TÜBİTAK

TÜRKİYE BİLİMSEL VE TEKNOLOJİK ARAŞTIRMA KURUMU

THE SCIENTIFIC AND TECHNOLOGICAL RESEARCH COUNCIL OF TURKEY

REALISTIC APPLICATIONS OF ACTION LANGUAGES FOR WORKFLOW MANAGEMENT

PROJE NO : 100E024

Elektrik, Elektronik ve Enformatik Araştırma Grubu

Electric, Electronics and Informatics Research Grant Committee

REALISTIC APPLICATIONS OF ACTION LANGUAGES FOR WORKFLOW MANAGEMENT

PROJE NO : 100E024

DOÇ. DR. FERDA NUR ALPASLAN

KASIM 2004

ANKARA

İÇİNDEKİLER

ÖZ

Bu projenin iki amacı vardır. Bunlardan ilki, kendi geliştirdiğimiz iş akışı tanımlama dilini kullanarak tanımlanan iş akışlarını C eylem dilinde yazılmış program haline getiren bir çevirmen sisteminin geliştirilmesi, C eylem dilindeki bu programlar için model(ler) bulma, ve bu programların nasıl çalıştığını görmek için geçiş şemalarını çizen bir araç geliştirmedir. Projenin diğer bir amacı ise, iş akışı konusunda elde ettiğimiz bilgi ve deneyimlerle başka problemlerin çözümü için C eylem dilini ve onun getirdiği altyapıyı kullanmaktır. Bu amaçla, bir ön çalışma olarak, gerçek hayattan alınma bir trafik problemini C dilini kullanarak çözmüş bulunmaktayız.

İş akışı yönetimi sistemleri kurumların otomasyon işlemleri için ümit vaad eden bir çözümdür. İş akışını modellemek ve çalıştırmak için pek çok ticari ürün mevcuttur. Ancak, iş akışı yönetim sistemlerinin kuramsal temelleri eksiktir. Diğer bir yandan, eylem dilleri, eylem sistemlerini tanımlama problemine en son yaklaşımlardan biridir. C dili, nedenselliğin açıklanması kuramını temel alan, bir yüksek düzey eylem tanımlama dilidir.İş akışı süreçlerinin biçimselleştirilmesi için yapılan pek çok öneriden biri eylem tanımlama dili C'nin kullanılmasıdır. Bu projede, iş akışı süreçlerinin eylem tanımlama dili C kullanılarak biçimselleştirilmesinde kullanılması amacı ile bir yüksek düzey iş akışı tanımlama dili olan WoDeL tasarlanmıştır ve WoDeL dilinden C diline bir çevirim tanımlanmıştır. Ayrıca, çevirim işinin otomatik olarak yapılabilmesini sağlamak için bir çevirim aracı geliştirilmiştir. Bu biçimselleştirme, iş akışı süreç tanımları üzerinden akıl yürütme işlemlerinin yapılabilmesine olanak sağlayan bir zemin oluşturmaktadır. Ayrıca, önerilen biçimselleştirme geliştirilerek iş akışı süreçlerinin doğrulanması ve en iyilenmesi amaçları için kullanılabilir.

Ayrıca iş akışı sürecini görsel olarak izlemeye yarayan bir araç geliştirilmiştir. C diline çevrilen programları farklı model bulucularla denemek mümkündür.

ABSTRACT

This project has two aims: First, we intend to develop a translation tool which translates a workflow defined using our workflow specification language into action language C, find the possible models of the system by using satisfiability-solvers, and visualize the execution of the C program via transition diagrams. Then, with this, we aim to formalize real-world applications using C. As a preliminary result, we have (to a reasonable extent) solved a real-world problem, namely TRAFFIC scenario, which can be thought of as a specific workflow problem.

Workflow specification using C is introduced as a real-world application area of action languages. Workflow is concerned with the automation of procedures where documents, information or tasks are passed between participants according to a defined set of rules to achieve, or contribute to, an overall business goal. A workflow management system defines, manages and executes workflows through the execution of software whose order of execution is driven by a computer representation of the workflow logic. To be suitable for workflow management, a workflow specification language (WODEL) has been defined. Processes defined using this language is translated into C. This translation makes it possible to preserve the power of representing actions and their effects as well as reasoning about them. Graphical representation of C programs helps the users to visualize the entire workflow. All possible plans can be generated using different satisfiability-checkers.

1.GİRİŞ

Eylemler ve ve bunların etkileri ile ilgili çıkarımlar yapmak yapay zeka konuları içinde yer alan bilgi modelleme (knowledge representation) ve planlama (planning) ilgili önemli çalışmalardır. Akıllı bir etmen (intelligent agent) değişen bir dünyada yaptığı eylemleri ve bunların sonuçlarını gözelmelek, kaydetmek, ve bunlarıla ilgili çıkarımlar yapmak ihtiyacı duyar. Bilgi gösteriminin önemli problemlerinden biri, bu tür bilgileri, çeşitli çıkarım mekanizmaları ile kullanılabilecek bir gösterim kullanarak ifade etmektir [1].

Eylem dilleri eylemler ve etkilerini ifade etmek için kullanılan formal dillerdir. Gelfond ve Lifschitz tarafından, A, B, ve C adı verilen üç ayrı eylem dili tanımlanmıştır [2]. Bu projede, bu dillerin sonuncusu olan C diline ağırlık verilmiştir. C dili McCain ve Turner tarafından geliştirilen etkisel açıklama (causal explanation) teorisine dayanmaktadır [3]. Gelfond ve Lifschitz C'nin ifade gücünü ortaya çıkarmak için bu dili çeşitli örnek problemlere uygulamışlardır [4].

Eylemlerin gösterimi, çıkarım mekanizmalarının otomatikleştirilmesi, ve çeşitli problemlerin C dilinde çözümü gibi bu konuda sürdürülen araştırmalar [3,4,6], cevap kümesi mantığı (answer set semantics) altında mantık programlama (logic programming) kullanarak çözümler üretmek yönünde gelişim göstermektedir [5]. C dilinden mantık programlarına çevirim yapmak, SMODELS [7], dlv[8], v.b. gibi cevap kümesi mantığına dayalı sistemler için bir temel oluşturmaktadır[9].

Bilgi modelleme yapay zekanın en önemli konularından birisidir. Tasarlanan bir ortamda, akıllı davranışlar sergileyen bir program ya da makinanın bu ortam ile bilgileri bir şekilde kaydedip kullanması gerekir. Bunun için de bu bilgiyi kuşkuya yer bırakmayacak şekilde modelleyip kullanabilen bir dile gereksinim vardır. Böylelikle sistemin bilgilerden çıkarımlar yapması, sorgulara cevap vermesi, ve tuttuğu bilgileri gerektiğinde güncelleyebilmesi mümkün olacaktır [10].

Başlangıçta bu amaçla klasik mantık (classical logic) kullanılmıştır. Ancak zamanla bunun monoton olmayan çıkarımlar (nonmonotonic reasoning) yapmak için yeterli olmaması nedeniyle yeni arayışlar içine girilmiştir. Çünkü gerçek hayatta, önceden yaptığımız çıkarımların sonradan geçerliliğini yitirmesi olasıdır. Halbuki klasik mantık gibi monoton çıkarım mekanizmaları kullanan sistemlerde bu mümkün değildir. Monoton olmayan matık sistemleri (nonmonotonic logics) bu motivasyonla ortaya çıkmıştır. Bunların içinde en çok bilinenler *circumscription* [11,12,13], *default logic* [14], ve *nonmonotonic modal logics* tir [15,16,17]. Bu konuda 1987 yılına dek yapılan tüm çalışmaları topluca [18,19] da bulmak mümkündür.

Bu konuda farklı yöndeki bir çalışma da Green [20], Hayes [21] ve Kowalski [22] tarafından başlatılan ve bir mantık programlama [23] dili olan Prolog'un [24]ortaya çıkmasına yol açan araştırmadır.

İş akşı yönetimi çeşitli endüstriyel sektörlerde kullanılan ve hızla gelişen bir teknolojidir. İnsan-makina etkileşiminin olduğu aktivitelerde proses otomasyonu için kullanılan bu teknolojiler özellikle IT uygulamalarında öne çıkmaktadır. İş akışı sistemlerinin temel özelliği bir iş kolundaki prosesleri tanımlamaya yarayan gereçler sunmaktır. Bir proses, kendi içinde mantıksal bir bütünlüğü olacak şekilde, bir ya da daha fazla aktivite içerir. Aktiviteler arasındaki ilişki iş akışlarıyla belirtilir. Bir aktiviteden diğerine koşullu ya da koşulsuz olarak geçmek mümkündür. Bir prosesi oluşturan aktiviteler sırayla (sequential) ya da paralel (parallel) olarak yürütülebilir. İş akışı yönetimi için geliştirilen pek çok ticari yazılım bulunmakla birlikte bunların pek çoğu teorik bir altyapıdan yoksundur.

Tanımlanan iş akışlarının yoruma açık olmasını engellemek ve bu iş akışları ile ilgili mantıksal çıkarımlar yapmak için teorik tabana dayalı bir çalışma gerekmektedir [28,35]. Bu amaçla çeşitli çalışmalar yapılmıştır [26,27,28,29,30]. Bu proje kapsamında yapılan çalışmada ise iş akışı tanımlamak için eylem dili C kullanılmaktadır. Bir iş akışı içinde ortak çalışan etmenlerin tanımlanabildiği, aktiviteler ve bunların sonuçlarınınn gösterildiği bir çalışma [30]'da mevcuttur.

Bu projede bir mantık dili olan C dilini kullanarak teorik bir temeli olan bir iş akışı yönetim geliştirmeyi hedefledik. Bu amaçla Mayıs 2001 tarihinde başlayan University of Texas-Austin

ile birlikte yürüttüğümüz proje çalışmalarımız kapsamında aşağıda üç ana başlıkta tanımlanan çalışmalar başarıyla tamamlandı.

Proje önerimizde de belirtildiği üzere bu projenin amacı eylem dillerinde (action languages) iş akışı (workflow) tanımlanmasını sağlayan ve bunu görsel bir araçla destekleyen bir sistem geliştirmektir. Bu sistemi oluşturulacak yazılımlar da şu şekilde tanımlanmıştır:

1. Eylem dillerinde yazılmış programların geçiş diyagramlarının çizme gereci(Visualization of transition diagrams)
2. İş Akışı tanımlama gereci (Workflow definition tool)
3. İş akışı tanımlarının C eylem diline çeviren çevirmen (Translator)
4. Plan bulma gereci (Plan generator)
5. İş akışının doğruluğunu kontrol etme gereci (Verifier)

Sözü edilen çalışmalar bir araştırma projesi olarak geliştirilmiş ve iş akışlarına teorik bir temel kazandırmayı hedeflemiştir. Böylelikle C dili kullanılarak tanımlanan iş akışlarını gerçekleştirecek mantıksal modelleri bulmak mümkün olmuştur. Bunun için C dilinde yazılan bir programın cevap kümelerini bulmaya yarayan CCALC (Causal Calculator) yazılımı kullanılmıştır. Verilen bir iş akışı için bulunan cevap setleri bu iş akışlarının gerçekleştiği modelleri göstermektedir. Projeye konu olan çalışma tamamen akademik bir çalışma olup ticarileştirilebilir değildir.

Aşağıda bu çalışmalardan ayrıntılı bir biçimde sözedilmektedir.

2. GELİŞME

1. Eylem dillerinde yazılmış programların geçiş diyagramlarının çizme gereci:

Eylem dilleri günümüz yapay zeka çalışmalarında önemli bir yer tutmaktadır. Eylem dillerinde program yazmak, sebep-sonuç teorilerini oluşturmak kadar bu dilleri öğrenmede ve öğretmede de karşılaşılan en temel sorunlardan biri geçiş diyagramlarının (transition diagrams) elle çizilmesi ya da zihinde canlandırılmaya çalışılmasıdır. Bu sorunu gidermek için geçiş diyagramlarını görsel bir şekilde çizen TDV (Transition Diagram Visualizer) adında bir yazılım bu proje kapsamında geliştirilmiştir.TDV, AT&T'nin GraphViz yazılımını da kullanarak C dilinde yazılmış programların geçiş diyagramılarını çizmektedir [31].

2. İş Akışı tanımlama gereci:

İş akışı, bilgilerin, işlemlerin, ve dökümanların önceden belirlenen kurallara göre bir sıra takip ederek tanımlanan bir işi yerine getirme sürecidir. İş akışı yönetim sistemleri genellikle prosedürel otomasyonu sağlamak üzere bilgisayar destekli olup, bir işi oluşturan aktivitelerle bu aktiviteler için gereken insan/bilgi teknolojisi kaynaklarını organize eder.

Bu proje kapsamında çeşitli ticari iş akışı yönetim sistemlerinin yaptığı işi C eylem dili ile yapabilen bir iş akışı tanımlama gereci geliştirilmiştir [34]. Bu gereç, ortaya çıkacak olan iş akışı için gereken bütün bilgilerin tanımlanmasına olanak sağlayacak şekilde tasarlanmıştır. Tanımlanan bir iş akışı daha sonra yine bu proje kapsamında geliştirilen ve aşağıda ayrıntılı olarak anlatılan WoDeL (Workflow Definition Language) diline çevrilir. Bir sonraki adım ise yine aşağıda tanımlanan çevirmen aracılığıyla WoDeL'de tanımlanmış bir iş akışını C eylem diline çevirmektir [32]. Daha sonra bu program CCALC ya da başka bir gereçle çalıştırılarak iş akışını gerçekleştirecek modeller elde edilir. WoDeL'in tanımı Ek-1'de yer almaktadır.

3. İş akışı tanımlarının C eylem diline çeviren çevirmen:

İş akışı yönetimi sistemleri kurumların otomasyon işlemleri için ümit vaad eden bir çözümdür. İş akışını modellemek ve çalıştırmak için pek çok ticari ürün mevcuttur. Ancak, iş akışı yönetim sistemlerinin kuramsal temelleri eksiktir. Diğer bir yandan, eylem dilleri, eylem sistemlerini tanımlama problemine en son yaklaşımlardan biridir. C dili, nedenselliğin açıklanması kuramını temel alan, bir yüksek düzey eylem tanımlama dilidir.

İş akışı süreçlerinin biçimselleştirilmesi için yapılan pek çok öneriden biri C eylem dilinin kullanılmasıdır. Bu proje kapsamında, iş akışı süreçlerinin eylem dili C kullanılarak biçimselleştirilmesi amacıyla bir yüksek düzey iş akışı tanımlama dili olan WoDeL tasarlanmıştır. WoDeL dilinden C diline bir çevirim tanımlanmış ve çevrim işinin otomatik olarak yapılabilmesini sağlamak için bir çevirim aracı geliştirilmiştir [32].

Bu biçimselleştirme, iş akışı süreç tanımları üzerinden akıl yürütme işlemlerinin yapılabilmesine olanak sağlayan bir zemin oluşturmaktadır. Ayrıca, önerilen biçimselleştirme geliştirilerek iş akışı süreçlerinin doğrulanması ve en iyilenmesi amaçları için kullanılabilir.

Bu çalışmalara ilişkin bir örnek Ek-2, 3, ve 4'te yer almaktadır.

4. Plan bulma gereci (Plan generator)

CCALC için geliştirilmiş çeşitli plan bulma gereçleri (Satisfiability solvers-RelSat / SATO) bulunmaktadır. Bu gereçler aracılığıyla, C eylem dilinde yazılmış bir program çalıştırılarak bu programı gerçekleyen modeller/planlar bulunur. Var olan plan bulma gereçlerine alternatif olarak SMODELS (Stable Model Semantics) ile model bulma üzerinde detaylı çalışmalar da yapılmıştır[33].

Plan oluşturma Yapay Zeka alanında önemli bir araştırma konusudur. Planlamada yeni bir yaklaşım problemlerin eylem dilleri(action languages) kullanılarak tanımlanmasıdır. Texas-Austin Üniversitesi tarafından geliştirilmiş olan CCALC, eylem dili C veya 'causal theory' ile tanımlanmış olan problemler için planlama yapan bir sistemdir. Girdi eylem dili C ile verildiği

takdirde CCALC bunu sebep-sonuç teorisine (causal theory) çevirir. Daha sonra "literal completion" adı verilen metod ile bu sebep-sonuç teorisine karşılık gelen niceleme mantığı tanımlamasını bulur ve sistemin modelini bulmak için relsat, sato veya chaff gibi plan bulma gereçlerini (satisfiability solver) kullanır.

Planlamada diğer bir yaklaşım ise mantık progamlama (logic programming) kullanılmasıdır. Mantık programlama dilleri bizim problemlerimizi daha rahat ve verimli tanımlayabilmemizi sağlarlar. Bu çalışmanın amacı eylem dili C ile tanımlanmış olan bir sistemi sebep-sonuç teorisine çevirip daha sonra da bu sebep-sonuç teorisine karşılık gelen mantık programını bularak planlama yapmaktır. Bu durumda planlama problemi mantık programının cevap kümesini (stable model) bulmaya indirgenmiş olacaktır. Bu sistem CCALC'a eklenecektir. Mantık programının cevap setini bulmak için Helsinki Üniversitesi tarafından geliştirilmiş olan Smodels sistemi kullanılmıştır. Bu bağlamda aşağıdaki çalışmalar yapılmıştır:

- CCALC'dan sebep-sonuç teorisi kurallarının alınması
- Bu sebep-sonuç teorisinin mantık programına çevrilmesi
- Smodels sistemi kullanılarak cevap kümesinin bulunması
- Cevap kümesinden faydalanılarak planların bulunması

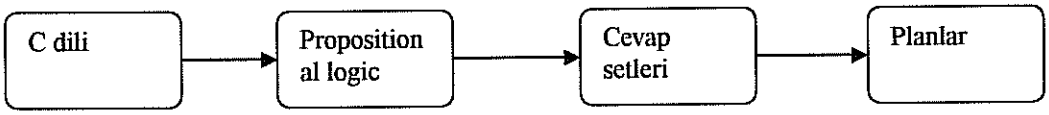
5. İş akışının doğruluğunu kontrol etme gereci (Verifier)

CCALC kullanılarak iş akışı problemlerinin çözülmesi sırasında yaşanılan en büyük zorluk bulunan planın (ya da elde edilen modelin) doğruluğunu kontrol etmektir. Çünkü doğal dilde yazdığımız cümleleri belli bir formalizm kullanarak ifade etmemize yarayan eylem dilleri gerçek hayatı tam olarak ifade etmemiz için yeterli değildir. Bu nedenle iş akışını C dili ile ifade edip elde ettiğimiz planın doğruluğunu formal olarak kanıtlamamız mümkün olmamıştır. Bu planın doğruluğunu araştırmak için ticari bir iş akışı yazılımı kullanarak elde edeceğimiz planla karşılaştırmak istediğimizde ise ticari yazılımların bir plan üretmek yerine, iş akışını her evresinde izleme (monitoring) amaçlı olduğu gözlemlenmiştir.

Yaşanan diğer bir sorun ise CCALC sisteminin problem alanındaki bütün kuralları değişkenlerin yerine olası bütün sabitleri koyarak yeniden yazmasından(grounding) kaynaklanmaktadır. Bu

problemin çok fazla büyümesine sebep olmaktadır. Daha sonra bu haliyle program SAT-solverlar tarafından çözümlenir ve cevap setleri bulunur. Ancak bilindiği gibi SAT NP-Complete bir işlemdir ve bu sebepten dolayı programın büyüklüğü bazı durumlarda çözüm bulunmasını zorlaştırmaktadır. Bu duruma bir çözüm PROLOG benzeri bir yapı kullanarak çözüm aramaktır. Bizim bu çalışmadaki amacımız *C* eylem dili kullanılarak yazılmış olan sistem üzerinde PROLOG benzeri bir metodla çözüm bulmaktır. Böylelikle, bulduğumuz planların doğruluğunu formal olarak kontrol etmenin de mümkün olup olmadığını araştırmış olacaktık. Ancak bu *C* eylem dilinin yapısı açısından bakıldığında çok zor bir çalışmadır. Amacımız *C* dilinin sadece bir altkümesi (workflowları ifade etmemize yeterli olacak kadarı) için böyle bir çalışma yapmaktır. Kısaca yapılacak işlem *C* dili ile tanımlanmış bir sistemi PROLOG benzeri bir sistemin çözebileceği bir sisteme dönüştürmektedir [33].

CCALC, *C* eylem dili kullanılarak tanımlanmış olan problem alanlarının cevap setlerini bulur. Bu işlemi yapmak için bütün kurallar için karşılık gelen sabit kurallar yazılır daha sonra da bu program CNF haline dönüştürülür. CNF halindeki programın cevap setleri SAT-solver'lar kullanılarak bulunur. Aşağıda bu çalışma mekanizmasının blok diyagramı verilmiştir.



CCALC'da problem büyük olduğu zaman karşılaştığımız zaman probleminin çözümü hem de bulunan planın formal olarak doğruluğunun araştırılması amacıyla *C* dilinin PROLOG benzeri bir yapı ile çözülebilecek bir programa dönüştürülmesi üzerinde çalışılmıştır. Yapılacak sistemin aşama aşama nasıl olması gerektiği incelenmiştir. Bu amaçla "WfMC TC-1016-X The representative business example" dökümanında tanımlanmış olan örnek iş akışı ve bunun PROLOG çözümü Ek-5'te verilmiştir. CCALC-PROLOG dönüşümü problemlerin çözümü üzerinde önemli bir katkı yapmıştır. Bu çeviriler bütün iş akışı yapıları için (AND-SPLIT, OR-

SPLIT ve döngü) doğrulanmıştır. Bu çevirim sayesinde PROLOG sisteminin hızı ile cevap-seti mantığının birleştirilmesi sağlanmıştır.

Proje önerimizde belirtilmeyen ek bir çalışmada ise, SMODELS için yeni heuristic tanımlama konusunda araştırmalar yapılmıştır [35].

