Automatização da Análise Global no modelo Granlog

Silvana Campos de Azevedo
Jorge Luis Victória Barbosa
{silc, barbosa}@atlas.ucpel.tche.br
Universidade Católica de Pelotas
Escola de Informática
Caixa Postal 402 – CEP 96010-000
Pelotas, RS, Brasil

Cláudio Fernando Resin Geyer
geyer@inf.ufrgs.br
Universidade Federal do Rio Grande do Sul
Instituto de Informática
Caixa Postal 15064 – CEP 91591-970
Porto Alegre, RS, Brasil

Resumo

Este trabalho apresenta a integração dos modelos ParTy, MODA e Granlog. O ParTy (*Parallel Types Analyzer*) é um interpretador abstrato de tipos. O MODA (*Modes and Dependencies Analyzer*) é um interpretador abstrato de modos e dependências. O Granlog (*Granularity Analyzer for LOGic programming*) é um analisador automático de granulosidade na programação em lógica. O modelo Granlog necessita da análise estática de modos, tipos, medidas e dependências dos argumentos de um programa Prolog. A análise estática pode ser realizada através da técnica de interpretação abstrata. Esta técnica simula a execução de um programa segundo um domínio abstrato obtendo informações sobre o seu comportamento. O ParTy realiza a interpretação abstrata de tipos e o MODA realiza a interpretação abstrata de modos e dependências dos argumentos de um programa Prolog. As integrações ParTy-Granlog e MODA-Granlog tornam automática a análise global do Granlog.

Palavras Chaves: Interpretação Abstrata, Programação em Lógica e Processamento Paralelo.

Abstract

This work presents the integration of the models ParTy, MODA and Granlog. ParTy (Parallel Types Analyzer) is a static analyzer of types, based on abstract interpretation. MODA (Modes and Dependencies Analyzer) is an abstract interpreter of modes and dependencies. Granlog is an automatic granularity analyzer for logic programming. The Granlog model needs the static analysis of modes, types, measures and dependencies of the arguments of a Prolog program. The static analysis can be accomplished through the technique of abstract interpretation. This technique simulates the execution of a give program over an abstract domain, obtaining information about its behavior. ParTy accomplishes the abstract interpretation of types and MODA accomplishes the abstract interpretation of modes and dependencies of the arguments of a program Prolog. The integrations ParTy-Granlog and MODA-Granlog turn automatic the global analysis of Granlog.

Keywords: Abstract Interpretion, Logic Programming and Parallel Processing.

1 Introdução

A interpretação abstrata é uma forma de análise global que simula a execução de um programa segundo um domínio abstrato obtendo informações do seu comportamento [COR97], [DAM97], [LAB96]. Esta técnica auxilia na otimização, depuração e paralelização de programas.

A programação em lógica, em especial a linguagem Prolog, conta com várias fontes de paralelismo implícito, estimulando a exploração automática do paralelismo [KER94].

A obtenção de um melhor desempenho no processo de paralelização em um programa Prolog necessita da análise do comportamento das variáveis em relação ao modo de chamada dos procedimentos e em relação aos tipos de elementos a que se instanciam. Isto deve-se ao fato da programação em lógica não possuir distinção entre argumento de entrada e argumento de saída. Esta característica dificulta a determinação do fluxo de controle e consequentemente a determinação das dependências entre as variáveis de um programa. Além disso, na programação em lógica as variáveis não são tipadas. Isto dificulta à determinação dos custos de comunicação entre as partes de um programa. A preocupação com a determinação das dependências e com os custos de comunicação deve-se ao fato deste trabalho se encontrar dentro do âmbito do Projeto Opera [BRI90], [MOR96], [YAM94], o qual opta pela exploração do paralelismo em um ambiente de memória distribuída de forma automática.

O Granlog é um analisador automático de granulosidade na programação em lógica. A completa automatização do modelo Granlog necessita da interpretação abstrata de modos, tipos, medidas e dependências dos argumentos de um programa Prolog [BAR96a].

O ParTy é um interpretador abstrato de tipos para a programação em lógica [CAS97], [CAS97a]. O modelo MODA é um interpretador abstrato de modos e dependências dos argumentos de um programa Prolog [AZE98a]. Este trabalho apresenta a integração destes dois modelos ao Granlog visando a sua automatização.

