itüdergisi***/d***
mühendislik
Cilt:4, Sayı:1, 103-111
Şubat 2005

Prolog'un paralel mantık programlamaya genişletilmesi

Nazım KOÇ*, Şakir KOCABAŞ

İTÜ Uçak ve Uzay Bilimleri Fakültesi, Uzay Mühendisliği Bölümü, 34469, Ayazağa, İstanbul

Özet

Bu çalışmamızda, CCND dil ailesi içinde bulunan, yeni bir paralel mantık programlama dili öneriyoruz. Bu paralel mantık programlama dili Berk Prolog ve MIMD mimarisindeki uygulaması da Berk Sistem olarak adlandırılmıştır. Berk Prolog, söz-dizimi ve semantik olarak klasik dillerden ayrılmaktadır: Senkronizasyon mekanizması farklıdır. Bir proses baş veya guard tarafından askıya alınamaz. Guard kısmı, cümle seçiminden çok esas hesaplama kısmını oluşturur. Şartlı olarak değişken talep etme ve körleme bekleme kavramları ortaya atılmıştır. Guard çağrıları sayesinde sıralı Prolog, paralel Prolog'un bir alt kümesi haline getirilmiştir. Kararlı bir sistem olduğu için, if-then-else yapıları ve bu yapılarla ilgili temel fonksiyonlar gerçekleştirilebilmiştir.
***Anahtar Kelimeler**: Prolog, paralel Prolog, CCND dili, paralel mantık programlama.*

An extension of prolog to parallel logic programming

Abstract

In our work we introduce a new parallel logic programming language in the class of CCND programming languages. We named our parallel logic programming language as "Berk Prolog" and its MIMD implementation as the "Berk System". Berk Prolog differs from other parallel logic programming langauges developed until today in many respects, primarily in its syntax and semantics. Berk Prolog also differs from other related languages in its synchronization mechanism, in wich a process cannot be suspended by head unification or guard evaluation. A process can be invoked if all the input variables are bound. Guard evaluation is the main subject of computation rather than clause selection. Berk Prolog introduces the concept of conditional demand of external variables. It also brings flexibility on clause mapping and distribution of processes. By its guard evaluation, Berk Prolog treats sequential Prolog as its subset. In Berk Prolog, if-then-else and related predicates can be implemented quite easily. The Berk System, developed as an implementation of Berk Prolog, is a model that works on TCP/IP network medium. In this model we introduce the concepts of solution network and network heap. Some familar benchmark programs have been written in Berk Prolog and executed in the Berk System successfully.
***Keywords**: Prolog, parallel prolog, CCND language, parallel logic programming.*

---

*Yazışmaların yapılacağı yazar: Nazım KOÇ. nazim@aractakip.biz; Tel: (216) 395 68 55.
Bu makale, birinci yazar tarafından İTÜ Uçak ve Uzay Bilimleri Fakültesi'nde tamamlanmış "Prolog'un paralel mantık programlamaya genişletilmesi" adlı doktora tezinden hazırlanmıştır. Makale metni 28.07.2003 tarihinde dergiye ulaşmış, 15.12.2003 tarihinde basım kararı alınmıştır. Makale ile ilgili tartışmalar 30.06.2005 tarihine kadar dergiye gönderilmelidir.

Giriş

Prolog programlama, mantık programlama dilleri içinde bilinen en klasik bilgisayar programlama dildir. Prolog'u paralelleştirme çalışmaları (Pollard, 1981) Prolog'un ortaya çıkışı ile başlamıştır. Japonların Beşinci Kuşak Bilgisayar Projesi sayesinde paralel Prolog en kuvvetli, en görkemli günlerini 1982-1992 yılları arasında yaşamıştır. Beşinci Kuşak bilgisayar projesinde paralel işleme ile zeki bir sistem geliştirmek için paralel mantık programlama bir araç olarak seçilmiştir.

Japonların beşinci kuşak bilgisayar projesinin tamamlanması ile CCND dilleri konusunda yapılan çalışmalar da neredeyse son bulmuştur.

CCND dilleri açık paralelliğe sahiptir. Açık paralel dillerde, paralelliği sağlayabilmek için kullanıcı, yazmış olduğu paralel bilgisayar programına bazı eklentiler yapmak zorundadır.

CCND Dilleri

CCND dilleri, akışlı ve-paralel bir hesaplama modeli sunarlar. Bu modelde, çağrılar arasında bulunan ortak değişkenler prosesler arasında bir iletişim kanalı kurarlar. Bir çağrının işlenebilmesi için bütün giriş değişkenlerinin belirli, atanmış olması gereklidir. Bu tanım, açıkça CNND dillerini veri-akışlı diller içine katmaktadır. Ayrıca bu dillerin bir diğer özelliği ise daimi (perpetual) proseslere destek vermesidir.

Bütün CCND dillerinin söz-dizim kuralı 1'deki yazım şekline uyarlar.

H :- G1, G2, ..., G n | B1, B 2, ..., Bm. (1)
n, m >= 0

H, Gi ve Bi ifadeleri klasik Prolog'un terimleri gibidir. Tek bir atomdan veya bir adet yüklem adı ve belirli sayıda argümandan oluşur. “|” işaretine seçme operatörü denir.