SMODELS, bir mantık programının çözüm kümelerini bulan bir programdır. Çözüm kümelerini aramada kullandığı yöntem şöyledir. Mantık programına göre bir aday çözüm kümesi seçer. Bu aday kümede çözüm kümesinin aksine programın bütün atomlarının doğruluk değerleri belirli değildir. Değerleri bilinmeyenler, programın kurallarına (rule sentences) göre bir değer verilerek aday kümeye eklenir. Bu küme genişletme işlemi aday küme bir çözüm kümesi olana kadar devam eder. Fakat bu işlem sırasında bazen SMODELS mantık programının kurallarına göre yeni atom bulamadığında belirli olmayan atomlardan birini seçmek ve ona doğru yada yanlış değeri vermek zorunda kalabilir. Bu seçimin yapıldığı yerlere seçim noktaları (choice points) denir. Bu noktalarda seçilen bazı atomlar çözüm kümesinin hızlı bir şekilde bulunmasını sağlarken, bazıları çözüm kümesi olmayan kötü bir yola sapılmasını sağlar ve arama işlemi çok uzun sürebilir. Bu yüzden seçim noktalarında yapılan seçimler SMODELS'ın hızlı çalışması açısından çok önemlidir. Bu noktalarda genelde heuristic fonksiyonları kullanılır.

Bir mantık programının çözüm kümelerini ararken, programdaki atomların çözüm kümesinde olup olmadığını bulma zorluğu seçim noktalarında kullanmak için çok yararlı bir bilgi olabilir. Bu aşamada atomların kritik değerlerinin bulunup ve bu değerleri SMODELS'ın seçim noktalarında hangi atomu seçeceğine karar vermesi için başvuracağı bir heuristic fonksiyonu olarak kullanması SMODELS'ın daha verimli çalışmasını sağlayacaktır.

İtalya IRST enstitüsünde geliştirilen RESISTOR programı planlama problemleri için soyutlama hiyerarşilerini (abstraction hierarchies) otomatik oluşturur. RESISTOR bir terimin kritik değeri kavramını (criticality of a literal) kullanır. Bir planlama problemindeki herhangi bir atomun kritik değeri, o atomu oluşturacak bir plan bulmanın ne kadar zor olduğuna dair sayısal bir yaklaşımdır. Amaç planda bulunması zor olan atomları belirlemektir. Bu atomlar hiyerarşik planlayıcıya girdi

olarak verilen hiyerarşide en üst seviyeye yerleştirilirler. Bu yöntemle oluşturulan hiyerarşiler problemin hızlı bir şekilde çözümlenmesini sağlar.

Bu proje kapsamında RESISTOR sistemindeki kritik değerlerin hesaplanış yöntemini mantık programları için uygulanıp uygulanamayacağı araştırılmıştır. Kritik değerler planlama problemleri için kullanıldığından, bunmları mantık programlarını planlama problemleri haline getiren bir yöntem geliştirilmiştir. Bunun için programdaki her bir kural bir planlama operatörüne (planning action) dönüştürülür. Kuralın neden kısmı (body) operatörün koşulları (preconditions) olarak; sonuç kısmı (head) ise operatörün etkileri (effects) olarak kabul edilir. Böylece mantık programındaki bütün atomların kritik değerlerini hesaplamak mümkün olur. Fakat araştırmalarımız sırasında RESISTOR'daki yöntemin uygun olmadığı noktaların olduğunu gözlemlemiş bulunmaktayız.. Atomların değillemelerinin (negation-as-failure kullanarak) kritik değerlerinin hesaplanması ve SMODELS'a has seçim (choice rules) ve sınırlayıcı (constraint/killing rules) kurallarının ayrıca ele alınması gerekir.

Bu konudaki araştırmalarımızın sonucunda kritik değerlerini heuristic olarak kullanan yeni bir SMODELS versiyonu olan CSMODELS (Criticality SMODELS) adında bir prototip geliştirilmiştir. Program, kritik değerlerinin hesaplanmasını ve bulunan değerlerin sonar seçim noktalarında heuristic olarak kullanılmasını sağlayan kodlar SMODELS'ın kaynak kodu içine gömülerek oluşturulmuştur.

CSMODELS çeşitli problemlerde denenmiş ve performansı SMODELS'la karşılaştırılmıştır. Bu problemler planlama alanından seçilmiş bilinen problemlerdir. Bu problemler Towers of Hanoi, Robot-box alanı ve Blocks-world alanıdır. Problemlerden özellikle Blocks-world alanında küçükten büyüğe farklı boyuttaki problem örnekleri çözülmüştür. Böylece performans karşılaştırması daha sağlıklı olmuştur.

Temelde sonuçlar geliştirdiğimiz bu yeni heuristiğin beklendiği gibi yanıt kümesi bulan sistemlerin performansını arttırdığını gösterir. CSMODELS'ta seçim noktalarında kullanılan heuristic SMODELS'in seçtiği literallara gore daha iyi literallar (eldeki problemin ana kısmıyla ilgili olup çözüme daha çabuk götüren) seçmiştir; bu da CSMODELS'ın performansının SMODELS'tan daha iyi olmasını sağlamıştır. Aşağıdaki tablo performans artışının küçük

problemlerde kıyaslanabilir ya da iki sistemin performansının hemen hemen aynı olduğunu gösterir. Fakat problem büyüyünce performans artışı belirgin olarak görülebilmektedir. Aşağıdaki tabloda yer alan sayılar sistemlerin problemin yanıt kümesini kaç CPU saniyesinde bulduğunu gösterir.

	SMODELS	CSMODELS
Hanoi (4 disk)	4.630	5.540
Robot-box	39.940	40.800
BlocksWorld 1. representation (11 blok)	46.350	36.690
BlocksWorld 2. representation (11 blok)	73.400	57.040
BlocksWorld 3. representation (15 blok)	38.500	37.390
BlocksWorld 3. representation (17 blok)	77.330	61.550
BlocksWorld 3. representation (19 blok)	174.660	90.620

Ayrıca CSMODELS "colorability" ve "n-queens" problemleri ile de denenmiştir. Böylece planlama problemleri dışındaki problemlerle için de CSMODELS'ın performansını test etmek mümkün olmuştur. Sonuçlar daha önceki yaptığımız deneylerle uyumludur. Hatta bu problemlerde performans artışı önceki sonuçlara gore çok daha belirgindir. Hemen hemen tüm problemlerde CSMODELS SMODELS'dan daha kısa zamanda çözüme ulaşmıştır. Aşağıda yapılan denemelerde sistemlerin problemlerin yanıt kümesini kaç CPU saniyede bulduğunu gösteren tablolar vardır.

	p100	*p300*	*p600*	*p1000*	*p3000*
SMODELS	0.530	3.530	12.380	35.210	305.060
CSMODELS	0.540	3.050	10.080	26.070	230.370

Colorability problemi sonuçları (kolonlar her örnek problemdeki node sayısını gösterir, birim CPU saniyesidir)

	8x8	*18x18*	*20x20*	*22x22*
SMODELS	0.020	1.830	9.360	117.730
CSMODELS	0.030	1.530	5.490	31.170

N-queens problemi sonuçları (kolonlar problemlerdeki tahta büyüklüklerini gösterir, birim CPU saniyedir).

CSMODELS sistemiyle ilgili tüm bilgileri ve kaynak kodları içeren bir web sitesinin adresi http://www.ceng.metu.edu.tr/~orkunt/csmodels dir.

3. SONUÇ

İş akışı, bilgilerin, işlemlerin, ve dökümanların önceden belirlenen kurallara göre bir sıra takip ederek tanımlanan bir işi yerine getirme sürecidir. Bu proje kapsamında C eylem dili tabanlı bir iş akışı tanımlama gereci geliştirilmiştir. Bu gereç, ortaya çıkacak olan iş akışı için gereken bütün bilgilerin görsel bir şekilde tanımlanmasına olanak sağlayacak şekilde tasarlanmıştır. Görsel olarak tanımlanan bir iş akışı aynı anda WoDeL (Workflow Definition Language) diline çevrilir. WoDeL bu proje kapsamında geliştirilen bir yüksek düzey iş akışı tanımlama dilidir. Yine projede geliştirilen çevirmen aracılığıyla WoDeL'de tanımlanmış bir iş akışı C eylem diline çevirilmektedir. Daha sonra bu program CCALC ya da başka bir model bulma programı çalıştırılarak iş akışını gerçekleştirecek modeller elde edilir.

CCALC kullanılarak iş akışı problemlerinin çözülmesi sırasında yaşanılan en büyük zorluk bulunan planın (ya da elde edilen modelin) doğruluğunu kontrol etmektir. Çünkü doğal dilde yazdığımız cümleleri belli bir formalizm kullanarak ifade etmemize yarayan eylem dilleri gerçek hayatı tam olarak ifade etmemiz için yeterli değildir. Bu nedenle iş akışını C dili ile ifade edip elde ettiğimiz planın doğruluğunu formal olarak kanıtlamamız mümkün olmamıştır. Bu planın doğruluğunu araştırmak için ticari bir iş akışı yazılımı kullanarak elde edeceğimiz planla karşılaştırmak istediğimizde ise ticari yazılımların bir plan üretmek yerine, iş akışını her evresinde izleme (monitoring) amaçlı olduğu gözlemlenmiştir. Ayrıca, hem bulduğumuz planların doğruluğunu formal olarak kontrol etmek hem de CCALC sisteminin problem alanındaki bütün kuralları değişkenlerin yerine olası bütün sabitleri koyarak yeniden yazması (grounding) problemini ortadan kaldırmak için C eylem dili kullanılarak yazılmış olan sistem üzerinde PROLOG benzeri bir metodla çözüm bulmak üzerine çalışmalar yapılmıştır.

CCALC-PROLOG dönüşümü problemlerin çözümü üzerinde önemli bir katkı yapmıştır. Bu çeviriler bütün iş akışı yapıları için (AND-SPLIT, OR-SPLIT, sıralı işlem, ve döngü) doğrulanmıştır. Bu çevirim sayesinde PROLOG sisteminin hızı ile cevap-seti mantığının birleştirilmesi sağlanmıştır.

Bir mantık programının çözüm kümelerini bulan bir başka program da SMODELSdır. Proje önerimizde belirtilmeyen ek bir çalışmada, SMODELS için yeni heuristic tanımlama konusunda araştırmalar yapılmıştır. Bu konudaki araştırmalarımızın sonucunda kritik değerlerini heuristic olarak kullanan yeni bir SMODELS versiyonu olan CSMODELS (Criticality SMODELS) adında bir prototip geliştirilmiştir. Program, kritik değerlerinin hesaplanmasını ve bulunan değerlerin sonar seçim noktalarında heuristic olarak kullanılmasını sağlayan kodlar SMODELS'ın kaynak kodu içine gömülerek oluşturulmuştur. Yapılan deneyler sonucunda CSMODELS'in SMODELS'ten daha iyi sonuçlar verdiği gözlenmiştir.

Kaynakça

[1] C. Baral, M. Gelfond and A. Provetti, Reasoning about actions: laws, observations and hypotheses, Journal of Logic Programming, 31, pp. 201-244, 1997.

[2] M. Gelfond and V. Lifschitz. Action languages. Electronic Transactions of AI, 3(16), 1998.

[3] N. McCain and H. Turner, Causal theories of action and change. In Proc. AAAI-97, pp. 460-465, 1997.

[4] E. Giunchiglia and V. Lifschitz, An action language based on causal explanation: preliminary report. In Proc. AAAI-98, pp. 623-630, 1998.

[5] V. Lifschitz and H. Turner, Representing transition systems by logic programs. Proc. Fifth International Conference on Logic Programming and Nonmonotonic Reasoning, 1999.

[6] N. McCain and H. Turner, Satisfiability planning with causal theories. Proc. Sixth International Conference on Knowledge Representation and Reasoning, pp. 212-223, 1998.

[7] I. Niemela and P. Simons. Efficient implementation of the well-founded and stable model semantics. Proc. Joint International Conference and Symposium on Logic Programming, pp. 289-303, 1996.

[8] T.Eiter, N.Leone, C.Mateis, G. Pfeifer, and F. Scarello. The KR system DLV: Progress report, comparisons, and benchmarks. Proceedings of the Sixth International Conference on Knowledge Representation and Reasoning, pp. 406-417, 1998.

[9] V. Lifschitz. Action languages, answer sets and planning. In The Logic Programming Paradigm: a 25-year Perspective. Springer Verlag, 1999.

[10] C. Baral and M. Gelfond, Logic programming and knowledge representation. Journal of Logic Programming , 19(20), pp. 74-148, 1994.

[11] J. McCarthy. Circumscription-a form of non-monotonic reasoning. Artificial Intelligence, 13(1, 2), pp. 27-39,171-172, 1980.

[12] J. McCarthy. Applications of circumscription to formalizing common sense knowledge. Artificial Intelligence, 26(3), pp. 89-116, 1986.

[13] V. Lifschitz. Computing circumscription. In Proc. IJCAI-85, pp. 121-127, 1985.

[14] R. Reiter. A logic for default reasoning. Artificial Intelligence, 13(1,2), pp. 81-132, 1980.

[15] D. McDermott and J. Doyle. Nonmonotonic logic I. Artificial Intelligence, 13(1,2), pp. 41-72, 1980.

[16] D. McDermott. Nonmonotonic logic II: Nonmonotonic modal theories. Journal of the ACM, 29(1), pp. 33-57, 1982.

[17] R. Moore. Semantical considerations on nonmonotonic logic. Artificial Intelligence, 25(1), pp. 75-94, 1985.

[18] M. Ginsberg, editor. Readings in Nonmonotonic Reasoning. Morgan Kaufmann, CA, 1987.

[19] R. Reiter. A theory of diagnosis from first principles. Artificial Intelligence, 32(1), pp. 57-95, 1987.

[20] C. Green. Theorem proving by resolution as a basis for question - answering systems. In B. Meltzer and D. Michie, editors, Machine Intelligence 4, pp. 183-205. Edinburgh University Press, New York, 1969.

[21] P. Hayes. Computation and deduction. In Proc. Second Symposium on Mathematical Foundations of Computer Science, pp.105-118. Czechoslovakian Academy of Sciences, 1973.

[22] R. Kowalski. Predicate logic as a programming language. Information Processing 74, pp. 569-574, 1974.

[23] J. Lloyd. Foundations of Logic Programming. Springer-Verlag, second edition, 1987.

[24] A. Colmerauer, H. Kanoui, R. Pasero, and P. Roussel. Un systeme de communication homme-machine en Francais. Technical report, Groupe de Intelligence Artificielle Universitae de Aix-Marseille II, Marseille, 1973.

[25] P. Köksal, N. Çicekli, and H. Toroslu. Workflow Specification Using the Action Description Language C, Technical report, METU, 2000.

[26] M. Singh, G. Meredith, C. Tomlinson, and P. Attie. An event algebra for specifying and scheduling workflows, Proc. 4th Intl. Conf. On Database Systems for Advances Applications (DASFAA), Singapore, April 1995.

[27] D. Harel. StateCharts: A visual formalism for complex systems. Science of Computer Programming, 8, pp.231-274, 1987.

[28] H. Davulcu, M. Kifer, C.R. Ramakrishnan, and I.V. Ramakrishnan. Logic based modeling and analysis of workflows. In *ACM Symposium on Principles of Database Systems*; Seattle, Washington, ACM Press, 1998.

[29] N. Kesim-Çiçekli. A temporal reasoning approach to model workflow activities. Lecture Notes in Computer Science, no. 1649, eds. R.Y. Pinter and S. Tsur, NGITS'99, Zikhron-Yaakov, Israel, 1999.

[30] C. Baral and J. Lobo. Formalizing Workflows as Cooperative Agents. Dynamics '97, Workshop in ILPS, 1997.

[31] Özcan Koç (M.Sc.), Visualizing transition diagrams of action language programs, July 2001.

[32] Emel Hasdal (M.Sc.), Translation of workflow specifications to the action description language C, July 2001.

[33] Semra Doğandağ (M.Sc.), Using Stable Model Semantics (SMODELS) in the Causal calculator (CCALC), January 2002.

[34] Handan (Seyrek) İnalpolat (M.Sc.), Workflow Process definition tool, January 2002.

[35] Orkunt Sabuncu (M.Sc.), Using criticalities as a heuristic for answer set programming, September 2002.

EK 1

WoDeL Dil Yapısı

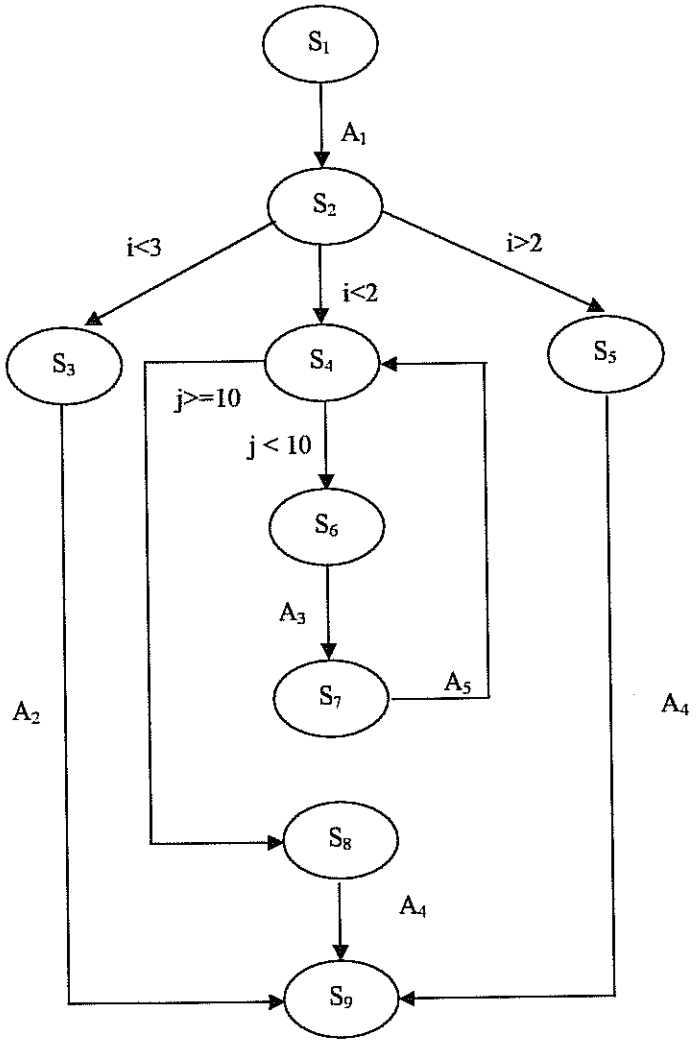
```
<wf_definition>      ::=  WORKFLOW <wf_name> {
                            <declaration_part>
                            <domaindefinition_part>
                            <activitydefinition_part>
                            }
<declaration_part>   ::=  DECL {
                            <agent_declaration>
                            <activity_declaration>
                            <qualification_declaration>
                            <variable_declaration>
                            }
<agent_declaration>  ::=  AGENT <agent_name> (, <agent_name>)*;
<activity_declaration>    ::=  ACTIVITY <act_name> (, <act_name>)*  ;
<qualification_declaration>    ::=  QUALIFIED
                            ( <agent_name>, [ <act_name> (, <act_name>)*] )
                            (,( <agent_name>, [ <act_name> (, <act_name>)*] ))* ;
<variable_declaration>    ::=  <var_type> <var_name>[ = <integer>]
                            ( , <var_name> [ = <integer>])*;
<domaindefinition_part>   ::=  DOMAIN_DEF{
                                  <seq_block>
                                  }
<block>              ::=   <seq_block>
                           | <and_block>
                           | <xor_block>
                           | <while_block>
<seq_block>               ::=  SEQ_BLOCK{
                           (<act_name> ; | <block>)+
                           }
<and_block>               ::=  AND_BLOCK{
                            (<act_name> ; | <block>)+
                           }
<path>               ::=   IF <condition>

                           (<act_name>;|<block>)
                           ENDIF
<xor_block>               ::=  XOR_BLOCK{
                           <path>
                           <path>+
                           }
<while_block>             ::=  WHILE<condition>{
                           <seq_block>
                           }
<activity_definition>     ::=  ACTIVITY<act_name>{
                           COMPLETE        (<statement>;)+
                           ABORT           (<statement>;)+
                           DURATION        (<duration_declaration>;)+
                           }
<statement>          ::=   <assignment>
<duration_declaration>    ::=   MAX_EXEC_TIME          (AGENT <agent_name>)
                           = < p_integer >;
<assignment>         ::=   <var_name> = <expression>    ;
<expression>         ::=   <term> + <term>
                           | <term> - <term>
<term>               ::=   <var_name>
                          | < p_integer >
<p_integer>          ::=   (0..9)+
<condition>          ::=   (<term><operator> <term>)
<operator>           ::=   < | > | = | <= | >=
<agent_name>         ::=   <name>
<act_name>           ::=   <name>
<var_name>           ::=   <name>
<var_type>           ::=   INT
<wf_name>            ::=   <name>
<name>               ::=   [a-zA-Z]+| [a-zA-Z] [a-zA-Z0-9]+
```

EK 2

Örnek İş Akışı (C Çevirimi)

Aşağıda tanımı verilen MyWorkflow isimli iş akışında agentX, agentY, agentZ olarak adlandırılan üç ajan bulunmaktadır. Her bir ajan A1, A2, A3 ve A4 aktivitelerini yapabilecek niteliktedir. A5 aktivitesi ise döngü değişkenini arttırmakla yükümlü olup herhangi bir ajanla ilişkilendirilmemiştir. S_1 - S_9 ajanların içinde bulundukları durumları göstermektedir. i ve j ise iş akışı prosesinde kullanılan iki ayrı değişkendir.



Bu iş akışının proje kapsamında geliştirilen WoDeL dilinde tanımı ise aaşağıdaki şekilde yapılmaktadır.