O artigo possui a seguinte organização. A segunda seção mostra uma visão sobre os modelos Granlog, ParTy e MODA. A terceira seção apresenta os pontos relevantes das integrações ParTy-Granlog e MODA-Granlog. A última seção apresenta as conclusões deste trabalho.

2 Descrição dos Modelos

2.1 Modelo Granlog

A análise de granulosidade realizada pelo Granlog consiste no estudo dos tamanhos dos grãos, ou seja, a determinação da complexidade dos módulos seqüenciais. A partir da determinação desta informação o escalonador pode tomar decisões quanto a execução paralela ou seqüencial destes módulos. O Granlog gera informações de granulosidade para auxiliar na paralelização de programas em lógica [BAR94], [BAR96].

O modelo é composto de três módulos: Analisador Global (AGL), Analisador de Grãos (AGR) e Analisador de Complexidade (AC). A figura 1 apresenta os módulos do Granlog.

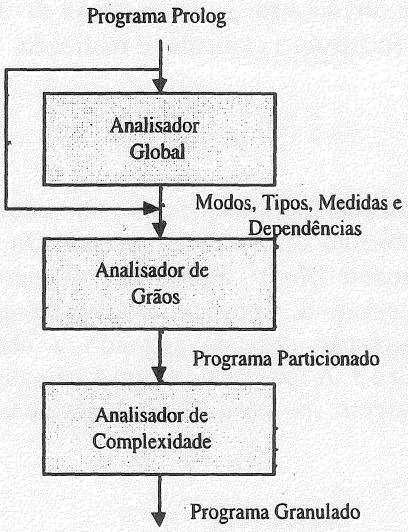

Figura 1 - Módulos do Granlog

O AGL realiza a análise de modos, tipos, medidas e dependências dos argumentos de um programa Prolog. Esta tarefa pode ser realizada através da técnica de análise global denominada interpretação abstrata. O domínio abstrato sob o qual realiza-se a interpretação abstrata representa o escopo onde deseja-se fazer a análise do comportamento dos argumentos. O domínio abstrato do Granlog utiliza quatro modos de argumentos, ou seja: entrada (*i*), saída (*o*), entrada/saída (*io*) e indefinido (*?*). O domínio abstrato de tipos para o modelo Granlog é o seguinte: inteiro (*int*), lista (*list*), estrutura (*struct*), ponto-flutuante (*float*), átomo (*atom*), variável (*var*), indefinido (*?*) e entrada/saída (*io*). As medidas de tamanho utilizadas pelo Granlog são as seguintes: valor de um inteiro (*int*), tamanho de lista (*length*), número de constantes em um termo (*size*), profundidade de uma estrutura (*depth*) e irrelevante (*void*) [BAR96a]. Segundo Barbosa o modelo Granlog necessita que o programador adicione as informações de modos, tipos e medidas ao código fonte [BAR96a]. A figura 2 apresenta o exemplo destas anotações para uma configuração do predicado *append*.

```
%-----------------------------------------------------
:- mode(append/3, [i, i, o]).
:- type(append/3, [list(?, [int])]).
:- measure(append/3, [length]).
%-----------------------------------------------------
append ([ ], L, L).
append ([HL], L1, [HR]) :- append (L, L1, R).
```

Figura 2 - Possível anotação para o *append*

Com base nessas informações fornecidas pelo programador, o sistema Granlog realiza uma análise de dependências conforme descrito em [BAR96a]. Essa análise não utiliza a técnica de interpretação abstrata o modelo cria um grafo de dependências, gerado a partir da estrutura das cláusulas e dos

modos dos procedimentos, o qual é obtido através da anotação de modos adicionada ao código-fonte. Desse modo, apenas a análise local (cláusula a cláusula) é realizada.

2.2 Modelo ParTy

O modelo ParTy prove estaticamente informações dos tipos dos argumentos de programas Prolog [CAS97], [CAS97a]. O domínio abstrato do ParTy é apresentado a seguir: ausência de informação (*$bottom$*), inteiro (*int*), ponto-flutuante (*float*), átomo (*atom*), estrutura (*struct*), lista (*list*), qualquer tipo simples do domínio (*top*), intervalo de tipos (*[minor_type, major_type]*).

