf{N} B(\phi_i) = B(\phi_{i+1})$	Next: ϕ has to hold at the next state. (X is used synonymously.)	
$\mathbf{F} \phi$	$\diamond \phi$	$\mathcal{F} B(\phi) = (\mathbf{true} \cup B)(\phi)$	Future: ϕ eventually has to hold (somewhere on the subsequent path).	
$\mathbf{G} \phi$	$\Box \phi$	$\mathcal{G} B(\phi) = \neg \mathcal{F} \neg B(\phi)$	Globally: ϕ has to hold on the entire subsequent path.	
$\mathbf{A} \phi$	$\forall \phi$	$(AB)(\psi) = (\forall \phi : \phi_0 = \psi \rightarrow B(\phi))$	All: ϕ has to hold on all paths starting from the current state.	
$\mathbf{E} \phi$	$\exists \phi$	$(EB)(\psi) = (\exists \phi : \phi_0 = \psi \wedge B(\phi))$	Exists: there exists at least one path starting from the current state where ϕ holds.	

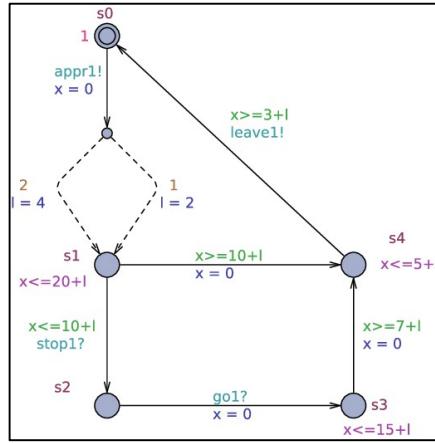
Şekil A.3: Zamansal mantık ve matematiksel mantık tablosu



Şekil A.4: UPPAAL basit doğrulama örneği

A.2.UPPAAL

UPPAAL, Uppsala Üniversitesi ve Aalborg Üniversitesi tarafından ortaklaşa geliştirilen gerçek zamanlı sistemlerin doğrulanması için bir araç kütusudur. Java kullanıcı arayüzü ve bir doğrulama motoruna sahiptir C++ ile yazılmıştır. UPPAAL'ın ilk versiyonu 1995 yılında yayımlanmıştır, yayınlandığı günden beri sürekli olarak geliştirilmektedir.



Şekil A.5: Özellikleri kullanılmış UPPAAL örneği

UPPAAL üç ana bölümden oluşur: bir **açıklama dili**, bir **simülatör** ve bir **model denetleyici**. Açıklama dili, veri türleri (ör. Sınırlı tamsayılar, diziler, vb.) içeren deterministik olmayan korumalı bir komut dilidir. Sistem davranışını saat ve veri değişkenleriyle genişletilmiş otomatik veri ağları olarak tanımlamak için bir modelleme veya tasarım dili olarak hizmet eder. Simülatör, erken tasarım (veya modelleme) aşamalarında bir sistemin olası dinamik uygulamalarının incelenmesini sağlayan ve böylece sistemin kapsamlı dinamik davranışını kapsayan model denetleyicisi tarafından doğrulanmadan önce hata algılama aracı sağlayan bir doğrulama aracıdır. Model denetleyicisi, bir sistemin durum uzayını, yani kısıtlamalarla temsil edilen sembolik durumlar açısından erişilebilirlik analizini keşfederken değişmez ve erişilebilirlik özelliklerini kontrol edebilir.

	1998	2000	DBM	Min	Ctrl	Act	PWL	State	2002
Fischer 5	126.30	13.50	4.79	6.02	3.98	2.13	3.83	12.66	0.19
Audio	-	2.23	1.50	1.79	1.45	0.50	1.57	2.28	0.45
Power Down	*	407.82	207.76	233.63	217.62	53.00	125.25	364.87	13.26
Collision Detection	128.64	17.40	7.75	8.50	7.43	7.94	7.04	19.16	6.92
TDMA	108.70	14.36	9.15	9.84	9.38	6.01	9.33	16.96	6.01

Şekil A.6: UPPAAL versiyonlarının zaman içinde hızlanması (sn)

Model denetleyicisi UPPAAL, zamanlanmış otomata teorisine dayanmaktadır. Zamanlanmış bir otomat, sonlu durumlu bir makinedir. UPPAAL’da bir sistem, birkaç zamanlanmış otomata ağı olarak modellenmiştir. Model, durumun bir parçası olan ayrık değişkenlerle daha da genişletilmiştir. Bu değişkenleri programlama dillerinde olduğu gibi kullanır: Okunurlar, yazılırlar ve genel aritmetik işlemlere tabidirler[4].

A.2.1.UPPAAL’da Modelleme

Bu başlık altında UPPAAL üzerinde modelleme yaparken kullanabileceğimiz özelliklerini mantıksal ve veri özellikleri olarak inceleyeceğiz.

A.2.1.1.Mantıksal Modelleme

A.2.1.1.1.Location(state)lerin Özellikleri

Invariant: Invariantı yazılan state’e ilerleyebilmek için kontrol edilmesi istenilen şartlar burada tanımlanır. Guard özelliği ile benzer çalışır tek farkı geçiş değil state’e özgü olmasıdır, aynı state’i hedef alan birden fazla geçiş olabileceği durumlarda her geçiş aynı şartın koyulması yerine invariant özelliği kullanılabilir.

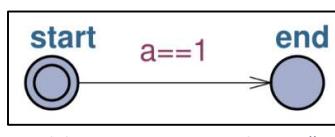
Initial: Başlangıç state’ine verilen özelliktir.

Committed: Bu özelliğin verildiği state üzerinde beklenme yapılamaz, mecburen -eğer bir yol var ise- bir yol seçip ilerlemek zorundadır.

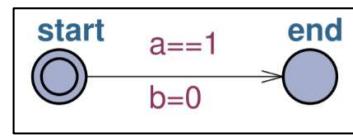
Urgent: Urgent hemen gerçekleşmesi gereken bir işlemin yapılmasında kullanılır. İstenilen işlem doğrultusunda Clock tanımlanmış gibi davranışır.

A.2.1.1.2.Geçişlerin Özellikleri

Guards: Değişkenleri kullanarak modelin hangi durumda ilerleyeceğini belirleyen kenardır özellikleidir.



Şekil A.7: UPPAAL Guards örneği



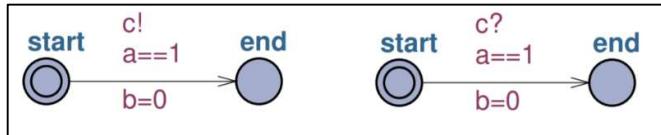
Şekil A.8: UPPAAL Update örneği

Update: Kenarın çalışacağı durumda verilen değerin güncellenmesini sağlayan kenar özellikleidir.

Synchronization: İki veya daha fazla sürecin eylemini koordine eder. 3 tip senkronizasyon çeşidi vardır ve channel tipi üzerinden çalışır. Sistem içerisinde channel arrayleri de tanımlamak mümkündür.

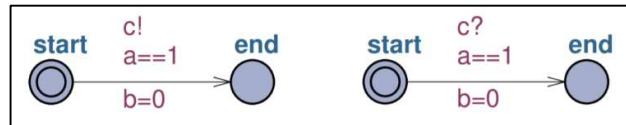
Clock: Birden fazla işlemin zamana bağlı olarak birbirleri ile çalışmasını sağlayan değişken tipidir.

Regular Channel(chan c): Bir süreçte soru işaretü ile gösterilen kenarın gerçekleşmesi için ünlemli kenara sahip diğer bir sürecin gerçekleşmesi gereklidir.



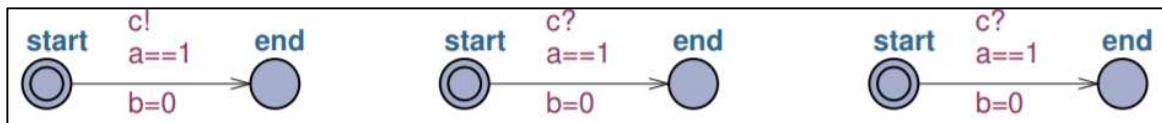
Şekil A.9: UPPAAL Regular Channel örneği

- 1) **Urgent Channel(urgent chan c):** Genel olarak Regular Channel'a benzerler. Kaynak durumundayken geciktirmek mümkün değildir. Clock bu kanallar üzerinde kullanılamaz.



Şekil A.10: UPPAAL Urgent Channel örneği

- 2) **Broadcast Channel(broadcast chan c):** Bir süreçte soru işaretü ile gösterilen kenarın(c?) gerçekleşmesi için ünlemli kenara(c!) sahip bir ya da daha fazla sürecin gerçekleşmesi gereklidir. Eğer ünlemli herhangi bir kenar diğer süreçlerde yoksa soru işaretine sahip kenar yine de çalışır[6].



Şekil A.11: UPPAAL Broadcast Channel örneği

A.2.1.2.Veri Modelleme

UPPAAL'ın sistemi içerisinde kullanılan değişken tipleri, matematiksel açıdan çoğu programlama diline kıyasla basit kalıyor. Matematiksel değişkenler dışında ise model checking yöntemini desteklemek amacıyla tanımlanmış senkronizasyon değişken tipleri bulunuyor. (Bkz. A.2.1.1)

Sistem içerisinde kullanılmamıza izin verilen bilgi saklayan değişkenler ve yapılar;

- Boolean
- Integer
- Array (Static)
- Struct, olarak karşımıza çıkıyor. [11]

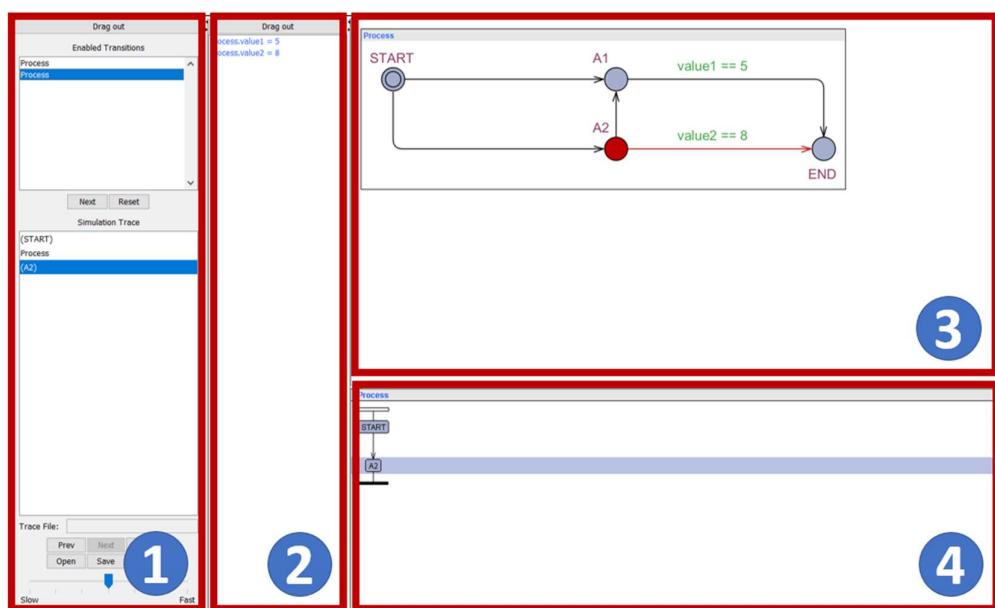
- Integer değerler için alt-üst sınırlar belirtilebiliyor. Örneğin “int[1,8] myValue;” şeklinde integer tanımlaması yapılabiliyor. Bu özellik modellemenin sağlığı açısından önemli bir yer tutuyor.
- Struct yapıları ise verilerin karışmaması bir arada durabilmesi için programı kullanmayı kolaylaştırın özelliklerden bir tanesi. Struct arraylerinin de kullanılmasına sistem izin veriyor.
- Aynı zamanda bu değerler constant olarak da tanımlanabiliyor.

A.2.2. UPPAAL Simülatör

UPPAAL’ın simülatör sekmesinde oluşturduğumuz modeli adım adım inceleme ve işlem yolları oluşturma imkanımız oluyor. Bu yollar ile de sistemimizin davranışının spesifik durumlar için bilgi almış oluyoruz.

Simülatör paneli Şekil A.12 de görebileceğimiz gibi 4 parçadan oluşuyor;

- 1- İlk bölümümüzde model üzerinden ilerlenecek yol tercihini ve yol üzerinde ileri-geri olabilecek şekilde simülatörün hareketlerini kontrol edebiliyoruz. Aynı zamanda rastgele seçimlerle otomatik bir yol oluşumu sağlayabiliyoruz. Ve oluşturduğumuz yolların takibini yapabiliyoruz.
- 2- Bu bölümde declarations kısmında tanımladığımız değişkenlerin durumları anlık olarak gösteriliyor.
- 3- Oluşturduğumuz templateler üzerinde görsel olarak ilerlediğimiz yolu burada takip edebiliyoruz. Birden fazla template ile çalışan sistemler için de bu kısımda bütün sistemleri aynı anda izleme imkanımız da oluyor.
- 4- Son bölümümüzde de ilerlediğimiz yolu ve yaptığım geçişleri (path) ayrıntılı ve görsel bir şekilde sunuluyor.