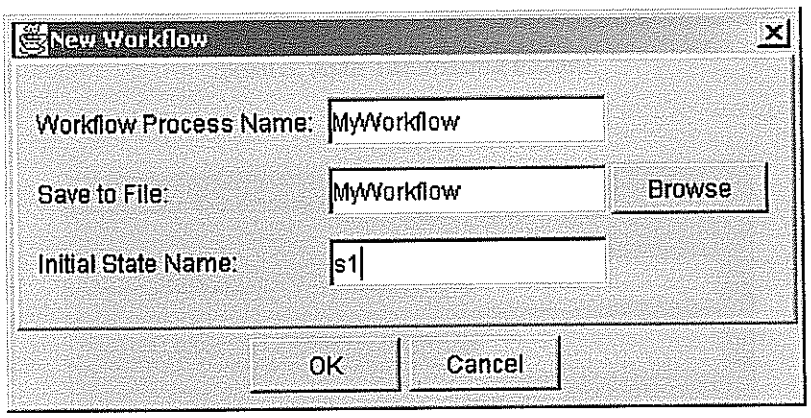
```
WORKFLOW MyWorkflow
{
        DECL {
                AGENT agentX, agentY, agentZ;
                ACTIVITY A1, A2, A3, A4, A5;
                QUALIFIED (agentX,[A1,A2,A3]), (agentY,[A1,A3,A4]),
                (agentZ,[A2,A4]);
                INT i=0, j=0;
        }

        DOMAIN_DEF {
                SEQ_BLOCK {
                        A1;
                        AND_BLOCK {
                                IF (i<3) THEN
                                        A2;
                                ENDIF
                                IF (i<2) THEN
                                        SEQ_BLOCK {
                                                WHILE (j<10) {
                                                        SEQ_BLOCK {
                                                                A3;
                                                                A5;
                                                        }
                                                }
                                                A4;
                                        }
                                ENDIF
                                IF (i>2) THEN
                                        A4;
                                ENDIF
                        }
                }
        }
```

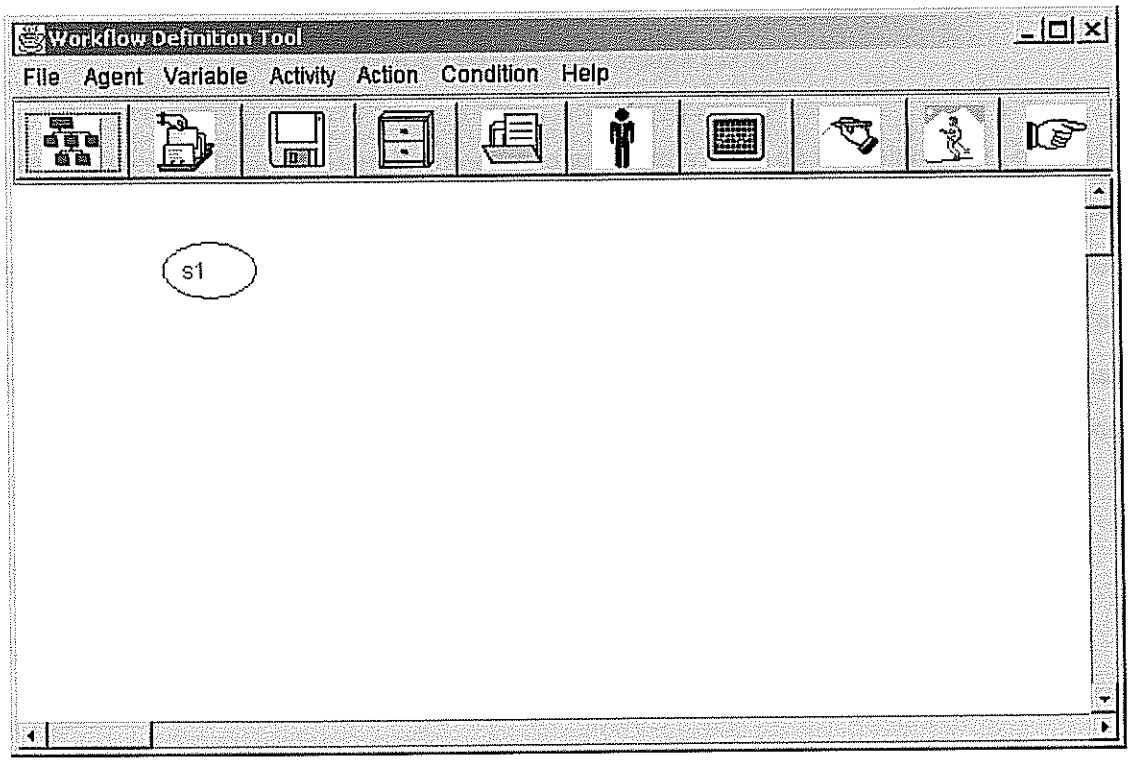
```
        ACTIVITY A1 {
                COMPLETE:   i = i + 3;
                ABORT:          i = i +1;
                DURATION:
                        MAX_EXEC_TIME( AGENT agentX) =100;
                        MAX_EXEC_TIME( AGENT agentY) =200;
        }
        ACTIVITY A2 {
                COMPLETE:   i = i + 5;
                DURATION:
                        MAX_EXEC_TIME( AGENT agentX) =100;
                        MAX_EXEC_TIME( AGENT agentZ) =50;
        }
        ACTIVITY A3 {
                COMPLETE:   j= j + 3;
                DURATION:
                        MAX_EXEC_TIME( AGENT agentX) =30;
                        MAX_EXEC_TIME( AGENT agentZ) =50;
        }
        ACTIVITY A4 {
                COMPLETE:   i = 10;
                DURATION:
                        MAX_EXEC_TIME( AGENT agentY) =100;
                        MAX_EXEC_TIME( AGENT agentZ) =20;
        }
        ACTIVITY A5 {
                COMPLETE:   i = i + 1;
        }
} //end of workflow process definition
```

İş akışı tanımlama gerecini kullanarak iş akışı tanımlama süreci aşağıdaki gibidir:

Tanımlama işlemi, aşağıdaki ekranda yeni bir iş akışı prosesi yaratmayla başlar (Şekil 1). Böylelikle iş akışı ekranda çizilmeye başlar (Şekil 2).

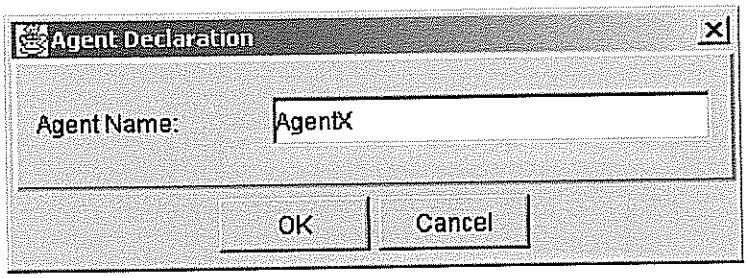


Şekil 1. Yeni İş Akışı Süreci Tanımlama Ekranı

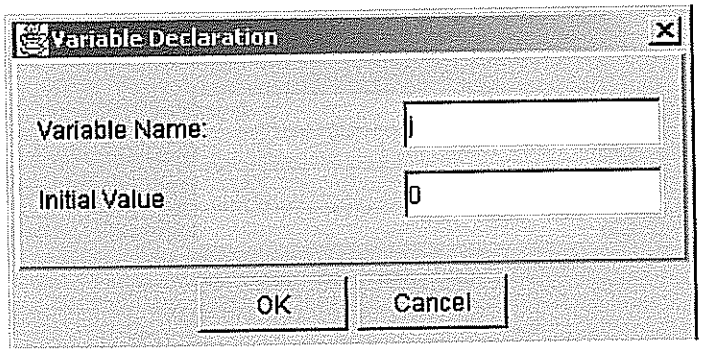


Şekil 2. İş Akışı Ekranı

Bundan sonra sıra ajanları ve değişkenleri tanımlamaya gelir. Bunlar da sırasıyla Şekil 3 ve Şekil 4'teki ekranlar aracılığıyla yapılır.

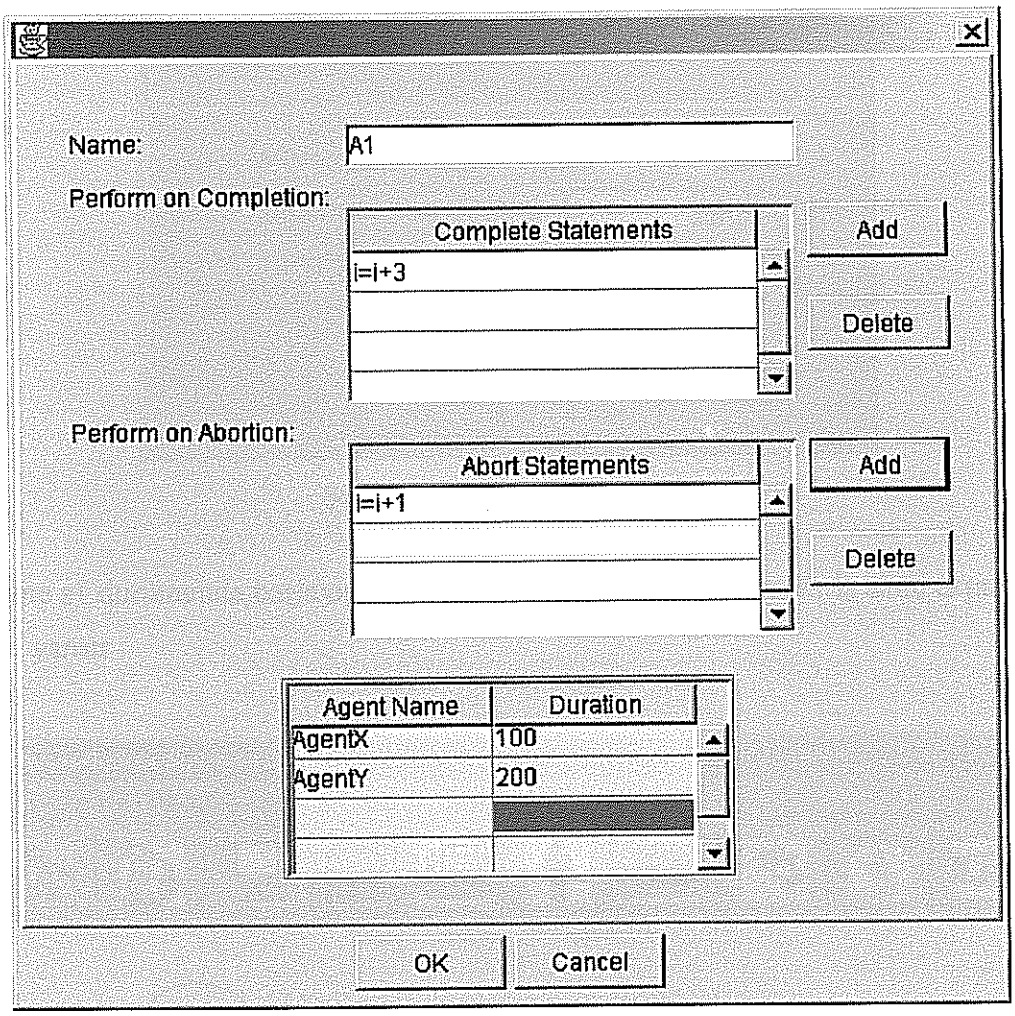


Şekil 3. Ajan Tanımlama Ekranı

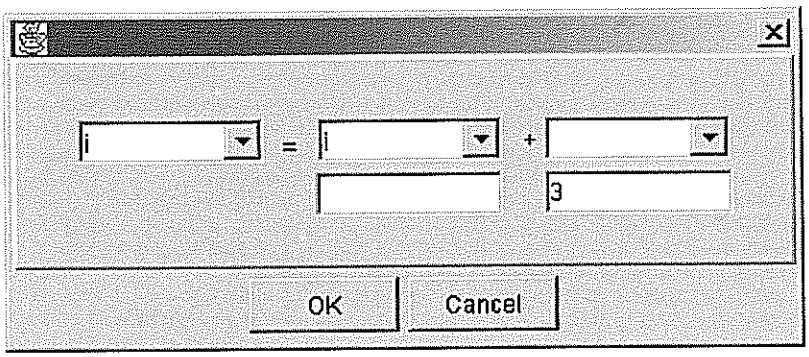


Şekil 4. Değişken Tanımlama Ekranı

Ajan ve değişken tanımları yapıldıktan sonra artık aktiviteler de tanımlanabilir. Bu tanımlama sürecinin herhangi bir aşamasında geri dönülüp daha önce tanımlanan elemanlar üzerinde değişiklik yapmak mümkündür. Şekil 5'te A1 aktivitesinin tanımlandığı ekran görülmektedir. *Complete* ve *Abort* cümleleri WoDeL dilindeki yapıya göre tanımlanmıştır. Bu ekranda hangi ajanların hangi aktiviteleri yapmakla yükümlü oldukları bilgisinin yanı sıra her ajan için öngörülen en fazla işlem zamanı da verilmektedir. Şekil 6'da döngü cümlesinin tanımlandığı ekran yer almaktadır. A2, A3, A4 ve A5 aktiviteleri de benzer şekilde tanımlanmıştır..

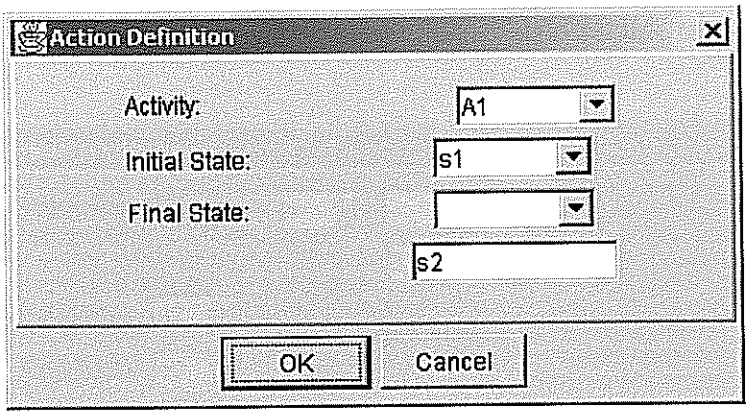


Şekil 5. Aktivite Tanım Ekranı



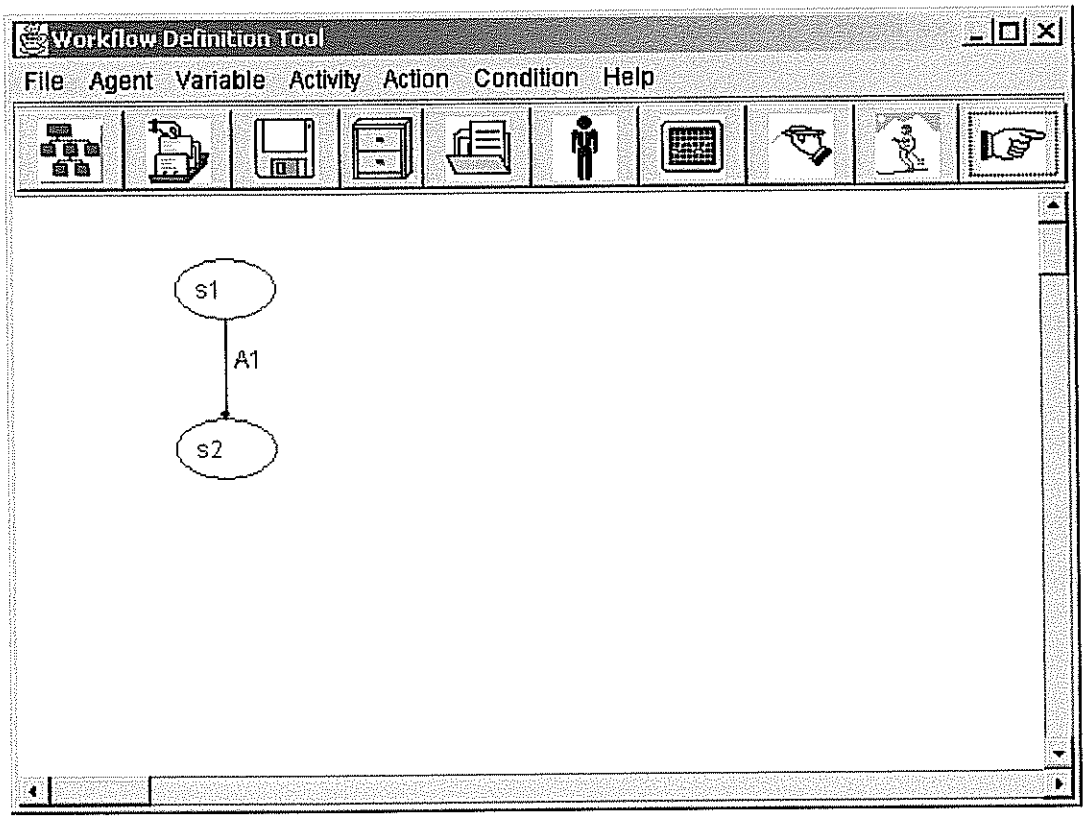
Şekil 6. İşlem Cümlesi Tanım Ekranı

Tanım aşaması bu şekilde tamamlandıktan sonra iş akışı sıra iş akışı bilgilerini girmeye gelir (Şekil 7).



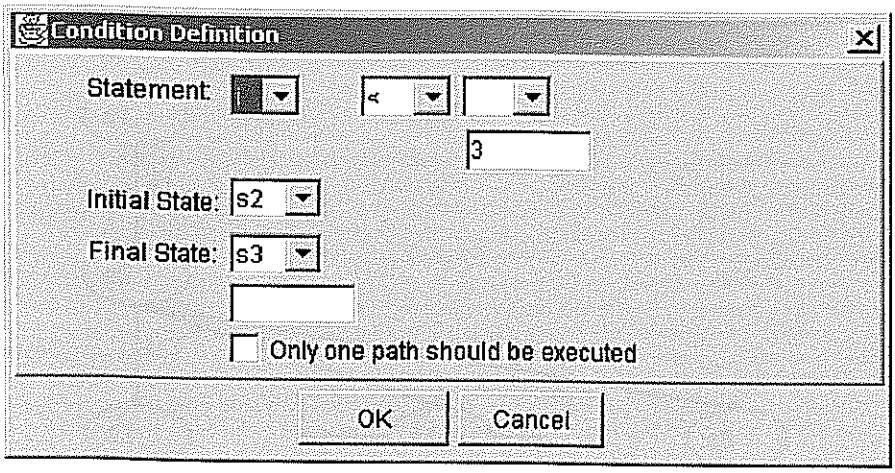
Şekil 7. İş Akışı Tanımlama ekranı

İlk aktivite s1 durumunda başlayıp s2 durumunda sona ermektedir. Bu İş akışı ekranında Şekil 8'deki gibi yansır



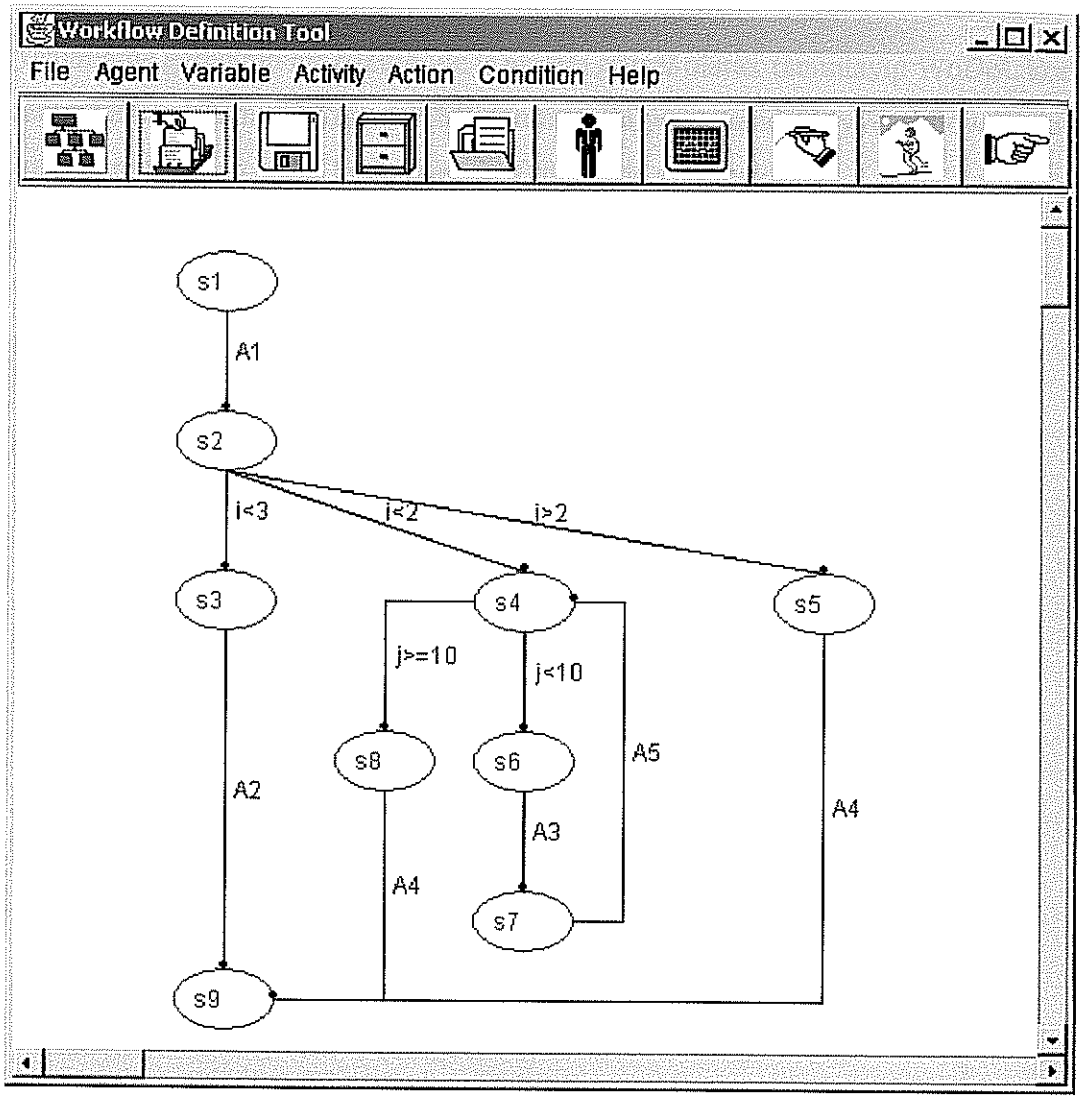
Şekil 8. İş Akışı Ekranı

A1 aktivitesinden sonra, i değişkeninin alacağı değere göre, iş akışı prosesi üç ayrı yoldan birisinde ilerler. Durum değişkenli döngülerin tümü Şekil 9'daki şekilde tanımlanır.



Şekil 9. Durum Değişkenli Döngü Tanımlama Ekranı

WHILE-döngüleri de yine benzer şekilde tanımlanır. MyWorkflow iş akışının son hali Şekil 10'da görülmektedir.