Este modelo realiza a interpretação abstrata segundo a abordagem de compilação abstrata [DEB88], [DEB94] conforme a figura 3. A técnica de compilação abstrata baseia-se na substituição das operações primitivas de uma linguagem (no caso Prolog) por suas respectivas operações primitivas abstratas.

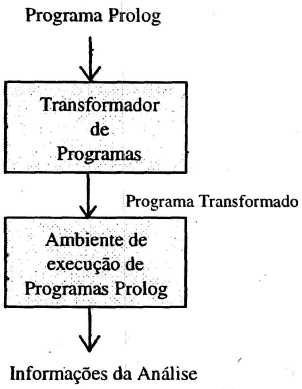

Figura 3 - Módulos do ParTy

Após esta etapa o programa transformado é acrescido do ponto de entrada para a realização da análise. O ponto de entrada do ParTy é apresentado na figura 4.

'__an_t_compute_fixpoint'('<nome/aridade>*',['__an_var'(*"variável1"*), '__an_var'(*"variável2"*), '__an_var'(*"variáveln"*)], [('__an_var'(*"variável1"*), tipo da variável1), ('__an_var'(*"variável2"*), tipo da variável2)]*, _).

Figura 4 – Ponto de entrada para o ParTy

O símbolo "_" no final da pergunta representa a variável *n*, a qual deseja-se saber o tipo após a execução do ParTy.

Após a inclusão do ponto de entrada o programa transformado é compilado e executado e desta execução resulta a análise desejada. A figura 5 exemplifica o padrão das informações geradas pelo ParTy para uma configuração do predicado *append*.

```
..... append/3
Xin= _an_t_pattern (list ($top$, int), list(5, atom(10)), $bottom$)
Xout= _an_t_pattern (list ($top$, int), list(5, atom(10)), list($top$, interval(int, atom(10))))
```

Figura 5 - Informações geradas pelo ParTy

2.3 Modelo MODA

O modelo MODA prove estaticamente informações de modos e dependências dos argumentos de um programa Prolog. A figura 6 mostra a estrutura deste modelo.

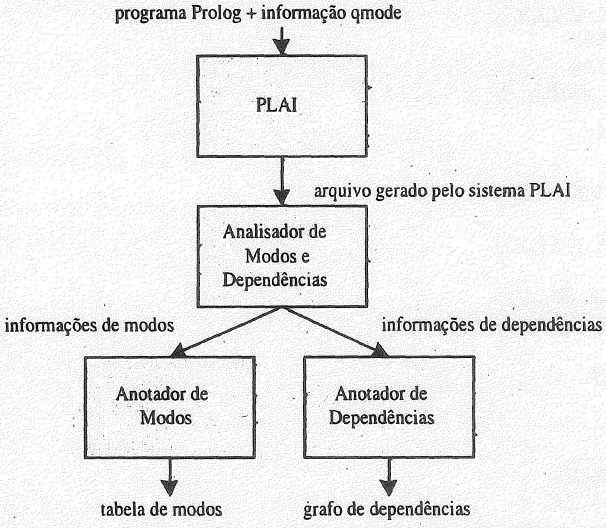

Figura 6 - Estrutura do modelo MODA

O sistema PLAI (*Programming Logic Analyzer Interpreter*), desenvolvido na Universidade Politécnica de Madrid, é um interpretador abstrato para a programação em lógica [LAB92]. Este sistema tem como objetivo fornecer informações de dependências entre as variáveis de um programa.

O PLAI realiza a análise segundo o domínio abstrato *sharing+freeness*. O domínio *sharing+freeness* é composto de um par de conjuntos. O primeiro elemento do par é um conjunto de conjuntos de variáveis que representam o domínio abstrato *sharing*. O segundo elemento é um conjunto de variáveis que representam o componente *freeness*.