Bu dillerde guard özelliği ve veri-akış senkronizasyonu prensibi ortaktır. Cümlelerde bulunan guard çok önemli bir özelliğe sahiptir. Genelde bu tür diller, guard'ın uygulanma tekniğine göre birbirlerinden ayrılırlar. Guard'ın cümle içindeki görevi kısaca şöyle açıklanabilir: Bir çağrı yapıldığında önce çağrıdaki terim ile H birleştirilir. Bu işlem başarılı olursa seçme operatörüne kadar olan bütün çağrılar sıralı veya paralel bir biçimde yürütülür (guard evaluation). Eğer bu yürütmenin tamamı başarılı olursa seçme operatörü devreye girer. Aslında bu operatör bir iş yapmaz. Bu operatörü sıralı Prolog'daki kesme operatörüne benzetebiliriz. Bu adımda sanki bir kesme işlemi yapılıyormuş gibi bütün alternatif çözüm yolları iptal edilir. Artık bu adımdan sonra geri dönüş yoktur, geriye-iz-sürme yoktur. İşte non-determinism'in bu tip uygulamasına "don't care nondeterminism" denir. Geriye-iz-sürme işlemi sadece seçme operatörüne kadar geçerlidir. Bundan dolayı CCND dillerinde seçim operatöründen sonra herhangi bir çağrı başarısız olursa bütün hesaplama başarısız olacaktır. Çünkü alternatif çözüm yolları, guard hesaplama bittikten hemen sonra iptal edilmiştir.

Seçme operatörüne kadar yapılan işlemler bir CCND dilinin karakteristiğini tespit eder. Seçme operatörüne kadar yapılacak işlemler her ne kadar uygulamadan uygulamaya değişse de, bu operatörden sonra yapılan işler bütün dillerde aynıdır. Bu opertörden sonraki her bir çağrı paralel olarak işletilir.

Üç adet klasik CCND dili mevcuttur. Parlog, Concurrent Prolog ve Guarded Horn Clause (GHC). Aşağıdaki bölümlerde her bir dilin özellikleri kısaca incelenecektir.

Parlog

Parlog (Clark ve Gregory, 1986) dilinde, birleştirmenin yönünü tespit etmek amacı ile yüklem seviyesinde mod tanımı yapılır. Yüklemin her bir argümanı giriş veya çıkış modları (? ve ^) ile tanıtılır. Baş birleştirilmesi sırasında giriş argümanıyla karşılaşılırsa ve bu değişkene değer atanmamışsa bu proses askıya alınır. Eğer yürütme seçim operatörüne kadar gelmiş ise gövde çağrıları işlenmeden evvel çıkış argümanlarına atama yapılır. Böylece cümlenin seçimi garanti edildikten sonra çıkış argümanları kullanıma açılır. Parlog programları güvenli olmak zorundadırlar. Diğer bir deyişle guard çağrıları giriş

modunda bulunan argümanları atamamalıdır. Cümlenin baş kısmı için tek yönlü birleştirme geçerlidir. Guard işleme kısmında ise standard birleştirme yapılır. Guard işleme sırasında genelde Veya (OR) prosesleri üretilir. Bütün guard işlemleri başarılı olursa artık bu cümle geri dönüşü olmayacak bir biçimde seçilmiştir. Daha sonra gövde içindeki her bir çağrı (atomic goal) için bir proses açılır.

Concurrent Prolog

Concurrent Prolog (CP) (Shapiro, 1983) veri-akışı kısıtı olarak değişken seviyesinde salt-okunur takısını kullanır. CP'nin senkronizasyon mekanizmasını oluşturan salt-okunur değişkenler veri-akış senkronizasyonunun (Ackerman, 1982) genelleştirilmiş bir halidir. CP'de, Parlog gibi yüklem seviyesinde mod tanımı mevcut değildir. Salt-okunur takı, sadece birleştirme sırasında ele alınır. Çünkü bir giriş değişkeni, diğer bir deyişle salt-okunur bir değişken, bir terimin herhangi bir yerinde ortaya çıkabilir. Ayrıca guard çağrıları bütün prosesi askıya alabilir. Birleştirme işlemi de zamana bağlıdır, ilk anda başarısız olan bir iş daha sonra başarılı olabilir. Değişken atama için hiçbir kısıt getirilmemiştir. Cümle içinde herhangi bir yerde herhangi bir zamanda değişken ataması yapılabilir. Bu da paralel programlamaya olağanüstü bir esneklik kazandırmaktadır. Fakat bu aşırı esnekliği sağlamak için çok karmaşık çoklu çevre yönetimi gerekmektedir. Ayrıca bazı ciddi uygulama problemleri de mevcuttur (Ueda, 1985). Bu zorlukların üstesinden gelebilmek için Flat Concurrent Prolog (FCP) tasarlanmıştır.