Şekil A.12: UPPAAL arayüzü ve bölümleri

A.2.3.UPPAAL Sorguları (Queryleri)

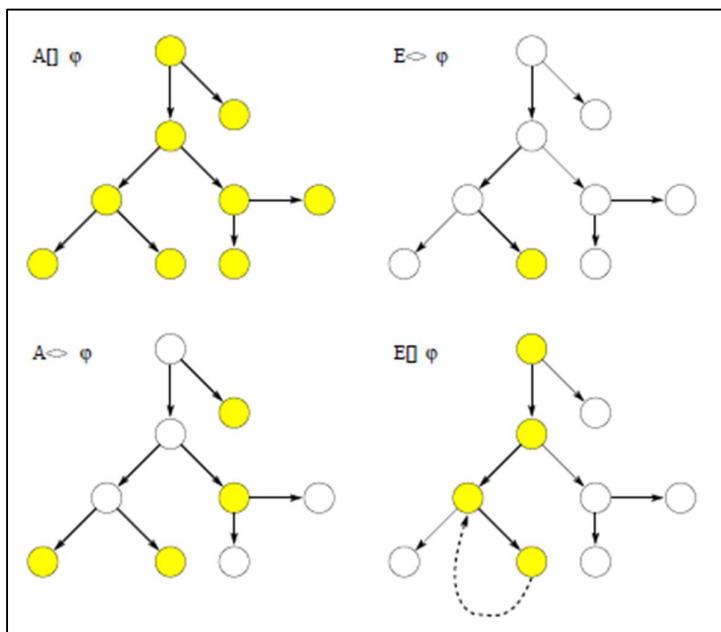
A.2.3.1.Sorgu Kavramı ve UPPAAL’ın Sunduğu Soru Sistemi

Model-checker’ın temel amacı istenilen gereksinimler doğrultusunda bir modeli doğrulamaktır. Modelde olduğu gibi iyi tanımlanmış ve makine tarafından okunabilir bir dilde gereksinimler tanımlanmalıdır. Bu koşulları kapsayan bazı mantıklar bilimsel doğrulamalarda kullanılıyor, UPPAAL ise TCTL (Timed Computational Temporal Logic)’nin basitleşmiş versiyonunu kullanıyor, sebebi ise UPPAAL tasarımcılarının programın verimliliğini mümkün olduğunca üst seviyede tutmak istemeleridir. Bu da sorguların karmaşıklığını nispeten azaltmıştır. TCTL de olduğu gibi UPPAAL sorgu dili de path formülleri ve state formüllerinden oluşuyor.

State formülü bir ifadedir, modelin davranışına bakmadan bir durum için değerlendirilir. Örnek vermek gerekirse $i == 5$ koşulu state in içerisinde i değeri 5’e eşitken doğrudur. Belirli bir özelliğin olup olmadığını kontrol etmek de mümkündür ($P.l$ P =sureç, l =konum). Guard özelliğinin syntaxıyla benzerdir ama guard özelliğinden farklı olarak yazılabilcek özellikler sınırlıdır.

Path formüllerini alt başlıklarda incelemek mümkündür;

- **Reachability properties:** En basit özelliktir. Belirli bir durum formülünün, ϕ , karşılanıp karşılanmayacağı sorarlar. 4 farklı türü vardır bunlar istenilen doğrulamaya göre bulundukları yola ve duruma göre değişkenlik gösterir.
- **Safety properties:** Belirli bir durum formülünün, ϕ , kesinlikle karşılanmayacağı doğrulamamızdır.
- **Liveness properties:** Belirli bir durum formülünün, ϕ , gerçekleşmesi sonucunda; bunun başka bir duruma yol açıp açmadığının kontrol edilmesidir.[4]



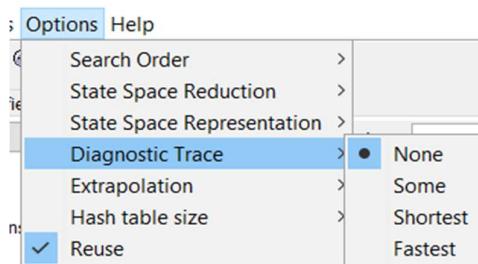
Şekil A.13: UPPAAL'da desteklenen yol formüllerini. Doldurulmuş durumlar(ϕ) durum formüllerini tutar, kalın yazılar ise formüldeki pathleri gösterir.

- E – herhangi bir yol için (UPPAAL'da “E”)
- A – bütün yollar için (UPPAAL'da “A”)
- G – bir yoldaki tüm durumlar (UPPAAL'da “[]”)
- F – bir yoldaki bazı durumlar (UPPAAL'da “<>”)

(Başlık B.5.Sorgular altında, OctoMap modeli için oluşturulan sorgular ve açıklamaları örnek olması açısından incelenebilir.)

UPPAAL verifier sekmesi üzerinden de bu sorguları çalıştırıp; doğrulanıp doğrulanmadığını kontrol edebiliyor, yazdığımız sorguları kaydedebiliyoruz.

Aynı zamanda Şekil A.14'de de görülebilen panelden “Diagnostic Trace” özelliği ile, girilen sorguların simülatör içerisindeki örneklerine ulaşılabilir. Bu özelliği açıp sorgu



Şekil A. 14: Ayarlar paneli

girildiğinde, sistem otomatik olarak sorgunun bir örneğinin gidiş yolunu simülatör üzerine yükler. Bu özellik, hata bulma işleminde çok yararlı bir özellik olarak karşımıza çıkarıyor.

A.2.3.2.Sorgular’ın Model Doğrulama İçerisindeki Yeri

Sorgular, basit yöntemlerle modelleyemeyeceğimiz büyük sistemlerin kurulum ve/veya çalışma sırasında çıkabilecek problemleri test etmemizi ve problemlerin kaynaklarına ulaşmamızı sağlar. Problemler dışında ise sistemin sağlamak zorunda olduğu “olmazsa olmaz” özelliklere sahip olup olmadığını doğrulamamıza yardımcı olur.

Bir örnek üzerinden anlatmamız gerekirse; İçerisinde yüzlerce insansız makinenin bulunduğu büyük bir fabrikamız olduğunu düşünelim ve bu fabrikanın istenilen ürünü vermesi beklenirken sorunlu ürün ürettiği bir durumda, hatayı bulmak için tek tek fabrika içerisindeki her robottu incelememiz bizimhaftalarımızı hatta belki de aylarımızı alacaktır. Bu kaybedilecek olan süre fabrikanın büyük bir zarara girmesine neden olabilir.

Böyle bir durumda fabrikamızın modeli program içerisinde implemente edilmiş bir şekilde bulunuyorsa, programın sorgu dili kullanarak kısa bir süre içerisinde problem ve kaynağı tespit edilebilir. Büyük çaplı ve karmaşık sistemlerde sorgu kullanımını ve doğrulama yapılması, işlerin hızlanması ve doğru/kesin dönüt alma konusunda oldukça önemlidir.

A.3.XML

XML, Genişletilebilir İşaretleme Dili anlamı taşıyan ve HTML dilinin de tasarımcısı olan Tim Berners Lee tarafından üretilmiş bir veri dokümanıdır. İnterneti kullanarak veri alışverişi yapan sistemler ve bu platformlar arasındaki veri iletişimini standart hale getirmek için tasarlanmıştır.[7]

Bir veriyi başka alana taşımak, taşındığı alanda okunabilir bir şeke bürünmesi için birden fazla işlemden geçmesi gerekmektedir. XML teknolojisi ise bu aşamaları çok daha basit hale getirmektedir.

XML dili günümüzde veri iletimiyle alakalı her alanda bolca kullanılmaktadır. Veri tabanları aktarımı, yazılım paketleri içerisindeki bağımlılıkların tanımlanması, verilerin iletimi, içerik depolanması gibi çok fazla alanda XML teknolojisi kullanılmaktadır.

A.3.1.XML'in Özellikleri

- Verileri yedekleme işlemi yapar.
- Verileri transfer edebilir.
- Verilerin paylaşılması, taşınması, yayılması ve saklanması gibi işlemler yapılabilir.
- Platform değişikliği basit bir şekilde yapılabilir.
- Verilerin kullanımını kolaylaştırır.

A.3.2.XML'in Projemizdeki Yeri

XML, projemizde UPPAAL'da oluşturulan sistemi dosya uzantısı olarak kullanılmaktadır. XML'den önce XTA formatı kullanıldı. Bu format syntax hatalı oluşturulan bir sistemi kaydetmeye izin vermiyordu ve sistemde bazı problemler oluşturuyordu. Bundan dolayı UPPAAL bu sıkıntıları önlemek için XML dosya formatına geçiş yaptı. UPPAAL'in XML kullanmasının diğer nedenlerinden birisi ise programda template, location, transitions ve label kullanımını sağlamıştır. Bunun yanı sıra XML, UPPAAL'in en önemli özelliklerinden birisi olan verification sistemini de desteklemektedir. [8]

A.4.OctoMap

A.4.1.Bir Haritalama Sistemi Olarak OctoMap

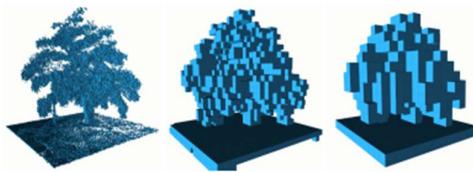
Octomap en basit tanımı ile; kendini diğer 3D mapping sistemlerinden ayıran özelliklere sahip, olasılığa dayanan, hızlı ve memory-friendly çalışmayı amaçlayan bir 3D haritalama sistemidir. Açık kaynak bir projedir ve octomap.github.io üzerinden ulaşılabilir.

OctoMap'in veri saklama yöntemi olan OcTree'nin çalışma prensibi; Şekil A.17 de görebileceğimiz gibi alınan veriyi bir ağaç yapısı içerisinde saklayıp, ilk olarak büyük parçaların boş-dolu bilgisini kaydederek, daha sonrasında ise **sadece** 'tamamen, boş ya da dolu' **olmayan** sub treelerden işleme devam etmesidir. Böylece tamamen dolu ya da boş kısımlar için

sub treeler oluşması engellenmiş oluyor bu da treeenin kapladığı alanı azaltıyor ve daha hızlı erişilebilirlik sağlıyor. Bu konuya başlık **A.4.2.1** de daha ayrıntılı olarak değinilmiştir.

OctoMap sistemi diğer mapping sistemlerinden farklı olarak dolu ve boş alanları tutmasının yanı sıra keşfedilmemiş alanları da hafızasında tutuyor. Bunun sağladığı fayda, hareketli mapping sistemlerinin hareketlerine daha optimal şekilde karar vermelerini sağlayabilmeleri. Keşfedilmemiş alanlar uninitialized nodelar olarak ifade ediliyor. Eğer gerekirse nodeların içerisinde renk, sıcaklık gibi bilgiler de saklanabiliyor.

Harcanılan alanı daha da azaltmak için; ağaçın node'ları içerisinde tutulan bilgilerin, ayrı iki node'da çok benzer veya aynı olduğu durumlarda, tek bir yapı oluşturulup iki node'un pointer'ı ile bu yapıya işaret ediliyor. Böylece tekrar edilen alanların bilgisinin ayrı ayrı saklanması gereklilik kalmıyor.



Şekil A.15: Octomap üzerinde oluşturulmuş ağaç modeli[5]

Aynı zamanda octomap'in yapımcıları, sensörlerin olasılığa dayanan veri analiz formülünü daha hızlı işlem yapmak için logaritma tabanlı bir formüle dönüştürüp(Şekil A.16) logaritma formülünü sisteme implemente etmişlerdir. Yenilenmiş formül çarpım işlemleri yerine (logaritmanın sağladığı özellik) toplama işlemi yapmaya olanak sağlıyor bunun sonucunda da işlemler daha hızlı yapılabiliyor [5].

$$P(n | z_{1:t}) = \left[1 + \frac{1 - P(n | z_t)}{P(n | z_t)} \frac{1 - P(n | z_{1:t-1})}{P(n | z_{1:t-1})} \frac{P(n)}{1 - P(n)} \right]^{-1}$$

Eq. (1) can be rewritten as

$$L(n | z_{1:t}) = L(n | z_{1:t-1}) + L(n | z_t),$$

with

$$L(n) = \log \left[\frac{P(n)}{1 - P(n)} \right].$$

Şekil A.16: Logaritma tabanlı yenilenmiş formül[5]

A.4.2.OctoMap'in Veri Saklama Yöntemi

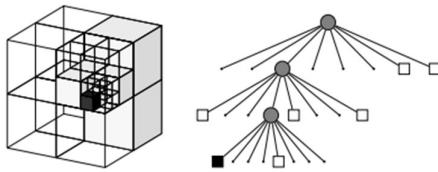
Bir önceki başlıkta da bahsettiğimiz gibi; OctoMap, saklaması gereken bilgiyi ocTree denilen ağaç yapıları içerisinde tutulup bu ağaç yapıları üzerinden veriye erişim sağlıyor. Octomapın temel yapı taşı olan ocTree, mapping alanında ilk olarak Meagher (1982) tarafından

kullanılmış ve daha sonrasında 2D ve 3D olarak dolu-boş alan tutan octreeler kullanılmaya devam edilmiştir [5].

A.4.2.1.OcTree

OcTreeler her parent birimin, 8 veya 0 adet çocuk birime sahip olduğu ağaç yapılarıdır. 3 boyutlu haritalar gibi büyük verilerin, veriden en az şekilde kayıp yaşayarak küçültülmesi ve saklanması konusunda çok kullanışlı bir veri yapısıdır.

OctoMap gibi haritalama sistemleri için konuşacak olursak; bir ocTree'nin tamamı kübik bir uzayı temsil eder ve ağacın ilk derinliğindeki 8 çocuk birim de bu kübik uzayı 8 farklı köşesindeki kübik alanları işaret eder. Bu şekilde ilerleyerek, oluşan her küçük kübik alan için de 8 adet çocuğun köşeleri temsil ettiğini varsayıyoruz. (Şekil A.17) Bu yöntemle elimizdeki küpü sürekli 8'er parçaya bölerek alan küçültüyoruz. İşlemimizi istenilen kadar tekrarladıkten sonra; ağacın en derin yerinde her birimin bir occupancy bilgisi (dolu, boş, bilinmeyen) olmuş oluyor. Daha sonrasında bütün çocuk birimlerinin occupancy değeri aynı olan parent birimler budanarak alt node'ları yok ediliyor, bunun sonucunda da toplu bir alanda aynı tipte bilgi tutuluyorsa gruplandırılmış, teke indirilmiş ve alandan kazanılmış oluyor.



