Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Laboratório de Linguagens de Programação

# Um Ambiente para Desenvolvimento de Programas em Machĭna Baseado no Paradigma do Estilo Literário [1]

por

Nahur Moraes da Fonseca
Mariza A. S. Bigonha

Relatório Técnico do Laboratório de
Linguagens de Programação LLP009/2000

Av. Antônio Carlos, 6627
31270-010 - Belo Horizonte - MG

12 de Abril de 2000

[1] Relatório Final de Iniciação Científica

## Resumo

Este relatório descreve os trabalhos desenvolvidos no âmbito do projeto HyperMachĭna, um ambiente de desenvolvimento de programas Machĭna baseado no paradigma do estilo literário.

Primeiramente é dado um histórico do projeto, que é uma extensão do projeto HyperPro, um ambiente de desenvolvimento de programas em ProLog baseado no paradigma do estilo literário. Em seguida, são apresentadas a metodologia de Máquinas de Estados Abstratas (ASM) e a linguagem Machĭna, uma concretização dessa metodologia. ASM é uma metodologia de especificação formal com forte base matemática e muito semelhante a uma especificação de algoritmos em uma linguagem convencional como C. Por fim, nós detalhamos a implementação da ferramenta HyperMachĭna, que foi baseada no editor de documentos estruturados *Thot*. Por questão de completeza, o sistema *Thot* também é apresentado neste relatório.



## Abstract

Firstly we give a brief historic about this project, which is an extension of the HyperPro project, an environment to develop programs written in ProLog based on the literate programming paradigm. Then, we present Abstract State Machines (ASM) and Machĭna, a programming language that concretize the ASM methodology. ASM is a methodology to formally specify systems. It is based on rigorous mathematical methods and is very similar to algorithms written in ordinary programming languages. Finnally, we give the details of the implementation of HyperMachĭna, which was based on the structured documents editor *Thot*. The *Thot* system is also presented in this report due to completeness.

In this report, we describe the work developed in the HyperMachĭna project, an environment to develop programs written in Machĭna based on the literate programming paradigm.

# 1 Introdução

Este documento descreve os trabalhos desenvolvidos no âmbito do projeto HyperMachĭna, um ambiente para desenvolver programas em Machĭna baseado no paradigma de estilo literário. Primeiramente, é apresentado um breve histórico desse projeto e então são apresentadas a linguagem Machĭna, as estruturas utilizadas bem como a implementação da ferramenta HyperMachĭna.

A linguagem Machĭna é uma concretização da metodologia de máquinas de estados abstratas (do inglês, *Abstract State Machine*, ou simplesmente ASM) [15, 16]. Um programa escrito em Machĭna, portanto, representa um modelo descrito em ASM. Com a vantagem de que um programa pode ser compilado e executado a fim de testar e verificar um modelo em ASM.

O paradigma de estilo literário [10, 11, 12] consiste em escrever programas juntamente com a sua documentação. A qualquer momento, é possível extrair uma visão que contenha apenas o programa em si, ou uma visão completa, com toda a documentação do programa, título, autores, data, seções, anexos e referências bibliográficas.

O objetivo deste projeto é criar um ambiente de desenvolvimento de programas escritos em Machĭna juntamente com sua documentação. Esse ambiente deve proporcionar acesso a ferramentas de compilação, verificação e execução dos programas em Machĭna.

Nosso trabalho é uma extensão do projeto HyperPro [1, 2, 3, 4, 5], um ambiente de desenvolvimento de programação em lógica baseado no paradigma de estilo literário. O HyperPro utiliza o *Thot* [6]. O *Thot* é um sistema aberto para editoração de documentos estruturados. Ele permite, por meio de programas escritos em um conjunto de linguagens proprietárias, definir novos tipos de documentos. Para se criar um novo tipo de documento, deve-se descrever sua estrutura, isto é, sua organização em termos de título, seções, parágrafos e outros elementos; a forma de apresentação desses elementos estruturais, por exemplo, tamanho e cor da fonte; bem como outras funcionalidades do editor, em particular, as diferentes visões sobre um documento e as ferramentas acessíveis por menus.

# 2 Histórico

Originalmente, minha contribuição neste projeto era finalizar a ferramenta HyperPro. Trabaho este que estava sendo desenvolvido por uma ex-aluna de graduação que se desligou do projeto para ingressar na curso de pós-graduação do Departamento de Ciência da Computação da UFMG (DCC/UFMG). O projeto do HyperPro faz parte do programa de cooperação internacional entre o *Institut National de Recherche en Informatique e Automatique* (INRIA) e o DCC/UFMG. Ele consiste na criação de um ambiente de desenvolvimento integrado de programas escritos em Prolog e CLP (do inglês, *Constraint Logic Programming*) baseado no estilo literário.