O componente *sharing* é composto de conjuntos de variáveis. Cada conjunto expressa a substituição abstrata dos argumentos. Uma substituição abstrata é definida como o mapeamento das variáveis do programa para elementos do domínio abstrato. Considerando que λ é o conjunto *sharing*, os conjuntos simbolizam as seguintes condições [COD97]:

- se uma variável X não aparece em nenhum dos conjuntos do componente *sharing*, então esta variável está totalmente instanciada sobre toda substituição concreta;

- se duas variáveis X e Y não aparecem juntas em nenhum conjunto de λ, então os termos a que X e Y estão ligados em tempo de execução nunca compartilham nenhuma variável;
- se X e Y aparecem juntas em algum conjunto de λ, existe a possibilidade destas variáveis compartilharem ao menos uma variável em tempo de execução.

O componente *freeness* fornece informação sobre as variáveis que estarão definitivamente ligadas a termos livres. Então as variáveis que aparecem no componente *freeness* são variáveis não instanciadas.

O sistema PLAI possui como entrada um programa Prolog acrescido, no início do código fonte, da anotação *:- qmode*. A figura 7 mostra o programa *qsort* com esta anotação.

```
:-qmode(qsort(X, Y), ([[Y]], [Y])).

qsort([X|L], R):-
        particion(L, X, L1, L2),
        qsort(L2, R2),
        qsort(L1, R1),
        append(R1, [X|R2], R).

qsort([], []).

particion([X|L], Y, [X|L1], L2):-
        X<Y, !,
        particion(L, Y, L1, L2).

particion([X|L], Y, L1, [X|L2]):-
        particion(L, Y, L1, L2).

particion([], _, [], []).

append([], X, X).

append([H|X], Y, [H|Z]):-
        append(X, Y, Z).
```

Figura 7 - Arquivo de entrada do sistema PLAI

A informação descrita pela notação *qmode* contém dois campos. O primeiro informa o nome do programa a ser analisado. O segundo contém a informação *sharing* e *freeness* para um ponto de entrada, ou seja, para uma determinada consulta.

O módulo Analisador de Modos e Dependências (AMD) do MODA recebe como entrada o arquivo gerado pelo sistema PLAI, o qual contém o código fonte acrescido dos resultados da análise em forma de conjuntos. Após receber este arquivo, o AMD realiza uma análise das informações de dependências gerando informações de modos e dependências correspondentes ao programa.

O módulo anotador de dependências recolhe as informações de dependências geradas pelo módulo AMD e realiza a geração da notação dos grafos de dependências. O anotador de modos recolhe as informações de modos geradas pelo módulo AMD e realiza a geração das informações de modos.

A figura 8 apresenta o padrão das informações geradas pelo MODA para a primeira cláusula do programa *qsort*.

```
qsort/2/1/1                [i, i, o, o]
qsort/2/1/2                [i, o]
qsort/2/1/3                [i, o]
qsort/2/1/4                [i, i, o]

:- dependency(qsort/2/1, [(I, 1), (1, 2), (1, 3), (2, 4), (3, 4), (4,O)]).
```

Figura 8 – Informações geradas pelo MODA

As informações de modos são organizadas em uma tabela contendo dois campos. O primeiro campo contém nome_predicado/aridade/posição_cláusula/posição_meta. O segundo campo a lista de modos. Na primeira linha da tabela de modos mostrada na figura 8, a primeira meta da primeira cláusula de *qsort/2* possui quatro argumentos, onde os dois primeiros são de entrada (i) e os dois últimos são de saída (o). A segunda e a terceira meta possuem dois argumentos, o primeiro é de entrada e o segundo é de saída. A quarta meta possui três argumentos, onde os dois primeiros são de entrada e o último é de saída. A notação de dependências segue o formalismo de teoria dos grafos. O padrão desta informação é:

***:-dependency*(nome_procedimento/aridade/posição_cláusula, [grafo de dependência]).**

A figura 9 mostra o grafo de dependências para a notação de dependência da figura 8.