Şekil A.17: OcTree yapısı[5]

Bu veri yapısı kullanılarak büyük veri yığınlarının saklanması ve kullanılması gereken senaryolarda alandan ve çalışma hızından büyük oranda kazanç sağlanmış oluyor.

A.4.2.2 OctoMap ve Hareketli Sistemler

Otonom araçlar gerek ortamın 3D yapısından gerekse aracın 3D konumundan dolayı ortamı anlamlandırmak ve verilen bir hedefe optimum yoldan gezinim yapabilmek için 3D ortam harita bilgisine ihtiyaç duymaktadırlar. OctoMap çoğunlukla bu araçların çevrelerini daha iyi analiz edebilmeleri için 3D haritalama süreçlerinde kullanılıyor. OctoMap'in diğer sistemlerden daha çok kullanılmasının nedenleri ise sistemin diğer uygulamalarda olmayan veya daha gelişmiş bulunan bazı özelliklerinin olmasıdır.

OctoMap;

- **Tam 3D haritalama** özelliği ile, haritalanacak alana ilişkin bir ön bilgiye ihtiyaç duyulmadığını dolayı keyfi ortamların haritalanabileceğini ifade eder.
- **Güncellenebilirlik** özelliği, ortamın bilgisinin olasılıksal olarak saklandığı ve harita güncellemesinin mümkün olduğunu ifade eder.
- **Esneklik** özelliği dinamik olarak değișebilen harita boyutunu ifade eder. Aynı zamanda esneklik ile bir minimum çözünürlük sınırı olmak şartıyla farklı çözünürlüklerde haritalama mümkündür.

- **Verimli alan kullanımı** harita bilgisinin disk ve hafızada verimli bir şekilde erişilebilir ve saklanabilir olduğunu ifade etmektedir.

OctoMap yöntemi otonom araçlar üzerinde bulunan derinlik kameralarından gelen nokta kümesi ve gMapping yönteminden üretilen konum bilgisine göre ortamın 3D harmasını OcTree veri yapısı ile saklamaktadır. Octree veri yapısının gösterimi Şekil A.17 ile verilmektedir. Şekilde OcTree ile 3D uzayın dolu hacimlerde alt düğümlere bölünerek ifade edildiği görülmektedir.

A.5.Projemizin Ders İle İlişkisi

Projemizin , Biçimsel Diller ve Otomata dersi ile en önemli ilişkisi, çalıştığımız ortam olan UPPAAL üzerinde DFA(Deterministic Finite Automata) modelleri oluşturup bunlarla işlemler gerçekleştirmemizdir. Modellerimiz herbiri sonlu durum makinesidir. UPPAAL zamana bağımlı çalışabilen bir araç olduğu için, oluşturduğumuz DFA'ler de zamana bağlı makineler olmuştur. Derste gördüğümüz DFA'ler ile UPPAAL'da oluşturduğumuz DFA'ler arasındaki bazı farklar Şekil A.18 ve Şekil A.19 'de görülebilir. Bu iki DFA arasında en önemli fark alfabe ve zamana bağımlılıktır. Basit bir DFA'de alfabe ile yapılan geçiş işlemleri, UPPAAL'daki DFA'ımızda daha gelişmiş bir şekilde kullanılmıştır. Guard, Synchronous, Update, Select işlemleri ile stateler arasındaki geçiş işlemleri sağlanır. Aynı zamanda UPPAAL ile iki ya da daha fazla DFA'in birbirleri ile zaman kavramı dahilinde düzgünce çalışmasını sağlar. Projemizde geliştirdiğimiz DFA'leri, derste öğrendiğimiz DFA'in , UPPAAL'ın DFA tanımına uygun hale getirerek oluşturduk.

Deterministic Finite Automata
□ DFA: five-tuple notation
$A = (Q, \Sigma, \delta, q_0, F)$
□ Q is a finite set of states
□ Σ is a finite alphabet (=input symbols/Alphabet)
□ δ is a transition function $\delta(q, a) \rightarrow p$
□ $q_0 \in Q$ is the start state
□ $F \subseteq Q$ is a set of final states

Şekil A.18: DFA tanımı [10]

Definition (Timed Automaton)
(L, ℓ_0, C, A, E, I) , where:
▶ L is a set of locations
▶ $\ell_0 \in L$ is the initial location
▶ C is the set of clocks,
▶ A is the set of actions (e.g. press!), co-actions (e.g. press?) and internal τ -actions
▶ $E \in L \times A \times B(C) \times 2^C \times L$ is a set of edges between locations with an action, a guard and a set of clocks to be reset
▶ $I : L \rightarrow B(C)$ assigns invariants to locations

Şekil A.19: UPPAAL DFA tanımı [9]

B.1.PROBLEM ÖZELLİKLERİ VE ÇÖZÜM YAKLAŞIMI

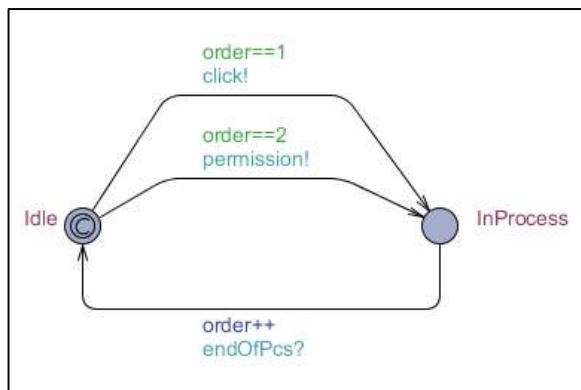
B.2.ÜZERİNDE ÇALIŞILACAK DURUM

Projemizde üzerinde çalışacağımız konu temel olarak octomap (octree) modelini bir doğrulama sistemine implemente etmektir. Spesifik olarak çalışacağımız durum ise koordinatları verilen bir noktadaki (grid, voxel) doluluk (occupancy) bilgisini döndürmesi istenilen bir sistemdir. Burada doluluk bilgisi olarak bahsettiğimiz bilgi, octomap sisteminde tanımlanan; dolu, boş ve bilinmeyen olabilen bir değişkendir. Doğrulama sistemi olarak da kullanacağımız program UPPAAL olarak belirlenmiştir.

B.3.UPPAAL ÜZERİNDE OLUŞTURULAN TEMPLATELER

B.3.1.IDLE

UPPAAL modelimizin başlangıç template'si Idle dır. Bu template'de başlangıç state'ini seçim yapmaya zorlamak ve bu state'de kalmasını önlemek amacıyla , başlangıç state'i committed olarak oluşturulmuştur.Kenarların guard özelliğine atanmış olan order değişkeninin değerine göre de seçim işlemi gerçekleştirilir.Bu özellikleri göz önüne alarak,Idle, ilk olarak sync click özelliği ile ikinci sırada çalışacak template'imiz olan Coordinate template' ini , click işlemini gerçekleştirmesi için uyarır. Coordinate template'i işlemini gerçekleştirdikten sonra işlemini bitirdiğini belirtmek için endOfPcs senkronizasyonunu tetikler.Bunun sayesinde Idle ikinci işlemini yapmak için başlangıç state'ine gelir.İkinci işlem olarak, üçüncü ve en önemli template olan , OctreeWalk'un çalışması için OctreeWalk'a permission senkronizasyonu ile izin verir.Kısacası, Idle , 2 ayrı işlem yapan templatelerin daha iyi bir düzende çalışması için oluşturulmuş bir ana template'dir.(Şekil B.1)



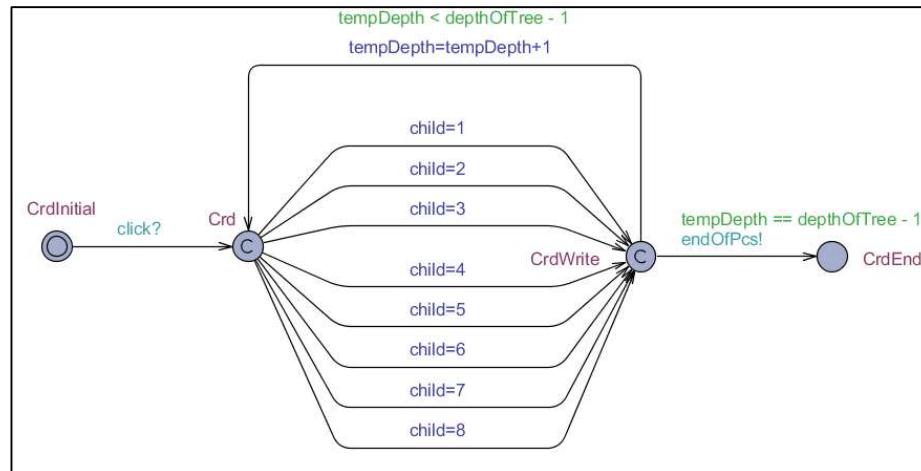
Şekil B.1: Idle Template



Şekil B.2: Edge Özellikleri

B.3.2.COORDINATES

İkinci template olan coordinates, 3 boyutlu bir harita baz alınarak girilen koordinat noktasını Octree'de bir path e dönüştürmeyi temsil eder. Buradaki modelimiz temsili bir modeldir. Octree'de kökten başlar ve 8 çocuktan hangisinden ilerlemesi gereği seçilerek Octree'nin derinliği kadar devam eder. Child seçiminin yapılmasını zorlamak amacıyla Crd adlı state committed olarak oluşturulmuştur. Bu template'de temsil edilen işlemi, yazdığımız C++ kodumuz gerekli koşulları ele alarak yapmaktadır.



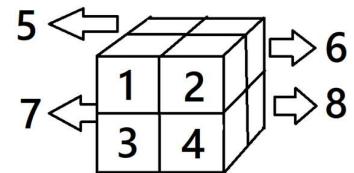
Şekil B.3: Coordinates Template

```

void crdToPath(int x, int y, int z, int path[treeDepth]) {
    bool xup, yup, zup;
    int xcomp = pow(2, treeDepth) / 2, ycomp = pow(2, treeDepth) / 2, zcomp = pow(2, treeDepth) / 2;
    int step = pow(2, treeDepth) / 4;
    for (int i = 0; i < treeDepth; i++, step /= 2)
    {
        if (x > xcomp) { xup = 1; xcomp += step; }
        else if (x == xcomp) { xup = 0; }
        else { xup = 0; xcomp -= step; }
        if (y > ycomp) { yup = 1; ycomp += step; }
        else if (y == ycomp) { yup = 0; }
        else { yup = 0; ycomp -= step; }
        if (z > zcomp) { zup = 1; zcomp += step; }
        else if (z == zcomp) { zup = 0; }
        else { zup = 0; zcomp -= step; }

        //cout << xup << " " << yup << " " << zup << endl;

        if (xup == 0 && yup == 0 && zup == 0) path[i] = 7;
        else if (xup == 0 && yup == 0 && zup == 1) path[i] = 3;
        else if (xup == 0 && yup == 1 && zup == 1) path[i] = 1;
        else if (xup == 1 && yup == 0 && zup == 1) path[i] = 2;
        else if (xup == 1 && yup == 1 && zup == 0) path[i] = 8;
        else if (xup == 1 && yup == 1 && zup == 1) path[i] = 6;
        else if (xup == 1 && yup == 0 && zup == 0) path[i] = 4;
        else if (xup == 0 && yup == 1 && zup == 0) path[i] = 5;
    }
}
  
```



Şekil B.4'de görünen kod 0,0,0(x,y,z) noktası yukarıda görülen küpün 7 ile numaralandırılmış kısmına denk gelmektedir. Uzayımız bütün eksenlerin pozitif tarafına yerleştirilmiştir.

Şekil B.4: crdToPath

B.3.3.OCTREEWALK (Ek 1)

Bu template'i kullanıcıdan gelen bir path input'u ile ilerleyecek şekilde tasarladık. Bu bilgiyi de path arrayi içinde inilmesi gereken her çocuğun indeksi(1-8) olarak tutuyoruz. (OcTree'nin kökü, pathde her zaman 1 olarak numaralandırılmıştır.)

OcTreeWalk template'i crdToPath fonksiyonu sonucu elde edilen OcTree path'inde ilerleyerek, istenilen noktanın doluluk bilgisini vermeye yarayan template'dir. Bu doluluk bilgileri;

occupancy = 0 Çocuklu node,

occupancy = 1 Bilinmeyen node,

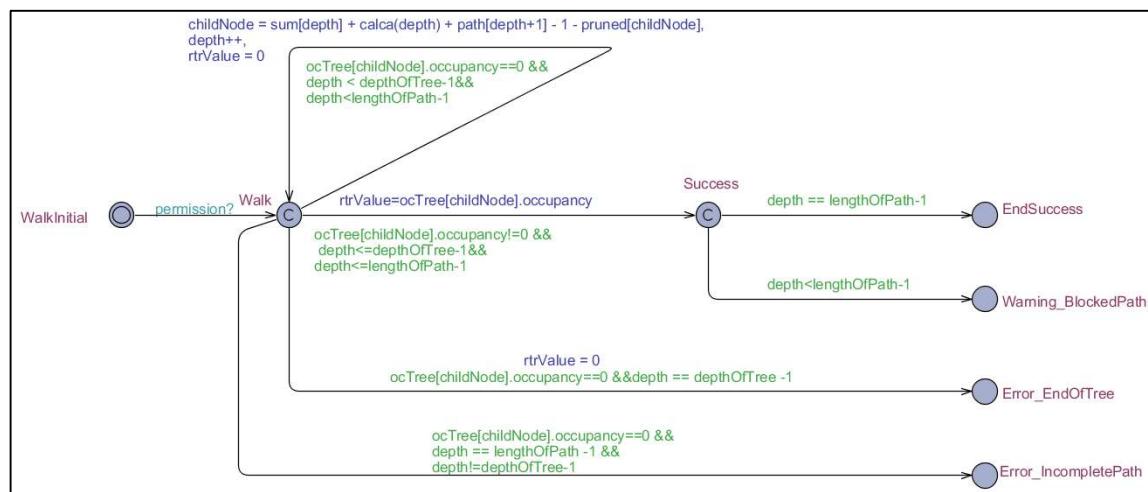
occupancy = 2 Boş node,

occupancy = 3 Dolu node dur.