Şekil 10. İş Akışı Ekranı

Bu iş akışını çalıştırmak için önce iş akışının WoDel dilinde yaratılan tanımı Çevirmen programı aracılığıyla C eylem dilinde çevrilir. C dilindeki bu program ise daha sonra CCALC model bulucuya verilerek bu iş akışını gerçekleştirecek modeller elde edilir.

İş akışı tanımlama gerecinin bir özelliği de iş akışını görsel olarak tanımlayıp WoDeL kodu yaratmasının yanı sıra aynı iş akışının XML tanımını da yaratmasıdır.MyWorkflow iş akışının bu gereç tarafından yaratılan XML kodu aşağıda Tablo 1'de verilmektedir.

Tablo 1. MyWorkFlow'un XML Tanımı

```
<?xml version="1.0" ?>
<workflow name="MyWorkflow">

  <declaration_part>

    <agent_dec>
        <agent>AgentX</agent>
        <agent>AgentY</agent>
        <agent>AgentZ</agent>
    </agent_dec>

    <activity_dec>
        <act_name>A1</act_name>
        <act_name>A2</act_name>
        <act_name>A4</act_name>
        <act_name>A5</act_name>
        <act_name>A3</act_name>
    </activity_dec>

    <qualification_dec>
        <agent_qualified name="AgentX">
            <qual_activity>A1</qual_activity>
            <qual_activity>A2</qual_activity>
            <qual_activity>A3</qual_activity>
        </agent_qualified>
        <agent_qualified name="AgentY">
            <qual_activity>A1</qual_activity>
            <qual_activity>A4</qual_activity>
        </agent_qualified>
        <agent_qualified name="AgentZ">
            <qual_activity>A2</qual_activity>
            <qual_activity>A4</qual_activity>
            <qual_activity>A3</qual_activity>
        </agent_qualified>

    </qualification_dec>

    <variable_dec>
        <variable type="INT" value="0">i</variable>
        <variable type="INT" value="0">j</variable>
    </variable_dec>

  </declaration_part>
```

```
<domain_def_part>

    <state name="s1">
        <act dest="s2">A1</act>
    </state>
    <state name="s2">
        <parent>s1</parent>
        <cond dest="s3">i<3</cond>
        <cond dest="s4">i<2</cond>
        <cond dest="s5">i>2</cond>
    </state>
    <state name="s3">
        <parent>s2</parent>
        <act dest="s9">A2</act>
    </state>
    <state name="s4" loop="yes">
        <parent>s2</parent>
        <cond dest="s6">j<10</cond>
        <cond dest="s8" alternate="yes">j>=10</cond>
    </state>
    <state name="s5">
        <parent>s2</parent>
        <act dest="s9">A4</act>
    </state>
    <state name="s6">
        <parent>s4</parent>
        <act dest="s7">A3</act>
    </state>
    <state name="s7">
        <parent>s6</parent>
        <act dest="s4">A5</act>
    </state>
    <state name="s8">
        <parent>s4</parent>
        <act dest="s9">A4</act>
    </state>
    <state name="s9">
        <parent>s8</parent>
        <parent>s3</parent>
        <parent>s5</parent>
    </state>

</domain_def_part>
```

```
<activity_def_part>

    <activity name="A1">
       <complete_stmt>i=i+3</complete_stmt>
       <abort_stmt>i=i+1</abort_stmt>
       <duration>
         <exec_time agent="AgentX" time="100" />
         <exec_time agent="AgentY" time="200" />
       </duration>
    </activity>

    <activity name="A2">
       <complete_stmt>i=i+5</complete_stmt>
       <duration>
         <exec_time agent="AgentX" time="100" />
         <exec_time agent="AgentZ" time="50" />
       </duration>
    </activity>

    <activity name="A3">
      <complete_stmt>j=j+3</complete_stmt>
      <duration>
         <exec_time agent="AgentX" time="30" />
         <exec_time agent="AgentZ" time="50" />
      </duration>
    </activity>

    <activity name="A4">
       <complete_stmt>i=10</complete_stmt>
       <duration>
         <exec_time agent="AgentY" time="100" />
         <exec_time agent="AgentZ" time="20" />
       </duration>
    </activity>

    <activity name="A5">
       <complete_stmt>i=i+1</complete_stmt>
    </activity>

  </activity_def_part>

</workflow>
```

Çevirmen tarafından C diline çevrilmiş iş akışı programı Ek 3'de, CCALC tarafından çalıştırıldığında bulunan model ise Ek 4'te yer almaktadır.

EK 3

İş Akışının C dilindeki çevrimi

```
:- include 'C.t'.

:- sorts
    activity;agent;number1;number2.

:- variables
   ag1,ag2                    :: agent;
   activity1,activity2              :: activity;
   _I1,_I2,_I3,_I4,_I5,_I6          :: number1;
   _J1,_J2                    :: number2.

:- macros
    maxInt2 -> 7;
    maxInt -> 3;
    sum(#1,#2,#3) -> #1 is min((#2)+(#3),maxInt2).

:- constants
A1,A2,A3,A4,A5,A6,A7            :: activity;
Agent1,Agent2,Agent3                  :: agent;
0..maxInt                       :: number1;
0..maxInt2                 :: number2;
qualified(agent,activity)             :: inertialFluent;
duration(agent,activity,number1)            :: inertialFluent;
executing(agent,activity,number1)           :: inertialFluent;
committed(activity)                   :: inertialFluent;
count_i(number2)                :: inertialFluent;
count_k(number2)                :: inertialFluent.

% GENERAL WORKFLOW DOMAIN  PROPOSITIONS
% An activity commits after executing its duration of times.
      caused  committed(activity1)  if duration(ag1,activity1,_I3) &&
      _I2 is _I1+1 && (_I2 > _I3)   after executing(ag1,activity1,_I1).

% An activity continues to execute if it  executed  n times and n is less than
its duration.
      caused executing(ag1,activity1,_I2) if  duration(ag1,activity1,_I3) &&
      _I2 is _I1 + 1 && (_I3>=_I2) after executing(ag1,activity1,_I1).

% At any step, an activity cannot be both executing and committed.
      never executing(ag1,activity1,_I1) && committed(activity1).


% At any step, an agent cannot execute two different activities at the same
time.
      never executing(ag1,activity1,_I1) && executing(ag1,activity2,_I2) && -
(activity1=activity2).
% At any time, if an activity starts to execute, it is executing at only one
time stamp i within its duration.
      caused  -executing(ag1,activity1,_I1) if executing(ag1,activity1,_I2) &&
-(_I1=_I2).
```

```
% If an activity is committed it is caused to be not executing.
      caused  -executing(ag1,activity1,_I1) if committed(activity1).


% PROPOSITIONS SPECIAL TO THAT WORKFLOW PROBLEM
% Activity A1 starts to be executed by agent ag1 at time stamp 1 within its
duration,
% iff agent ag1 is qualified to execute A1, A1 is not committed and A1 is not
already executing.
      caused executing(ag1,A1,1) after qualified(ag1,A1) && (/\_I1: -
executing(ag1,A1,_I1)) && -committed(A1).


% Activity A2 starts to be executed by agent ag1 at time stamp 1 within its
duration,
% iff agent ag1 is qualified to execute A2, A2 is not committed, A2 is not
already executing, A1 is committed
% and condition of the loop A2 executes in evaluates to true.
      caused executing(ag1,A2,1)  after  qualified(ag1,A2) && committed(A1)
&& count_k(_J2) && (_J2<2) && (/\_I1: -executing(ag1,A2,_I1)) && -
committed(A2).
% Activity A3 starts to be executed by agent ag1 at time stamp 1 within its
duration,
% iff agent ag1 is qualified to execute A3, A3 is not committed, A3 is not
already executing and A2 is committed.
      caused executing(ag1,A3,1)  after  qualified(ag1,A3)
&& committed(A2) && (/\_I1: -executing(ag1,A3,_I1)) && -committed(A3).

% Activity A4 starts to be executed by agent ag1 at time stamp 1 within its
duration,
% iff agent ag1 is qualified to execute A4, A4 is not committed, A4 is not
already executing, A1 is committed % and the condition i<5 holds.
      caused executing(ag1,A4,1)  after  qualified(ag1,A4)       &&
committed(A1) && (/\_I1: -executing(ag1,A4,_I1)) && -committed(A4)
&& count_i(_J1) && _J1<5.
% Activity A5 starts to be executed by agent ag1 at time stamp 1 within its
duration,
% iff agent ag1 is qualified to execute A5, A5 is not committed, A5 is not
already executing, A1 is committed and the condition i >= 5 holds.
      caused executing(ag1,A5,1) after   qualified(ag1,A5)  && committed(A1)
&& (/\_I1: -executing(ag1,A5,_I1)) && -committed(A5) && count_i(_J1) &&
(i>=5).

% Activity A6 starts to be executed by agent ag1 at time stamp 1 within its
duration,
% iff agent ag1 is qualified to execute A6, A6 is not committed, A6 is not
already executing and A5 is committed.
      caused executing(ag1,A6,1) after qualified(ag1,A6) && committed(A5) &&
(/\_I1: -executing(ag1,A6,_I1)) && -committed(A6).

% Activity A7 starts to be executed by agent ag1 at time stamp 1 within its
duration,
% iff agent ag1 is qualified to execute A7, A7 is not committed, A7 is not
already executing,
% A5 is committed, the loop before A7 is successfully terminated and A4 or A6
is committed.
```

```
      caused executing(ag1,A7,1) after   qualified(ag1,A7) && count_k(_J1) &&
_J1 >=2  && (committed(A4) ++ committed(A6)) && (/\_I1: executing(ag1,A7,_I1))
&& -committed(A7).

% Side effect of A1 : count_i is incremented by 1 when A1 is committed.
      caused  count_i(_J1)  if  committed(A1)  after  executing(ag1,A1,_I1)  &&
count_i(_J2) && sum(_J1,_J2,1).

% Side effect of A7 : count_i is decremented by 1 when A7 is committed.
      caused  count_i(_J1)  if  committed(A7)  after  executing(ag1,A7,_I1)  &&
count_i(_J2) && sum(_J2,_J1,1).

% Side effect of A3 : loopcount is incremented by 1 when A3 is committed and
loop is not completed yet.
      caused count_k(_J1) after committed(A3) && count_k(_J2) && (_J2<2) &&
sum(_J1,_J2,1).

% Loop : A2 and A3 are in a loop which must execute while loopcount < 2.
% After A3 commits, if the loop condition evaluates to true then loop
activities A2 and A3 becomes -committed. Therefore they can execute again.
      caused -committed(A2) && -committed(A3)  if count_k(_J2)
&& (_J2<2) after committed(A3).

% PROPOSITIONS TO SIMULATE FLUENTS WHICH CAN HAVE INTEGER VALUES.
      caused  -count_k(_J1) if count_k(_J2) && -(_J1=_J2).
      caused  -count_i(_J1) if count_i(_J2) && -(_J1=_J2).
      caused  -duration(ag1,activity1,_I1) if duration(ag1,activity1,_I2) && -
(_I1=_I2).
```

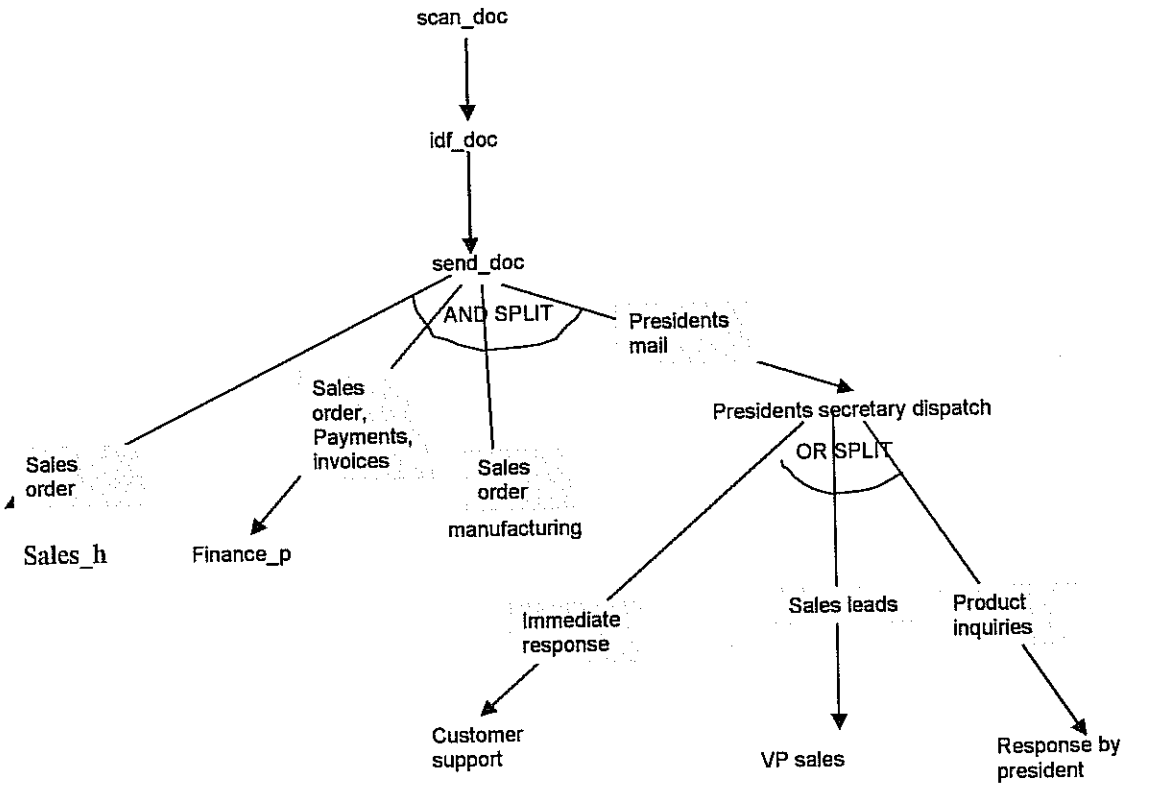
EK 4

İş Akışının CCALC Tarafından Bulunan Modeli

```
:- plan
facts ::
 0: (/\activity1: -committed(activity1)),
 0: (/\activity1: /\ag1: /\_I1: -executing(ag1,activity1,_I1)),
 0: qualified(Agent1, A1),
 0: -qualified(Agent2, A1),
 0: -qualified(Agent3, A1),
 0: qualified(Agent2, A2),
 0: -qualified(Agent1, A2),
 0: -qualified(Agent3, A2),
 0: qualified(Agent2, A3),
 0: -qualified(Agent1, A3),
 0: -qualified(Agent3, A3),
 0: qualified(Agent3, A4),
 0: -qualified(Agent1, A4),
 0: -qualified(Agent2, A4),
 0: qualified(Agent3, A5),
 0: -qualified(Agent1, A5),
 0: -qualified(Agent2, A5),
 0: qualified(Agent3, A6),
 0: -qualified(Agent1, A6),
 0: -qualified(Agent2, A6),
 0: qualified(Agent1, A7),
 0: -qualified(Agent3, A7),
 0: -qualified(Agent2, A7),
 0: duration(Agent1,A1,1),
 0: duration(Agent2,A2,1),
 0: duration(Agent2,A3,1),
 0: duration(Agent3,A4,2),
 0: duration(Agent3,A5,1),
 0: duration(Agent3,A6,1),
 0: duration(Agent1,A7,1),
 0: count_i(2),
 0: count_k(0);
goal ::
14: committed(A7).
```

EK 5

Örnek İş Akışı (PROLOG Çevirimi)



Bu örnekte bir işyerine gelen dökümanlar 'tom' isminde bir eleman tarafından incelenmekte (scan_doc), gruplanıyor(idf_doc) ve gönderilmektedir(send_doc). Döküman ise döküman tipine göre tüm gerekli birimlere gönderilmektedir(AND- SPLIT). Bölüm başkanı sekreterine (Presidents secretary dispatch) gelen dökümanlar ise yine döküman tipine göre ilgili bölüme sevkedilmekte ancak bu durumda birimlerden sadece birine gönderilmektedir (OR-SPLIT).

Aşağıda sıralı çalışma için *C* ile yazılmış çözüm ve buna karşılık gelen PROLOG çözümü verilmiştir.

CCALC çözümü:

```
% File 'workflow-test1'

:- sorts
   agent;
   activity.

:- objects
  scan_doc,
  idf_doc,
  send_doc      :: activity;
  tom           :: agent.


:- variables
  Ag1, Ag2              :: agent;
  Act1, Act2            :: activity.

:- constants
  execute(agent,activity)       :: action;
  qualified(agent,activity)     :: rigid;
  committed(activity)           :: inertialFluent.

:- show all.
% Common rules
 exogenous qualified(Ag1, Act1).

 nonexecutable execute(Ag1, Act1) if -qualified(Ag1, Act1).
 nonexecutable execute(Ag1, Act1) && execute(Ag1, Act2) if -(Act1=Act2).
 nonexecutable execute(Ag1, Act1) && execute(Ag2, Act1) if -(Ag1=Ag2).

 execute(Ag1, scan_doc) causes committed(scan_doc).
 execute(Ag1, idf_doc) causes committed(idf_doc).
 execute(Ag1, send_doc) causes committed(send_doc).

%Sequential execution
 nonexecutable execute(Ag1, idf_doc) if -committed(scan_doc).
 nonexecutable execute(Ag1, send_doc) if -committed(idf_doc).

noconcurrency.

:- query
label :: 1;
maxstep :: 1..10;
0: [/\Act1 |  -committed(Act1)];
0:  qualified(tom, scan_doc);
0:  qualified(tom, idf_doc);
0:  qualified(tom, send_doc);
maxstep : committed(send_doc).
```

PROLOG ÇÖZÜMÜ

```
qualified(tom, scan_doc).
qualified(tom, idf_doc).
qualified(tom, send_doc).

agent(tom).
activity(scan_doc).
activity(idf_doc).
activity(send_doc).

committed(_Act1, 0) :-  fail.
committed(_Act1, -1) :-  fail.

committed(scan_doc, T1) :-  time(T),  time(T1), T>=0, T1 is T+1,
                            agent(Ag1), execute(Ag1, scan_doc,T).

committed(idf_doc, T1) :-  time(T),  time(T1),  T>=0, T1 is T+1,
                            agent(Ag1), execute(Ag1, idf_doc,T).

committed(send_doc, T1) :-  time(T), time(T1),   T>=0, T1 is T+1,
                            agent(Ag1), execute(Ag1, send_doc,T).

committed(Act1, T1) :- time(T),  time(T1), T>=0, T1 is T+1,
                            committed(Act1,T), activity(Act1).

 execute(Ag1, scan_doc, T1) :-  time(T1),  T1>=0,
                                agent(Ag1),
                                qualified(Ag1, scan_doc),
                                not(committed(scan_doc,T1)).

 execute(Ag1, idf_doc, T1) :-   T1>=0, time(T1),
                                agent(Ag1),
                                qualified(Ag1, idf_doc),
                                 not(committed(idf_doc,T1)),
                                committed(scan_doc,T1).

 execute(Ag1, send_doc, T1) :-  T1>=0, time(T1),
                                agent(Ag1),
                                qualified(Ag1, send_doc),
                                not(committed(send_doc,T1)),
                                committed(idf_doc,T1).

 not(P) :- call(P), !, fail.
 not(_).
 time(-1). time(0). time(1). time(2). time(3).
```

CCALC çözümünde örnekte de görüldüğü gibi iki çeşit C kuralı kullanılmıştır bunlar:

(1) nonexecutable F if G

Aşağıdaki örneklerde olduğu gibi gösterilir.

```
nonexecutable execute(Ag1, Act1) if -qualified(Ag1, Act1).
nonexecutable execute(Ag1, Act1) && execute(Ag1, Act2) if -(Act1=Act2).
nonexecutable execute(Ag1, Act1) && execute(Ag2, Act1) if -(Ag1=Ag2).
```

(2) F causes G if H

ise aşağıdaki örneklerde olduğu gibi gösterilir.

```
execute(Ag1, scan_doc) causes committed(scan_doc).
execute(Ag1, idf_doc) causes committed(idf_doc).
execute(Ag1, send_doc) causes committed(send_doc).
```

(2) şeklindeki kuralları rahatlıkla çevirilebilir. Bunlar için yukarıda verilmiş olan ilgili kısım aşağıda tekrarlanmıştır.

```
committed(scan_doc, T1) :-  time(T),  time(T1), T>=0, T1 is T+1,
                            agent(Ag1), execute(Ag1, scan_doc,T).

committed(idf_doc, T1) :-  time(T),  time(T1),  T>=0, T1 is T+1,
                           agent(Ag1), execute(Ag1, idf_doc,T).

committed(send_doc, T1) :-  time(T), time(T1),   T>=0, T1 is T+1,
                            agent(Ag1), execute(Ag1, send_doc,T).

committed(Act1, T1) :- time(T),  time(T1), T>=0, T1 is T+1,
                       committed(Act1,T), activity(Act1).
```

(1) şeklindeki kurallar için ise karşılık gelen Prolog kurallarını yazmak çok daha zordur. Bu kurallar cevap seti bulan sistemlerin bütün çözümlerden istenilmeyen çözümleri elemesi için kullanılır ancak Prolog benzeri bir sistemde bir tek çözüm üzerinden sonuç bulmaya çalışılır. Bu durumda iki çeşit çözüm düşünülebilir.

(a) 'nonexecutable' kurallarını 'abnormality' olarak tanımlamak. Ancak bu çözüm kuralların yapısından dolayı sonsuz döngülere sebep olmaktadır. Sonsuz döngünün sebepleri burada tartışılmayacaktır, ancak bu çözüm kullanılmak istenildiği takdirde C kuralları üzerinde yeni sınırlandırmalar getirmek gerekecektir.