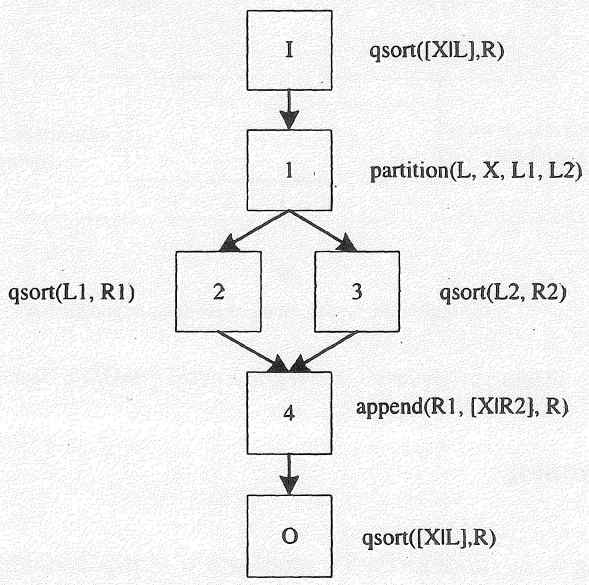

Figura 9 – Grafo gerado pelo MODA

3 Integração ParTy-MODA-Granlog

Esta seção apresenta a integração do modelo ParTy e do modelo MODA ao módulo analisador global (AGL) do modelo Granlog. Este módulo recebe como entrada um programa Prolog mais um ponto de entrada. O ponto de entrada representa a consulta para o início da análise. Esta consulta é relevante, pois os modelos ParTy e MODA realizam a análise segundo uma abordagem *top-down*. O ParTy é responsável pela análise dos tipos dos argumentos de um programa. O MODA é responsável pela análise dos modos e dependências entre as variáveis de um programa. A figura 10 apresenta a estrutura desta integração.

A subseção 3.1 apresenta os pontos relevantes da integração ParTy-Granlog. Por sua vez, a subseção 3.2 descreve a integração MODA-Granlog.

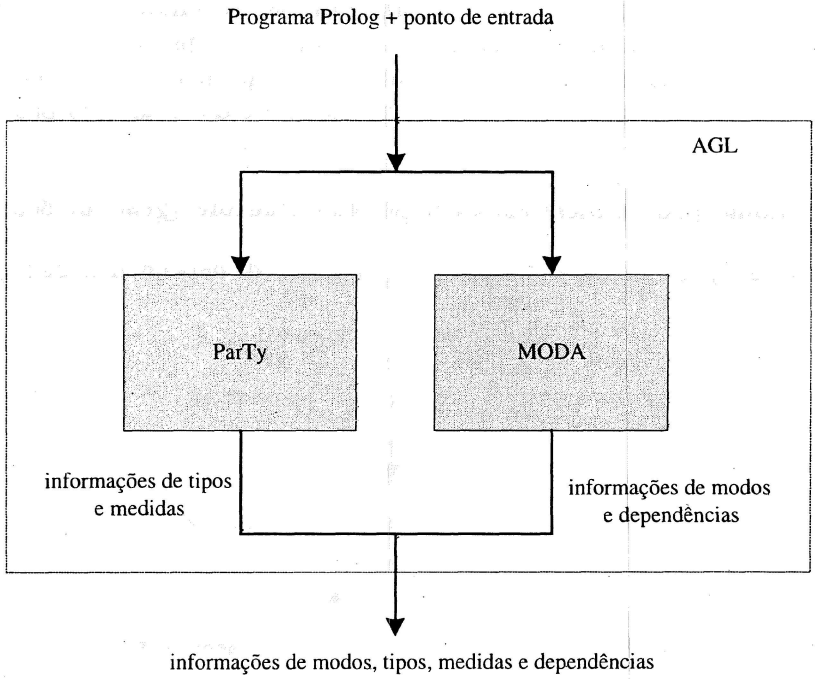

Figura 10 - Estrutura da integração ParTy-MODA-Granlog

3.1 Integração ParTy-Granlog

O modelo Granlog e do modelo ParTy analisam o comportamento das variáveis segundo un escopo diferente. Sendo assim a integração é feita através do conversor ParTy-Granlog, o qual realiza a conversão do domínio abstrato do ParTy para o domínio abstrato do Granlog [AZE98].