Após a conclusão da versão do HyperPro, a minha proposta de atuação seria construir uma ferramenta para desenvolvimento de programas em Java baseado no estilo literário, devido a crescente utilização dessa linguagem em diversas aplicações. No entanto, em paralelo à finalização de HyperPro, estava sendo concluída a definição da linguagem Machĭna pelo grupo de pesquisadores do Laboratório de Linguagens de Programação do DCC/UFMG, e com isso criou-se a necessidade do desenvolvimento de ferramentas para aplicação desta linguagem. A partir das discussões sobre o ambiente de Machĭna, houve um grande interesse na

construção de uma ferramenta para criação e documentação de programas baseados na metodologia de Máquinas de Estado Abstratas durante o III Simpósio Brasileiro de Linguagens de Programação por esse grupo.

A fim de atender a essas expectativas, fez-se a opção por desenvolver a ferramenta HyperMachĭna, descrita neste documento.

## 3 Máquinas de Estados Abstratas

Máquinas de Estados Abstratas, introduzidas por Yuri Gurevich em [15, 16], são usadas para modelar, especificar e verificar sistemas dinâmicos discretos. A metodologia ASM possui base matemática rigorosa, o que permite a demonstração formal de propriedades do sistema especificado. Por outro lado, uma especificação em ASM está mais próxima das implementações que seriam feitas em uma linguagem de programação convencional, o que facilita a sua utilização por programadores experientes, mas que não utilizam qualquer método formal. Outra vantagem é a possibilidade de executar a especificação, o que auxilia na verificação de sua correção em relação ao mundo real.

A importância de ASM pode ser vista pela sua facilidade de compreensão, pela possibilidade de modelar iterações de forma tão direta quanto em uma implementação em uma linguagem de programação convencional, e pela sua fundamentação em métodos matemáticos rigorosos, o que permite a verificação formal de sistemas. Dessa forma, programadores que nunca utilizaram métodos de especificação formal, podem fazê-lo com ASM, da mesma forma que descreveriam um algoritmo em uma linguagem de programação convencional. Não queremos, com isso, provar que as ASMs serão adotadas como solução definitiva para especificações formais. Mas elas são um caminho mais natural para aqueles que ainda não estão familiarizados com essa metodologia.

Existem vários exemplos de especificações feitas com ASM em diferentes áreas, dentre as quais podemos citar: arquiteturas [20, 21, 24], linguagens de programação [25, 26, 27, 29, 32], sistemas distribuídos [18, 19, 22, 30] e de tempo real [28, 31], entre outros [23].

A fim de aplicar a metodologia de ASM para especificação de sistemas reais, foi desenvolvida a linguagem Machĭna.

## 4 Linguagem Machĭna

Machĭna é uma linguagem de programação baseada na metodologia ASM. Ela foi desenvolvida pelo grupo de pesquisadores do Laboratório de Linguagens Programação do DCC/UFMG. Sua definição completa pode ser encontrada em [17].

Um programa escrito em Machĭna descreve uma máquina de estados abstrata. Cada estado é representado pelo conjunto de nomes de funções e relações do estado. Esse conjunto forma o vocabulário do estado. A mudança de estado da máquina é dada por uma regra de transição do estado origem. Uma regra de transição modifica a interpretação de alguns nomes de função do vocabulário formando, dessa forma, um novo estado. Dizemos assim, que um programa em Machĭna tem o estilo de programação algébrica, isto é, um estado de computação da máquina é descrito por uma álgebra, um conjunto de definições, e existem regras de transição que promovem a mudança de estados.

Um sistema real pode ser descrito em ASM por um programa Machĭna. No entanto, um programa Machĭna possui sintaxe própria, mais próxima de uma linguagem de programação

convencional, do que a linguagem que usamos para nos comunicar. Portanto, há a necessidade de ferramentas para desenvolver programas em Machĭna juntamente com a sua documentação, de modo que a especificação de um sistema seja efetiva, tanto em termos da legibilidade para os humanos, quanto em termos de precisão para um sistema de computação. Nesse sentido, este trabalho descreve a implementação da ferramenta HyperMachĭna, um sistema de desenvolvimento de programas em Machĭna baseado no estilo literário [10, 11, 12].