Guarded Horn Clause

Guarded Horn Clause (GHC) (Ueda, 1986), CP'ye bir alternatif olarak geliştirilmiştir. GHC'de değişken erişimini kısıtlamak için doğrudan veya açık bir biçimde takı kullanılmaz. Baş birleştirmesi sırasında bir değişkenle karşılaşıldığında veya guard işleme sırasında başta bulunan bir değişkene rasgelindiğinde, değişken değeri belli olana kadar bütün proses askıya alınır. Başta bulunan bir değişkene ancak cümle seçildikten sonra bir terim ataması yapılır. Flat GHC'de, guard kısmında yine sadece önceden tanıtılmış yüklemlere izin verilir. Bütün guard işlemleri sıralı yürütülür. Eğer guard işleme sırasında atanmamış değişkenle karşılaşılırsa, bütün proses yine askıya alınır. Flat GHC'nin bir diğer ismi ise KL1'dir.

Berk Prolog

Bu çalışmada standard Prolog programlama dili, paralel mantık programlama diline genişletilmiştir. Elde edilen amaç dil, CCND dil ailesinin bir üyesidir. Bu yeni üye Berk Prolog olarak adlandırılmıştır. Bu çalışmada sadece yeni bir programlama dilinin özellikleri ve kuralları verilmemiş aynı zamanda bu dilin dağıtık bir ortamda yürütülmesini sağlayan derleyicisi de yazılmıştır. Paralel derleyiciyi yazabilmek için GNU Prolog (Diaz, 2002) sıralı derleyicisi kullanılmıştır. Gerekli kütüphaneler yazılarak, GNU Prolog derleyicisi paralel Prolog programlarını dağıtık bir ortamda yürütebilecek bir hale getirilmiştir.

Berk Prolog bir CCND dilidir ve içinde guarded Horn cümlelerini barındırır. Berk Prolog'un sözdizimi 1'de verilen ifadeye benzemekle birlikte bazı farklılıklara sahiptir. Bu faklılıklarla elde edilen yeni guarded Horn cümlesine, orjinal guarded Horn cümlesinin genişletilmesi denilmektedir. Genişletilmiş guarded Horn cümlesinin sözdizimi 2 ifadesinde verilmiştir.

```
H :- G1, G2, ..., Gn                    (2)
    B1 Koşul1,
    B2 Koşul2,
    ...
    Bm Koşul.
```

n, m >= 0

2 ifadesi görünüşte 1'deki yapıya benzemesine rağmen H, G ve B terimlerinin tanıtılmasında pek çok farklılık arzeder.

Cümlenin baş kısmını temsil eden H ifadesi, f/n şeklinde, klasik bir Prolog terimidir. Bu terim @/2 operatörü ile genişletilmiştir. Baş kısım sadece H şeklinde olabileceği gibi H@Alias şeklinde de olabilir. Alias değeri ilgili cümlenin hangi bilgisayar ortamında veya indirgeme biriminde indirgeneceğini tespit eder. Alias değeri uygun bir ad olabileceği gibi bir liste de olabilir.

Cümlenin baş kısmında CP'de olduğu gibi salt-okunur operatör mevcut değildir. Parlog'da olduğu gibi cümle seviyesinde mod tanımı da mevcut değildir. Hem mod tanımı hem de salt okunur değişken eki iptal edilerek baş kısmın tamamen bir Prolog terimi olması sağlanmıştır. Böylece sıralı bir Prolog derleyicisi, baş birleştirmesini, kendi birleştirme algoritmasında hiç bir değişiklik yapmadan kullanabilecektir.

Baş birleştirilmesi sırasında bir proses askıya alınmaz. Çünkü giriş değişkenleri hazır olmayan bir çağrı yapılamaz. CP'de ise giriş değişkenleri hazır olsa da olmasa da çağrı yapılır. Eğer bir salt-okunur değişkenle karşılaşılırsa ve bu değişkenin değeri henüz mevcut değilse, çağrı ve dolayısıyla proses askıya alınır. Parlog'da ve GHC'de de aynı mantık silsilesi geçerlidir. Berk Prolog'ta ise proses senkronizasyonu, baş veya guard kısmında değil gövde kısmında ve prosesi çağırmadan evvel yapılmaktadır.

Baş kısmında @/2 operatörünün kullanılması farklı sonuçlar doğurmaktadır. CCND dillerinde tanımlanan bir P programı sonlu sayıda guarded Horn cümlelerinden kuruludur. Her bir proses bu cümleleri kullanarak indirgeme yapabilir. Berk Prolog'ta ise bu kavram daha da genişletilmiş ve daha kontrollü bir cümle dağıtımı getirilmiştir. Bir proses indirgemesi yapılırken, indirgeme biriminde, o yükleme ait cümlelerin bulunması gerekmektedir. Kullanıcı veya planlayıcı program hangi indirgeme biriminde hangi cümlenin bulunduğunu bilmelidir.

Sistolik programlamada kullanılan Logical Occam (Cohen vd., 1992) dilinde @/2 ile proses ataması mevcuttur. Ayrıca PMS-Prolog (Wise vd., 1992) dilinde ise @/2 kullanarak yüklem dağıtımı, önerilen anlamdan daha farklı bir biçimde mevcuttur. Berk Prolog'a her iki özellik te farklı bir biçimde eklenmiştir. Ayrıca gövde çağrılarına koşul getirerek @/2 operatörü ile beraber Horn cümlesini genişletilmiştir.