Bu bilgileri vermesinin yanında oluşabilecek tüm hataları ya da farklı durumları gözden geçirerek bunların bilgisini de verir. Bu template parametre olarak OcTree alır. Doğrulama işleminin yapılacak OcTree, declaration kısmından girilerek bu template'de çalıştırılabilir. OcTree gibi büyük bir veriyi UPPAAL gibi bir aracта declare ederken, hafıza alanından kullanımı düşürmek için geliştirdiğimiz algoritma, bazı node'ların bilgisini tutması gerektiğinden pruned array'i de parametre olarak alınır. Pruned arrayinde tuttuğumuz bilgiler ise yazdığımız C++ koduna OcTree'nin verilmesi sonucu elde edilir. Bu kullandığımız algoritma raporumuzun bir sonraki kısmında detaylıca anlatılacaktır. (Başlık B.4.3)

Name: OcTreeWalk	Parameters: OcTree &ocTree[sizeofTree],int &pruned[sizeofPruned]
------------------	--

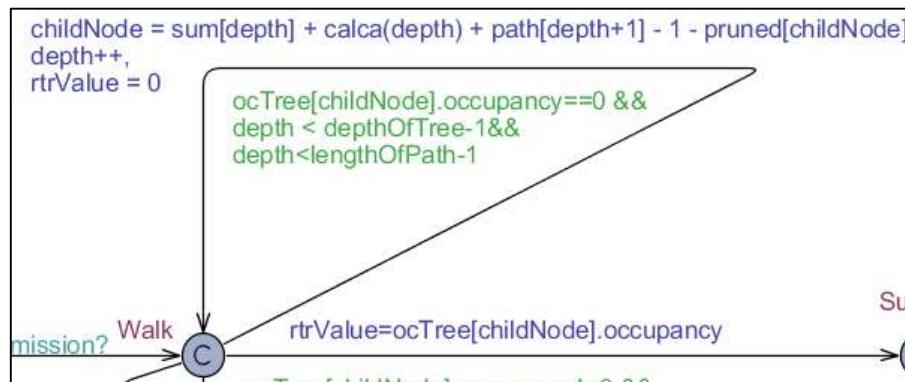
Şekil B.5: OcTreeWalk Parameters



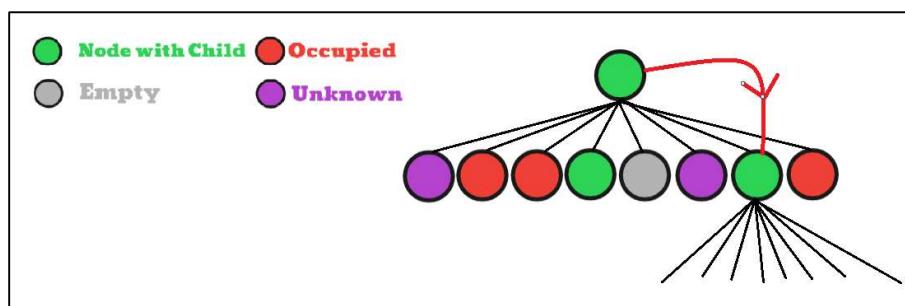
Şekil B.6: OcTreeWalk Template

OcTreeWalk template'i WalkInitial state'ının Idle dan gelecek permission senkronizasyonu sonucu Walk state'ine ilerlemesi ile başlar. Bu state'i seçim yapmaya zorlamak için state committed olarak oluşturulmuştur. Bu template'de en önemli değişkenler childNode ve onun doluluk bilgisidir. OcTree'de ilerleme işlemi sadece çocuklu nodelar üzerinden olacağı için occupancy'nin 0 değerine göre kontroller gerçekleştirilir. 0 olmadığı durum ise OcTree'de daha fazla ilerlenemeyeceği gerekli bilginin alındığını belirtir. Walk state'i OcTree'de , kökten başlar. Bu yüzden depth ve childNode 0 dır. Walk state'nin gidebileceği 4 kenar olsa da hatasız bir işlemde 2 kenara gidebilir.

Şekil B.7 OcTreeWalk görselinde görüldüğü üzere şu anki childNode'un eğer çocuğu(occupancy==0) var ise Walk state bu kenarı seçer. Hata durumlarından korunmak için OcTree'nin derinliğinin ve kullanıcı tarafından girilen path'in uzunluğunun aşılmaması da kontrol edilir. Bu kenarda şimdiki childNode'un çocuğu gerekli işlemler yapılarak bulunur.(Bu işlemler raporumuzun ilerleyen kısmında detaylı bir şekilde açıklanacaktır.(Başlık B.4)) Depth değişkeni artırılarak OcTree'de yeni çocuğun derinliğine inilir. Şekil B.8'de görüldüğü gibi OcTree'de çocuklu bir node'un çocuğuna yürüme işlemi gerçekleşmiş olur.



Şekil B.7: OcTreeWalk

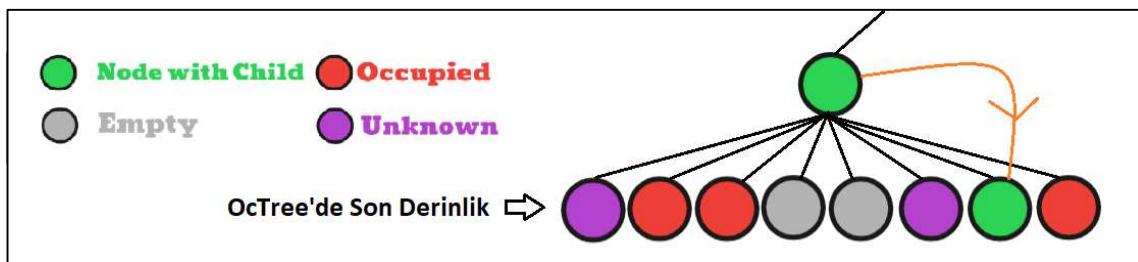


Şekil B.8: Bir alt node'a yürüme işlemi

OcTree'deki `childNode`' doluluk bilgisinin 0(`node`'un parent olduğu) olduğu 2 durum daha vardır. Bunlar hata durumlarıdır.

1.Durum

Eğer OcTree'nin son derinliğine gelinmiş ise ve buna rağmen childNode'un çocuğu var ise , OcTree'nin bittiği hatası Error_EndOfTree alınır. Şekil B.10'da bu durum gösterilmiştir. **OcTree'nin son derinliğindeki bütün çocukların doluluk bilgisi çocuklu bilgisi dışında bir bilgi(Bos,Dolu,Bilinmeyen) olmalıdır.** Bu kenara gidilen bir durumda OcTree'nin son derinliğinde parent node bulunduğuundan kullanıcı tarafından verilen OcTree'de hata vardır.



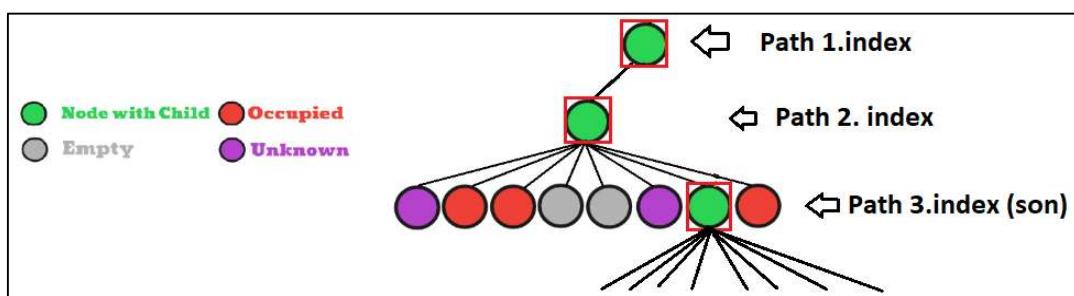
Şekil B.10: Son derinlikte çocuklu olma durumu

2.Durum

Eğer gidilmek istenen path'in sonuna ulaşılmış ise ve buna rağmen childNode'un çocuğu varsa(childNode.occupancy==0), kullanıcı tarafından girilen bir path'in hatalı olması sonucu oluşan, Error_IncompletePath(Şekil B.11) hatasıdır(Şekil B.12).Bu kenarda ayrıca OcTree'nin sonuna ulaşılmamış olması da gerekir. Bu kontrol OcTreeWalk template'sinin bir sonraki state'i seçmesindeki hataları önlemek amacıyla yazılmıştır .Bu kontrolün yapılmadığında, bazı durumlarda yanlış kenar seçilerek Error_EndOfTree hatası alınabiliyordu. Kısacası,Walk state'sinin seçme şansı ortadan kaldırılarak tüm durumlar kontrol edildi.



Sekil B.11: OcTreeWalk

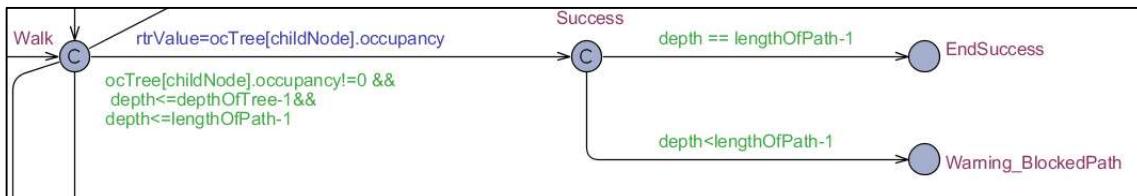


Şekil B.12: Gidilmek istenilen Path'in sonunda son node'un çocuklu olma

Hatasız bir OcTree'de path'e göre çocuklu childNode'lardan ilerleme işlemi gerçekleştiğten sonra path'in sonunda mutlaka çocuksuz node olan dolu, boş ya da bilinmeyen node'lardan biri gelmelidir. Bu özellikteki childNode'a OcTree'nin kökü hariç herhangi bir derinliğinde rastlanabilir. Ayrıca gidilmesi istenilen path'in herhangi bir noktasında da rastlanabilir. Bu yüzden bu kenardaki koşullarımız Şekil B.13 OcTreeWalk 1.5 görselindeki gibidir. Bu childNode'a gelindiğinde ilk aşama başarılı bir şekilde gerçekleştiği için Success state'ine gidilir. Molelimizin Success state'inde kalmaması gerekmekte ve Success state'ini seçim yapmaya zorlamak için bu state de committed olarak oluşturulmuştur. Bu aşamadan sonra karşımıza iki durum çıkarıyor.

1.Durum: Çocuksuz node'a path'in sonunda ulaşılmasıdır. Bu durumda istenilen yolda herhangi bir engel ile karşılaşmadan ilerlenmiştir. Sonuç başarılıdır ve son state olan EndSuccess'e gidilir.

2.Durum: Çocuksuz node'a path'in sonu olmayan bir aşamada ulaşılmasıdır. Örnek olarak, gidilmek istenen path 3 aşamalı olmasına rağmen 2 aşamadaki node çocuksuz bir node'dur ve pathin 3.aşamasına geçiş yapılamaz. Bu durumda gidilmek istenen path bloklanmışdır. Bu durumu hata olarak adlandırmak yerine uyarı olarak adlandırmayı tercih etti ve bu son state'e Warning_BlockedPath ismi verildi.



