Integração ParTy-Granlog: Interpretação Abstrata Aplicada a Paralelização de Programas em Lógica

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
Luis Fernando Pias de Castro
{geyer, lfcastro}@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 e Granlog. O ParTy (Parallel Types Analyzer) é um interpretador abstrato de tipos. 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. Atualmente a análise de modos, tipos e medidas não é automática, o programador adiciona estas informações ao código fonte.

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 para os argumentos de um programa Prolog. A integração ParTy-Granlog torna automática a análise de tipos no modelo Granlog.

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

ABSTRACT

This work presents the integration of the ParTy and Granlog models. ParTy (Parallel Types Analyzer) is a static analyser of types, based on abstract interpretation. Granlog (Granularity Analyzer for LOGic programming) is an automatic granularity analyzer for Logic Programming.

Granlog needs the static analysis of modes, types, measures and dependences over the arguments of a Prolog program. At the present moment, modes, types and measure analysis aren't automated, and thus programs have to be annoted by the programmer.

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 concerning its behavior. ParTy accomplishes the abstract interpretation of types for the arguments of a Prolog program. The integration between ParTy and Granlog provides for automatic type analysis in the Granlog model.

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. 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, permitindo a exploração automática do paralelismo ([KER 94]).

Para obter um melhor desempenho no processo de paralelização, em um programa Prolog, é preciso a análise do comportamento das variáveis em relação aos tipos de elementos a que se instanciam. Isto se deve ao fato das variáveis, na programação em lógica, não serem tipadas. Esta característica da linguagem Prolog dificulta a determinação dos custos de comunicação entre as partes de um programa. A preocupação com os custos de comunicação deve-se ao fato deste trabalho se encontrar dentro do âmbito do Projeto Opera, o qual opta pela exploração do paralelismo em um ambiente de memória distribuída ([BRI 90], [MOR 96], [YAM 94]).

O Granlog é um Analisador Automático de Granulosidade na Programação em Lógica. Para o Granlog realizar a análise de granulosidade é necessária a interpretação abstrata de modos, tipos, medidas e dependências dos argumentos de um programa Prolog. O ParTy é um Interpretador Abstrato de tipos para argumentos de um programa Prolog. Por esta razão o trabalho propõe a integração ParTy-Granlog visando a automatização da análise de tipos no modelo Granlog.

O artigo possui a seguinte organização. A segunda seção mostra uma visão sobre o modelo Granlog. A terceira seção apresenta um resumo sobre o modelo ParTy. A quarta seção aborda os pontos relevantes para a integração ParTy-Granlog e a última seção apresenta as conclusões do trabalho.

2 - Granlog

O Granlog (GRanularity ANalyzer for LOGic programming) é um analisador automático de granulosidade na Programação em Lógica ([BAR 96]). A análise de granulosidade consiste no estudo dos tamanhos dos grãos, ou seja, a determinação da complexidade adequada para os módulos que deverão ser executados seqüencialmente em um único processador. O Granlog gera informações de granulosidade para auxiliar na paralelização de programas em lógica ([BAR 94]), [BAR 96]).

O modelo é composto de três módulos: o Analisador Global (AGL), o Analisador de Grãos (AGR) e o Analisador de Complexidade (AC) como apresentado na figura 1.

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. Atualmente a análise de modos, tipos e medidas não é automática conforme proposto pelo modelo Granlog.

As informações de modos, tipos, medidas e dependências dos argumentos de um Programa Prolog são importantes, pois na programação em lógica não há distinção entre argumentos de entrada e saída (modos), o que dificulta a determinação do fluxo de controle, e os argumentos não são tipados (tipos), o que dificulta o cálculo dos custos de comunicação ([BAR 94], [BAR 96]).