(b) 'nonexecutable' kurallarında verilenler kurallara önşart olarak eklemek. Bu çözüme göre

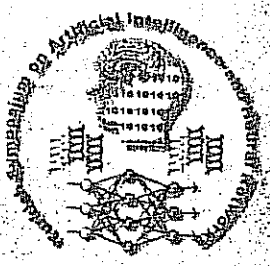
```
nonexecutable execute(Ag1, Act1) if -qualified(Ag1, Act1).
nonexecutable execute(Ag1, idf_doc) if -committed(scan_doc).
```

kuralları Prolog halinde aşağıdaki gibi yazılır.

```
execute(Ag1, idf_doc, T1) :-   T1>=0, time(T1),
                               agent(Ag1),
                               qualified(Ag1, idf_doc),
                                not(committed(idf_doc,T1)),
                               committed(scan_doc,T1).
```

EK 6

Projeden Kaynaklanan Yayınlar

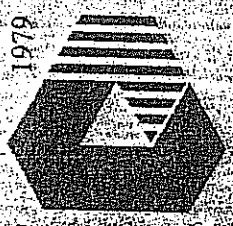


The Tenth Turkish Symposium on Artificial Intelligence and Neural Networks

Symposium Proceedings
Gazimagusa
June 21-22, 2001

TAINN 2001

Edited by
A. ACAN
I. AYBAY
M. SALAMAH



Using Stable Model Semantics (SMODELS) in the Causal Calculator (CCALC)

Semra Doğandağ*, F. Nur Alpaslan*, Varol Akman**

* Department of Computer Engineering
Middle East Technical University
Ankara, TR-06531 Turkey
{semra, alpaslan}@ceng.metu.edu.tr
** Department of Computer Engineering
Bilkent University
Ankara, TR-06533 Turkey
akman@cs.bilkent.edu.tr

ABSTRACT

Action Languages are formal methods of talking about actions and their effects on fluents. One recent approach in planning is to define the domains of the planning problems using action languages. The aim of this research is to find a plan for a system defined in the action language *C* by translating it into a causal theory and then finding an equivalent logic program. The planning problem will then be reduced to finding the answer set (stable model) of this logic program. This planner will be added as an extension to the Causal Calculator (CCALC) which is a model checker for the language of the causal theories.

I INTRODUCTION

Plan generation plays an important role in AI research. A lot of work has been done on representing actions and generating plans. While reasoning about action, some problems (such as the frame, ramification, and qualification problems) have to be taken into account. It has been shown [3] that these problems can be overcome to some extent by using causal knowledge. CCALC, which is a system written at the University of Texas, Austin, finds plans and reasons about the actions defined in causal theories. Programs written in the action description language *C* can be given as input to the CCALC system. CCALC translates them to the corresponding causal theory. The *C* language can express indirect effects, implicit preconditions, action preconditions, nondeterminism, concurrent actions, and noninertial fluents. CCALC gets the equivalent propositional representation of the problem by literal completion and finds the models of the system by using the available satisfiability solvers. (A user of CCALC currently has the choice of using either "relsat" [13] or "sato" [4] as the satisfiability solver).

Another perspective to planning is to use logic programming. The language of logic programming offers a reasonably expressive way, enabling us to describe the effects of actions. Also there is a

relation between logic programming and causal theories in that both are *noncontrapositive*[1]. As is mentioned in [17] the difference between

$$p \leftarrow not\ q \tag{1}$$

and

$$q \leftarrow not\ p \tag{2}$$

is apparent both in logic programming and in causal theories but is lost if propositional logic is used. The first sentence means that the absence of q is a cause for p, but this doesn't mean that the absence of p is a cause for q, which is what the second sentence says. To combine the ideas of logic programming with the use of expressive action description languages, the domain described in the action language can be translated into an equivalent logic program. Our research will realize this by first translating a program in C language to a causal theory and then the causal theory to the corresponding logic program. The answer set of the program will be used for planning or querying the plans found.

Causality is an important concept for nonmonotonic reasoning while working on action representations and their effects on the fluents. Situation calculus which specifies the effects of actions using only domain constraints is too weak and cannot deal with the well known problems such as the frame, qualification, and ramification problems. The reference [3] adresses this problems and intoduces a ternary predicate *Caused* to situation calculus to solve them. In this research the formalism introduced by [12] will be used. According to this formalism, one does not need to know the cause of facts in order to determine whether a world history is causally possible. It is sufficient to know the conditions under which the facts are caused. The causal theory is a set of causal rules which are expressions of the form

$$\phi \Rightarrow \psi \tag{3}$$

where ϕ and ψ are formulas of the underlying propositional language. The intended reading of (3) is: *Necessarily, if ϕ then the fact that ψ is caused.*

It is possible to formalize action domains using this nonmonotonic causal theory. To formalise action domains three pair-wise disjoint sets are defined: a set of *action names*, a set of *fluent names*, and a set of *time instances*. The atoms of the language are expressions of the forms a_t and f_t where a, f, t are action, fluent and time instances respectively. a_t means action occurs at time t and f_t means fluent f holds at time t. In this formalism, the causes of the actions are not taken into account and the following two schemas provide that the occurrence of actions is exogenous[2] to the causal theory:

$$a_t \Rightarrow a_t \tag{4}$$

$$f_t \Rightarrow f_t \tag{5}$$

Also the initial values of the fluents are exogenous to the causal theory and this can be represented using the following schemas:

$$f_0 \Rightarrow f_0 \tag{6}$$

$$\neg f_0 \Rightarrow \neg f_0 \tag{7}$$

[1] An implication of the form $\neg q \leftarrow \neg p$ is the contrapositive of $p \leftarrow q$. In a noncontrapositive theory these two statements don't have the same meaning.

[2] An action or fluent is *exogenous* if its cause lies outside of the theory.

Inertia, that is a fluent doesn't change its value unless there is a reason for it, can be expressed using the schema below, where σ is a meta-variable for inertial fluent.

$$\sigma_t \wedge \sigma_{t+1} \Rightarrow \sigma_{t+1} \tag{8}$$

also be described and things that change by themselves can be modeled.

One of the ways of expressing the non-monoticity and negation as failure in logic programming is the "model theoretic" approach. In this approach, the models of the program represent the semantics of the program. Instead of doing query evaluation directly, the intended meaning of the program is obtained by finding its model. As an example to the "model theoretic" approach the well-founded semantics, perfect model semantics, and stable model semantics can be given. Stable model semantics is related with auto-epistemic logic. The stable model of a system is said to be the rational beliefs of an agent. Every predicate in a stable-model is "supported", that is, have a reason to be there. The Smodels system [6] is an implementation of the stable model semantics for range-restricted function-free normal programs. It has two modules: one for finding the stable model of a grounded program and one for grounding the given range-restricted program. There are different front-ends of the Smodels but the richest of them in terms of features is lparse [15].

II PLANNING WITH STABLE MODEL SEMANTICS

The aim of this research is to use logic programming and stable model semantics for plan generation and add it as an option to the CCALC system. The given domain description in C will be translated to a logic program and the logic program will be grounded to obtain a ground logic program. The Smodels system accepts ground logic programs and finds their corresponding stable model. The stable model of the program will then be used to find the sought plan. There are different ways of doing all these; one can either take the input written in C and directly translate it to a logic program or let CCALC translate it to causal theory and then translate this causal theory to a logic program. In our research, the latter alternative is chosen. Moreover, there are two methods for grounding: let CCALC do grounding and get the ground causal theory or let lparse do grounding, which is the front-end parser of Smodels. In our research the latter method will be applied, because of efficiency consideration.
As a result, the execution steps will be in the following order

- Get the causal theory of the planning problem from CCALC
- Translate it to an equivalent logic program
- Call lparse to ground the program
- Call Smodels to get the answer set (stable model) of the program
- Extract the plan(s) from the stable model of the program

The translation task starts with getting the causal theory from CCALC system. During the translation, instead of using a different predicate for each action and fluent, two predicates will be used. These are *'holds'* and *'occurs'*. The *'holds'* predicate is in the form

$$holds(F, T) \tag{9}$$

meaning that fluent F holds at time T. The *'occurs'* predicate is in the form

$$occurs(A, T) \tag{10}$$

meaning that action A occurs at time T.
One problem is that the early versions of Smodels doesn't have classical negation, but this problem can be easily solved, as suggested in [17], by introducing new predicates standing for the negations of *holds* and *occurs*. The following four rules are used for handling the classical negation problem:

```
notholds(F,0) :- not holds(F,0), fluent(F).
notoccurs(A,0) :- not occurs(A,0), action(A).
holds(F,0) :- not notholds(F,0), fluent(F).
occurs(A,0) :- not notoccurs(A,0), action(A).
:- not holds(F,T), not notholds(F,T), fluent(F), time(T), T>0.
:- not occurs(A,S), not notoccurs(A,S), action(A), step(S), S>0.
```

The other rules of the logic program are obtained by translating the causal theory produced by CCALC. As an example, a solution to the Yale Shooting Problem(YSP) [15] will now be revisited[3].
The *constants* and *variables* are declared as follows[4]:

```
gun(g1).
gun(g2).

action(load(g1)).
action(load(g2)).
action(shoot(g1)).
action(shoot(g2)).
```

Time is represented using the predicates `time` `step` and `next`:

```
time(0). time(1). time(2).
step(0). step(1).
next(0,1). next(1,2).
```

`step` is used to make sure that no action takes place at the last time point.
All of the causal rules returned by CCALC are translated to the corresponding logic programming rules. For example, the rule:

```
shoot(G) causes -alive if loaded(G).
```

becomes

```
-h(alive,1) <- h(loaded(G),0) && o(shoot(G),0) && true
```

which in turn translated to:

```
notholds(alive,T1) :- holds(loaded(G),T),
      occurs(shoot(G),T),
      time(T),
      next(T,T1),
      gun(G).
```

[3]This example is for giving a complete overview of the translation task. More detailed information about the translation will be given in section 3 and 4
[4]Some of the declarations are omitted for the sake of simplicity.

The initial state and the goal state of the planning domain is written in the *compute* statement of Smodels. For example, for YSP the initial condition is that both of the guns are unloaded and the turkey is alive and the goal to be reached is that the turkey is dead at time 2. So the corresponding *compute* statement is:

```
compute all{ holds(unloaded(g1), 0), holds(unloaded(g2),0),
     holds(alive,0), notholds(alive,2)
   }.
```

The initial state is:

```
holds(unloaded(g1), 0) holds(unloaded(g2),0) holds(alive,0)
```

The goal state is:

```
notholds(alive,2)
```

III TRANSLATION OF CAUSAL RULES TO LOGIC PROGRAM

Causal rules returned by CCALC are in the form:

$$head \Leftarrow body \tag{11}$$

where head and body can be any kind of logical formulas constructed with the constructs &&, ++, $\rightarrow$, $\leftrightarrow$, $\vee$, $\wedge$. Here && stands for "and", ++ for "or", $\rightarrow$ for "only if", $\leftrightarrow$ for "if and only if", $\vee$ for "there exist" and $\wedge$ for "for all".
Translation starts first by eliminating the constructs $\vee$ and $\wedge$ if there is any. For each expression in the form

$$\vee X F(X) \tag{12}$$

where $F(X)$ is an expression having the free variable X. The instantiation of X results in the formula :

$$F(a_1) + + F(a_2) ... + + F(a_n) \tag{13}$$

where a_i is a constant in the domain of the variable X. The same method applies to the construct $\wedge$, but in this case ++ is replaced with &&.
The next step is placing this formula in the 'Disjunctive Normal Form' (DNF). The logic program statements are constructed as :

$$\begin{aligned} head &: -b_1 \\ head &: -b_2 \\ &\cdot \\ &\cdot \\ head &: -b_n \end{aligned} \tag{14}$$

where b_i represents a disjunct of the expression that was derived from the original expression by eliminating the constructs $\vee$ and $\wedge$. To illustrate this translation an example from the "gripper problem" can be given. In the CCALC formalization of the gripper problem in [11], the below rule written in C language says that: If the gripper of the robot is holding a ball it cannot pick up another ball. The rule is in the form :

```
nonexecutable pickUp(B,G) if \/B1:isHolding(B1,G).
```

The corresponding causal rule returned by CCALC is in the form:

```
(false<-(o(pickUp(B,G),0)&&\/B1:h(isHolding(B1,G),0))&&true)
```

Elimination of ∨ results in the expression:

```
o(pickUp(B,G),0) && ((h(isHolding(ball1,G),0)) \/
                     (h(isHolding(ball2,G),0)) \/
                     (h(isHolding(ball3,G),0)))
```

Placing the expression in DNF results in the expression:

```
(o(pickUp(B,G),0) && (h(isHolding(ball1,G),0))) \/
(o(pickUp(B,G),0) && (h(isHolding(ball2,G),0))) \/
(o(pickUp(B,G),0) && (h(isHolding(ball3,G),0))) \/
```

The corresponding logic program statements are given below:

```
false :- o(pickUp(B,G),0), (h(isHolding(ball1,G),0))
false :- o(pickUp(B,G),0), (h(isHolding(ball2,G),0))
false :- o(pickUp(B,G),0), (h(isHolding(ball3,G),0))
```

IV FORMALIZING PLANNING PROBLEMS AS LOGIC PROGRAMS

CCALC takes a planning problem as a set of facts and goal states. For each fact, the time instance at which it holds, and for each goal state, the time instance at which it should hold is given. On the other hand, the *compute* statement of Smodels system doesn't make any distinction between facts and goals. Writing a literal in the *compute* statement means that the model of the system should include this literal. Therefore the literals in the *compute* statement represent a conjunction. *Compute* statement should include only ground literals.
One of the planning problems for the "gripper problem" given as input to CCALC in [11] is as follows:[5]

```
:- plan
label ::
  0;
facts ::
  0: /\B: at(B,room1),
  0: /\B: color(B,white),
  0: /\B: onFloor(B),
  0: at(robot,room3);
goals ::
  15: (\/B: color(B,red) && \/B: color(B,white) && \/B: color(B,blue))
```

As in the previous case the translation starts with elimination of the constructs ∨ and ∧ if there is any. Then for each fact and goal its corresponding 'Conjunctive Normal Form'(CNF) is found. For example for the goal:

```
15: (\/B: color(B,red) && \/B: color(B,white) && \/B: color(B,blue))
```

[5]This is a simplification of the planning problem in [11].

the conjuncts are:

```
h(at(ball1,white),14) \/ h(at(ball2,white),14) \/ h(at(ball3,white),
h(at(ball1,red),14) \/ h(at(ball2,red),14) \/ h(at(ball3,red),14)
h(at(ball1,blue),14) \/ h(at(ball2,blue),14) \/ h(at(ball3,blue),14)
```

If the conjunct consists of only one literal it can be directly put in the *compute* statement, as in the planning problem in section 2. Otherwise, a new rule with the same head is created for each literal in the conjunct. The case for the above goal statement is illustrated below and the corresponding *compute* statement is given:

```
temp1:- h(color(ball1,white),14).
temp1:- h(color(ball2,white),14).
temp1:- h(color(ball3,white),14).

temp2:- h(color(ball1,blue),14).
temp2:- h(color(ball2,blue),14).
temp2:- h(color(ball3,blue),14).

temp3:- h(color(ball1,red),14).
temp3:- h(color(ball2,red),14).
temp3:- h(color(ball3,red),14).

compute 1{
h(at(ball1,room1),0), h(at(ball2,room1),0),
h(at(ball3,room1),0), h(color(ball1,white),0),
h(color(ball2,white),0), h(color(ball3,white),0),
h(onFloor(ball1),0), h(onFloor(ball2),0),
h(onFloor(ball3),0), h(at(robot,room3),0),
temp1, temp2, temp3
}.
```

V AN EXAMPLE

Applying the method described in the previous section to YSP, the following logic program is obtained:

```
% Yale Shooting Problem with two guns
gun(g1).
gun(g2).

action(load(g1)).
action(load(g2)).
action(shoot(g1)).
action(shoot(g2)).

inertialFluent(alive).
```

```
inertialFluent(loaded(g1)).
inertialFluent(loaded(g2)).

inertialFalseFluent(alive).
inertialFalseFluent(loaded(g1)).
inertialFalseFluent(loaded(g2)).

inertialTrueFluent(alive).
inertialTrueFluent(loaded(g1)).
inertialTrueFluent(loaded(g2)).

fluent(alive).
fluent(loaded(g1)).
fluent(loaded(g2)).

hide.
show o(A,T).
show h(F,T).
show noth(F,T).

time(0). time(1). time(2).
step(0). step(1).
next(0,1). next(1,2).

% Only one action can be executed at a time
1{o(A,S): action(A)}1 :- step(S).
:- noth(F,T), h(F,T), fluent(F), time(T).

h(F,0) :- h(F,0),fluent(F).
o(A,S) :- o(A,S), action(A), step(S).
noth(F,0) :- noth(F,0), fluent(F).
noto(A,S) :- noto(A,S), action(A), step(S).

% Rules for implementing the classical negation
noth(F,0) :- not h(F,0), fluent(F).
noto(A,S) :- not o(A,S), action(A), step(S).

h(F,0) :- not noth(F,0), fluent(F).
o(A,S) :- not noto(A,S), action(A), step(S).

:- not h(F,T), not noth(F,T), fluent(F), time(T), T>0.
:- not o(A,S), not noto(A,S), action(A), step(S), S>0.

% The following rules are written by translating the causal theory
% returned by CCALC.

h(loaded(G),T1) :- o(load(G),S), step(S), time(T1),next(S,T1),
  gun(G).
```

```
noth(alive,T1) :- h(loaded(G),S), o(shoot(G),S), step(S),time(T1),
                  next(S,T1), gun(G).

noth(loaded(G),T1) :-o(shoot(G),S), step(S), time(T1), next(S,T1),
     gun(G).

h(IT,T) :- h(IT,S), not noth(IT,T), step(S), time(T), next(S,T),
           inertialTrueFluent(IT).

h(DT,T) :- h(DT,T), defaultTrueFluent(DT), time(T), T>0.

noth(DF,T) :- noth(DF,T), defaultFalseFluent(DF), time(T), T>0.

noth(IF,T) :- noth(IF,S), not h(IF,T), step(S), time(T), next(S,T),
              inertialFalseFluent(IF).

% The initial and final states
compute all{
  h(alive,0), noth(loaded(g1),0), noth(loaded(g2),0), noth(alive,2)
}.
```

The Smodels system finds two stable models for this program. These are:

```
Answer: 1
Stable Model: noth(loaded(g1),0) noth(loaded(g2),0) h(alive,0)
noth(alive,2) o(shoot(g2),1) h(loaded(g2),1) noth(loaded(g2),2)
noth(loaded(g1),2) noth(loaded(g1),1) h(alive,1) o(load(g2),0)
Answer: 2
Stable Model: noth(loaded(g1),0) noth(loaded(g2),0) h(alive,0)
noth(alive,2) o(shoot(g1),1) h(loaded(g1),1) noth(loaded(g2),2)
noth(loaded(g2),1) noth(loaded(g1),2) h(alive,1) o(load(g1),0)
```

The first model gives a plan in which the gun g2 is loaded and shooted and the second model gives a plan in which the other gun g1 is loaded and shooted.

VI CONCLUSION

In this research, the aim is to use the action description language *C* to define actions and their effects on fluents. By translating *C* program to a logic program, it will be possible to combine the expressiveness of *C* with efficient computational techniques for logic programming. Finding the stable model of the logic program using Smodels will provide the required planning for a given domain.

REFERENCES

[1] Chitta Baral and Michael Gelfond. "Logic programming and knowledge representation." *Journal of Logic Programming*, 19(20), pp. 74-148, (1994).

[2] Enrico Giunchiglia and Vladimir Lifschitz. "An action language based on causal explanation: Preliminary report." In *Proc. AAAI-98*, pp. 623-630, (1998).

[3] Fangzhen Lin. "Embracing causality in specifying the indirect effects of actions." In *Proc. of IJCAI-95, 1985-1991* .

[4] Hanto Zhang. "SATO: An efficient propositional prover." In *Proc. of International Conference of Automated Deduction (CADE-97), (1997).*

[5] Henry Kautz and Bart Selman. Planning as satisfiability. In *Proc. of ECAI-92*, pp. 359-379, (1992).

[6] Ilkka Niemelä and Patrik Simons. "Efficient implementation of the well-founded and stable model semantics." In *Proc. of the Joint International Conference and Symposium on Logic Programming*, pp. 289-303, (1996).

[7] Ilkka Niemelä, Patrik Simons and Timo Soininen. Stable model semantics and weight constraint rules. In *Proc. of the Fifth International Conference on Logic Programming and Non-Monotonic Reasoning*, (1999).

[8] John McCarthy. "Elaboration tolerance." In progress, 1999. [Available at `http://www-formal.stanford.edu/jmc/elaboration.html`]

[9] Joohung Lee and Vladimir Lifschitz.Additive fluents(DRAFT),2000. [Available at `http://www.cs.utexas.edu/users/vl/mypapers/additive.ps`]

[10] Michael Gelfond and Vladimir Lifschitz. "Action languages." *Electronic Transaction on AI* 3, (1998).

[11] Norman McCain. Using the Causal Calculator(Draft),1999. [Available at `http://www.cs.utexas.edu/users/mccain/cc/ccalc/manual-C.ps`]

[12] Norman McCain and Hudson Turner. "Causal Theories of action and change." In *Proc. AAAI-97*, pp. 460-465, (1997).

[13] Roberto Bayardo and Robert Schrag. "Using the CSP look-back techniques to solve real-world SAT instances." *In Proc. of AAAI-97*, pp. 203-208, (1997).

[14] Patrik Simons. "Extending the stable model semantics with more expressive rules." In *Proc. of the 5th International Conference on Logic Programming and Non-monotonic Reasoning*, (1999).

[15] Tommi Syrjänen, "Lparse User's Manual (Draft 0.9)",1999.[Available at `http://www.tcs.hut.fi/Software/smodels/lparse/lparse.ps.gz`]

[16] Vladimir Lifschitz. "Missionaries and Cannibals in the Causal Calculator." In *Principles of Knowledge Representation and Reasoning: Proc. Seventh Int'l Conf.*, pp. 85-96, (2000).

[17] Vladimir Lifschitz. "Action languages, answer sets and planning." In *The Logic Programming Paradigm: A 25-Year Perspective*, Springer-Verlag, (1999).

ISCIS

Proceedings of

The 17th International Symposium on Computer and Information Sciences

Edited by

Ilyas Cicekli • Nihan Kesim Cicekli • Erol Gelenbe

ISCIS XVII
October 28-30, 2002
Orlando, Florida USA

Organized by
Computer Science Program
School of Electrical Engineering and Computer Science
University of Central Florida

CRC PRESS

Boca Raton London New York Washington, D.C.