# 5 HyperMachĭna

O sistema HyperMachĭna é uma ferramenta que provê um ambiente para desenvolvimento de programas em Machĭna baseado no paradigma de estilo literário. Portanto, um documento escrito com o HyperMachĭna possui os elementos que definem sua estrutura, por exemplo, título, autores e seções. entremeado por trechos de código de um programa em Machĭna.

Por questão de completeza, antes de descrevermos a estrutura de um documento escrito no HyperMachĭna, bem como a sua forma de apresentação, serão apresentadas as principais características de *Thot*, o sistema de editoração de documentos estruturados utilizado no desenvolvimento de HyperMachĭna. Após esta apresentação, concomitantemente com a definição de um documento HyperMachĭna, iremos mostrando os trechos dos programas utilizados pelo *Thot* para manter essas propriedades do documento.

## 5.1 O Sistema *Thot*

O sistema *Thot* foi desenvolvido por Quint e outros[8, 6]. Ele é um editor de documentos estruturados baseado no paradigma WYSIWYG (do inglês, *What You See Is What You Get*). Com ele é possível criar novos documentos, modificar documentos existentes, extrair informações de um documento ou visualizar partes de um documento. O *Thot* possui um *kit* de ferramentas constituído por bibliotecas escritas em C e cuja interface gráfica usa a biblioteca OSF/Motif.

Um documento é primeiramente considerado como uma estrutura abstrata composta por elementos tipados. Exemplos de elementos tipados são títulos, capítulos, seções, parágrafos e listas. A estrutura é dada pela hierarquia desses elementos.

Para descrever a estrutura, apresentação e outras propriedades de um documento, o sistema *Thot* possui vários modelos, ou linguagens proprietárias descritas a seguir:

**Linguagem S** descreve a estrutura de um documento, dada pela hierarquia dos elementos que o compõe, tais como, título, autores, data e seções.

**Linguagem P** descreve a apresentação de um documento. Por exemplo, o tamanho da fonte usada pelos elementos, a cor, as distâncias horizontal e vertical em relação às margens do documento ou a outro elemento.

**Linguagem A** descreve as funções que estarão disponíveis em forma de menus ou botões para os usuários do editor. Por exemplo, criar documento, salvar documento e salvar documento em outros formatos.

**Linguagem T** descreve como um documento *Thot* deve ser traduzido para outros formatos, por exemplo, HTML, LaTex e ASCII.

Nosso trabalho consiste no desenvolvimento dos programas, ou esquemas, S, P e A para a ferramenta HyperMachĭna, baseado na definição da linguagem Machĭna, em trabalhos sobre estilo literáro de programação e no projeto HyperPro.

## 5.2 Estrutura de um Documento HyperMachĭna

A estrutura de um documento HyperMachĭna deve ser criada em duas etapas. Primeiramente, é feita uma definição em alto nível. Nós baseamos a estrutura de HyperMachĭna, na estrutura de documentos HyperPro e HyperProC. O primeiro se refere à linguagem ProLog e CLP e o último se refere à linguagem C. Em seguida, é feita a implementação do programa .S que especifica a estrutura de um documento HyperMachĭnapara o editor *Thot*.

Três definições de estrutura com diferentes graus de granularidade foram feitas. Uma delas definia cada elemento sintático da gramática de Machĭna. Em outra abordagem, cada módulo de um programa HyperMachĭna devia ser especificado de uma só vez, como uma função em C. Ambas definições possuem deficiências, uma por detalhar demais os elementos, dificulta a usabilidade da ferramenta. A outra, por não permitir definir apenas trechos de programas Machĭna, não possibilita o reuso de código. Por isso, foi criada esta definição, apresentada a seguir, que foi inspirada no trabalho de Knuth [10].

A estrutura de um documento HyperMachĭna possui obrigatoriamente um título e uma ou mais seções. Opcionalmente, um documento pode conter uma data, os autores, a filiação dos autores, as palavras chaves, referências bibliográficas e anexos, conforme pode ser visto neste trecho do programa S:

```
HyperMachina (ATTR First_page_number = Integer;
              First_section_number = Integer) =
   BEGIN
      Document_title  = Lines;
      ? Document_date   = Lines;
      ? Authors         = LIST OF (Author);
      ? Affiliations    = LIST OF (Affiliation = Lines);
      ? Key_words       = Lines;
      Sectio ns_seq;
      ? Bibliography    = LIST OF (Citation_biblio = RefBib);
      ? Annexes         = LIST OF (Annexe);
   END;
```