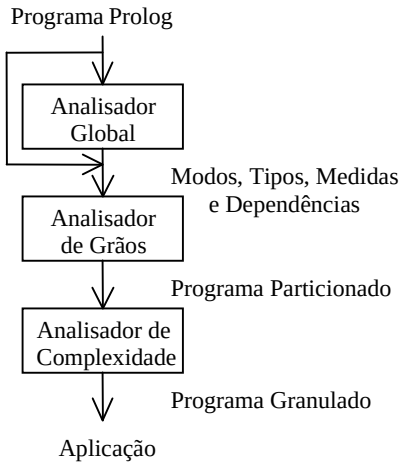

Figura 1 - Módulos do Granlog

3 - ParTy

A Interpretação Abstrata gera informações, segundo um domínio abstrato, sobre características do comportamento das variáveis do programa ([LAB 92], [DEB 94], [LAB 96], [COR 97], [DAM 97]).

O ParTy (Parallel Types Analyser) é um interpretador abstrato de tipos para os argumentos de programas Prolog ([CAS 96], [CAS 97a], [CAS 97b]). Este modelo realiza a interpretação abstrata segundo a abordagem de Compilação Abstrata ([DEB 88], [DEB 94]) conforme a figura 2. A técnica de compilação abstrata se baseia na substituição das operações primitivas de uma linguagem (no caso Prolog) por suas respectivas operações primitivas abstratas. O processo de transformação ocorre através de um único passo sobre o programa analisado. Após esta etapa o programa transformado será compilado e desta compilação resultará a análise desejada.

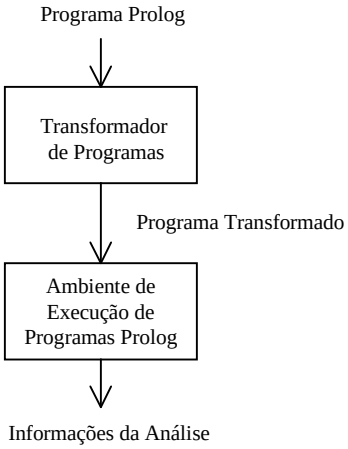

Figura 2 - Módulos do ParTy

4 - Integração ParTy-Granlog

A obtenção estática das informações de modos, tipos e medidas dos argumentos de um programa Prolog, torna automática a análise de granulosidade feita pelo Granlog.

A integração ParTy-Granlog torna transparente a análise de tipos. Isto reduz o número de informações que serão inseridas no código-fonte, facilitando assim o trabalho do programador.

4.1 - O módulo Analisador Global do Granlog

A figura 1 apresenta os três módulos do modelo Granlog. O Analisador Global recebe como entrada um programa Prolog acrescido das **anotações de modos, tipos e medidas** e tem como saída a **lista descritiva e o grafo de dependência,** como mostra a figura 3 ([VAR 95]).

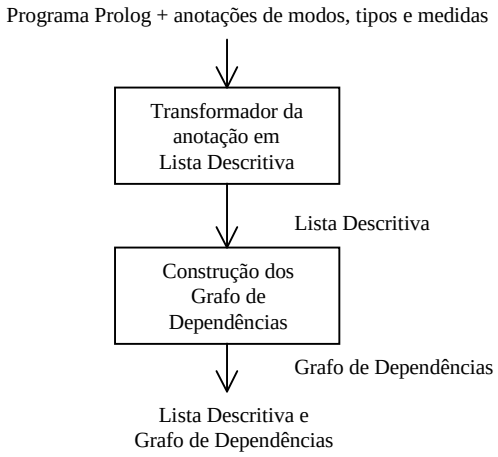

Figura 3 – Organização do módulo Analisador Global

Atualmente as informações de modos, tipos e medidas são inseridas no código-fonte pelas seguintes notações:

:- mode(p/n, lista_de_modos).
:- type(p/n, lista_de_tipos).
:- measure(p/n, lista_de_medidas).