O domínio ParTy possui um tipo que representa um intervalo. Este tipo denota uma relação de ordem no domínio de tipos simples. A relação considera a utilização de memória necessária para o armazenamento de cada tipo nas pilhas da máquina abstrata WAM [AÏT90]. A informação da quantidade de memória necessária é relevante para o cálculo dos custos de comunicação entre as partes

de um programa durante a execução paralela em um ambiente distribuído. Neste tipo de ambiente os custos de comunicação assumem grande importância. A máquina abstrata WAM é o padrão das implementações eficientes do Prolog. Em implementações baseadas nesta máquina, o código fonte em Prolog é compilado na linguagem da WAM, a qual é baseada em pilhas [AÏT90].

O conversor ParTy-Granlog recolhe as informações geradas pelo ParTy, analisa e transforma os tipos do domínio ParTy para os tipos correspondentes no domínio Granlog, conforme a tabela de conversão apresentada na figura 11.

ParTy	Granlog
$bottom$	?
int	int
float	float
atom	atom
list	list
struct	struct
top	?
[minor_type, major_type]	major_type

Figura 11 - Tabela de conversão dos domínios

A conversão do tipo intervalar do ParTy em um tipo correspondente no domínio Granlog utiliza a abordagem do pior caso. Este fato deve-se ao modelo Granlog basear-se nesta abordagem.

O tipo top, para o Granlog, não oferece informação alguma sobre o tipo que a variável terá em tempo de execução, por esta razão será convertido para o tipo indefinido do domínio Granlog. O tipo *var* e o tipo *io* do domínio Granlog não possuem correspondentes no domínio ParTy. Porém o tipo *io* é obtido através da comparação das informações geradas pelo ParTy.

Através dos tipos gerados pelo ParTy pode-se obter informações sobre as medidas dos argumentos de um programa Prolog. Por exemplo a análise do predicado *append* percorre os seguintes passos:

I. As informações geradas pelo ParTy são recebidas pelo conversor. A figura 5 apresenta o padrão de saída do ParTy para o predicado *append/3*. Em *Xin*, o primeiro argumento é uma lista de tamanho desconhecido e composta de elementos do tipo inteiro. Esta informação está representada por *list (top, int)*. O segundo argumento é uma lista de tamanho cinco e de elementos do tipo átomo de tamanho dez, representado por *list (5, atom(10))* O terceiro mostra a informação *$bottom$* , indicando que não se sabe nada a respeito deste argumento. Em *Xout* o primeiro argumento é uma lista de tamanho desconhecido e composta de elementos do tipo inteiro. Esta informação está representada por *list (top, int)*. O segundo argumento é uma lista de tamanho cinco e de elementos do tipo átomo de tamanho dez, representada por *list (5, atom(10))*. O terceiro argumento, é uma lista de tamanho indefinido e o tipo dos elementos desta lista pode ser qualquer tipo do intervalo, entre inteiro e átomo de tamanho dez.

II. A conversão precisa da comparação das informações de *Xin e Xout*. Existem três casos à serem considerados. O primeiro quando as informações de *Xin* e *Xout* para um mesmo argumento são idênticas. O segundo caso quando *Xin* é *$bottom$* e *Xout* é outro tipo. Neste caso a informação relevante é a de *Xout*. O terceiro caso quando a informação de *Xin* é um tipo diferente de *$bottom$* e diferente do tipo presente em *Xout*. Um exemplo deste caso poderia ser *Xin* igual a *list(3, $bottom$)* e *Xout*, para o mesmo argumento, igual a *list(3, int)*. Este caso representa um argumento do tipo *io*

segundo o modelo Granlog e por este motivo precisa da notação do tipo de entrada (*Xin*) e do tipo de saída (*Xout*).

III. Depois da comparação a informação relevante (*Xin* ou *Xout)* é convertida para os tipos correspondentes no domínio Granlog, usando a tabela de conversão apresentada na figura 11.