Exemplo 1

Uma seção no HyperMachĭna é composta pelo título da seção e o corpo da seção. Opcionalmente, ela pode possuir outras sub-seções. O corpo de uma seção é composto por parágrafos ou trechos de código, de acordo com o estilo literário. A distinção entre parágrafos e trechos de código é o que permitirá a criação de visões específicas que permitem mostrar apenas o programa, facilitando o exame do código ou o artigo completo, como ele seria submetido a uma conferência. O seguinte trecho do programa S define uma seção do HyperMachĭna.

```
Section =
   BEGIN
      Section_title = TEXT;
      Section_body =
         LIST OF (Paragraphs_or_Code =
```

```
            CASE OF
               Paragraphs_seq;
               Code_seq;
            END);
      ? Sections_seq;
   END;
```

Exemplo 2

Um parágrafo pode ser quase qualquer coisa. Uma seqüência de palavras, uma imagem, uma fórmula, uma figura, um bloco de texto, ou um objeto de outra natureza. Ele serve para descrever a especificação de um programa. Ele pode vir antes ou após um trecho de código e pode ainda conter referências bibliográficas ou notas de rodapé. A definição deste elemento, bem como a dos elementos que o compõe foram extraídas da ferramenta HyperPro, e podem ser vistas parcialmente no código abaixo:

```
Paragraphs_seq = LIST [0..*] OF (Paragr);
   Paragr =
      CASE OF
         Paragraphe;
         Titled_group =
            BEGIN
               Group_title = Lines;
               Group = LIST OF (Paragr);
            END;
         Image = PICTURE;
         Numbred_formula = Math;
         Figure;
         Other_nature = NATURE;
      END;
```

Exemplo 3

O próximo elemento é o trecho de código. Ele pode ser formado por um ou mais blocos. Cada bloco possui um nome, o qual será usado em futuras referências para esse bloco, por exemplo, para ligar um bloco a outro, formando assim o programa completo. Um bloco possui ainda uma barra de versões, com referências para uma das definições do bloco. Ele ainda possui a definição corrente, um conceito que indica qual a versão de definição está sendo utilizada no momento. A especificação de um trecho de código do HyperMachĭna pode ser vista abaixo.

```
Code_seq = LIST OF (CodeBlock);

CodeBlock =
   BEGIN
      CodeBlockName       = TEXT;
      Versions_bar        = LIST [1..*] OF (Version);
      Current_def_ref     = REFERENCE (Code_def);
      Definitions         = LIST [1..*] OF (Code_def);
   END;
```

Exemplo 4

Uma definição de bloco é um trecho de programa Machĭna. Um programa Machĭna é composto por módulos e dentro de cada módulo podem existir até quatro partes. As partes que compõem um módulo são chamadas de *Import*, *Algebra*, *Transition* e *Invariant*. A parte chamada *Algebra* é usada para definir até seis entidades: *Tipos*, *Funções Estáticas*, *Funções Dinâmicas*, *Funções Externas*, *Ações* e *Inicializações*. A fim de capturar essa estrutura dos programas foi necessário criar os elementos respectivos na estrutura do documento *Thot*. Dessa forma, podem ser feitas projeções sobre o documento, que destaquem apenas uma parte, em particular, um projeção poderia exibir apenas as definições, ou a parte *Algebra* de um programa em Machĭna. Os elementos criados para representar essa estrutura são mostrados a seguir.

```
Code_def =
   BEGIN
      ? Informal_comments = LIST [1..*] OF (Paragr);
      Module_part;
   END;

Module_part =
   CASE OF
      Plain_Code       = LIST [1..*] OF (Code_or_Macro);
      Import_part      = LIST [1..*] OF (Code_or_Macro);
      Algebra_part;
      Transition_part  = LIST [1..*] OF (Code_or_Macro);
      Invariant_part   = LIST [1..*] OF (Code_or_Macro);
   END;

Algebra_part =
   CASE OF
      Type_sec             = LIST [1..*] OF (Code_or_Macro);
      Static_function_sec  = LIST [1..*] OF (Code_or_Macro);
      Dynamic_function_sec = LIST [1..*] OF (Code_or_Macro);
      External_sec         = LIST [1..*] OF (Code_or_Macro);
      Action_sec           = LIST [1..*] OF (Code_or_Macro);
      Init_sec             = LIST [1..*] OF (Code_or_Macro);
   END;
```

Exemplo 5

Por fim, cada um dos trechos de código descritos acima, pode ser um trecho do programa Machĭna, com a sua sintaxe própria, ou uma referência para um outro bloco definido em outra parte do documento. Esse tipo de referência foi chamado de macro por Knuth em [10]. Assim, é possível criar uma cadeia de trechos que juntos formam o programa Machĭna completo. A definição do elemento a seguir ilustra essa idéia.

```
Code_or_Macro =
   CASE OF
      Code_Text = TEXT;
      Macro_ref = REFERENCE (CodeBlock);
   END;
```

Exemplo 6

A estrutura completa de um documento HyperMachĭna pode ser vista na Figura 1.