Onde **p** representa o nome do predicado e **n** sua aridade. A **lista_de_modos** contém os modos dos argumentos do predicado, a **lista_de_tipos** os tipos e a **lista_de_medidas** as medidas. A figura 4 apresenta o exemplo das anotações de modos, tipos e medidas 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 4 – Possível anotação para o *append*

A notação *mode(append/3, [i, i, o])*, na figura4, indica que o primeiro e o segundo argumento são de entrada (i) e o terceiro argumento é de saída (o). A notação *type(append/3, [list(?, [int])])* indica que os argumentos são listas de tamanho indefinido e do tipo inteiro. A notação *measure(append/3, [length])* indica que a medida dos argumentos é length, isto é, os argumentos são mensurados pelo número de elementos que compõem a lista.

A interpretação abstrata é realizada segundo um domínio abstrato que representa o escopo sob o qual deseja-se fazer o estudo do comportamento das variáveis

do programa ([CAS 97a]). O domínio abstrato sob o qual o Granlog analisa as variáveis de um programa Prolog é o seguinte:

- **?:** indica que o argumento não pode ser definido.
- **int:** indica que o argumento é um inteiro.
- **float***:* indica que o argumento é um número de ponto-flutuante.
- **atom(size):** indica que o argumento é um átomo cujo parâmetro *size* contém o número de caracteres do identificador do átomo.
- **var:** o argumento é uma variável livre.
- **list(size, types)***:* indica que o argumento é uma lista cujo *size* é o número de elementos que compõe a lista e *types* é uma lista contendo o tipo de cada um dos argumentos.
- **struct(functor, elements, types)***:* o argumento é uma estrutura cujo *functor* contém o número de caracteres do seu identificador. *Elements* determina o número de elementos da estrutura. *Types* é uma lista contendo o tipo de cada um dos argumentos.
- **io(entrada, saída):** indica que o argumento possui duas notações de tipos. O parâmetro *entrada* descreve a instanciação antes da chamada. O parâmetro *saída* indica o estado de instanciação após a chamada.

4.2 - O ParTy

O formato das informações geradas pelo ParTy são relevantes para a integração. A figura 5 exemplifica o padrão destas informações para o predicado *append*.

Para o modelo Granlog utilizar tais informações é preciso haver a conversão do domínio abstrato do ParTy para o domínio abstrato do Granlog, já que estes analisam o comportamento das variáveis segundo um escopo diferente. O domínio abstrato do ParTy é apresentado a seguir:

- **$bottom$:** ausência de informação a respeito da variável. Menor elemento do conjunto de tipos.
- **int:** representa elementos do tipo inteiro.
- **float:** representa elementos do tipo ponto-flutuante.
- **atom(size)***:* representa átomos de tamanho *size.*
- **struct(size, arity, [list_of_types]):** representa estruturas de aridade *arity*, cujos argumentos têm os tipos descritos na lista *[list_of_types]*, e cujo functor tem tamanho *size.*
- **list(size, type):** representa listas de tamanho *size* e tipo *type.*
- **top:** representa variáveis que podem ser ligadas a qualquer tipo de objeto em tempo de execução.
- **[minor_type, major_type]:** representa um intervalo de tipos.

```
. . . . . 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

O domínio abstrato do ParTy possui um tipo que representa um intervalo. Este tipo denota uma relação de ordem no domínio de tipos simples. A figura 6 apresenta a ordem dos tipos. A relação considera a utilização de memória necessária para o armazenamento de cada tipo nas pilhas da máquina abstrata WAM ([AÏT 90]). 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 uma das implementações mais eficientes do Prolog. Em implementações baseadas na WAM, o código fonte em Prolog é compilado na linguagem da WAM, a qual é baseada em pilhas. Um emulador portável desta máquina abstrata, permite um rápido e portável sistema Prolog ([YAM 94]).

1. $bottom$
2. int
3. float.
4. atom(size).
5. struct(size, arity, [list_of_types]).
6. list(size, type).
7. top.

Figura 6 - Relação de ordem do domínio abstrato do modelo ParTy

4.3 - Proposta de Integração

A integração ParTy-Granlog é feita através de um conversor que recolhe as informações geradas pelo ParTy, analisa e transforma os tipos do domínio ParTy para os tipos correspondentes no domínio Granlog, conforme mostra a figura 7.

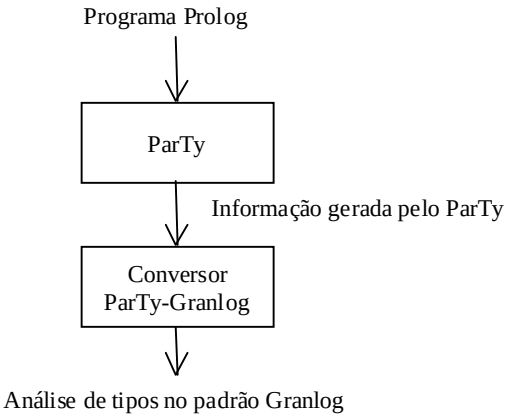

Figura 7 - Representação modular do modelo

As informações geradas pelo conversor são depositadas em uma estrutura de dados do AGL denominada Lista Descritiva. A Lista Descritiva é utilizada pelo AGL para geração dos Grafo de Dependências. A figura 8 apresenta uma ampla visão da integração ParTy-Granlog.

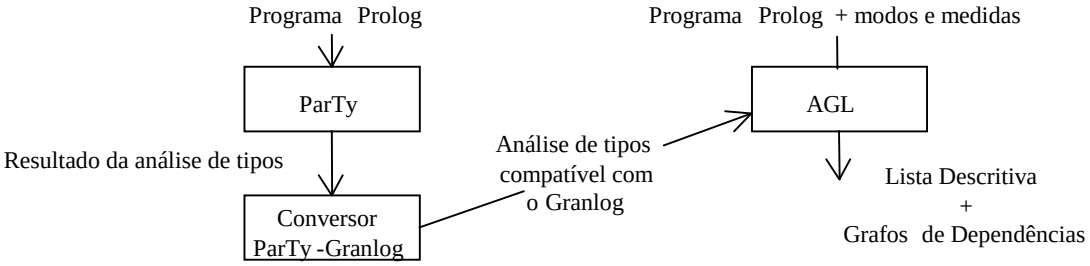

Figura 8 - Integração ParTy-Granlog

A conversão de tipos entre os domínios ParTy/Granlog é realizada através da Tabela de Conversão apresentada na figura 9.

ParTy	Granlog
$bottom$	?
int	int
float	float
atom(size)	atom(size)
list(size, type)	list(size, type)
struct(size,arity,[list_of_types])	struct(functor, elements, types)
top	?
[minor_type, major_type]	maior tipo do intervalo

Figura 9 - Tabela de conversão dos domínios

A conversão do tipo intervalar do ParTy em um tipo correspondente no domínio Granlog, considera duas abordagens, ou seja, o melhor caso e o pior caso.

A primeira opta em trabalhar com o menor valor do intervalo, ou seja, o menor tipo que uma variável pode assumir em tempo de execução. Esta é uma abordagem otimista, porém de pouco auxílio, pois saber o tamanho mínimo que uma variável ocupa na máquina abstrata WAM não contribui para um cálculo correto dos custos de transporte desta variável.

A segunda abordagem 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, conclui-se que trabalhar com a abordagem do pior caso é mais útil, pois oferece uma informação mais abrangente sobre o custo necessário para a transferência.

O tipo top, segundo o domínio do 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 entre as informações de *Xin* e *Xout.*

4.4 - A Conversão

A conversão é realizada através da análise das informações presentes em *Xin e Xout*, geradas pelo ParTy. 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 esta 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 é quando as informações de *Xin* e *Xout* para um mesmo argumento são idênticas. O segundo caso é quando *Xin* é *$bottom$* e *Xout* é outro tipo. Neste caso a informação relevante é a de *Xout.* O terceiro caso é 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 9. A figura 10 mostra o resultado da conversão do predicado *append/3*.