Visualizing Transition Diagrams of Action Language Programs

Özcan Koç, Ferda N. Alpaslan, Nihan K. Çiçekli
Department of Computer Engineering
Middle East Technical University,
06531 Ankara, TURKEY
Phone: +90–312–210–2080
Fax: +90-312–210–1259
okoc@udel.edu, alpaslan@ceng.metu.edu.tr, nihan@cs.ucf.edu

Abstract

The subject of action languages is one of the prominent research topics in current Artificial Intelligence (AI) research. One of the problems in teaching and learning action languages as well as writing causal theory expressions is the difficulty of visualizing transition diagrams in mind. A tool, called TDV, which extends CCALC [GL98b] and uses GraphViz[KN91] software, is developed to visualize transition diagrams of $\mathcal{C}$ programs.

Keywords : Action Languages, causal theories, transition diagrams, visualization.

1 Introduction

The subject of action languages is one of the prominent research topics in current Artificial Intelligence (AI) research. An action language allows its users to study the change and properties of actions by means of fluents and causal relations among them. A *fluent* is a judgement about the status of objects in the world—*i.e.* a logical world model. Causal relationships among fluents are expressed using logical propositions. For example,

Shoot(gun) causes $\neg$*Alive* if *loaded(gun)*

describes the effect of *Shoot* action on *Alive* fluent. Action languages and related research are discussed in many recent papers [GL98a], [BG97]. $\mathcal{C}$ is an action language developed by Enrico Giunchiglia and Vladimir Lifschitz[GL98b] and uses the idea of causal theories. $\mathcal{C}$ is a language having two kinds of *laws*. Static laws are of the form

caused F if G

and dynamic laws are of the form

caused F if G after H

where F and G are propositional combinations of fluent names and H is a propositional combination of fluent and action names. The dynamic laws are used to show the direct effects of the actions. The $\mathcal{C}$ language also has some additional expressions that can be written in the form of static and dynamic laws. These are itemized below:

U causes F if G
inertial F
always F
nonexecutable U if F
default F if G
U may cause F if G

where F and G are propositional combinations of fluent names and U is a propositional combination of fluent and action names. Examples of simple planning problems that are solved using $\mathcal{C}$ can be found in [McC99]. A $\mathcal{C}$ program defines a *transition system* which consists of sets of fluents and relationships among these fluent sets. Although $\mathcal{C}$ offers a complete system in formalizing action language domains, it lacks a tool to obtain a visual appearance of the resulting transition systems. Drawing a transition diagram is the best way of representing the whole state space of the problem domain. Doing it manually is time consuming and open to errors. Moreover, an automated tool would be helpful for the learners and teachers of action languages.

The Causal Calculator (CCALC) [McC99] is a system, written in Prolog, for query answering and satisfiability in the context of planning. It is de-

veloped for the language of causal theories [MT97]. The input to CCALC is given in $\mathcal{C}$ which is translated to causal theory by using rewrite rules. The causal theory is grounded to obtain a ground causal theory. The ground causal theory is translated to propositional logic by means of literal completion. Then, the propositional logic formulas are put into the conjunctive normal form (CNF). To find a model of the system, CCALC uses a so–called *satisfiability solver* (SAT). These are propositional provers based on Davis–Putnam method and expect their input in CNF. The models found by SATs are then used for planning. The idea behind this approach can be found in [KS92].

We have developed a software, called TDV(Transition Diagram Visualizer), to draw transition diagrams defined by programs written in action language $\mathcal{C}$. TDV extends the CCALC package and uses GraphViz[KN91] package for drawing.

The rest of this paper is organized as follows: Section 2 explains transition diagrams, Section 3 discusses implementation details, Section 4 compares the performance of algorithms, Section 5 presents the visual customizations and Section 6 contains the conclusion.

2 Transition Diagrams

Programs written in action languages consist of causal clauses which constitue a causal theory. Every causal theory defines a transition system. A *transition system* is a set of transitions of the form $<s, A, s'>$, where s is initial state A is a set of states and s', is resulting state

At each state, every fluent has a value of *true* or *false*. Hence, a state s is a set of all literals, *i.e.* positive or negative fluents. A is a, possibly empty, set of concurrent actions that a causal theory allows to execute concurrently. If we assume that there are n fluents and m actions, there are $2^n \times m \times 2^n$ possible transitions for non–concurrent case and $2^n \times 2^m \times 2^n$ possible transitions for the concurrent case. Among these, some are *causally explained*[1]. A *transition diagram* refers to a set of causally explained transitions.

```
:- include 'C.t'.
:- sorts latch.
:- variables L  :: latch.
:- constants
 l1, l2          :: latch;
 up(latch)       :: defaultFalseFluent;
 open            :: inertialFluent;
 toggle(latch)   :: action.
caused open if up(l1) && up(l2).
caused -open if -up(l1) ++ -up(l2).
toggle(L) causes up(L) if -up(L).
toggle(L) causes -up(L) if up(L).
```

Figure 1: $\mathcal{C}$ code for suitcase domain

We can represent transition diagrams by means of labeled directed graphs. In such a representation, nodes correspond to states and edges represent transitions. Figure 2 presents the transition diagram of Lin's suitcase domain [Fan95] whose code is presented in Figure 1 as an example. In the suitcase domain, there is a spring–loaded suitcase with two latches. The suitcase is open whenever both latches are up. According to Figure 2, the domain has 3 fluents—*open*, *up(l1)* and *up(l2)*—and 2 actions—*toggle(l1)* and *toggle(l2)*. Since this is a concurrent example[2], there are $2^3 \times 2^2 \times 2^3 = 256$ possible transitions. Among these 14 of them are causally explained. In $\mathcal{C}$, states may change (or remain same) without executing any action. Transitions labeled with 0 represent such null actions. Note that *up(l1)* and *up(l2)* fluents are defined as `defaultFalseFluent`. If we modify this definition as `up(latch) :: inertialFluent;` then, we would obtain a transition diagram as illustrated in Figure 3. In this case, transition diagram has again 14 causally explained transitions with certain differences, *i.e.* some of the transitions are different. These two figures emphasize the importance of formalization.

Although the Figures 2 and 3 are strongly connected directed graph, transition diagrams may be unconnected. A directed graph is said to be strongly connected if and only if there is a path—in the sense of graph theory—between every two vertices in the graph. Figure 4 illusrates a transi-

[1] A transition $< s, A, s' >$ is *causally explained* if its resulting state s' is the only interpretation of σ^{fl}, *i.e* fluent symbols, that satisfies all formulas caused in this transition[GL98b].

[2] $\mathcal{C}$ assumes that any program is concurrent unless it contains a noconcurrency—or equivalently nonexecutable _A && _A1 if _A0<_A1—statement.

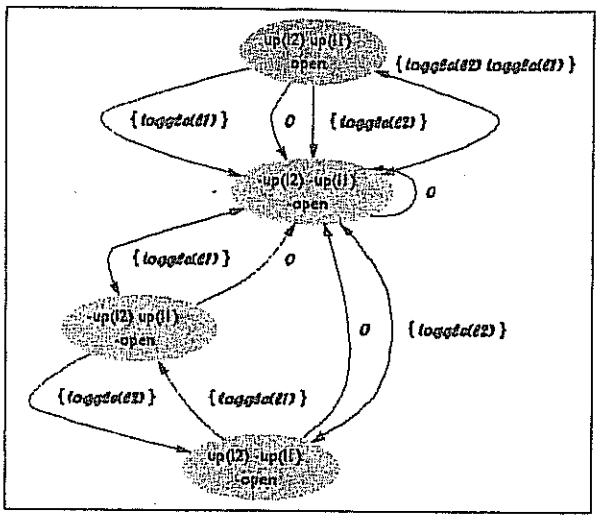


Figure 2: Transition diagram of suitcase example

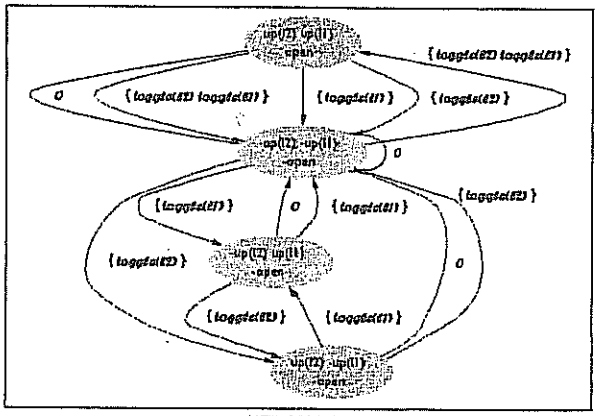


Figure 3: Transition diagram of suitcase example (with inertial fluents)

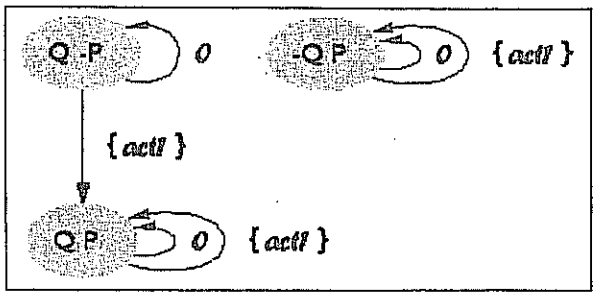


Figure 4: A transition diagram which is not strongly connected.

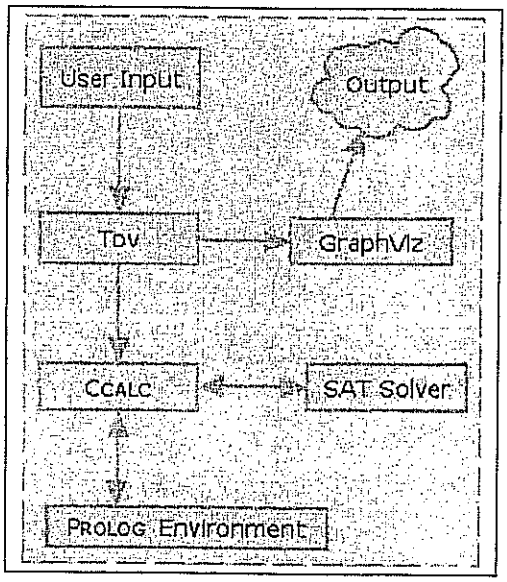


Figure 5: TDV architecture

tion diagram which is not strongly connected. This graph is not strongly connected, because it is not possible to reach $\{\neg P\ \neg Q\}$, or any other node, from $\{P\ Q\}$.

3 Visualization of Transition Diagrams

In order to visualize the transition diagram of a $\mathcal{C}$ program, TD software extends the CCALC software as illustrated in Figure 5. CCALC passes the information about the nodes and transitions among the nodes to TDV. After extracting the node and transition information, the result is written to a file in a format readable by AT&T's GraphViz software and the file is sent to *dot*[3] as input. *dot* produces the transition diagram. CCALC neither produces nodes or transitions, nor uses this information. Hence, this information is obtained by posing planning problems to CCALC as outlined in algorithm, named `algorithm 1`, below:

Algorithm

1. Ask CCALC to find a plan of length 1 with initial condition `true` and goal `true`. CCALC

[3] *dot* is a part of GraphViz package.

produces a plan with an initial state s_0, a final state s_1 and a set of actions A_0.

2. While there are more transitions, construct a new planning problem with

   - current state is s,
   - goal state s' is different from previously found goals,
     i.e. $s' \neq s_1 \wedge s' \neq s_2 \wedge \ldots s' \neq s_n$ where s_i are goals found so far,
   - set of actions A is different than previously found action sets, *i.e.* $A \neq A_1 \wedge A \neq A_2 \wedge \ldots A \neq A_n$ where A_i are actions found so far.

   After finding the new transition $< s, A, s' >$, add s' to the set of non–processed nodes if we see this node for the first time.

3. When there are no more transitions, mark current state s as processed, take another non–processed node as new s, and go to step 2.

4. Repeat step 3 until all nodes are processed.

5. Output the resulting nodes and transitions to a file.

As we discussed before, a state may change, or remain same, without performing any action. In such a case, how to specify that at least one action should be used in order to get a different transition is a problem needs special handling. This is formulated by the following statement:

```
0: \/V_A1: o(V_A1,0)
```

This statement specifies that at least one action should occur, and it solves the problem. Although the algorithm outlined above is correct, transition diagrams produced by the algorithm may be incomplete. As we discussed in the previous section, transition diagrams may be unconnected. The above algorithm, however, finds only a connected subgraph of the transition diagram. To overcome this difficulty, we can generate all states by taking permutations of 2^n literals beforehand, and use them as non–processed nodes. This algorithm is named `algorithm 2`. However, this would decrease the performance considerably for graphs which are not fully connected. A hybrid of these two algorithms, named `algorithm 3`, is also possible. In this case, a set of desired nodes is given to the system as non–processed nodes. System discovers the remaining nodes by applying the first algorithm. All of these 3 algorithms were implemented in TDV.

4 Performance Issues

In order to discuss the complexities of the algorithms, let us consider a domain with n fluents, m actions, T causally explained transitions and N nodes. We assume N and T are actual number of nodes and edges, respectively, in the resulting transition diagrams, hence they satisfy

- $0 \leq N \leq 2^n$ (concurrent and non–concurrent case)
- $0 \leq T \leq N \times 2^m \times N$ (concurrent) or $0 \leq T \leq N \times (m+1) \times N$ (non–concurrent case)

Therefore, `algorithm 1` runs in $O(T)$[4] in general and in $O(N \times 2^m \times N)$ for the worst case. `Algorithm 2` first discovers *all* possible nodes ($O(2^n)$), then it finds all causally explained transitions ($O(T)$). So, `algorithm 2` runs in $O(2^n) + O(T)$, which impiles a worst case of $O(2^n \times 2^m \times 2^n)$ and a best case of $O(2^n)$. `Algorithm 3`, like `algorithm 1`, runs faster than `algorithm 2`. This algorithm runs in $O(T_1 + T_2 + T_3 + \cdots + T_n)$, where each T_i denotes the number of causally explained transitions in discovered subgraph i. As we can easily see, the summation satisfies the $0 \leq T_1 + T_2 + T_3 + \cdots + T_n \leq T$ inequality, since the summation of number of causally explained transitions in subgraphs cannot exceed the total number of causally explained transitions.

Our experiments, *see* appendix, showed that our program runs reasonably fast for domains with 2^{11} or less nodes. Each of the examples took less than six minutes on an average Pentium II PC with 64 MB RAM. We observed that, most of this time is spent for file I/O operations of CCALC. We believe that running times may be decreased to 30–75% if CCALC is compiled with the SAT solvers. On the other hand, we should note that the problem is, *by definition*, exponential; *i.e.* there can be up to $2^n \times 2^m \times 2^n$ transitions. So, for larger domains, TDV cannot produce results fastly, or even indefinitely. For example, TDV is not able to produce the

[4] If the graph is not connected, T is the number of causally explained transitions in the subgraph discovered by the algorithm.

transition diagram of airport problem[LMRT00], since there are 42 fluents—which impiles 2^{42} = 4.398.046.511.104 possible nodes.

5 Visual Customizations

GraphViz package offers some customizations about the visual appearance of graphs, and we have an interface for these options. TDV supports the customizations of shape of nodes, node and edge fonts, size/color of node and edge fonts, multi-line labels for nodes and edges, thickness of edges, scale of graphs and it is able to produce a smaller version of transition diagrams by merging[5] bi-directional edges, *etc.*

In addition to these customization, *dotty* program of GraphViz package allows its users to modify the layout of the graph.

6 Conclusion and Suggested Work

Action laguages is one of the important topics in current AI research. In order to help the study of action languages research a tool which is capable of displaying the transition diagrams of $\mathcal{C}$ programs is developed. We will improve our tool by implementing algorithm 3. Three different algorithms were developed to achieve speed and completeness. Also, several visual customizations are offered for convenience. System is tested in SWI-Prolog under Windows NT/2000. Minor revisions may be done to run the program under different operating systems. Furthermore, we believe that it would be useful to develop a web based interface to achive a cross-platform system.

Acknowledgements

We are grateful to Vladimir Lifschitz and Esra Erdem for their suggestions and useful comments.

[5] If there are two nodes X and Y such that there is a transition from X to Y with action set A and another transition from Y to X with the same set of actions , *i.e.* A, TDV can display a single edge with two arrowheads which reduces the number edges displayed. Such kind of edges are called bi-directional edges.

Appendix

Yale Shooting Domain

```
:- sorts gun.
:- variables G :: gun.
:- constants
 g1, g2                 :: gun;
 load(gun), shoot(gun)  :: action;
 alive, loaded(gun)     :: inertialFluent.
load(G) causes loaded(G).
shoot(G) causes -alive if loaded(G).
shoot(G) causes -loaded(G).
```

$\mathcal{C}$ code for yale shooting example

References

[BG97] Chitta Baral and Michael Gelfond. Reasoning about effects of concurrent actions. *Journal of Logic Programming 31*, 1997.

[Fan95] Lin Fangzhen. Embracing causality in specifying the indirect effects of actions. *Proc. of IJCAI-95*, pages 1985–1991, 1995.

[GL98a] Michael Gelfond and Vladimir Lifschitz. Actions languages. *Electronic Transactions on AI*, page 3, 1998.

[GL98b] Enrico Giunchiglia and Vladimir Lifschitz. An actions language based on causal explanation: Preliminary report. *Proc. AAAI-98*, pages 623–630, 1998.

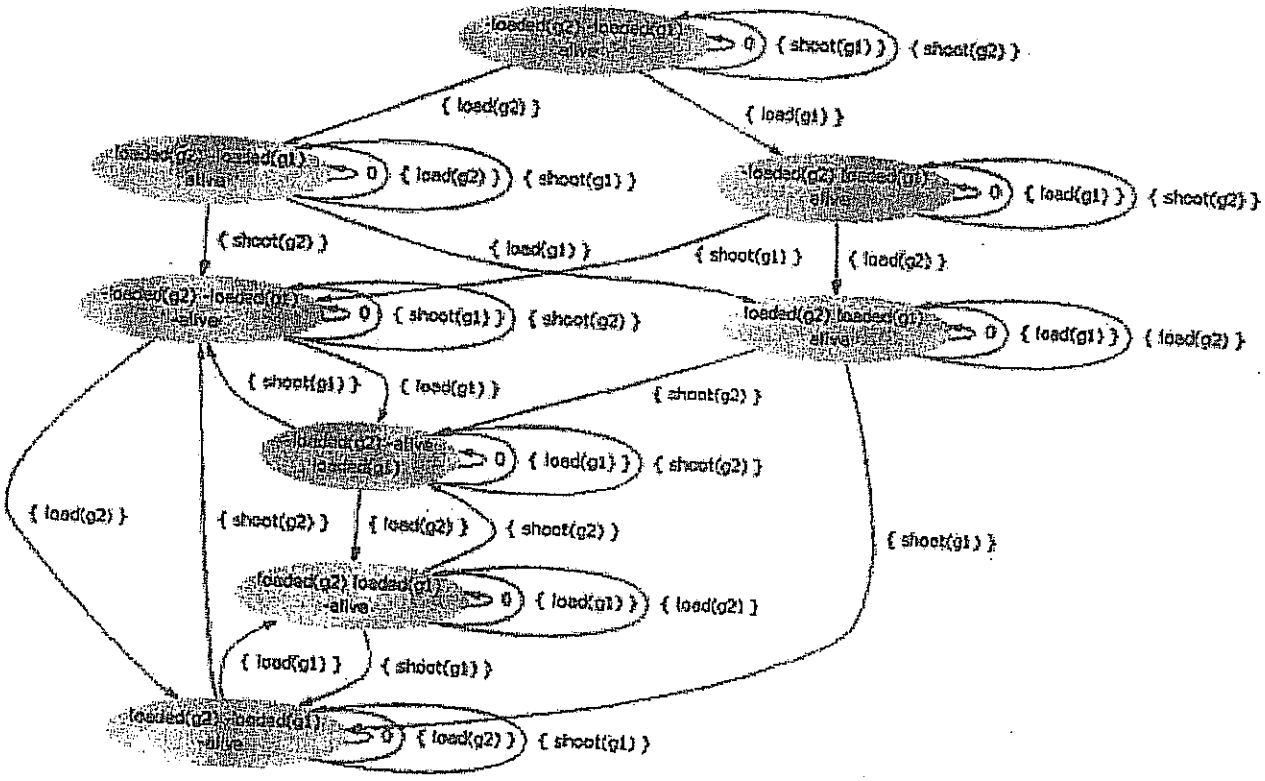
[KN91] Eleftherios Koutsofios and Stephen North. Drawing graphs with dot. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, September 1991.

[KS92] H. Kautz and B. Selman. Planning as satisfaibility. *Proc. of ECAI-92*, pages 359–379, 1992.

[LMRT00] Vladimir Lifschitz, Norman McCain, Emilio Remolina, and Armando Tacchella. Getting to the airport: the oldest planning problem in AI. *Logic-Based Artificial Intelligence*, pages 147–165, 2000.

[McC99] Norman McCain. Using the causal calculator with the $\mathcal{C}$ input language. Technical report, 1999.

[MT97] Norman McCain and Hudson Turner. Causal theories of action and change. *Proc. AAAI–97*, pages 460–465, 1997.



Transition diagram of yale shooting example

Lecture Notes in Artificial Intelligence 2923

Edited by J. G. Carbonell and J. Siekmann

Subseries of Lecture Notes in Computer Science

Vladimir Lifschitz Ilkka Niemelä (Eds.)

Logic Programming and Nonmonotonic Reasoning

7th International Conference, LPNMR 2004
Fort Lauderdale, FL, USA, January 6-8, 2004
Proceedings

Springer

Using Criticalities as a Heuristic for Answer Set Programming

Orkunt Sabuncu[1], Ferda N. Alpaslan[1], and Varol Akman[2]

[1] Department of Computer Engineering
Middle East Technical University,
06531 Ankara, Turkey
{orkunt,alpaslan}@ceng.metu.edu.tr
[2] Department of Computer Engineering
Bilkent University,
06800 Ankara, Turkey
akman@cs.bilkent.edu.tr

Abstract. Answer Set Programming is a new paradigm based on logic programming. The main component of answer set programming is a system that finds the answer sets of logic programs. During the computation of an answer set, systems are faced with choice points where they have to select a literal and assign it a truth value. Generally, systems utilize some heuristics to choose new literals at the choice points. The heuristic used is one of the key factors for the performance of the system.
A new heuristic for answer set programming has been developed. This heuristic is inspired by hierarchical planning. The notion of criticality, which was introduced for generating abstraction hierarchies in hierarchical planning, is used in this heuristic. The resulting system (CSMODELS) uses this new heuristic in a static way. CSMODELS is based on the system SMODELS. The experimental results show that this new heuristic is promising for answer set programming. A comparison of search times with SMODELS demonstrate CSMODELS' usefulness.