Şekil B.13: OcTreeWalk

```

const int sizeofTree=81;
Octree octree1[sizeofTree] = {
{0},
{3}, {3}, {0}, {3}, {2}, {0}, {3}, {1},
{3}, {0}, {3}, {0}, {0}, {2}, {3}, {0},
{3}, {0}, {2}, {0}, {0}, {3}, {3}, {2},
{1}, {3}, {2}, {3}, {3}, {1}, {1}, {1},
{3}, {1}, {1}, {3}, {2}, {3}, {2}, {2},
{1}, {1}, {3}, {3}, {3}, {2}, {1}, {3},
{2}, {1}, {1}, {2}, {1}, {2}, {3}, {1},
{3}, {1}, {0}, {1}, {2}, {1}, {1}, {1},
{3}, {3}, {2}, {2}, {1}, {1}, {1},
{3}, {1}, {1}, {3}, {1}, {2}, {3}, {1}};
const int sizeofPruned=25;
int pruned1[sizeofPruned] = {0,0,0,16,0,0,32,0,0,0,184,0,192,192,0,0,208,0,344,0,352,352,0,0,0};

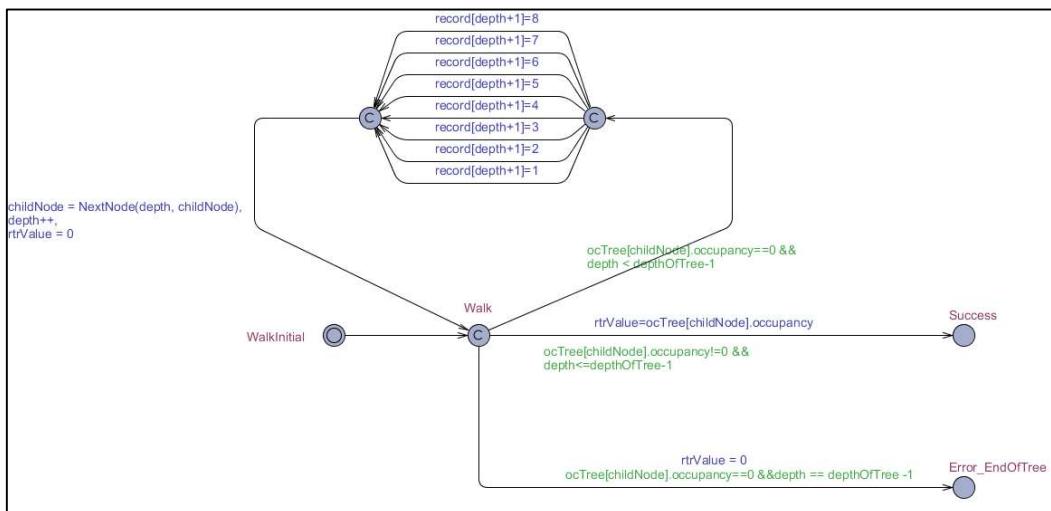
const int lengthOfPath=4;
const int[1,8] path[lengthOfPath] = {1,6,2,3};
  
```

Şekil B.14: OcTreeWalk Declaration

B.3.4 OcTreeWalkDynamic (Ek 2)

OcTreeWalkDynamic template’imiz B.3.3’de bahsedilen OcTreeWalk ile neredeyse aynı işlemi yapmaktadır. **Bu iki template arasındaki en önemli fark path arrayinin dinamik bir şekilde girilmesidir.** OcTreeWalk template’inde path arrayi declaration kısmında verilirken, bu template’imizde path arrayi geçen node’ları tutacak şekilde model üzerinden doldurulur. Template’imizin asıl amacı OcTree üzerinden geçilecek node’lar için daha kapsamlı sorgular yazmaktır. OcTreeWalk template’inde de rtrValue değişkeni üzerinde bulunduğu node’un doluluk(occupancy) bilgisini tutar. Bunun yanında OcTreeWalk’da kullanılan bir sonraki childNode’u bulmaya yarayan formülümüz, bu template’de NextNode fonksiyonu olarak adlandırılmıştır. Aralarında herhangi bir fark yoktur.

Şekil B.15 ‘de görüldüğü üzere çocuklu(occupancy==0) bir node ile karşılaşıldığında çocuklu node’ un 8 çocuğundan herhangi biri istege göre seçilip path belirlenebilir. Çocuksuz bir node ile ağacın kök hariç herhangi bir noktasında karşılaşıldığında ise (Boş,Dolu,Bilinmeyen) işlemimiz başarılı bir şekilde sonuçlanmış olur ve Success state’ine gider.OcTree’nin son derinliğinde çocuklu bir node ile karşılaşıldığında ise Error_EndOfTree state’ine gidilir ve işlem başarısız olur.



Şekil B.15: OcTreeWalkDynamic

B.3.5 MoveUpdate (Ek 3)

Bu başlık altında inceleyeceğimiz sistem; üst başlıkların aksine sıralı bir işlem yolu yerine, iki sistemin birbiri ile haberleşerek aynı anda çalıştığı bir sistem olacaktır. Bu sistemi kurmaktaki motivasyonumuz, octoMap sisteminin, hareketli ve buna bağlı olarak anlık harita güncelleme yapan robot sistemler için kullanılması durumunda, güncelleme, hareket kararı verme ve hareket etme işlemlerini temsili bir şekilde modellemek.

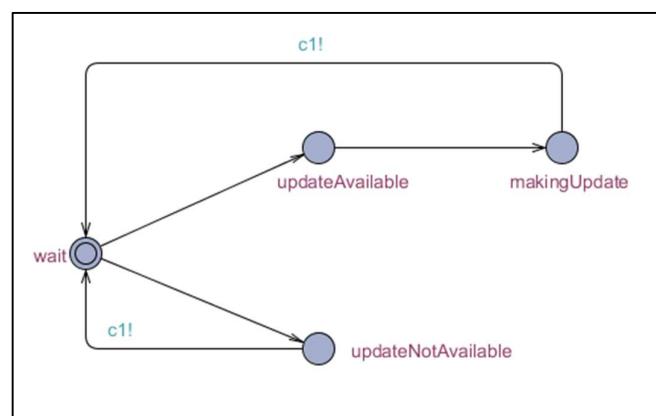
Bu modelimizde ön plana çıkan özellik, iki farklı sistemin channellar kullanılarak birbiri ile sürekli iletişim içerisinde olması. Bu model, UPPAAL üzerinde oluşturulabilecek senkronize sistemlere de güzel bir örnek oluyor.

Şekil B.16 deki template güncelleme döngüsünü, Şekil B.17 deki template ise robotun bulunulan şartlar doğrultusunda hareket kararı vermesi, hareket edilecek alanın ilerlenebilirliğinin kontrol edilmesi ve duruma göre hareket işleminin yapılmasını temsil ediyor.

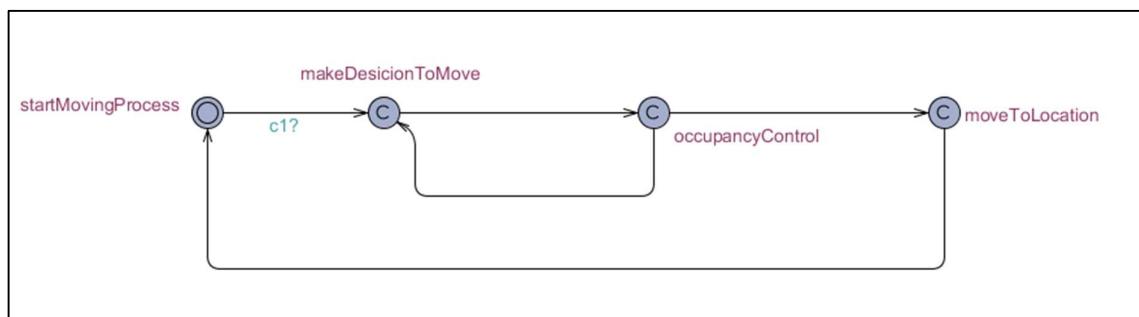
Sistemimiz çalışmaya Şekil B.16 yani update döngüsünün bulunduğu sistem ile başlıyor ve tüm update işlemleri bittiği zaman wait konumuna geçip geçerken de c1 channel'ını uyarıyor. Bu uyarı da Şekil B.17 deki aksiyonun alınabilir olmasını sağlıyor. Hareket aksiyonu bitirdikten sonra update sitemimizden c1 channel'ı yolu ile izin geldiği sürece ilerleme işlemimiz devam ediyor.

İkili sistemimizi tasarlarken; sistemin sağlaması gereken özellik, “update işlemleri yapılan durumlarda hareket işlemlerinin geçici olarak durdurulması” olarak belirlendi. Yani Şekil B.16 wait durumunda değil ise hareket işlem state’lerinde bulunmamız mümkün olmuyor.

Kurduğumuz sistem; spesifik bir çalışma mekanizmasını temsil ediyor olup, farklı octoMap kullanım durumları için uyarlanabilir.



Şekil B.16: Template 1



B.4.OCTREE VERİ YAPISININ UPPAAL ÜZERİNDE MODELLENMESİ

Doğrulama ortamımızı hazırlarken, başlık **B.1** başlığında bahsettiğimiz gibi; UPPAAL üzerinde sistemimizin temsili bir modelini oluşturmamız gerekiyor. Bu başlıkta entegre edeceğimiz model, fiziksel ya da algoritmik bir problemden çok, temel olarak bir veri yapısı problemi olarak karşımıza çıkıyor.

Bahsettiğimiz “veri yapısı” problemini biraz daha açmak gerekirse; OctoMap projesinin üzerine kurulu olduğu ocTree ağaç yapısını burada modelimizin kaynağı olarak ele alıyoruz. Hedef modelimizi ise başlık **A.2.1.2**’de bahsettiğimiz, UPPAAL’ın kullanılabilir veri tipleri üzerinden, olabildiğince kaynağa bağımlı kalarak oluşturmamız gerekiyor. Daha sade bir şekilde şöyle diyebiliriz ki; C++ üzerinde pointerlar kullanılarak oluşturulan bir ağaç yapısını, UPPAAL üzerinde temsili modelini oluşturuyoruz.

B.4.1.Geçirilecek Özellikler

Kaynak sistemden yani ocTree ağaç yapısından, UPPAAL modelimiz üzerine geçirmemiz gereken “olmazsa olmaz” olarak karar verdığımız özelliklerimizden biraz bahsedelim.

- İlk olarak pointer sistemini sağlıklı bir şekilde modellemek için, parent parent’lardan child’lara bir adımda yani O(1) de ulaşabilmemiz gerekiyor.
- İkinci özellikimiz ise veri büyülüğu olarak bir octree nasıl tasarruf yapıyorsa bizim modelimizin de aynı şekilde gereksiz yere fazlalık veri saklamaktan kaçınması gerekiyor.

Bu iki şartı modelimizin temelleri olarak aldığımız zaman modelleme işleminin tam anlamıyla püf noktası diyebeceğimiz kısım başlamış oluyor.

B.4.2.Kullanılabilir UPPAAL Veri Yapıları

Başlık **A.2.1.2**’de bahsettiğimiz yapılara bakacak olursak, ocTree modellerken kullanabileceğimiz ileri seviye yapılar olarak elimizde sadece struct ve array’ler kalıyor. Bir ocTree node’unu temsil etmesi için struct yapısını, verileri birbirine bağlı şekilde tutabilmek için ise array yapısını kullanıyoruz. Sonuç olarak temel yapımız bir struct arrayi olmuş oluyor. Buradaki önemli kısım ise order (1) de işlem yapabilen bir array formülasyonunu en az alan kullanarak implemente etmek oluyor.

B.4.3.Modelleme Yöntemimiz

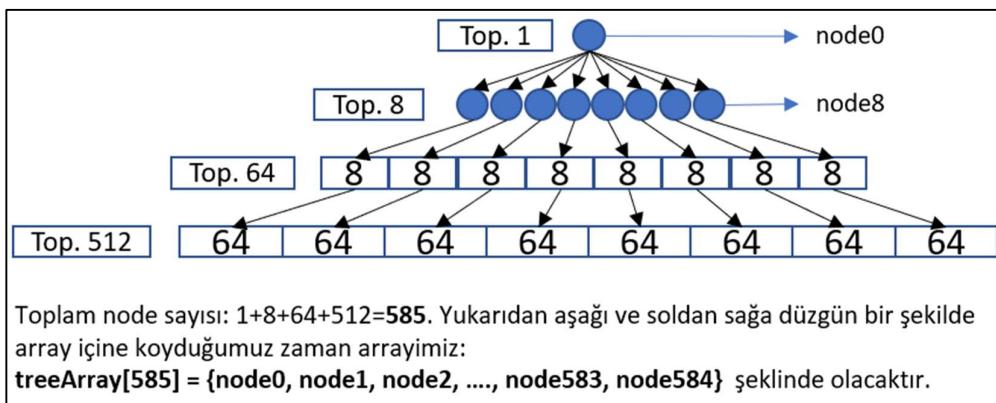
Şimdiye kadar üzerinden geçtiğimiz konulara bakacak olursak; bir tarafta modellenmesi gereken bi ağaç yapısı, diğer tarafta ise bu model için kullanabileceğimiz array ve struct yapıları var. Burada karşılaştığımız durum, sektörün her alanında da karşımıza çıkan time-space ikilemi oluyor. Tamamen hızlı çalışmaya yönelik bir modelleme tasarlayarak ocTree içerisindeki bilgiyi, 3 boyutlu bir point cloud (3D array) olarak saklasak, inanılmaz bir alan harcaması yapmamız gerekiyor ve sistemimizin verimliliği açısından kötü bir duruma düşmüş oluyoruz. Diğer taraftan hiç fazladan alan kullanmadan sadece ağacın içerisindeki bilgiyi sıralı

bir şekilde tutan bir array tasarlayıp işlemlerimizi array üzerinde arama yaparak gerçekleştirirsek bu sefer de sistemimizin çalışma hızı büyük oranda etkileniyor.

Bu şartlar altında bizim getirdiğimiz çözüm ise; elimizdeki bilgiye oranla az bir fazladan alan harcaması yaparak, ağaç üzerinde ilerleme işlemini tek adımda $O(1)$ gerçekleştirmemizi sağlıyor.

Modellememizi array üzerinden yapacağımıza kesinleştiğine göre şimdiki problemimiz array üzerinden bilgiye nasıl ulaşacağımız oluyor. Başlık A.4.2.1'de bahsettiğimiz gibi bir ocTree içerisinde budanmış dalların olması muhtemel. Bu da demek oluyor ki array içerisinde bir çocuğa ilerlerken kullanacağımız matematiksel formül karmaşıklaşıyor.

Öncelikle hiç budanmamış bir ocTree ağaç yapısını ele alalım Şekil B.18 de gördüğümüz ağaç düz bir array içerisinde koyacak olursak her node'un 8 çocuğu olduğunu bildiğimiz için 8 sayısı üzerinden bir formülasyon gerçekleştirebiliriz. Fakat üzerinde çalıştığımız ağaçlar budanmış ağaçlar olacağı için burada ekstra bir bilgiye ihtiyacımız olacak. Bu bilgiyi ise "pruned" ismini verdığımız array içerisinde saklıyor olacağız. "pruned" array hakkında daha ayrıntılı bilgi vermeden önce, "eğer ağaç hiç budanmamış olsaydı formülümüz nasıl olurdu?" sorusunu cevaplayıp onun üzerine pruned arrayimizi getirerek devam edelim.