Guard kısmı yine klasik olarak, Ve operatörü ile birleşmiş, p(A1, A2, ..., An) çağrılarından oluşmaktadır. Berk Prolog flat bir dil değildir. Bundan dolayı guard çağrıları aynen CP'de olduğu gibi her zaman ve her yerde, her türlü değişkeni atayabilirler. Bu değişkenler baş veya gövde değişkeni de olabilir. Diğer bir deyişle Berk Sistemde güvenli olma gibi bir kavram mevcut değildir. Flat diller, guard'da sadece önceden tanımlanmış yüklemlere izin verirler. CP'de ise guard için hiç bir kısıt yoktur. Berk Polog'ta ise guard kısmında sadece klasik Prolog çağrılarına izin verilmektedir. Flat olmayan diğer CCND dillerinde olduğu gibi guard kısmında paralel Prolog yüklemleri değil sadece standard Prolog yüklemleri çağrılmaktadır. Bu kısıt basit bir ihtiyaçtan doğmuştur: Sıralı bir Prolog derleyicisini kullanarak, üzerinde hiç bir değişiklik yapmadan, paralel bir Prolog sistemi oluşturmak.

Yukarıdaki paragraflarda bahsettiğimiz gibi, baş kısmının standart bir Prolog terimi olması ve guard kısmında da sadece standart Prolog çağrılarının yapılmasından dolayı, 2 ifadesinde verilen cümle, guard operatörüne kadar sıralı Prolog ile yürütülebilir.

Bu kısıtlardan çok basit bazı sonuçlar çıkmaktadır. Baş veya guard terimlerinde bulunan değişkenlerde senkronizasyonu sağlamak için salt-okunur eki mevcut değildir. Bundan dolayı baş veya guard çağrıları bir prosesi askıya alamaz. Diğer CCND dillerinde ise, sadece baş veya guard çağrıları prosesi askıya alabilirler. Berk Prolog'ta, guard içindeki çağrılar sıralı Prolog tarafından işleneceği için, guard çağrıları sıralı yürütülür.

Klasik CCND dilleri sıralı Prolog'u dilin bir parçası olarak kabul etmezler. Sadece CP dilinin sıralı Prolog ile basit bir etkileşim mekanizması mevcuttur.

CP'de olduğu gibi baş birleştirmesi ve guard hesaplaması atomiktir. Bu işlemler sırasında hiç bir indirgeme birimi hesaplanan değişkenlere erişemez. Aynı zamanda cümle seçimi başarısız olursa bütün hesaplanan değişkenler geri alınır.

Bu çalışmada kavramsal olarak farklı bir yaklaşım önerilmektedir. Klasik CCND dillerinde baş birleştirme ve guard hesaplama cümle seçimi

amacı ile kullanılmaktadır. Esas hesaplama gövde yüklemlerinin argümanları tarafından yapılmaktadır. Berk Prolog'ta ise asıl hesaplama kısmının guard tarafında sıralı Prolog ile yapılması ve gövde yüklemlerinin hesaplamadan ziyade prosesler arası iletişim ve senkronizasyon amacı ile kullanılması önerilmektedir.

Seçme operatörü bilinen klasik anlamında kullanılmıştır. Eğer yürütme guard operatörüne kadar gelirse, bu çağrıya ait bütün alternatif çözümler iptal edilir. Diğer bir deyişle CCND dillerinin ortak özelliği olan "don't care nondeterminism" uygulanmıştır.

2 ifadesinde gövde kısmı, "B1, B2, ..., Bn" şeklinde, "Ve" ile bağlanmış, terimlerden oluşmaktadır. Berk Prolog'ta B terimleri, klasik atomik çağrılar olabilir. Böylece Berk Prolog, guarded Horn cümle yapısını tam olarak içermektedir. Kullanıcı Bi çağrılarını, @/2 operatörü ve koşul koyarak 3'teki gibi genişletebilir.

f(A1, A2, A3, ...,An)@Alias with Koşul (3)

3 ifadesinde standrad guarded Horn cümlesi gösterimi, @/2 ve with/2 operatörleri ile genişletilmiştir. Bu operatörlerin kullanımı tamamen çözülen probleme özgüdür. Alias değişkeni bir indirgeme birimi ismi veya isim listesi olabilir. p(X)@[r1, r2, r3] with [need(X)] ifadesinde p/1 çağrısı aynı anda paralel olarak r1, r2 ve r3 adlı indirgeme birimlerinde işlenecektir. Fakat klasik CCND dillerinde olduğu gibi, bu çağrı önce yapılıp sonra da ilgili indirgeme biriminde askıya alınmaz. Berk Prolog'da çok basit bir kural vardır, eğer giriş değişkenleri mevcut değilse çağrı bekletilir, yapılmaz. Bundan dolayı indirgeme sırasında guard operatörüne gelene kadar hiç bir çağrı, prosesi askıya alamaz. Çünkü bütün giriş değişkenleri hazır olduktan sonra çağrı yapılır.