IV. As medidas são obtidas através da análise dos tipos dos argumentos, ou seja, se um argumento é um inteiro então sua medida é *int.* Se um argumento tem o tipo *list* então sua medida será *length.* Se o argumento tiver outro tipo assume-se como medida *void.*

Após a conversão as informações geradas são depositadas em uma estrutura de dados do AGL, que descreve todos os dados relevantes dos predicados, denominada lista descritiva. A lista descritiva é passada juntamente com os grafos de dependências ao módulo AGR. A figura 12 mostra as informações geradas pelo conversor ParTy-Granlog para o programa *append.*

```
:- type(append/3, [list(?, [int]), list(5,[atom(10)], list(?, [atom(10)]]).
:- measure(append/3, [length, length, length]).
```

Figura 12 – Informações geradas pelo conversor ParTy-Granlog

3.2 Integração MODA-Granlog

Nota-se que a informação gerada pelo modelo MODA permite somente a obtenção dos modos *i* e *o*. Este característica é incompatível com o domínio abstrato proposto pelo modelo Granlog. Porém, Nai-Wei Lin defende em [LIN93] a utilização de apenas dois modos, ou seja, o modo fechado e o modo aberto. Outra característica defendida por Nai-Wei Lin é a de um procedimento na programação em lógica poder ser chamado por múltiplos modos. Apesar desta característica não ser comum, a detecção de múltiplos modos propicia uma informação mais precisa e abrangente sobre os modos de um procedimento.

Segundo Barbosa o Granlog não considera a informação de múltiplos modos. Porém, a geração deste tipo de informação foi proposta pelo modelo como uma melhoria a ser realizada em trabalhos futuros [BAR96a]. Sendo assim, a integração MODA-Granlog gera informações de múltiplos modos para o modelo Granlog.

O modelo MODA foi desenvolvido visando um futura integração ao modelo Granlog. Esta característica torna a integração MODA-Granlog uma tarefa simples, ou seja, realizada através de um *parser* que recolhe as informações de modos e dependências geradas pelo MODA e deposita estas informações na estrutura de dados do AGL (lista descritiva), a qual também recebe as informações de tipos e medidas obtidas pela integração ParTy-Granlog.

4 Conclusão

O modelo de integração apresentado neste trabalho viabiliza a automatização do módulo analisador global do modelo Granlog. A utilização da técnica de interpretação abstrata permite a obtenção em tempo de compilação de várias características dos programas Prolog, tais como: modos, tipos, medidas e

dependências das variáveis. Essas características são utilizadas na determinação do fluxo de controle do programa e no cálculo dos custos de comunicação em um ambiente distribuído.

As principais conclusões obtidas através da realização deste trabalho são as seguintes:

- a diferença entre os domínios ParTy/Granlog requer a realização de uma conversão entre estes domínios. A conversão do tipo intervalar do ParTy para um tipo correspondente no Granlog enfoca a abordagem do pior caso. Esta característica pode causar uma perda na precisão da análise;
- a utilização da abordagem do pior caso oferece a informação do maior tipo que uma variável pode assumir em tempo de execução e, consequentemente o maior espaço de memória que poderá ocupar na WAM. Como a informação de tipo serve para calcular o custo computacional de transferência da variável de um procedimento para outro, esta abordagem oferece uma informação mais abrangente sobre o custo necessário para a transferência. No entanto, em alguns casos esta abrangência pode ocasionar a não transferência da variável em vista dos custos serem altos;
- a obtenção de múltiplos modos representa uma melhoria ao modelo Granlog;
- a obtenção das informações de dependências através da técnica de interpretação abstrata proporciona uma análise mais precisa;

Como trabalhos futuros propõe-se o desenvolvimento de um interpretador abstrato específico para a análise de medidas. O desenvolvimento deste interpretador é importante devido ao fato da informação de medidas auxiliar no cálculo dos custos de comunicação.

Referências Bibliográficas

[AÏT90] AÏT-KACI, H. **The WAM: A (Real) Tutorial.** Paris, Digital Equipment Corporation, 1990.

[AZE98] AZEVEDO, S. C.; BARBOSA, J. L. V.; GEYER, C. F. R. **Integração ParTy-Granlog: Interpretação Abstrata Aplicada a Paralização de Programas em Lógica.** Neuquen, anais: IV Congresso Argentino de Ciência da Computação, p. 551-560, 1998.

[AZE98a] AZEVEDO, S. C.; BARBOSA, J. L. V.; COSTA, C. A.; YAMIN, A. C. **MODA: Um Analisador Estático de Modos e Dependências para Programação em Lógica.** Pelotas, Universidade Católica de Pelotas, 1998. (trabalho de graduação).