Şekil B.18: Budanmamış ağaç formülasyonu

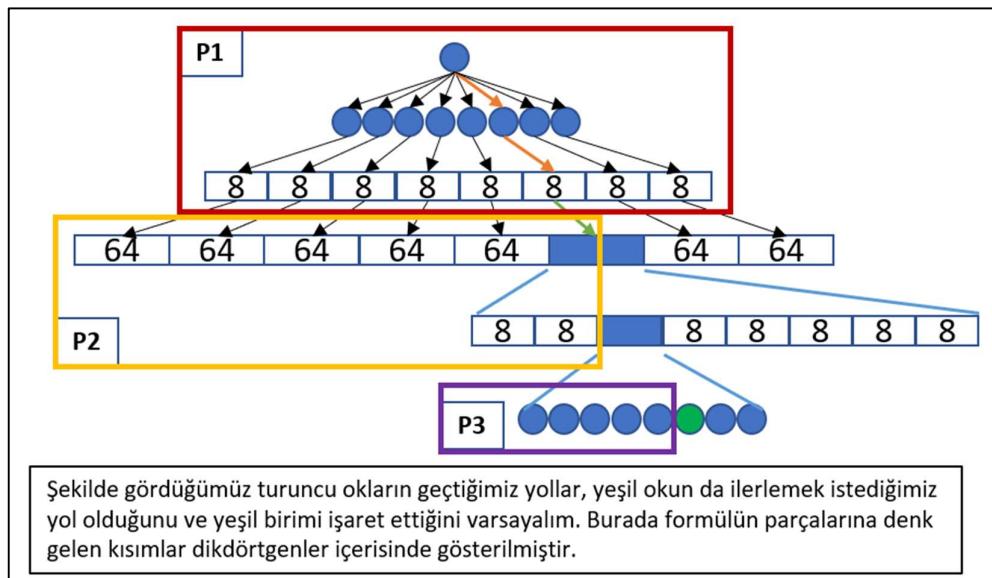
Eğer ağaçımız hiç budanmamış olsaydı ağaç üzerinde çocuk node'a ilerleme formülümüz üç parçadan oluşacaktı;

1. Ağacın bulunduğu derinlikten üstteki derinliklerin toplam node sayısı
2. Yine bulunduğu derinliğe bağlı olarak içerisinde bulunduğu 8'lik bütünü sol tarafındaki nodelar'ın toplamı
3. Son olarak ilerleyeceğimiz çocuğun indexi

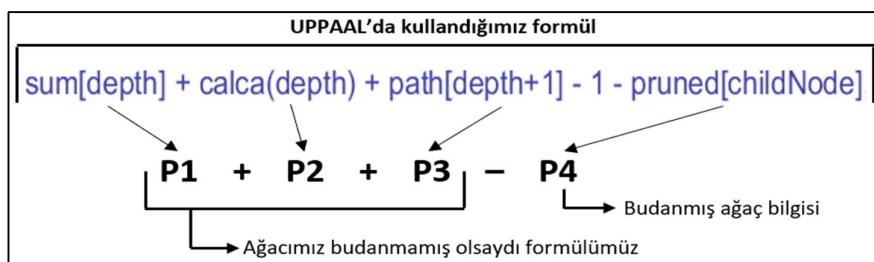
Şekil B.19'da formül parçalarının temsil ettikleri alanları görebilirsiniz.

Bahsettiğimiz bu 3 formül parçasının toplamı ise görebileceğimiz gibi bize budanmamış bir ağaç yapısını formülize etmeye yetiyor. B.4.3.1 başlığında formülü ayrıntılı olarak inceleyeceğiz.

Bu 3 formül parçasından sonra ise “**pruned**” olarak adlandırdığımız budanmış ağaç bilgisi devreye giriyor ve formülün sonuna negatif bir değer olarak eklenip kesilmiş yerlerin değerlerini ana formülden çıkartarak **4.** Bir formül parçası olarak formülümüzü tamamlamış oluyor. (Şekil B.20)



Şekil B.19: Ağaç modeli



Şekil B.20: Formül

B.4.3.1. Formül Ayrıntıları

Bu başlık altına formülümüzün her bir parçasını ayrıntılı olarak inceleyeceğiz.

Parça 1 – Üstte Kalan Nodelar

Parca 2 – İçerisinde Bulunulan 8'lik Dışarısında Solda Kalan Nodelar

Parça 3 – İçerisinde Bulunulan 8’lik İçerisinde Solda Kalan Nodelar

Parça 4 – Budanmış Nodelar’ın Eksiltilmesi

Formül: Parça 1

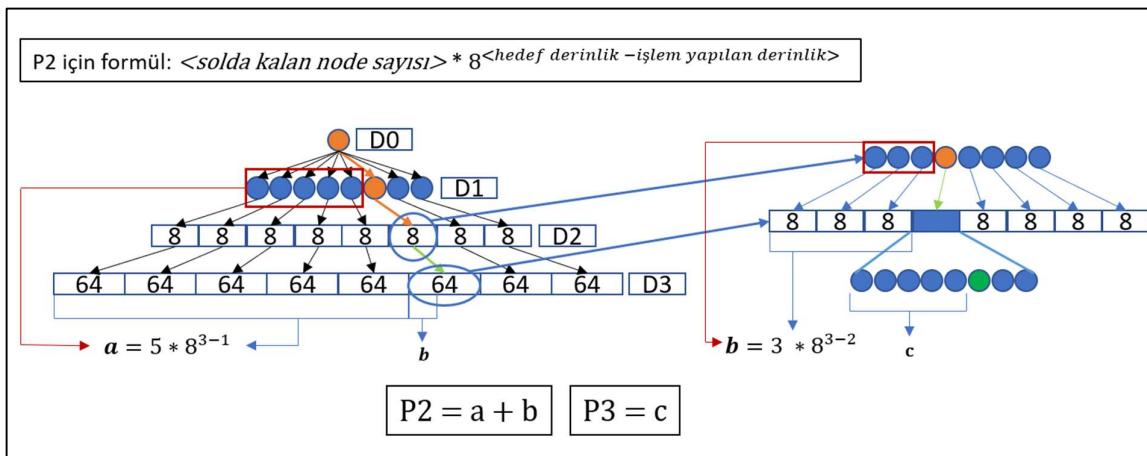
Bu parçada Şekil B.19'de gördüğümüz 1. Kısmı formüle ekliyoruz. Bu kısım ağacın yapısından tamamen bağımsız ve sadece derinliğe bağlı olarak değişen bir sayı. Derinlik dışında herhangi bir şart olmaksızın tek adımda erişebileceğimiz bir değer.

Kendi implemantasyonumuzda bu değeri bir array içerisinde koyup, arrayin index değerlerini de derinliğe bağlı şekilde ayarladık. (Bkz. Şekil B.20) Bu şekilde tek adımda birinci parçamıza ulaşmış oluyoruz.

Formül: Parça 2

Bu parçada Şekil B.20'de gördüğümüz 2. Kısmı formüle ekliyoruz. Yine ağacın yapısından bağımsız olarak çalışan bu parça da parça 1'e benzer şekilde derinliğe bağlı şekilde çalışıyor ve hangi çocuklara ilerlediğimizin bilgisini yani "path" arrayini de kullanıyor.

Bu parçayı hesaplamak içerisinde bulunduğu 8'lik bütününe solda kalanının, sonuç aradığımız derinlik seviyesine yansiyacak olan çocukların sayısını alıyoruz. Fakat sadece içerisinde bulunduğu 8'lik yetmeyeceği için ağaç üzerinde geldiğimiz yola bakarak bu işlemi 1. Derinliğe -yani sadece 8 node olan derinlige- gelene kadar tekrarlayarak, parça 2 yani solda kalan node sayısını buluyoruz.(Bkz. Şekil B.21)



Şekil B.21: Ağaç üzerinde detaylı formül gösterimi

Formül: Parça 3

Bu parçamız en kolay parçamız olarak basitçe bulunan parent'dan hangi indexdeki child'a ineceğimiz bilgisini formüle ekliyor ve ağacımız budanmamış olsaydı gitmemiz gereken index ne olurdu sorusunun cevabını buraya kadar buluyoruz.

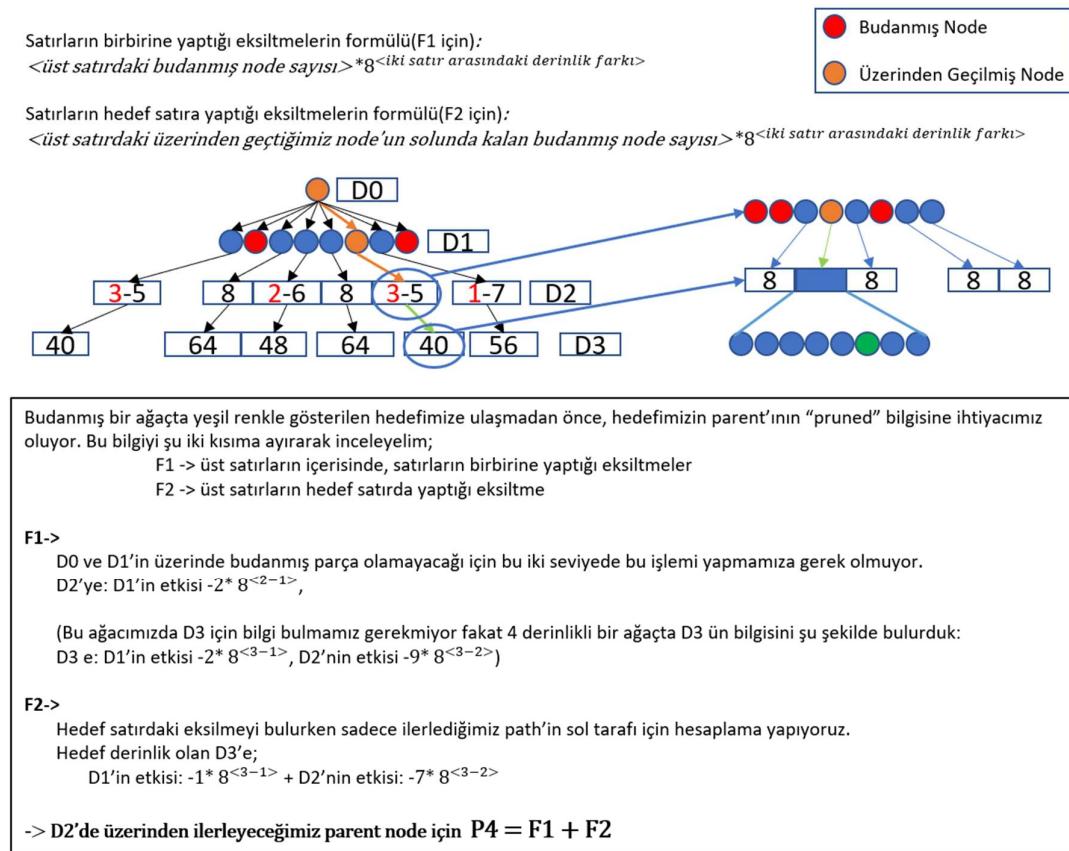
Formül: Parça 4

4. parçamız ise bize kesilmiş ağaç bilgisini veren parçamız "pruned" arrayinin parent node'a denk gelen bilgisini formülden çıkarttığımız zaman formülümüz sonlanmış oluyor. Bu

başlık altında asıl olarak inceleyeceğimiz şey bu bilgiyi “pruned” arrayine yazarken yaptığımız işlemler.

“pruned” arrayini bir parent’dan child’a ilerlerken kullanacağımız için bilgiyi sadece parent olan node’lar için tutuyor, bu da demek oluyor ki ağacımızın en derin seviyesi için bu bilgiyi tutmamız gerekmıyor ve önemli bir alandan tasarruf ederek ihtiyacımız olan bilgiyi tutmamızı sağlıyor. Her node’un karşılığında bu bilgiyi tutmayacağımız için bu bilgiyi her node’u temsilen tuttuğumuz struct yapısına koymak yerine ayrı bir array olarak tutuyoruz.

“pruned” arrayini her parent için tuttuğumuzu söylemişik. Arrayimizin oluşturulma aşamasından biraz bahsetmemiz gerekiyor. Arrayi oluşturmak için üzerinde çalıştığımız ocTree yapısının son derinliği dışında bütün node’larını search etmemiz gerekiyor. Biz bu işlemi kendi tasarladığımız rastgele ocTree oluşturan C++ kodumuz içerisinde ocTree oluşturulurken gerçekleştiriyoruz. Bu işlemi bir kere gerçekleştirdikten sonra tekrarlamaya gerek olmuyor ve aynı ağaç üzerinde işlemlerimizi $O(1)$ de yapma imkanımız oluyor.