1 Introduction

Answer Set Programming is a new programming paradigm. It is based on logic programming but solutions of a problem are not extracted from a proof session [1]. It is a model-theoretic approach. One writes a logic program for the problem at hand in such a way that the intended models of the program correspond to the solutions of the problem. The semantics used for selecting the intended models in answer set programming is *stable model semantics*. The models found by the semantics are called *stable models* or *answer sets* and each of the models correspond to a solution of the problem.

The main component of answer set programming is a system that finds the answer sets. Such systems can be seen as the implementations of stable model semantics.

The main algorithm for finding an answer set is common to almost all systems. The idea is a simple generate-and-test cycle [2]. In the generation phase, a candidate model is constructed for the input logic program. This model is a partial model. The aim is to transform this candidate model into an answer set. The system periodically checks whether the model at hand is an answer set while augmenting it in the test phase.

At *choice points* (see Sect. 3) the system chooses an uncovered atom and assigns it *true* or *false* as an interpretation to augment the candidate model. Some choices cause the system find an answer set very quickly, but some cause it to enter an incorrect search path and consume lots of time before it backtracks. So, heuristics are usually used for choosing an uncovered atom. They greatly affect the performance of the system.

Working on new heuristics is important for developing better systems for answer set programming [2]. In this work, a new system called CSMODELS[1] (Criticality SMODELS), which is based on SMODELS [3], is developed. CSMODELS uses a new heuristic whose foundation is *criticality*. The notion of criticality has been used in hierarchical planning [4].

Hierarchical planning is a way to deal with search space complexity of planning problems. Hierarchical planners attack the problem at different levels of detail. An *abstraction hierarchy* is used to define these levels.

There are successful abstraction hierarchies that improve the performance of the hierarchical planner, but there are also poor ones which cause considerable backtracking between levels. In a good abstraction hierarchy, the upper levels (the most abstract ones) should deal with the hardest part of the planning problem so the planner tries to solve them first. This property will limit the amount of backtracking between the levels during the plan finding process [5]. The notion of criticality was introduced in [4] for automatically generating abstraction hierarchies based on this property. Criticality of a literal in a planning problem is a numerical value and approximates the difficulty of finding a plan that achieves this literal.

There is also the difficulty of finding a literal in an answer set. Selecting the 'hardest' literals at the first choice points can limit backtracking and can help the system find an answer set quickly as in the case of hierarchical planning. This is the main motivation for our work. In CSMODELS, criticalities of the literals of a logic program are calculated to approximate the difficulty of finding them in an answer set. Then, these values are used as a heuristic at the choice points. It is not trivial to calculate the criticalities this time because the original method was for planning problems, not for logic programs. A method for applying criticality calculation for answer set programming is also developed.

The experimental results obtained with CSMODELS are encouraging. Generally, CSMODELS finds an answer set of a program more efficiently than SMODELS. This increase in the performance of search time is significant for especially large problems.

Background information about criticalities and how to calculate them are given in the next section. Section 3 describes the main algorithm of SMODELS. The essential contribution of this work, which is applying the notion of criti-

[1] http://www.ceng.metu.edu.tr/~orkunt/csmodels/ .

V. Lifschitz and I. Niemelä (Eds.): LPNMR 2004, LNAI 2923, pp. 234–246, 2004.

cality to answer set programming, is presented in Section 4. Section 5 describes *CSMODELS*. Section 6 includes the experimental results. In the final section, conclusions can be found.

2 Criticalities

Criticalities are used for generating abstraction hierarchies for planners. Planners use abstraction to reduce the complexity of the planning problem [6,7]. ABSTRIPS, ABTWEAK, ALPINE, and RESISTOR are some hierarchical planners [6].

Hierarchical planners use an abstraction hierarchy for the planning problem to generate and solve subproblems. An *abstraction hierarchy* divides the whole problem into pieces. A hierarchy level indicates which parts of the problem should be neglected or taken into account to form a subproblem corresponding to that level. After finding an abstract plan, the next step is to go one level below and find a plan for that level. The information that is neglected in the higher level is considered now as part of the problem. This is called *refinement*. This process continues level by level until the ground level (i.e., the lowest level) is reached.

Results of using abstraction in planning are generally encouraging. In some problems it can lead to an exponential reduction in the search space, improving the efficiency of plan finding [8].

There are also discouraging results [9,10]. The main cause of inefficiency is *trashing* [5] between the levels of abstraction hierarchy. During refinement, if no plan can be found in a level, backtracking to a more abstract level (the upper one) to search for another abstract plan is inevitable. There may be many possible abstract plans which cannot be refined. This will cause numerous backtrackings, leading to trashing. A good abstraction hierarchy should avoid trashing. To limit trashing, a system should try to solve a subproblem which constitutes the hardest part of the original problem first [5].

Bundy, Giunchiglia, Sebastini, and Walsh [4] provide a method for generating good hierarchies and provide details of an implementation called RESISTOR. RESISTOR sorts the precondition literals of a planning problem according to their difficulties to achieve (i.e., their costs). So, it partitions the problem into parts in terms of difficulty to generate a good abstraction hierarchy.

The method discussed in [4] uses a numerical simulation of the plan finding process. This method has introduced the notion of *criticality*. Criticality captures the cost of a literal; criticality of a literal is an approximation of the cost of achieving it. Criticalities have numerical values; smaller values correspond to easier-to-achieve literals, larger values correspond to harder-to-achieve literals.

The *criticality function* $C(p,n)$ gives the criticality value of literal p. There is an interpretation of $C(p,n)$ as *the difficulty of finding a plan of length* $\leq n$ *achieving* p. This interpretation gives rise to a group of criticality function definitions.[2] Our work is based on RESISTOR's criticality functions. After defining the criticality functions, criticality values are calculated in an iterative way. The iteration is on n starting from 0. So, we basically see $C(p,n)$ as a function that gives the criticality value of literal p at the n-th iteration. Note that the variable n refers to plan length in the interpretation just to find criticality function definitions.

RESISTOR's definition of $C(p,n)$ is inspired by electrical resistors. In calculating the difficulty of achieving p, operators whose effect is p can be regarded as electrical resistors connected in parallel. Fig. 1 shows this circuit. The more operators there are, the more paths there will be for achieving p. Consequently, finding p becomes less difficult. As in equation (1), $C(p,n)$ is calculated by the parallel sum (total resistance of parallel connected resistors) of the criticalities of operators whose effect is p and the initial criticality $C(p,0)$ (the difficulty of finding a plan of length 0).

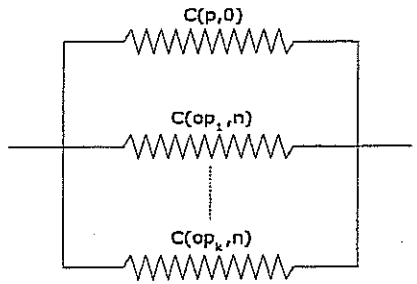


Fig. 1. Criticality circuit for the precondition p

$$\frac{1}{C(p,n)} = \frac{1}{C(p,0)} + \frac{1}{C(op_1,n)} + \ldots + \frac{1}{C(op_k,n)} \quad (1)$$

Equation (1) introduces *the notion of criticality of an operator*. $C(op,n)$ for an operator op is defined to make calculations simple. It is interpreted as the *difficulty of finding a plan of length 1 to n which ends with the occurrence of operator op*. For calculating the criticality of an operator, preconditions of that operator can be regarded as electrical resistors connected in series (Fig. 2).

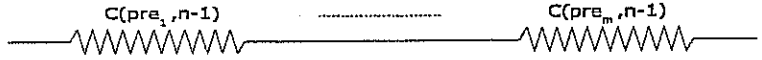


Fig. 2. Criticality circuit for the operator op

$$C(op,n) = C(pre_1,n-1) + \ldots + C(pre_m,n-1) \quad (2)$$

[2] Other definitions and detailed information can be found in [4].

Since the interpretation of $C(op, n)$ expects the occurrence of op at the n-th step, all the preconditions must be achieved up to that point. This is why criticalities of preconditions at the $(n-1)$-th iteration are used in equation (2).

Using equations (1) and (2), criticalities of all the precondition literals are calculated iteratively. Initial criticality values for every literal (i.e., $C(p, 0)$ terms) are assigned to 1. Since criticality functions are monotonically decreasing and convergent [4], the limiting values of criticalities will lie in the interval [0,1] after some iterations.[3] RESISTOR continues to iterate until no change occurs in the values (within some predefined *accuracy*). This is a computationally practical way of terminating iterations.

Abstraction hierarchies are generated by sorting the final criticality values. However, what is important for our work is just how they are calculated.

3 SMODELS: An Answer Set Programming System

SMODELS has been developed at Helsinki University of Technology, and is now one of the most popular answer set programming systems. The system implements stable model semantics [11] for logic programs. Reference [3] explains SMODELS' philosophy and implementation in detail.

Input to SMODELS is variable free logic programs. Programs with variables are transformed to ground logic programs by a front-end system called LPARSE[4] [12]. SMODELS uses a candidate model which is an empty set at the beginning. It tries to augment this candidate model by adding literals deterministically according to the program by using the properties of stable model semantics. This process of generating new literals deterministically is equivalent to *expanding* the model at hand.

Expanding a model can cause situations in which both an atom and its negation are in the model at the same time. This contradicting situation is called a *conflict*.

At an intermediate stage of the search process, an atom can take one of three different possible values [2]: *true*, *false*, or *undefined*. If an atom has a value of undefined, it is an *uncovered* atom.

At the end of first expansion the candidate model corresponds to a *well-founded model* [13] of the program. The aim is to expand the candidate model until it becomes a stable model. The decision criterion for a candidate model to be a stable model is that all the atoms of the program should be covered by it and there should be no conflicts. At several stages during the whole search process there are uncovered atoms and SMODELS cannot deterministically assign values to these atoms (i.e., true or false). These stages are called *choice points*. At choice points, SMODELS should just select one of the uncovered atoms and give an interpretation as true or false. Then, the main algorithm will again try to expand the newly generated candidate model with the addition of the chosen atom. Remember that there can be conflicts after expansion. An incorrect choice at a previous choice point can be the reason for a conflict. So, in case of a conflict SMODELS backtracks to the previous choice. Fig. 3 shows a simplified version of SMODELS' main algorithm [3].

```
function smodels(CM)  {CM: Candidate Model}
  CM := expand(CM)
  if conflict(CM) then
      return false
  else if no atom is undefined in CM then
      return true  {CM is a stable model}
  else
      c := Choose an uncovered atom
      if smodels(CM ∪ {c}) then
          return true
      else
          return smodels(CM ∪ {not c})
      end if
  end if
```

Fig. 3. Main algorithm of SMODELS

SMODELS' heuristic to choose an atom is based on minimizing the remaining search space after a choice is made [3]. SMODELS selects every uncovered literal temporarily and adds it to the candidate model at a choice point. It does so in order to determine what happens if it chooses that uncovered literal actually. After expanding the candidate model, there will possibly be newly derived literals. The number of these new literals is the heuristic score of the chosen literal. If the chosen literal is positive then the score of the literal is called *positive score* of that atom. Similarly, the score of a negative literal of the same atom is *negative score* of that atom.

Given the minimum of positive and negative scores of each atom, SMODELS' heuristic chooses the maximum one to guarantee that it reduces the search space maximally.

The heuristic of SMODELS is a *dynamic heuristic* [2]: at every choice point it recomputes the scores.

4 Applying the Criticality Notion for Answer Set Programming

Criticalities make it possible to find hard-to-achieve literals and to work on them first during hierarchical plan finding. This will reduce the amount backtracking between the levels of abstraction hierarchy.

The same idea can also be applied to answer set programming. During the answer set finding process, choosing the hard-to-find literals at the early choice

[3] Actually, if the initial criticality values are 1, then all the criticalities will be in this interval.

[4] Available at http://www.tcs.hut.fi/Software/smodels/ .

points can reduce the backtracking and permits the system to find an answer set quickly. There are costs of finding literals in an answer set similar to the costs of achieving literals in a planning problem. Knowing the costs of literals of an input logic program can be useful for a system at the choice points. This intuition is the main motivation for the heuristic developed in our work.

Every atom in the stable model has to be *grounded*. If an atom has at least one rule that has generated it, then it is grounded. Rules that have an atom in the head are possible *generators* of it.

The only condition for a rule to generate the literal in its head is that its body must be true. So, if we accept the rule's body as a *precondition* we can transform a rule into a *planning operator (action)*.[5] The *effect* of this planning operator is the head atom. Let all the generator rules for literal a below constitute a portion of a logic program.

$$a \leftarrow b,\ not\ c \quad (rule_1) \tag{3}$$

$$a \leftarrow d \quad (rule_2) \tag{4}$$

The corresponding planning operator for rule (3) is given below.

$$a \leftarrow^{r_1} b,\ not\ c \quad \Longrightarrow \quad \begin{array}{l} \text{Operator : } r_1 \\ \text{Preconditions : } b,\ not\ c \\ \text{Effect : } a \end{array}$$

When we consider the above rules as planning actions, equations (5–7) are used for calculating the criticality of literal a. Note that the initial criticality of a, $C(a,0)$, is set to 1 like all the other literals' initial criticalities.

$$C(rule_1,n) = C(b,n-1) + C(not\ c,n-1) \tag{5}$$

$$C(rule_2,n) = C(d,n-1) \tag{6}$$

$$\frac{1}{C(a,n)} = \frac{1}{C(a,0)} + \frac{1}{C(rule_1,n)} + \frac{1}{C(rule_2,n)} \tag{7}$$

There is a term representing criticality value of a negative literal in equation (5). How can the criticality of a negative literal be found? There are no rules that can generate a negative literal in the sense of generating positive ones. In fact *negation-as-failure* means that if it is not possible to achieve an atom a, then *not* a is assumed to be true. Considering the properties of stable model semantics, we can define the necessary conditions for *not* a to be in the stable model as no generator rule will have a chance to generate a. The bodies of all generator rules must be interpreted as false, so literal *not* a is interpreted as true.

[5] Lin and Reiter [14] define logic rules as planning actions in the context of defining semantics for logic programs using situation calculus.

Here is the only rule generating *not* a formed by taking all the generator rules (3–4) of a into consideration:[6]

$$not\ a \leftarrow (not\ b \vee c),\ not\ d \quad (rule_{not\ a}) \tag{8}$$

Based on rule (8), the criticality of literal *not* a is calculated using the following equations:

$$C(rule_{not\ a},n) = C((not\ b \vee c),n-1) + C(not\ d,n-1) \tag{9}$$

$$\frac{1}{C(not\ a,n)} = \frac{1}{C(not\ a,0)} + \frac{1}{C(rule_{not\ a},n)} \tag{10}$$

Calculating the criticality value of a negative literal is no different than calculating a positive literal except for compound literals with disjunction. Achieving a compound literal as a whole should be easier than achieving each one of the basic literals individually. So, the criticality value of a compound literal should be less than each of the criticalities of its basic literals. Calculating the criticality of compound literal ($notb \vee c$) by the equation $C((notb \vee c),n) = C(notb,n) \times C(c,n)$ is a meaningful approximation, since the multiplication of two or more real numbers in the interval [0,1] is always smaller than each number (or equal to the smallest one at least).

A *constraint rule* is a rule that has no atom in the head (i.e., a rule that has an implicit false in the head). They are not used in criticality calculations, since they cannot be generators for any atom.

Using the above equations, we can calculate the criticalities of all the literals of a logic program if all the rules in the program are normal. But SMODELS enlarges the syntax and semantics of logic programs by extended rules [3]. These extended rules are *choice, cardinality,* and *weight* rules. Our system, CSMODELS, supports choice rules, but not cardinality or weight rules.

$$\{a\} \leftarrow b,\ not\ c \tag{11}$$

Rule (11) is a choice rule. A choice rule's head may not be true although its body is satisfied by the model unlike normal rules. One can rewrite the choice rule using only normal rules. But this translation introduces new atoms to the input program [3]. Choice rule (11) can be rewritten as two rules:

$$\begin{array}{l} a \leftarrow not\ c\_a,\ b,\ not\ c \\ c\_a \leftarrow not\ a,\ b,\ not\ c \end{array} \tag{12}$$

The atom c_a is a newly introduced atom. By treating choice rules as if two normal rules of the form (12), we can handle choice rules for criticality calculations. In this way we do not enlarge the program. Answer sets in the output do not contain the introduced atoms, since they are only used for criticality calculations.

[6] Body of the rule (8) is the inverse of the completion of literal a [15]. We can see the inverse of the completion of literal x as a generator rule for literal *not* x.

5 CSMODELS (Criticality SMODELS)

CSMODELS uses criticalities as a heuristic and is based on the SMODELS system. It implements the method of applying criticality calculation to logic programs of SMODELS described in Sect. 4. Implementation details of CSMODELS can be found in [16]. The main algorithm is the same as that of SMODELS except the parts related to the choice points.

CSMODELS calculates the criticality values at the first choice point. Unlike SMODELS, CSMODELS' heuristic is not dynamic. So, the same heuristic scores calculated at the first choice point are used at all the other choice points. At the first choice point, the candidate model at hand corresponds to a well-founded model (WFM) of the program. WFM covers some of the atoms of the input program, so there is no need to calculate the criticality values of the already covered atoms. We set the criticality of a literal in WFM to 0, indicating that it is very easy to achieve that literal or the literal is already achieved. Also, we set the criticality of the inverse of that literal to 1 indicating the impossibility or difficulty of achieving the literal.[7]

$$\text{If literal } l \in \text{WFM} \implies \begin{array}{l} C(l) = 0 \\ C(not\ l) = 1 \end{array}$$

After calculating criticality values of all uncovered literals, CSMODELS finds the *criticality heuristic scores* to select a literal at the choice points. It temporarily selects every uncovered literal one by one and adds them to the candidate model just like SMODELS computes heuristic scores. Let x be an uncovered literal by the WFM, the expansion of the model formed by adding x to WFM will probably generate new literals that have been uncovered previously. Let these literals be

$$NewLiterals(x) = \{a, b, \ldots, not\ k, not\ l, \ldots\}. \tag{13}$$

CSMODELS finds the criticality heuristic score of literal x by the following equation:

$$\begin{array}{rl} Score(x) = & C(a) + (1 - C(not\ a)) + \\ & C(b) + (1 - C(not\ b)) + \ldots + \\ & C(not\ k) + (1 - C(k)) + \\ & C(not\ l) + (1 - C(l)) + \ldots \end{array} \tag{14}$$

The purpose of the score of a literal is to approximate the *difficulty of choosing that literal at a choice point*. Knowing that choosing x generates literal a, the system faces the cost of achieving a and avoiding the generation of *not* a when choosing x. That is why the terms $C(a)$ and $1 - C(not\ a)$ are added to the score. This is done for all the literals in $NewLiterals(x)$. Just like SMODELS, there are positive and negative criticality heuristic scores of an atom.

[7] Note that 1 is the maximum criticality that a literal can have.

The set $NewLiterals$ for an uncovered literal can be different at different choice points. Thus, the same literal can have different criticality heuristic scores at different choice points. However, if we calculate the criticality values and the criticality heuristic scores at every choice point, the proportion of the time used by these calculations to total search time will be high. Because of these costly calculations, CSMODELS uses a static heuristic.

In order to limit the amount of backtracking, CSMODELS selects hard-to-choose literals first. So, the system sorts the uncovered atoms according to their heuristic scores. The *sorting criterion* affects the performance of the heuristic. The main sorting criterion of CSMODELS, which is called *maxmin* sorting criterion, is the same as that of SMODELS. Atoms are sorted according to minimum of their positive and negative criticality heuristic scores in descending order. Sorting atoms according to the sum of positive and negative heuristic scores in descending order is another criterion named *maxsum*. Experiments showed that using maxsum helped CSMODELS to find an answer set in time less than using maxmin for problems having many answer sets.

Fortunately, using maxsum in problems that do not have many answer sets usually leads to conflict at the first choice point. CSMODELS first uses the maxsum criterion. If it ends up with an immediate conflict, it switches back to maxmin. This causes CSMODELS to behave identically for all problems.

6 Experimental Results

CSMODELS has been tested to find out its performance. Test problems are taken from common application domains of answer set programming. We compare results of CSMODELS with those of SMODELS. Since both systems take the same ground logic program for the experiments, grounding time is the same for both. The grounding times are not included in the search time results of the experiments.

The main measure for the tests is the duration which states how long the search for an answer set took in CPU seconds. Another measure for comparison is the number of choice points. This number shows how many times a system used its heuristic to find an answer. The same ratio between the number of choice points may not be observed for the search times in comparisons of CSMODELS and SMODELS since there is not a direct correlation. However, the reduction in the number of choice points generally leads to performance gains in terms of search times.

Tests for planning problems have been performed on a 200 MHz Pentium computer with 256 MB of main memory running Linux 2.4.5; for colorability and n-queens problems a 733 MHz Pentium III computer with 256 MB of main memory running Linux 2.2.19 has been used.

Several well-known planning problems are tested. These are Towers of Hanoi, simple robot-box domain, and blocks-world planning. In addition to planning problems, several tests of colorability and n-queens problems have been performed. If the original logic programs for these problems have cardinality rules,

they are rewritten using normal rules without affecting the answer sets. Tables 1, 2 and 3 report the results of CSMODELS and SMODELS for all the problem instances. The results for choice points are shown in parentheses in tables.

Towers of Hanoi problem with 4 disks has been tested. It can be solved in 15 steps optimally. The logic program representing the simple robot–box domain is based on the reference [6].[8] The optimal solution has 13 steps. For the blocks-world planning problem, two different domain representations have been tested. One representation is taken from the reference [17] where concurrency is allowed. Three different problem instances with 15, 17, and 19 blocks from the reference [18] are tested (rows BW_2 (15), BW_2 (17) and BW_2 (19)).[9] They have optimal plans with 8, 9 and 10 steps, respectively. Another representation is the one used in reference [19]. A problem instance with 11 blocks from the same reference has been used (row BW_1 (11)).[10] It has an optimal solution with 9 steps. Unlike the first representation, this one does not allow concurrent moves.