[BAR94] BARBOSA, J. L. V.; WERNER, O.; GEYER, C. F. R. **Automatic Granularity Analysis in Logic Programming.** Zürich, Institut Für Informatik der Universität Zürich, In: Tenth Logic Programming Workshop, p. 85-88, 1994.

[BAR96] BARBOSA, J. L. V.; GEYER, C. F. R. **GRANLOG: Um Modelo para Análise Automática de Granulosidade na Programação em Lógica.** Porto Alegre, GPGCC-UFRGS, 1996. (Dissertação de Mestrado).

[BAR96a] BARBOSA, J. L. V.; GEYER, C. F. R. **Análise Global na Programação em Lógica.** Belo Horizonte, anais: I Simpósio Brasileiro de Linguagens de Programação, p. 89-102, 1996.

[BRI90] BRIAT, J.; FAVRE, M.; GEYER, C. et al. **Opera: a Parallel Prolog system and its Implementation on Supernode.** Grenoble, Laboratorie de Genie Informatique de Grenoble/CAP-Gremini-Inovation, 1990. (Technical Report).

[CAS97] CASTRO, L. F. P.; GEYER, C. F. R. **Um Modelo de Analisador Estático Baseado na Interpretação Abstrata Direcionado à Paralelização de Programas em Lógica.** Porto Alegre, CPGCC-UFRGS, 1997. (Dissertação de Mestrado).

[CAS97a] CASTRO, L. F., GEYER, C. F. R. **ParTy – A Parallel Types Analyzer.** [S.L], In: Workshop on Parallelism and Implementation Technology for Logic Programming Languages, 1997.

[COD97] CODISH, M.; LAGOON, V.; BUENO, F. **An Algebraic Approach to Sharing Analysis of Logic Programs.** Paris, In: IV International Symposium SAS'97, 1997.

[COR97] CORTESI; FILÉ, G.; GIACOBAZZI, R.; PALAMIDESSI, C.; RANZATO, F. **Complementation in Abstract Interpretation.** New York, In: ACM Transactions on Programming Languages and Systems, v. 19, n. 1, p.7-47, 1997.

[DAM97] DAMS, D.; GERTH, R. **Abstract Interpretation of Reactives Systems.** New York, In: ACM Transactions on Programming Languages and Systems, v. 19, n. 2, p.253-291, 1997.

[DEB88] DEBRAY, S. ; WARREN, D. S. **Automatic Mode Inference for Prolog Programs.** New York, In: Journal of Logic Programming, v.5, n.3, p.207-229, 1988.

[DEB94] DEBRAY, S.; RAMAKRISHNAN, R. **Abstract Interpretation of logic Programs Using Magic Transformations.** [S.L.], In: Journal of Logic Programming, p.149-176, 1994.

[KER94] KERGOMMEAUX, J. C.; CODOGNET, P. **Parallel Logic Programming Systems.** Grenoble, Université Joseph Fourier-Grenoble I, 1994. (Technical Report).

[LAB92] LA BANDA, M. J. G.; **Implementación de um Intérprete Abstrato de Programas Prolog sobre el Dominio Sharing+Freeness.** Madrid, Universidad Polotécnica de Madrid, 1993. (Trabajo Fin de Carreira).

[LAB96] LA BANDA, M.; HERMENEGILDO, M.; BRUYNOOGHE, M.; DUMORTIER, V.; JANSSENS, G.; SIMOENS, W. **Global Analysis of Constraint Logic Programs.** New York, ACM Transactions on Programming Languages and Systems, v. 18, n. 5, p.564-614, 1996.

[LIN93] LIN, N. **Automatic Complexity Analysis of Logic programs.** Arizona, University of Arizona, 1993. (PhD Tesis).

[MOR96] MOREL, É.; et al. **Side-effects in PloSys or-parallel Prolog on distributed memory machines.** [S.L.], In: Compulog Net Meeting on Parallelism and Implementation Technology, 1996.

[YAM94] YAMIN, A. C. **Um Ambiente para Exploração do Paralelismo na Programação em Lógica.** Porto Alegre, CPGCC-UFRGS, 1994. (Dissertação de Mestrado).