Parlog'da cümle seviyesinde, CP'de salt-okunur ek ile, GHC'de ise nonvar/var denetimi ile yapılan prosesler arası senkronizasyon, Berk Prolog'da gövde çağrılarında ve with/2 operatörünün içine eklenecek need*/n mesajları ile sağlanır. Berk Prolog'da with/2 operatörünün başka kullanım alanları da vardır.

with/2 operatörü içinde kullanılan bir diğer özellik de çıkış değişkenlerinin açık olarak kullanıcı tarafından belirtiliyor olmasıdır. CP'de sadece salt-okunur değişkenler kullanıcı tarafından tespit edilir, GHC'de kullanıcı hiçbir değişkene müdahalede bulunmaz, hem giriş hem de çıkış değişkenleri kullanım yerine göre tespit edilir. Örtük paralelliğe en yakın sistem GHC'dir. Parlog'da ise bir üst seviyede hem giriş hem de çıkış argümanları kullanıcı tarafından tespit edilir. mod/1 tanımı ile yapılan bu müdahalede, birleştirmenin yönü dolayısı ile değişkenlerin tipi tespit edilmektedir. Berk Prolog'da ise çıkış değişkenleri veya üretici değişkenler, açıkça share* iletileri ile, çağrı seviyesinde tespit edilir. Berk Prolog bu kullanım şekli ile kullanıcıya bir miktar fazla yük bindirmekte fakat programlama esnekliğini artırmaktadır.

Klasik CNND dillerinin ortak özelliklerinde olduğu gibi, prosesler arası iletişim yine akışlar vasıtası ile sağlanır. Prosesler arası senkronizasyon need* iletisinin açık olarak belirtilmesi ile, çıkış değişkenlerinin tespiti ise share* iletisi ile sağlanır.

Şu anda sistemde son derece basit bir planlayıcı mevcuttur. @/2 operatörü ile açık olarak verilmemiş olan bütün çağrılar yerel makinede indirgenirler.

Bir proses şartlı bir biçimde değişken talebinde bulunabilir. Örneğin need_and([X, Y, Z]) mesajında, her üç değişken de aynı anda belli olmadan bu değişkeni bekleyen proses ayağa kaldırılmaz. Ya da, need_or([X, Y, Z]) şeklindeki bir mesajda X, Y veya Z değişkenlerinden ilk önce hangisi değer almışsa bu değişkeni bekleyen proses ayağa kaldırılır. need_or([X, Y, Z]) ile need_or([Y, X, Z]) bu yüzden farklıdır. Diğer bir deyişle need_or mesajının elemanları arasında sıraya bağımlılık mevcuttur. Bu ilginç özellik ile, kararlı bir sistem olan Berk Sistem'de sonlu beklemeli merge/3 algoritması gerçekleştirilmiştir.

Berk Sistem'de, kullanıcının yazdığı paralel Prolog programları yürütme için uygun değildir. Bundan dolayı yazılan programlar Berk Sistem tarafından bir ön-işleme aşamasından geçirilir.

Berk Sistem kararlı bir sistem olduğu için, if-then-else yapısı ve dolayısı ile not/1 yüklemi problemsiz bir biçimde uygulanmıştır. Burada if-then-else'deki then kısmı için bir kısıt mevcuttur: then kısmında sadece standard prolog yüklemleri çağrılabilir.

Berk Sistem'de ortak değişkenin birden fazla üreticisi olabilir. Eğer üretilen ortak değişkenler paylaşılıyorsa, basit bir çakışma denetimi yapılarak sistemin bütünlüğü korunmaya çalışılır.

Berk Prolog'a körleme bekleyiş olarak adlandırdığımız bir kavram eklenmiştir. Bununla bir proses tarafından hesaplanan bir değişken belirli bir indirgeme birimine doğrudan yönlendirilmektedir. Değişkeni bekleyen indirgeme birimi, değişkeni beklediğine dair hiç bir yere bir mesaj veya bilgi göndermez. Değişkeni hesaplayan proses, bu değişkenin hangi indirgeme birimi tarafından beklendiğini bilir ve oraya gönderir. Bu daha çok uygulamaya yönelik bir kavramdır, çıkış veya log bilgilerinin belirli bir makinede tutulması amacı ile kullanılabilir.

İndirgeme

Sonraki kısımda açıklanacak olan Berk Sistem'de indirgeme sadece ilgili istemcide yapılır. Sunucu indirgeme işlemine müdahale etmez. İndirgenecek atom ile ilgili cümleler, yukarıdan aşağıya, kullanıcının yazdığı sırada taranır. Berk Sistem bu şekildeki taramayı her zaman garanti eder. Bu tür sistemlere kararlı sistem denir. Berk Sistem kararlı bir sistemdir. Böylece if-then-else yapıları kolay bir biçimde uygulanabilmektedir. İndirgencek Atom, önce yazılan sırada yukarıdan aşağıya kontrol edilir. Head ile Atom bilinen klasik yolla birleştirilir. Eğer birleştirme işlemi başarılı olursa guard içindeki atomlar tek tek çağrılır. Bu işlem doğrudan ISO Prolog'un call/1 yüklemi ile yapılır. Eğer call(Guard) çağrısı başarılı olursa, commit operatörü devreye girer. Bu operatör klasik cut operatörü gibi çalışır ve bütün alternatif çözümleri yok eder. Yüklemin bu cümlesi, artık seçilmiştir. Eğer varsa, sonraki bütün cümleler gözardı edilir. Guard, klasik Prolog çağrılarından oluşmaktadır ve Berk Prolog yüklemlerini çağıramaz.