Şekil B.22

Array içerisinde bir değeri oluştururken içeriye istediğimiz bilgiler parça 2 deki algoritmamıza benzer şekilde içerisinde bulunduğuuz 8’liğin sol tarafındaki kesilmiş node’ları -yani occupancy değeri dolu, boş ya da bilinmeyen olan-, daha sonrasında ise path üzerinden geriye ilerleyerek ağacın en üstüne kadar her 8’lik bütününe, kendi pozisyonumuza

göre sol tarafında kalan kesilmiş node'larını sayıyor. Ve bulduğu değerin üzerine de hedef dışındaki bütün derinlik seviyelerindeki budanmış node'ların birbirine etkisini ekleyerek bir parent için çocuğuna ilerlerken kullanılacak olan “pruned” değerini bulmuş oluyoruz. (Bkz. Şekil B.22)

Parça 2'den farklı olarak bu algoritmada sadece hedef derinliğe yönelik bir sayma yerine, her budanmış node'un her derinliğe etkisi derinlik farkına göre hesaplanıp arraye işleniyor. Bunu yapmamızın sebebi her ne kadar bir ağaç yapısı üzerinde çalışıyor olarak görünsek de sade bir array üzerinde çalışıyoruz ve her derinlikteki eksilme bir sonraki derinliğe tamamen aktarılıyor. (Bkz. Şekil B.22)

B.4.3.2. Formülün Bütünleşmesi

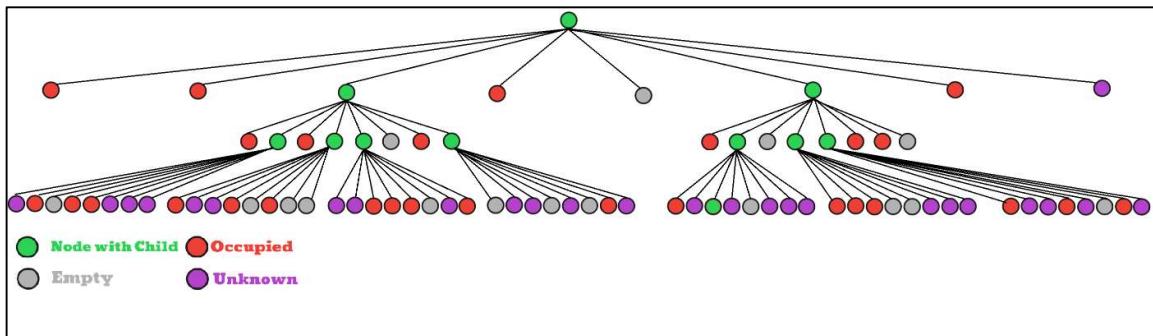
Bütün parçaları bir araya getirdiğimizde kaynak projedeki ocTree ağaç yapısının pointer kullanarak $O(1)$ zamanda çocuk node'a ilerlemesini temsil eden bir formül oluşturmuş oluyoruz. Sistemimizin kaynak ile uyumluluğu bu şekilde minimum ekstra alan kullanarak, tek adımda erişim yaparak sağlamış oluyoruz.

B.4.4. Sonuç

Sonuç olaak UPPAAL üzerinde kullanacağımız struct yapısını node yapısına karşılık olarak, arrayi de ağaç sistemi içerisindeki bilgileri tutan bağlantısal sisteme karşılık olarak kullandık. Maksimum verimliliği sağlamak için işlem sırasında, $O(1)$ de işlem yapmamızı sağlayacak bir formül geliştirdik. Bu formülü geliştirmek için tutmamız gereken “pruned” arrayi ise ağacımızın tüm bilgisine oranla az yer tuttuğu için verimlilik beklenilerimizi karşılayıp, kullanılabilir bir özellik haline geliyor. Elimizdeki imkanlar dahilinde, “çalışma hızı – harcanılan alan” ikileminin üstesinden gelmek için geliştirebildiğimiz en verimli ve kaynak sistem (OCTOMAP->ocTree) ile en tutarlı yöntem olarak, bu modelleme metodunu sunuyoruz.

B.5.SORGULAR

Bu bölümde **B.3** başlığı altında oluşturulan template'ler üzerinde sorgulanan formülleri sunacağız.



B.5.1 IDLE, COORDINATES, OCTREEWALK

Başlık **B.3.1**, **B.3.2** ve **B.3.3**'de bahsedilen template'lerin birleşiminden oluşan ilk sistemimiz üzerinde yaptığıımız sorgular:

- **{1, 3, 4, 5} path inputu için:**

Şekil B.24' de gördüğümüz sorgular her depth için node'un beklenen indexini Walk state'i içerisinde kontrol etmemizi sağlıyor. Şekil B.24'de sorgularımızın hepsinin doğrulandığını görebiliriz.

```
A<>TREEWALK.childNode==0&&TREEWALK.depth==0&&TREEWALK.Walk  
Property is satisfied.  
A<>TREEWALK.childNode==3&&TREEWALK.depth==1&&TREEWALK.Walk  
Property is satisfied.  
A<>TREEWALK.childNode==12&&TREEWALK.depth==2&&TREEWALK.Walk  
Property is satisfied.  
A<>TREEWALK.childNode==37&&TREEWALK.depth==3&&TREEWALK.Walk  
Property is satisfied.
```

Şekil B.24: Sorgular

Elimizdeki ağacın index değerlerine baktığımızda

Depth – Index ilişkisinin

0 – 0

1 – 3

2 – 12

3 – 37

Şeklinde olması bekleniyor.

- {1, 6, 4} path inputu için:

Şekil B.25' de gördüğümüz ilk 3 sorgu her depth için node'un beklenen indexini Walk state'i içerisinde kontrol etmemizi sağlıyor. 4. Sorgu ise path arrayinin boyutunun ağacın derinliğinden küçük olduğu durumlarda doğru uyarı state'ine geçip geçmediğini kontrol ediyor ve Şekil B.25'de de görüldüğü üzere sorgumuz doğrulanıyor.

```
A<>TREEWALK.childNode==0&&TREEWALK.depth==0&&TREEWALK.Walk  
Property is satisfied.  
A<>TREEWALK.childNode==6&&TREEWALK.depth==1&&TREEWALK.Walk  
Property is satisfied.  
A<>TREEWALK.childNode==20&&TREEWALK.depth==2&&TREEWALK.Walk  
Property is satisfied.  
A<>(lengthOfPath<depthOfTree&&TREEWALK.Error_IncompletePath)  
Property is satisfied.
```

Depth – Index ilişkisinin
0 – 0
1 – 6
2 – 20
Şeklinde olması bekleniyor.

Şekil B.25: Sorgular

- {1, 6, 7, 8} path inputu için:

Bu sorgumuzda blockedPath uyarı state'ine doğru koşullar altında ulaşıp ulaşmadığımızı kontrol ediyoruz.

```
A<>TREEWALK.depth<lengthOfPath-1 && rtrValue != 0 && TREEWALK.Warning_BlockedPath  
Property is satisfied.
```

Şekil B.26: Sorgular

Sorgumuzu (Şekil B.26) incelememiz gerekirse; rtrValue değeri, 0 dışında bir değer (1,2,3)lığında eğer input olarak gelen path'in sonuna kadar ilerlenmemiş ise bu state'e varılmasını bekliyoruz. İlk şart derinlik ile path boyutunu karşılaştırıyor, ikinci şart rtrValue değerini kontrol ediyor , son şart ise içerisinde bulunulması gereken state'i temsil ediyor.

- {1, 6, 2, 3} path inputu için:

Bu durumda, varmamız beklenen state Error_EndOfTree olmalıdır. (Şekil B.27) İlk şartta istenilen state'e varılması, ikincide bulunulan derinliğin en derin nokta olup olmadığı, üçüncü ve son şartta ise rtrValue değerinin 0 olup olmadığı kontrol ediliyor. Beklendiği gibi sorgumuz doğrulanıyor.

```
A<>TREEWALK.Error_EndOfTree && TREEWALK.depth == depthOfTree - 1 && rtrValue == 0  
Property is satisfied.
```

Şekil B.27: Sorgular

Bu input altındaki diğer sorgularımız ise yine childNode'un alabileceği değerler ile alakalı olarak beklenen sonucu veriyor. (Şekil B.28)

```

E<>TREEWALK.childNode == 6
Property is satisfied.
E<>TREEWALK.childNode == 18
Property is satisfied.
E<>TREEWALK.childNode == 15
Property is not satisfied.

```

Şekil B.28: Sorgular

Bir diğer sorgumuzda da global olarak koordinat sistemi ile walk sistemi arasındaki ilişki hakkında bir doğrulama yapıyoruz. Şekil B.29'de gösterilen sorgumuzun anlamı koordinat sisteminin bir üyesi olan Crd ile TreeWalk sisteminin bir üyesi olan Walk state'lerinin aynı anda aktif olup olamayacaklarını sorgulayıp, bekleniği gibi hayır cevabını alıyoruz.

```

E<>TREEWALK.Walk && CRD.Crd
Property is not satisfied.

```

Şekil B.29: Sorgular

B.5.2 OcTreeWalkDynamic

B.5.2.1 Derinlik-Index İlişkisi

Başlık B.3.4 içinde bahsedilen OcTreeWalkDynamic template'si üzerinde yaptığımız sorgular:

```

E<> TREEWALK.depth==2 && TREEWALK.Walk && TREEWALK.childNode == 15
Property is satisfied.
E<> TREEWALK.depth==2 && TREEWALK.Walk && TREEWALK.childNode == 30
Property is not satisfied.
E<> TREEWALK.depth==2 && TREEWALK.Walk && (TREEWALK.childNode > 24 || TREEWALK.childNode < 9)
Property is not satisfied.

```

Şekil B.30: Sorgular

Görülebileceği gibi ilk sorgu grubumuzda(Şekil B.30), ağaçın her derinliği için childNode'un alabileceği index değerlerini kontrol ediyoruz.

Şekil B.23'deki ağaç yapısı üzerinde de görüldüğü üzere;

Derinliğe bağlı olarak alınabilecek index değerleri aşağıda gösteriliyor.

D1 -> 0, D2-> [1, 8], D3-> [9, 24], D4-> [25, 80]

Bunlara bağlı olarak; ilk sorgumuz olarak 2. derinlikte childeNode değerinin 15 olmasının olası olduğunu, fakat 30 olamayacağını bekleniği gibi doğruladık.

Üçüncü sorgumuzda ise derinlik 2 iken, `childNode` değerinin, 24 den büyük ya da 9 dan küçük olma durumunun var olup olmadığını sorgulayarak beklediği gibi hayır cevabını alıyoruz.

```
E<> TREEWALK.depth==3 && TREEWALK.Walk && (TREEWALK.childNode > 80 || TREEWALK.childNode < 25)
Property is not satisfied.
E<> TREEWALK.depth==3&& TREEWALK.Walk && TREEWALK.childNode == 24
Property is not satisfied.
E<> TREEWALK.depth==3&& TREEWALK.Walk && TREEWALK.childNode == 25
Property is satisfied.
E<> TREEWALK.depth==3&& TREEWALK.Walk && TREEWALK.childNode == 80
Property is satisfied.
E<> TREEWALK.depth==3&& TREEWALK.Walk && TREEWALK.childNode == 81
Property is not satisfied.
```

Şekil B.31: Sorgular

Şekil B.31'de de derinlik 2 için yaptığımız kontrol işlemlerini derinlik 3 için yapıyoruz ve doğrulamasını yapmış oluyoruz.

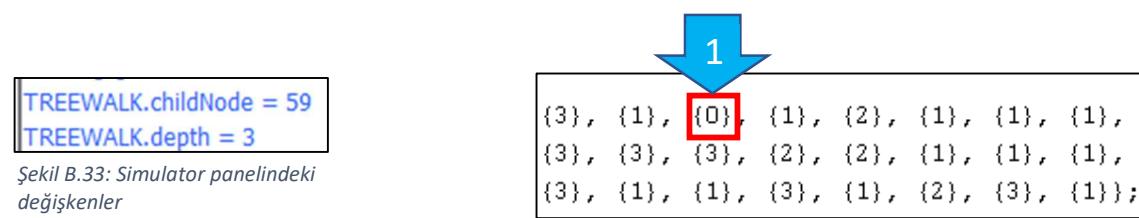
B.5.2.2 Ağacın Doğruluğunun Sınanması

Bu başlık altında inceleyeceğimiz sorgu şekil B.32 de görülebilir. Burada yazdığımız sorgunun, doğru bir ağaç yapısı üzerinde “not satisfied” yanıtını vermesini bekleriz. Çünkü sorguladığımız şey derinlik 3 olduğunda yani, ağacımızın son derinliğinde occupancy değeri olarak 0 barındıran bir node olma ihtimalidir. Fakat Şekil B.32 de gördüğümüz gibi satisfied yanıtı almaktayız.

```
E<> TREEWALK.depth==3 && TREEWALK.Walk && TREEWALK.ocTree[TREEWALK.childNode].occupancy == 0
Property is satisfied.
```

Şekil B.32: Sorgu

Beklemediğimiz bir durum olduğunda, “Diagnostic Trace” özelliği ile sorgumuzun durumunun, UPPAAL simülatör üzerine yüklenmesini sağlayarak hatamıza yol açan ilerleme sürecini görmüş oluyoruz. Bunu yaparak hatamıza gittiğimizde UPPAAL simülatörde `childNode` değerini 59 olarak buluyoruz (Şekil B.33). Şekil B.23'de de görülen ağaç üzerinden kontrol ettiğimizde, ağacımızın son derinliğinde 59 indexli node'da hata olduğunu fark edebiliyoruz.



Şekil B.34: octree declaration parçası

Hatanın yeri bulunduktan sonra declaration kısmından gerekli düzenlemeyi yaparak (Şekil B.34), tekrar sorgumuzu çalıştırduğumda, beklediğimiz gibi “not satisfied” dönütünü alabiliyoruz.

```
E<> TREEWALK.depth==3 && TREEWALK.Walk && TREEWALK.ocTree[TREEWALK.childNode].occupancy == 0  
Property is not satisfied.
```

Şekil B.34: Aynı soru tekrar çalıştırıldığında

B.5.3 Move – Update

B.3.5 başlığı altında incelenen move-update ikili sistemi hakkında bu başlık altında, ilişkisel çalışma ve senkronize olma özellikleri sınanacak.