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

Figura 10 - Conversão da análise do predicado *append*

5 - Conclusões

O modelo de integração apresentado neste trabalho viabiliza a análise automática dos tipos dos argumentos no sistema Granlog.

A utilização da técnica de Interpretação Abstrata permite a obtenção em tempo de compilação de várias características da linguagem, 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**, isto pode causar uma perda na precisão da análise.
- Além disso a abordagem do pior caso pode ocasionar uma perda na precisão dos cálculos dos custos de transporte das variáveis.

A completa automatização do Granlog necessita da obtenção das informações de modos e medidas através da Interpretação Abstrata. Por este motivo propõe-se como trabalhos futuros a criação de um Interpretador Abstrato que faça a análise de modos e medidas para o modelo Granlog.

Referências Bibliográficas

[AÏT 90] AÏT-KACI, H. **The WAM: A (Real) Tutorial.** Paris: Digital Equipment Corporation, 1990.

[BAR 94] BARBOSA, J. L. V.; WERNER, O.; GEYER, C. F. R. **Automatic Granularity Analysis in Logic Programming.** Zürich: Institut Für Informatik der Universität Zürich, 1994, Proceedings of the Tenth Logic Programming Workshop, p. 85-88.

[BAR 96] BARBOSA, J. L. V. **GRANLOG: Um Modelo para Análise Automática de Granulosidade na Programação em Lógica.** Porto Alegre: GPGCC-UFRGS, 1996. (Dissertação de mestrado).

[BRI 90] 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).

[CAS 96] CASTRO, L. F. P. **Interpretação Abstrata e sua Aplicação na Programação em Lógica.** Porto Alegre: CPGCC-UFRGS, 1996. (Relatório Técnico).

[CAS 97a] CASTRO, L. F. P. **Um Modelo de Analisador Estática Baseado na Interpretação Abstrata Direcionado à Paralelização de Programas em Lógica.** Porto Alegre: CPGCC-UFRGS, 1997. (Dissertação de Mestrado).

[CAS 97b] CASTRO, L. F., GEYER, F. R. **ParTy – A Parallel – Types Analyzer.** [S.L]: Workshop on Parallelism and Implementation Technology for Logic Programming Languages, 1997.

[COR 97] CORTESI; FILÉ, G.; GIACOBAZZI, R.; PALAMIDESSI, C.; RANZATO, F. **Complementation in Abstract Interpretation.** New York: ACM Transactions on Programming Languages and Systems, v. 19, n. 1, p.7-47, 1997.

[DAM 97] DAMS, D.; GERTH, R. **Abstract Interpretation of Reactives Systems.** New York: ACM Transactions on Programming Languages and Systems, v. 19, n. 2, p.253-291, 1997.

[DEB 88] DEBRAY, S. ; WARREN, D. S. **Automatic Mode Inference for Prolog Programs.** New York: Journal of Logic Programming, v.5, n.3, p.207-229, 1988.

[DEB 94] DEBRAY, S.; RAMAKRISHNAN, R. **Abstract Interpretation of logic Programs Using Magic Transformations.** [S.L.]: Journal of Logic Programming, p.149-176, , 1994.

[KER 94] KERGOMMEAUX, J. C.; CODOGNET, P. **Parallel Logic Programming Systems.** Grenoble: Universite Joseph Fourier-Grenoble I, 1994. 52p. (Technical Report).

[LAB 92] LA BANDA, M. G. **Implementación de un Intérprete Abstracto de Programas Prolog sobre el Dominio Sharing + Freeness.** Madrid: Universidad Politecnica de Madrid, 1992. (Trabajo Fin de Carreira).

[LAB 96] 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, 1996, v. 18, n. 5, p.564-614.

[MOR 96] 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. Proceedings... [S.l.:s.n.], 1996.

[VAR 95] VARGAS, P. K. **Implementação de um Analisador de Granulosidade para Prolog.** Porto Alegre: Universidade Federal do Rio Grande do Sul, 1995. (Trabalho de Conclusão de Curso).

[YAM 94] YAMIN, A. C. **Um Ambiente para Exploração do Paralelismo na Programação em Lógica.** Porto Alegre: CPGCC-UFRGS, 1994. (Dissertação de Mestrado).