Shapiro'nun sistemi (Shapiro, 1983) ile Berk Prolog arasındaki en önemli farklardan biri salt okunur değişkenler için "?" operatörünün kullanılmamasıdır. İkinci önemli fark ise guard içinde Berk Prolog çağrılarının bulunmamasıdır. Guard içinde salt okunur değişken ve Berk Prolog çağrısı mevcut olmadığı için guard içinde bulunduğu prosesi askı durumuna geçiremez. Bu özellik ise Berk Prolog ile CP arasındaki üçüncü önemli farkı oluşturur. İşte bu üç önemli fark, CP ve Berk Prolog'u birbirinden kesin çizgilerle ayırmaktadır.

Guard içinde bütün klasik Prolog çağrıları bulunabileceği gibi cut kontrol operatörü de bulunabilir. Eğer Berk Sistem'in üzerinde çalıştığı klasik Prolog sisteminin call/1 uygulaması, cut işlevini gerçeklerse, guard içinde bulunan cut karakteri de bilinen yan etkilerini gösterir. Diğer durumlarda ise bu kontrol operatörü göz ardı edilir. Diğer bir deyişle guard'ın davranışı, Berk Sistem'in yazıldığı klasik Prolog sistemindeki call/1 çağrısının davranışına bağlıdır.

Gövde kısmı ise concurrency'nin asıl gerçekleştiği yerdir. Gövde atomları, guard atomlarından faklı olarak hem Berk Prolog hem de sıralı Prolog çağrıları yapabilirler. Baş ve guard çağrıları her zaman soldan sağa doğru klasik Prolog'daki gibi işlenirler. Gövde kısmı concurent Prolog'a geçiş kısmıdır. Gövdenin yazım şekline ve planlamaya göre gövde içindeki çağrılar, yerel ve uzak olarak iki ayrı gruba ayrılırlar. Uzak çağrı yapan atomlar hemen sunucuya gönderilirler. Sunucu da atomları indirgenmek üzere ilgili istemcilere dağıtır.

Berk Sistem

Berk Prolog'un TCP/IP ortamındaki yürütme modeli Berk Sistem olarak adlandırılmıştır. Bu yürütme modelinde, standart Prolog sisteminin kendi yüklemleri kullanılarak paralel bir kütüphane yazılmıştır. Bu paralel kütüphaneyi kendisine ekleyen her standart Prolog sistemi, doğrudan paralel Prolog sistemine dönüşecektir.

Berk Sistem'de kullanılan paralelleştirme kütüphanesi klasik Prolog sisteminin yapısında herhangi bir değişiklik yapmamaktadır. Kütüphane doğrudan orjinal sistemin yüklemleri ile yazıldığıdan, bu sistemin kaynak kodlarına ulaşma problemi de olmayacaktır. Ayrıca orijinal sistemde bulunan, DCG, CLP gibi genişletmeler de doğrudan paralel sisteme geçmiş olacaktır.

Berk Sistem MIMD (Multiple Instruction Multiple Data) tipinde bir paralelliğe sahiptir ve tam bir istemci/sunucu uygulamasıdır. Bu özelliği ile Beowulf tipi sistemlerde çalışmaya son derece elverişlidir.

Bir bilgisayarda Berk Sistem'in çalışabilmesi için IP yığın yüklü herhangi bir Linux makine olması yeterlidir. Berk Sistem ile derlenmiş kodun paralel çalışabilmesi için bir adet makine sunucu en az bir adet makine de istemci olarak seçilmek zorundadır. Sunucu görevini üstlenen makinenin ana işlevi indirgenecek olan atomları ilgili makinelere göndermek ve değişken alışverişi için bir santral görevi görmektir. Teorik olarak istemci sayısında bir sınır yoktur.

Bir ve yalnız bir sunucu, en az bir adet istemci ve bir port numarasından oluşan yapıya "çözüm ağı" denir. Aynı çözüm ağı içindeki bütün istemcilerin aynı sunucu adını ve aynı portu kullanmaları zorunludur. Sunucu adı aynı olmasına rağmen farklı bir port numarası verilirse, bu ayrı bir çözüm ağına karşı gelir. Diğer bir deyişle bir sunucu aynı anda birden fazla çözüm ağına destek verebilir. Bir çözüm ağı içinde geçen herhangi bir değişken aynı anda iki veya daha fazla istemci tarafından kullanılabilir.

Sunucuyu ağ alanı gibi düşünebiliriz. Çözüm ağı içinde bulunan sunucu, bütün paylaşılan değişkenlerin birer kopyasını kendi içinde barındırır. İhtiyacı olan istemciye istenen değişken gönderilir. Bu özellik çözüm ağının ve Berk Sistem'in temel taşlarından birini oluşturmaktadır.