Şekil B.35 de görülen ilk sorguda; bir update işlemi (state’i) gerçekleştiği sırada, bir hareket işleminin gerçekleşme olasılığının olup olmadığıdır. Sistemin tasarımını üzerine aynı anda herhangi bir update işlemi içerisinde, bir sonraki hareket durumuna, update sistemi wait konumuna geçene kadar izin verilmemesi bekleniyor. İlk sorgumuzun “not satisfy” olması bekłentilerimizi karşılıyor.

Daha sonrasında, ikinci sorgumuzda, update sistemi wait konumunda iken, bir hareket işleminin yapılmama durumunun olup olamayacağı soruluyor ve bekleniği üzere evet cevabı alınıyor.

```
E<> U.makingUpdate and M.moveToLocation  
Property is not satisfied.  
E<> U.wait and M.moveToLocation  
Property is satisfied.
```

Şekil B.35: Sorgular

B.5.4.Sorgu Sonuçları Hakkında

Yukarıdaki başlıklar altında sorguladığımız durumların bekleniği gibi dönüt verdiği, vermediği durumlarda da hatamızı nasıl arayabileceğimiz gösterildi. Her sistem için yazılan sorgular ihtiyaca göre daha da çeşitlendirilebilir.

Bizim istediğimiz sorgular sistemlerin temel özelliklerine yönelik genel kontrollerden oluşuyordu, sistemlerin daha da ayrıntısına inilerek ve yeni gereksinimler eklenerek, sisteme sorulacak sorgular da zenginleştirilebilir.

C.SONUÇLAR

Projemizin ilk safhalarına; öncelikle doğrulama (verification) kavramının tam anlamıyla anlaşılmaması, daha sonrasında ise octoMap teknolojisini çalışılıp, modele dökülebilecek kadar bilgi toplanması gerekti. Bu iki ayrı çalışma kolu üzerinde yeterli bilgiye sahip olunduğunda, bilgiler bir araya getirilerek, projemizin ana safhasına geçildi ve doğrulama, UPPAAL kullanımı ve octoMap konuları üzerine de ortak bir çalışma örneği olan projemiz gerçekleşmiş oldu.

Çalışmamızın her bölümünde tasarladığımız sistem ve yöntemler; sadece üzerinde çalışılan spesifik durumun ihtiyaçlarını karşılamak yerine, daha genel durumlar için kapsayıcı sabit kurallar çerçevesinde oluşturulmaya çalışıldı. Bunun sebebi çalıştığımız durumların daha temiz ve anlaşılır olmasını ve oluşturduğumuz sistemlerin, oluşturulabilecek başka sistemlere örnek niteliği taşımamasını amaçlamamızdır.

Aldığımız sonuçlar açısından konuşacak olursak, UPPAAL üzerinde takibini yaptığımız sistemlerin karşılaşması gereken özelliklerini sorguladığımızda, sağlıklı ve beklenileri karşılayacak şekilde davranış gösterdiklerini gözlemledik. Özellikle başlık **B.5** altında çalışmalarımızın sonucu niteliğinde kullandığımız sorguları bulabilirsiniz.

Tasarladığımız modeller yine aynı konu(octoMap-haritalama) içerisinde çok daha çeşitlendirilip, odakları değiştirilebilir. Bu proje dahilinde bütün algoritmalar için modelleme yapma imkanımız olmadığından, fakat oluşturulan temel taşlar sayesinde, modelleri daha ileriye taşımak mümkündür.

D.PROJE EKİBİ DEĞERLENDİRMESİ

Proje ekibimizde alınan bütün kararlar yapılan toplantılar ile belirlendi. Ekibimizdeki herkes elinden geldiği kadar sıkı çalıştı. Zorlanılan konularda, problemler el birliği ile çözüldü, projemizin genel hedefi ve gidişatı, Zekeriyya Hoca'mızın da katıldığı düzenli olarak yapılan değerlendirme toplantılarında konuşuldu.

(Koordinatör: Muhammed Talha Şahin)

Ön rapor sonrası JIRA zaman raporu :

Key		Summary		Original Estimate	Σ	Est. Time Remaining		Σ	Time Spent		Σ	Accuracy	Σ
<input checked="" type="checkbox"/>	VMOU-6	DONE	2. rapor hazırlanması ve proje çözümüne yönelik çalışma	-	2d 5h	-	2h	-	2d 2h 30m	-	30m		
	VMOU-7	DONE	↳ 2.Raporun Oluşturulması(Muhammed Talha Şahin)	-	-	0m	-	1h	-	?	-		
	VMOU-9	DONE	↳ UPPAAL üzerinde modelleme işlemi yapılması(Hüseyin Can Ergün)	4h 30m	-	0m	-	3h	-	1h 30m	-		
	VMOU-10	DONE	↳ Kullanılacak fonksiyon ve algoritmaların C++ üzerinden yazılması(1)(Mu...)	1d	-	0m	-	1d	-	on track	-		
	VMOU-11	DONE	↳ UPPAAL üzerinde modelleme işlemi yapılması(2)(Yusuf Kenan Aksan)	4h 30m	-	1h	-	3h 30m	-	on track	-		
	VMOU-12	DONE	↳ Kullanılacak fonksiyon ve algoritmaların C++ üzerinde yazılması(2)(Efek...)	4h	-	1h	-	3h	-	on track	-		

Key		Summary	Original Estimate	Σ	Est. Time Remaining		Σ	Time Spent		Σ
☒	VMOU-13	DONE Bug Fix (Efekan Sargin)	2h	2h	0m	0m	4h 30m	4h 30m		
☒	VMOU-14	DONE Bug Fix (Muhammed Talha Şahin)	2h	2h	0m	0m	5h	5h		
☒	VMOU-15	DONE Bug Fix(Hüseyin Can Ergün)	2h	2h	0m	0m	4h 45m	4h 45m		
☒	VMOU-16	DONE Bug Fix (Yusuf Kenan Aksan)	2h	2h	0m	0m	3h	3h		
✓	VMOU-17	DONE UPPAAL Sorgularının Araştırılması , Raporlanması (Hüseyin Can Ergün)	3d	3d	0m	0m	3d 3h	3d 3h		
✓	VMOU-18	DONE Modellemenin Raporlanması , Gerekli Araştırmaların Yapılması(Muhammed...)	1d	1d	0m	0m	3d 15m	3d 15m		
✓	VMOU-19	DONE Templatelerin Raporlanması , Gerekli Araştırmaların Yapılması(Efekan Sargin)	1d	1d	0m	0m	3d 5h	3d 5h		
✓	VMOU-20	DONE XML UPPAAL Bağlantısı & Ders İlişkisi Araştırılması , Raporlanması(Yusuf Ke...)	1d	1d	0m	0m	3d 3h	3d 3h		
✓	VMOU-21	DONE UPPAAL Sorgularının Templatelere Göre Yapılıp Modellemesи(Hüseyin Can...)	5h	5h	1h	1h	4h	4h		
✓	VMOU-22	DONE UPPAAL Sorgularının Templatelere Göre Yapılıp Modellemesи(Efekan Sargin)	5h	5h	1h	1h	4h	4h		
✓	VMOU-23	DONE UPPAAL Sorgularının Templatelere Göre Yapılıp Modellemesи(Muhammed...)	5h	5h	1h	1h	4h	4h		
✓	VMOU-24	DONE UPPAAL Sorgularının Templatelere Göre Yapılıp Modellemesи(Yusuf Kenan...)	5h	5h	1h	1h	4h	4h		

Projemizin JIRA Üzerinden Zaman-Takip-Görev Raporu

	Efekan Sargin	M. Talha Şahin	H. Can Ergün	Y. Kenan Aksan
Top. Adam-Gün	5.1	5.25	4.75	4.65

TOPLANTILAR

- 1) 12.04.2021 – Zekeriyya Hocamız ile Zoom platformu üzerinden projenin gidişatı hakkında bilgi verildi ve geri dönüş alındı. Proje hedefimiz spesifikleştirildi.
- 2) 23.04.2021 – UPPAAL kullanımı ve projenin çözümü hakkında fikir alışverişi yapıldı. Olası UPPAAL modelleri gözden geçirildi. Oluşturulacak modele karar verildi.
- 3) 25.04.2021 – Oluşturulan rapor gözden geçirildi.
- 4) 29.04.2021 – Zekeriyya hocamızla yapılan zoom toplantılarında elde ettiğimiz bilgiler paylaşıldı ve proje hakkında geri dönüşleri doğrultusunda neler yapacağımıza karar verildi.
- 5) 6.05.2021 – Zekeriyya hocamızla yapılan zoom toplantılarında projede gelinen UPPAAL sistemi tanıtıldı ve error handling bölümünün yetersiz olduğunda karar kılınıp karşılaşabileceğimiz error’leri çeşitlendirme kararı alındı. Rapor hakkında görüşler alınarak küçük düzeltmeler yapıldı.
- 6) 11.05.2021 – Proje gidişatını konuşmak üzere grup üyeleriyle beraber toplantı yapıldı. Bu toplantıda sorgular üzerinden hata ayıklama ve sistemi daha verimli/düzenli yapma konuları hakkında konuşuldu.
- 7) 14.05.2021 – Projenin eksikleri hakkında konuşuldu ve rapordaki eksikliklerin giderilmesi üzerine iş bölümü yapıldı.

8) 16.05.2021 – Toplanan/öğrenilen bilgiler doğrultusunda grup içerisinde öğrenilen bilgiler paylaşıldı ve ilerleyen günlerde yeni bilgilerin sisteme geçirilmesi/raporlanması konusunda anlaşıldı.

9) 17.05.2021 – Raporlama aşamasında grup üyelerinin görüşleri alındı ve yeni bilgiler rapora eklendi.

-> Kaynak kodları geliştirilirken takip sistemi olarak Bitbucket kullanıldı. Kod deposunun son hali aşağıdadır.

Name	Size	Last commit	Message
01.cpp	2.21 KB	2021-04-29	first versions of C++ code and xml file of UPPAAL model
1.4.cpp	4.28 KB	2021-05-01	pruned succ.
1.5.cpp	4.34 KB	2021-05-01	pruned succ.
model.v1.2.xml	7.12 KB	2021-04-29	first versions of C++ code and xml file of UPPAAL model
model.v1.3.xml	5.5 KB	2021-04-29	model with octree impl. no data
model.v1.4.xml	6.21 KB	2021-05-01	pruned init.
model.v1.5.xml	6.62 KB	2021-05-01	pruned succ.
model.v1.6.xml	7.01 KB	2021-05-05	parametre verildi. calc fonksiyonu düzlenlendi
model.v1.7.xml	7.86 KB	5 days ago	model.v1.7 Error Handling
model.v1.8.xml	7.46 KB	3 days ago	model.v1.9 Bug Fixed
model.v1.9.q	719 B	3 days ago	model.v1.9 Bug Fixed
model.v1.9.xml	7.45 KB	3 days ago	model.v1.9 Bug Fixed
model.v2.0.q	466 B	2 days ago	2.0 queries
model.v2.0.xml	8.05 KB	2 days ago	görünüş düzenlenmesi
model.v2.1.q	466 B	4 minutes ago	model v2.1 queries
model.v2.1.xml	8.13 KB	4 minutes ago	free path model bug fix
model.x.1.0.q	183 B	2 minutes ago	abstract model x queries added
model.x.1.0.xml	2.33 KB	3 minutes ago	abstract model x added
tree1.txt	634 B	2 minutes ago	auto generated tree examples
tree2.txt	1.5 KB	2 minutes ago	auto generated tree examples

Bitbucket Proje Commitleri

E.KAYNAKÇA

1. Schneider Gerardo, Model Checking: A Complement to Test and Simulation
(<http://www.cse.chalmers.se/~gersch/model-checking/What-is-Model-Checking.html>)
2. Clarke Jr, Edmund M., et al. Model checking. MIT press, 2018.
(https://books.google.com.tr/books?hl=tr&lr=&id=qJl8DwAAQBAJ&oi=fnd&pg=PR7&dq=model+checking&ots=ssX-g-n3CI&sig=k3t7XR5BVIuiqhZIBrk_H2waRAU&redir_esc=y#v=onepage&q=model%20checking&f=false)
3. Şadi Evren Şeker, Zamansal Mantık
(<http://bilgisayarkavramlari.com/2009/05/06/zamansal-mantik-temporal-logic/>)
4. Gerd Behrmann, Alexandre David, and Kim G. Larsen (A Tutorial on Uppaal 4.0) Kasım 28, 2006 (<http://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf>)
5. Armin Hornung · Kai M. Wurm · Maren Bennewitz · Cyrill Stachniss · Wolfram Burgard (OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees) Aralık 31, 2012 (<http://www.arminhornung.de/Research/pub/hornung13auro.pdf>)
6. Proenza Julian , The UPPAAL Model Checker, sf(35-38/56-59)
(<http://ppedreiras.av.it.pt/resources/empse0809/slides/TheUppaalModelChecker-Julian.pdf>)
7. Barış Dayak, XML Nedir? XML Neden Kullanılır?,(Erişim Tarihi 15.05.2021)
(<https://www.barisdayak.com/xml-nedir-xml-neden-kullanilir-xml-dosya-turu/>)
8. (<https://people.cs.aau.dk/~marius/utap/syntax.html>) (Erişim Tarihi 15.05.2021)
9. Verifying Real-Time Systems – The UPPAAL Model Checker , sf 12, (Erişim Tarihi 16.05.2021)(http://groups.di.unipi.it/~maggiolo/Lucidi_TA/VerifyingTA-Uppaal.pdf)
10. Ahmet Yazıcı, Biçimsel Diller ve Otomata Finite Automata, (2_BDO_FiniteAutomata.pdf) Ders Dökümanı ,sf 13, (Erişim Tarihi 16.05.2021)
11. (<https://www.seas.upenn.edu/~lee/09cis480/lec-part-4-uppaal-input.pdf>) (Erişim Tarihi 05.05.2021)