Table 1. A comparison of search time (CPU seconds) and number of choice points for planning problems

	SMODELS	CSMODELS
Hanoi	4.630 (12)	5.540 (8)
Robot–Box	39.940 (11)	40.800 (8)
BW_1 (11)	73.400 (7)	57.040 (5)
BW_2 (15)	38.500 (29)	37.390 (7)
BW_2 (17)	77.330 (27)	61.550 (10)
BW_2 (19)	174.660 (4111)	90.620 (6)

The results show that for small size problems like Towers of Hanoi and simple robot-box domains, CSMODELS' performance is almost the same as SMODELS' performance. When the sizes of the problems become larger, criticality heuristic of CSMODELS helps the system solve the problems more efficiently. This claim is supported especially by the problem instances of the blocks-world representation BW_2. By increasing the number of blocks from 15 to 19, the performance difference between CSMODELS and SMODELS becomes obvious. Also the reduction in the number of choice points is consistent with the performance gains.

Other than the planning domain, colorability (4-colorability problem instances are used in experiments) and n-queens problems have been used in testing. The logic programs for these problems are based on I. Niemela's representations.[11]

[8] This domain is about controlling a robot to move boxes between rooms.

[9] They correspond to I. Niemela's bw-large.c, bw-large.d and bw-large.e problems, respectively [18].

[10] This problem instance corresponds to E. Erdem's P4 [19].

[11] The logic program for n-queens problem is actually from [17]. The one for colorability problem is adapted from [18] by rewriting choice rules.

Table 2. A comparison of search time (CPU seconds) and number of choice points for colorability problem

	p100	*p300*	*p600*	*p1000*	*p3000*
SMODELS	0.530 (32)	3.530 (89)	12.380 (177)	35.210 (310)	305.060 (860)
CSMODELS	0.540 (38)	3.050 (79)	10.080 (162)	26.070 (235)	230.370 (787)

Table 3. A comparison of search time (CPU seconds) and number of choice points for n-queens problem

	8×8	*18×18*	*20×20*	*22×22*
SMODELS	0.020 (3)	1.830 (302)	9.360 (1483)	117.730 (16156)
CSMODELS	0.030 (7)	1.530 (152)	5.490 (571)	31.170 (2401)

Several tests have been carried out ranging from small size problem instances to large sized ones for colorability and n-queens. Results show that CSMODELS is more efficient than SMODELS for colorability and n-queens problems. For larger sized problem instances, the performance gains obtained by CSMODELS are more significant.

7 Conclusion

Answer set programming systems utilize some heuristics to compute answer sets. Heuristics affect the search path that the system explores within the entire search space. As some paths lead to efficient solutions and some not, heuristics are one of the key ingredients influencing the performance of a system.

In our work, a new heuristic for answer set programming has been developed. The resulting system called CSMODELS uses this new heuristic. The main insight of the heuristic comes from hierarchical planning. The notion of criticality, which is introduced for generating abstraction hierarchies, is applied for answer set programming in the new heuristic. The experimental results indicate that CSMODELS outperforms SMODELS in terms of efficiency. The performance difference becomes more significant when the problem size gets larger. Generally, the results show that this new heuristic is promising for answer set programming.

The choice rules from the extended rules of SMODELS' new versions are supported by CSMODELS. However, cardinality and weight rules are not supported. Rewriting cardinality and weight rules using normal rules lead to an exponential growth in the number of rules. Handling cardinality and weight rules in CSMODELS is a topic for future work.

Heuristics for answer set programming are similar to the heuristics developed for satisfiability solvers. Although there is a substantial amount of work done for the latter, there is not much for the former [2]. New heuristics will help systems to be more efficient and make answer set programming more applicable.

References

1. V. Lifschitz. Answer set planning. In *International Conference on Logic Programming*, pages 23–37, 1999.
2. W. Faber, N. Leone, and G. Pfeifer. A comparison of heuristics for answer set programming. In *Proc. of the 5th Dutch-German Workshop on Nonmonotonic Reasoning Techniques and their Applications (DGNMR 2001)*, pages 64–75, 2001.
3. P. Simons. Extending and implementing the stable model semantics. Research Report 58, Helsinki University of Technology, Helsinki, Finland, 2000.
4. A. Bundy, F. Giunchiglia, R. Sebastiani, and T. Walsh. Calculating criticalities. *Artificial Intelligence*, 88(1–2):39–67, 1996.
5. F. Giunchiglia. Using ABSTRIPS abstractions – where do we stand? Technical Report 9607–10, IRST, Trento, Italy, 1996.
6. C. A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68(2):243–302, 1994.
7. F. Giunchiglia, A. Villafiorita, and T. Walsh. Theories of abstraction. *AI Communications*, 10(3–4):167–176, 1997.
8. C. A. Knoblock. Abstracting the Tower of Hanoi. In *Working Notes of AAAI-90 Workshop on Automatic Generation of Approximations and Abstractions*, pages 13–23, 1990.
9. D. E. Smith and M. A. Peot. A critical look at Knoblock's hierarchy mechanism. In *Proc. of 1st International conference Artificial Intelligence Planning Systems (AIPS-92)*, pages 307–308, 1992.
10. C. Backstrom and P. Jonsson. Planning with abstraction hierarchies can be exponentially less efficient. In *Proc. of the 14th International Joint Conference on Artificial Intelligence*, pages 1599–1604, 1995.
11. M. Gelfond and V. Lifschitz. The stable semantics for logic programs. In R. Kowalski and K. Bowen, editors, *Proceedings of the 5th International Conference on Logic Programming*, pages 1070–1080. The MIT Press, 1988.
12. T. Syrjanen. LPARSE 1.0 user's manual. Available at http://www.tcs.hut.fi/Software/smodels/lparse.ps.
13. A. van Gelder, K. Ross, and J. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
14. F. Lin and R. Reiter. Rules as actions: A situation calculus semantics for logic programs. *Journal of Logic Programming*, 31(1–3):299–330, 1997.
15. K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, 1978.
16. O. Sabuncu. Using criticalities as a heuristic for answer set programming. MS Thesis, Middle East Technical University, Department of Computer Engineering, Ankara, Turkey, 2002.
17. I. Niemela and M. Trunszczynski. Answer-set programming: a declarative knowledge representation paradigm. In Lecture notes of ESSLLI 2001 Summer School, 2001.
18. I. Niemela. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3–4):241–273, 1999.
19. E. Erdem. Applications of logic programming to planning: Computational experiments. Unpublished draft, http://www.cs.utexas.edu/users/esra/papers.html, 1999.

Planning with Preferences Using Logic Programming*

Tran Cao Son and Enrico Pontelli

Department of Computer Science
New Mexico State University
Las Cruces, NM 88003, USA
{tson,epontell}@cs.nmsu.edu

Abstract. We present a declarative language, $\mathcal{PP}$, for the specification of preferences between possible solutions (or trajectories) of a planning problem. This novel language allows users to elegantly express non-trivial, multi-dimensional preferences and priorities over them. The semantics of $\mathcal{PP}$ allows the identification of *most preferred trajectories* for a given goal. We also provide an answer set programming implementation of planning problems with $\mathcal{PP}$ preferences.

1 Introduction

Planning—in its classical sense—is the problem of finding a sequence of actions that achieves a predefined goal. Most of the research in AI planning has been focused on methodologies and issues related to the development of efficient planners. To date, several efficient planning systems have been developed (e.g., see [18]). These developments can be attributed to the discovery of good domain-independent heuristics, the use of domain-specific knowledge, and the development of efficient data structures used in the implementation of the planning algorithms. Logic programming has played a significant role in this line of research, providing a declarative framework for the encoding of different forms of knowledge and its effective use during the planning process [24].

However, relatively limited effort has been placed on addressing several important aspects in real-world planning domains, such as *plan quality* and *preferences about plans*. In many real world frameworks, the space of feasible plans to achieve the goal is dense, but many of such plans, even if executable, may present undesirable features. In these frameworks, it may be simple to find a solution; rather, the challenge is to produce a solution that is considered satisfactory w.r.t. the needs and preferences of the user. Thus, feasible plans may have a measure of quality, and only a subset may be considered acceptable. These issues can be illustrated with the following example:

Example 1. It is 7 am and Bob, *a Ph.D. student, is at home. He needs to be at school at 8 am. He can take a bus, a train, or a taxi to go to school, which will take him 55, 45, or 15 minutes respectively. Taking the bus or the train will require* Bob *to walk to the nearby station, which may take 20 minutes. However, a taxi can arrive in only 5 minutes. When in need of a taxi,* Bob *can call either the* MakeIt50 *or the* PayByMeter *taxi company.* MakeIt50 *will charge a flat rate of $50 for any trip, while* PayByMeter *has a fee schedule of $20 for the trip to school. If he takes the bus or the train, then* Bob *will spend only $2.*

* The research has been partially supported by NSF grants EIA0220590, EIA0130887, CCR9875279, CCR9820852, and CCR9900320.

V. Lifschitz and I. Niemelä (Eds.): LPNMR 2004, LNAI 2923, pp. 247–260, 2004.

USING CRITICALITIES AS A HEURISTIC FOR ANSWER SET PROGRAMMING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

ORKUNT SABUNCU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF COMPUTER ENGINEERING

SEPTEMBER 2002

Approval of the Graduate School of Natural and Applied Sciences.

Prof. Dr. Tayfur Öztürk
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Ayşe Kiper
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Ferda Nur Alpaslan
Supervisor

Examining Committee Members

Prof. Dr. Varol Akman

Assoc. Prof. Dr. Ferda Nur Alpaslan

Assoc. Prof. Dr. İsmail Hakkı Toroslu

Assist. Prof. Dr. Bilge Say

Dr. Ayşenur Birtürk

ÖZ

KRİTİKLERİN YANIT KÜMESİ PROGRAMLAMADA BULUŞSAL YÖNTEM OLARAK KULLANILMASI

Sabuncu, Orkunt

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. Ferda Nur Alpaslan

Eylül 2002, 55 sayfa

Yanıt kümesi programlama temeli mantık programlama olan yeni bir yaklaşımdır. Ana bileşeni mantık programlarının yanıt kümelerini bulan dizgedir. Programın yanıt kümelerine karşılık gelen durağan modeller (stable models) durağan model anlambilimi kullanılarak bulunur. Sistemler yeni harfler (literals) seçebilmek için genelde bazı buluşsal yöntemler (heuristics) kullanırlar ve seçilen harfleri aday modele (candidate model) yanıt kümesi bulmak için eklerler. Kullanılan buluşsal yöntemler sistemin performansı açısından önemli bir unsurdur.

Bu çalışmada yanıt kümesi programlama sistemleri için yeni bir buluşsal yöntem geliştirilmiştir. Bu yöntemin ana fikri için hiyerarşik planlamadan esinlenilmiştir. Soyutlama hiyerarşilerinin oluşturulmasında kullanılan kritik kavramı (criticality) geliştirilen buluşsal yöntemin temelini oluşturur. SMODELS sistemi üzerinde geliştirdiğimiz CSMODELS bu yeni yöntemi kullanır. Deneysel sonuçlar yeni buluşsal yöntemin ümit verici olduğunu gösteriyor. Genelde CSMODELS'ın yanıt kümesi bulması SMODELS'dan daha az zaman alıyor.

Anahtar Kelimeler: yanıt kümesi programlama, mantık programlama, SMODELS, durağan model anlambilimi, kritikler, soyutlama hiyerarşisi

USING STABLE MODEL SEMANTICS (SMODELS) IN THE CAUSAL CALCULATOR (CCALC)

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

SEMRA DOĞANDAĞ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF COMPUTER ENGINEERING

JANUARY 2002

Approval of the Graduate School of Natural and Applied Sciences.

Prof. Dr. Tayfur Öztürk
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Ayşe Kiper
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Ferda Nur Alpaslan
Supervisor

Examining Committee Members

Prof. Dr. Varol Akman

Prof. Dr. Adnan Yazıcı

Assoc. Prof. Dr. Ferda Nur Alpaslan

Assis. Prof. Dr. Ahmet Coşar

Dr. Ayşenur Birtürk

ÖZ

CAUSAL CALCULATOR'DA (CCALC) DURAĞAN MODEL ANLAMBİLİMİNİNİN (SMODELS) KULLANIMI

Doğandağ, Semra

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Assoc. Prof. Dr. Ferda Nur Alpaslan

Ocak 2002, 63 sayfa

Eylem dilleri eylemler ve bunların durum değişkenleri üzerindeki etkilerini konu alan biçimsel yöntemlerdir. Planlamada yeni bir yaklaşım olarak problemlerin alanı eylem dilleri kullanılarak tanımlanır. Bu çalışmanın amacı eylem dili C ile tanımlanmış olan bir dizgeyi önce nedensel teoriye çevirmek, daha sonra da karşılık gelen mantık programını bularak planlama yapmaktır. Bu durumda planlama problemi mantık programının yanıt setini bulmaya indirgenmiş olacaktır. Bu dizge nedensel teoriler için model bulan Causal Calculator (CCALC)'a eklenmiştir.

Anahtar Kelimeler: eylem dilleri, durağan model anlambilimi, mantıksal programlama

WORKFLOW PROCESS DEFINITION TOOL

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

HANDAN SEYREK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE
OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF COMPUTER ENGINEERING

JANUARY 2002

Approval of the Graduate School of Natural and Applied Sciences.

Prof. Dr. Tayfur Öztürk
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Ayşe Kiper
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Ferda Alpaslan
Supervisor

Examining Committee Members

Prof. Dr. Adnan Yazıcı (Chairman)

Prof. Dr. Varol Akman

Assoc. Prof. Dr. Ferda Alpaslan

Asst. Prof. Dr. Ahmet Coşar

Dr. Ayşenur Birtürk

ÖZ

İŞAKIŞI İŞLEMİ TANIMLAMA ARACI

Handan Seyrek

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. Ferda Alpaslan

Ocak 2002, 57 sayfa

Bu tez, kullanıcı arayüzü aracılığıyla işlem spesifikasyonlarını alarak tanım diline çeviren bir işakışı işlemi tanımlama aracı geliştirilmesi ile ilgilidir. İşakışı Yönetimi Koalisyonu (WFMC), bir işakışındaki aktiviteler arasında olabilecek bütün geçişleri tanımlamıştır. Bu çalışmada, bütün geçişler nedensel açıklama teorisi tarafından ortaya koyulan durum geçiş diyagramlarıyla gösterilmiştir. Araç, işakışı işlemlerini geçiş sistemleriyle modellerler ve tanımlarını hem XML (Genişletilebilir İşaretleme Dili) hem de WODEL (İşakışı Tanımlama Dili) formatında üretir.

Anahtar kelimeler: İşakışı, Aksiyon dilleri, Nedensel teori, XML

VISUALIZING TRANSITION DIAGRAMS OF ACTION LANGUAGE PROGRAMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÖZCAN KOÇ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF COMPUTER ENGINEERING

JULY 2001

Approval of the Graduate School of Natural and Applied Sciences.

Prof. Dr. Tayfur Özturk
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Ayşe Kiper
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Ferda N.
Alpaslan
Supervisor

Examining Committee Members

Assoc. Prof. Ferda N. Alpaslan	
Assoc. Prof. Nihan K. Çiçekli	
Prof. Dr. Varol Akman	
Prof. Dr. Adnan Yazıcı	
Selim Erdogan, M.S.	Selim Tevhan Erdoğan

ÖZ

EYLEM DİLLERİNDE YAZILMIŞ PROGRAMLARIN GEÇİŞ DİYAGRAMLARININ GÖRSEL OLARAK ÇİZİLMESİ

Koç, Özcan

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Assoc. Prof. Ferda N. Alpaslan

Temmuz 2001, 106 sayfa

Eylem dilleri günümüz yapay zeka (YZ) çalışmalarında önemli bir yer tutmaktadır. Eylem dilleri programları ve nedensel kuramlar oluşturmak kadar bu dilleri öğrenme ve öğretmede karşılaşılan en temel sorunlardan biri geçiş diyagramlarının elle çizilmesi yada zihinde canlandırılmaya çalışılmasıdır. Bu tezde, geçiş diyagramlarını otomatik olarak ve görsel bir biçimde çizecek, TDV adında bir program geliştirildi. TDV, CCALC [18] programını geliştirerek `GraphViz`[21] yazılımını kullanmakta ve *C* programlarının geçiş diyagramlarını çizmektedir.

Anahtar Kelimeler: Eylem Dilleri, nedensel kuramlar, planlama, bilgi gösterimi, geçiş diyagramları, Görsel Çizimler, *C*, CCALC, TDV, GRAPHVIZ.

TRANSLATION OF WORKFLOW SPECIFICATIONS TO THE ACTION DESCRIPTION LANGUAGE C

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

OF

MIDDLE EAST TECHNICAL UNIVERSITY

BY

EMEL HASDAL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF COMPUTER ENGINEERING

JULY 2001

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Tayfur Öztürk
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Ayşe Kiper
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Ferda
Nur Alpaslan
Supervisor

Examining Committee Members

Assoc. Prof. Ferda N. Alpaslan

Assoc. Prof. Nihan K. Çiçekli

Prof. Dr. Varol Akman

Prof. Dr. Adnan Yazıcı

Selim Erdoğan,M.S.

ÖZ

İŞ AKIŞI TANIMLARININ C EYLEM TANIMLAMA DİLİNE ÇEVİRİLMESİ

Hasdal, Emel

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Assoc. Prof. Dr. Ferda Nur Alpaslan

Temmuz 2001, 72 Sayfa

İş akışı yönetimi sistemleri kurumların otomasyon işlemleri için ümit vaad eden bir çözümdür. İş akışını modellemek ve çalıştırmak için pek çok ticari ürün mevcuttur. Ancak, iş akışı yönetim sistemlerinin kuramsal temelleri eksiktir. Diğer bir yandan, eylem dilleri, eylem sistemlerini tanımlama problemine en son yaklaşımlardan biridir. *C* dili, nedenselliğin açıklanması kuramını temel alan, bir yüksek düzey eylem tanımlama dilidir.

İş akışı süreçlerinin biçimselleştirilmesi için yapılan pek çok öneriden biri eylem tanımlama dili *C*'nin kullanılmasıdır. Bu tezde, iş akışı süreçlerinin eylem tanımlama dili *C* kullanılarak biçimselleştirilmesinde kullanılması amacı ile bir yüksek düzey iş akışı tanımlama dili olan WODEL tasarlanmıştır ve WODEL dilinden *C* diline

bir çevirim tanımlanmıştır. Ayrıca, çevirim işinin otomatik olarak yapılabilmesini sağlamak için bir çevirim aracı geliştirilmiştir.

Bu biçimselleştirme, iş akışı süreç tanımları üzerinden akıl yürütme işlemlerinin yapılabilmesine olanak sağlayan bir zemin oluşturmaktadır. Ayrıca, önerilen biçimselleştirme geliştirilerek iş akışı süreçlerinin doğrulanması ve en iyilenmesi amaçları için kullanılabilir.

Anahtar Kelimeler: İş akışı süreçleri, eylem dilleri, planlama.

PROJE ÖZET BİLGİ FORMU

Proje Kodu : 100E024

Proje Başlığı : Realistic Applications of Action Languages for Workflow Management

Proje Yürütücüsü ve Yardımcı Araştırıcılar :
Doç.Dr.Ferda Nur Alpaslan

Projenin Yürütüldüğü Kuruluş ve Adresi :
ODTÜ Bilgisayar Mühendisliği Bölümü

Destekleyen Kuruluş(ların) Adı ve Adresi :
TÜBİTAK (Türkiye) ve NSF(ABD)

Projenin Başlangıç ve Bitiş Tarihleri : 1/5/2001-1/5/2003

Öz : (en çok 70 kelime)

Projenin amacı, University of Texas-Austin (UT-Austin) tarafından geliştirilen C eylem dilini kullanarak gerçek hayat problemlerine çözüm bulmaktır. Bu amaçla iş akışlarını C dilinde programlara çeviren ve bu programları çalıştırarark olası modeller bulan bir sistem geliştirilmiştir. Ayrıca C dilindeki programların çalışmasını görsel olarak izleyebilmek için, programların geçiş şemalarını üreten bir araç ta geliştirilmiştir. Bunlardan elde edilen deneyimle başka gerçek hayat problemleri üzerinde çalışılmış, bu arada ortaya çıkan sorunların çözümü için projenin başlangıç amaçları içinde yer almayan bir takım ek çalışmaları yapmak ta gerkmiştir. Bunlar proje raporunda detaylı olarak yer almaktadır.

Anahtar Kelimeler: Eylem dilleri, iş akışı, model bulucular

Projeden Kaynaklanan Yayınlar :

- Özcan Koç, Visualizing transition diagrams of action language programs, Y. Lisans Tezi, ODTÜ, Temmuz 2001.
- Emel Hasdal, Translation of workflow specifications to the action description language C, Y. Lisans Tezi, ODTÜ, Temmuz 2001.
- Semra Doğandağ , Using Stable Model Semantics (SMODELS) in the Causal calculator (CCALC), Y. Lisans Tezi, ODTÜ, Ocak 2002.
- Handan (Seyrek) İnalpolat, Workflow Process definition tool, Y. Lisans Tezi, ODTÜ, Ocak 2002.

- Orkunt Sabuncu, Using criticalities as a heuristic for answer set programming, Y. Lisans Tezi, ODTÜ, Eylül 2002.
- Semra Doğandağ, Ferda N. Alpaslan, Varol Akman. Using Stable Model Semantics (SMODELS) in the Causal Calculator (CCALC). Proc. TAINN 2001, Haziran 21-22, 2001, Gazimagusa, Kıbrıs, pp. 312-321.
- Özcan Koç, Ferda N. Alpaslan, Nihan K. Çiçekli. Visualizing Transition Diagrams of Action Language Programs. Proc. ISCIS XVII, 28-30 Ekim 2002, Orlando, Florida, USA, pp. 181-186.
- Sabuncu, O., F.N. Alpaslan, V.Akman. "Using Criticalities as a Heuristic for Answer Set Programming". Proc. 7th . International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-7), Fort Lauderdale, Fl, USA, January 6-8, 2004.

Bilim Dalı : Bilgisayar Mühendisliği
Doçentlik B. Dalı Kodu :