Bir terimi ground hale getirmek için bu terimin bütün değişkenleri ground bir terim ile değiştirilir. Bu işleme dondurma denir. Bu işlemin tersine de eritme denir. Ertime işlemini kolaylaştırmak için, dondurma sırasında her değişkene karşı gelen bir sayı tutulur. Böylece eritme işlemi sırasında aynı değişkenlere aynı adresin verilmesi garantilenmiş olur. Bir terim içindeki bir atanmamış değişken kendi heap alanı içinde anlamlıdır. Heap dışına çıkıldığı an bu değişkenin hiçbir anlamı kalmaz. Berk Sistem terimleri istemciler arasında taşımak için benzer bir teknik kullanır. Ek olarak, bir istemcide bulunan bir değişkenin başka bir istemcide de aynı değişken olmasının garantilenmesi gerekir. Berk Sistem bunu sağlamak için bütün değişkenlere özel bir numara verir.

Çözüm ağını kontrol etmek amacı ile yapılan bazı işlemler mevcuttur. Bu işlemler (freeze, continue, stop, fail) sunucu tarafından denetlenir. Herhangi bir istemci veya konsol karşısındaki bir kullanıcı sunucudan freeze talebinde bulunabilir. Bu durumda bütün çözüm ağı geçici bir süre için askıya alınacaktır. Kullanıcı herhangi bir istemciden veya sunucudan klavyeyi kullanarak sistemin o anki durumunu inceleyebilir. Daha sonra da sunucudan “cont” komutunu girerek bütün sistemi ayaklandırır ve çözüm ağı kaldığı yerden çözüme devam edebilir. Ya da bütün sistem uykuya daldıktan ve sistem durumu incelendikten sonra bütün çözümün hatalı bitmesi istenebilir.

Kütüphane yüklemleri

Giriş akışları

Paralel mantık programlamadaki en temel ihtiyaçlardan birisi kullanıcı girişlerinin akış şeklinde sağlanabilmesidir. Diğer kütüphane fonksiyonlarında olduğu gibi bu fonksiyon da tek bir istemci içinde çalışır. Elde edilen sonuçların diğer istemcilere dağıtılması kullanıcının sorumluluğundadır.

Çıkış akışları

outstream/1 fonksiyonu bir akışı ekranda göstermek amacı ile kullanılır. Fakat ekrana yazılacak ifade bir akış değil de tek bir ifade ise bu fonksiyonu kullanmak uygun değildir. Bu tür çıkışlar için guard kısmında klasik write/1 fonksiyonu kullanılabilir. Gövde kısmında da write/1 'in yaptığı işin aynısını yapan output/1 fonksiyonu tanımlanmıştır.

Akışların birleştirilmesi (Stream Merging)
Pek çok concurrent programlama dili prosesler arası iletişimi sağlamak için akışları kullanırlar. Akışlar, oku/yaz işlemlerini gerçekleştirecek veri yapılarıdır. Berk Sistem de yine prosesler arası iletişim için akışları kullanır. Akışlar, yapı itibariyle listelere benzemesine rağmen, listelerden çok büyük bir farkı vardır. Akışlar, hesaplamanın herhangi bir anında parçalı olarak belirlenmiş veri yapılarıdır. CCND dillerinde olması gereken en önemli özelliklerden birisi iki akışın birleştirilerek tek bir akış haline getirilmesidir. Concurrent programlama dillerinde stream merging işlemleri ya hazır bir kütüphane fonksiyonu olarak verilir veya o dilin kuralları kullanılarak normal bir yüklem olarak sisteme eklenir. Berk Sistem'de ikinci yöntem kullanılmıştır. Akışların birleştirilmesi için merge/3 yüklemi yazılmıştır. Bu yüklem iki adet akışı birleştirmektedir. 2 adet akışı birleştirmek ile n adet akışı birleştirmek arasında prensipte bir farklılık yoktur. Bundan dolayı sadece iki akışın birleştirilmesi üzerinde durulmuştur. Ayrıca birleştirme için bir öncelik tanımlanmamıştır. Argümanların sırasını değiştirmekle bu özellik de merge/3 yüklemi içine kolayca eklenebilir. Akışların birleştirilmesi özelliği Shapiro'nun (1983) raporunda ayrıntılı olarak incelenmiştir. Berk Sistem kararlı bir sistemdir. Shapiro'ya (1983) göre kararlı bir sistemde akışların birleştirilmesi sonlu bir bekleme ile mümkün değildir. Bundan dolayı sistemin en azından zayıf kararlı veya kararsız olması gereklidir. Kararlı bir makinede, eğer her iki akışta da aynı anda veri mevcutsa her zaman birinci akıştaki elemanlar işlenecektir. Hiçbir zaman ikinci akışa geçilemeyecektir. Fakat Berk Sistem'de uygulanan mimari gereği bu problem kendiliğinden ortadan kalkmaktadır. Şöyle ki, her iki akışta da eleman olmasına rağmen akışları birleştirme programı need_or kavramı sayesinde, akışları aynı anda değil teker teker yerel belleğe çekmektedir. Aynı anda iki adet akış olsa dahi her zaman yerel bellekte tek akış gözükecektir. Akışların isimleri need_or parametrelerinde ters yüz edilerek tekrar çağrıldığında bu sefer diğer akış elde edilecek ilk akışta veri olmasına rağmen, yerel makine tarafında ilk akış unbounded gibi gözükecektir. Eğer ilk akış tarafında bir eleman bekleniyorsa, ikinci akış tarafına bir eleman geldiğinde, sistem hala ilk akışı bekleyecektir. Böylece ilk akışa bir 'ilk eleman' gelmediği sürece sistem tıkanacaktır. Berk Prolog'ta öne sürülen need_or kavramı bu problemlerin üstesinden gelmektedir.

Sonuçlar ve tartışma

Bu çalışmada, Prolog programlama dili paralel mantık programlamaya genişletilerek, CCND dilleri ailesine yeni bir dil kazandırılmıştır. Bu yeni dilde, kararlı bir makinenin de sonlu bir bekleme ile akışları birleştirebileceği gösterilmiştir.

Birden fazla değişkenin şartlı olarak talep edilmesi kavramı öne sürülmüştür. Bir çağrı yapılmadan evvel çağrıya ait bütün değişkenler elde edilmekte sonra çağrı yapılmaktadır. Böylece esas çağrı yapılmadan evvel, mevcut değişkenlerin durumları incelenerek, yüklemin uygun cümlesi çağrılabilmektedir.

Hem proseslerin hem de yüklemlerin indirgeme makinelerine kullanıcı tarafından keyfi bir biçimde dağıtabilmesi sağlanarak programlama açısından esneklikler getirilmiştir. Sıralı Prolog üzerinde çok az bir değişiklik yaparak paralel mantık programlama diline geçiş sağlanmıştır. Böylece paralel mantık sistemlerinin, derleyicilere bağımlılığı azaltılmıştır.

Berk Prolog'un dağıtık bir ortamda yürütülebilmesi amacı ile geliştirilen bir yürütme modeli tanıtılmıştır. Bu yürütme modeli, TCP/IP ortamında çalışan dağıtık bellekli bir yürütme modelidir. Bu model için, sıralı bir Prolog derleyicisi kullanılarak paralel bir derleyici geliştirilmiştir. Bu derleyici, önerdiğimiz paralel programlama dilini kullanarak pek çok örnek programı yürütebilmiştir.

Berk Prolog'un ve TCP/IP ortamında uygulanan yürütme modeli olan Berk Sistem geliştirmeye açıktır. Berk Prolog'un son derece serbest olan semantiği daha kesin sınırlar ile tanıtılmalıdır. Dağıtık artık toplama sisteme eklenmelidir. Berk Sistemi'n verimini, planlayıcı doğrudan etkilemektedir. Bundan dolayı hem sistemin

verimli çalışması hem de kullanıcının üzerindeki yükü azaltmak amacı ile, Berk Sistem'e bir planlayıcının eklenmesi gerekmektedir. Dağıtık hesaplama için gerekli olan, merge, giriş, çıkış gibi yüklemler geliştirilmeli ve yenileri yazılmalıdır. Ayrıca, Berk Sistem'in kullandığı kütüphanenin tamamı sıralı Prolog ile yazılmıştır. Hızı artırmak için pek çok yüklem, doğrudan C ile yazılabilir. Berk Sistem için, dağıtık bellekli MIMD yürütme modeli üzerinde çalışılmıştır. Ortak bellekli modellerde sistemin çok iyi bir performans göstereceği aşikardır. Çünkü indirgeme birimleri arasında değişken akışı zaman kaybına sebep olmayacaktır. Ortak belleğe sahip bir makine için yeni bir yürütme modeli üzerinde de çalışılmalıdır.

Kaynaklar

Ackerman, W.B., (1982). Data flow languages, *IEEE Computer* 15, **2**, 15-25.

Clark, K.L. and Gregory, S., (1986). Parlog: Parallel programming in logic, *ACM Transaction on Programming Languages and Systems*, 8, 1, 1-49

Cohen D., Huntbach, M.M., Ringwook G.A, (1992), Logical Occam, Implementation of distributed prolog, Willey.

Diaz, D., (2002). GNU Prolog User Manuel, *A Native Prolog Compiler with Constraint Solving over Finite Domains,* Edition 1.7.

Pollard, G.H., (1981). Parallel execution of horn clause programs, *Ph.D. Thesis,* Department of Computing, Imperial College.

Shapiro, E.Y., (1983). A subset of concurrent prolog and ıts interpreter, *ICOT Techincal Report TR-003*.

Ueda, K., (1986). Introduction to guarded horn clauses, *ICOT Techincal Report,* TR-209.

Ueda, K., (1985). Concurrent prolog re-examined, *ICOT Techincal Report, TR-102.*

Wise, M.J., Jones, D.G., Hintz, T., (1992). PMS-Prolog: A distributed, coarse -grain-parallel prolog wiht processes, modules and streams, *Implementation of distributed prolog*, Willey.