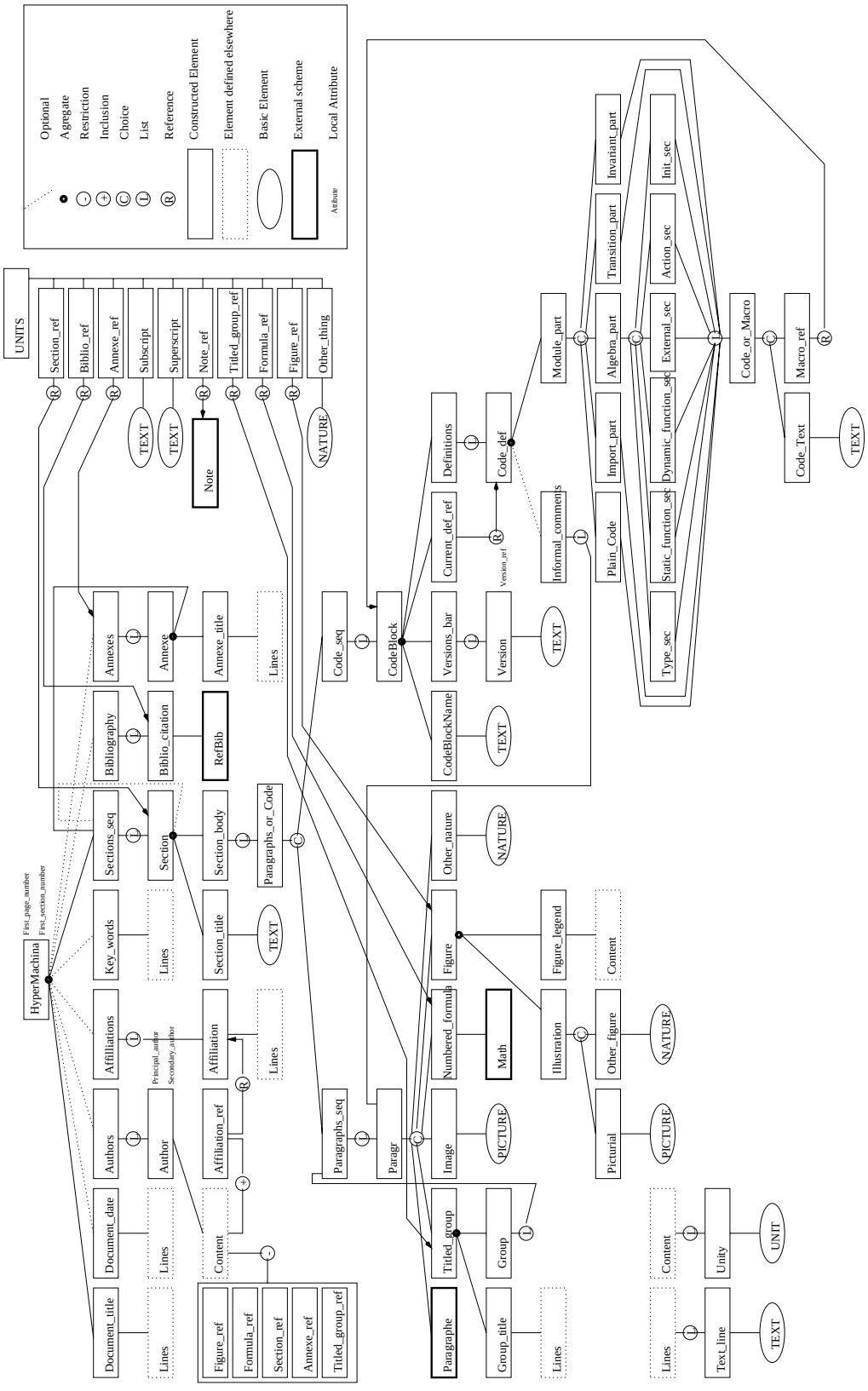


Figura 1: Árvore de Estrutura do HyperMachĭna

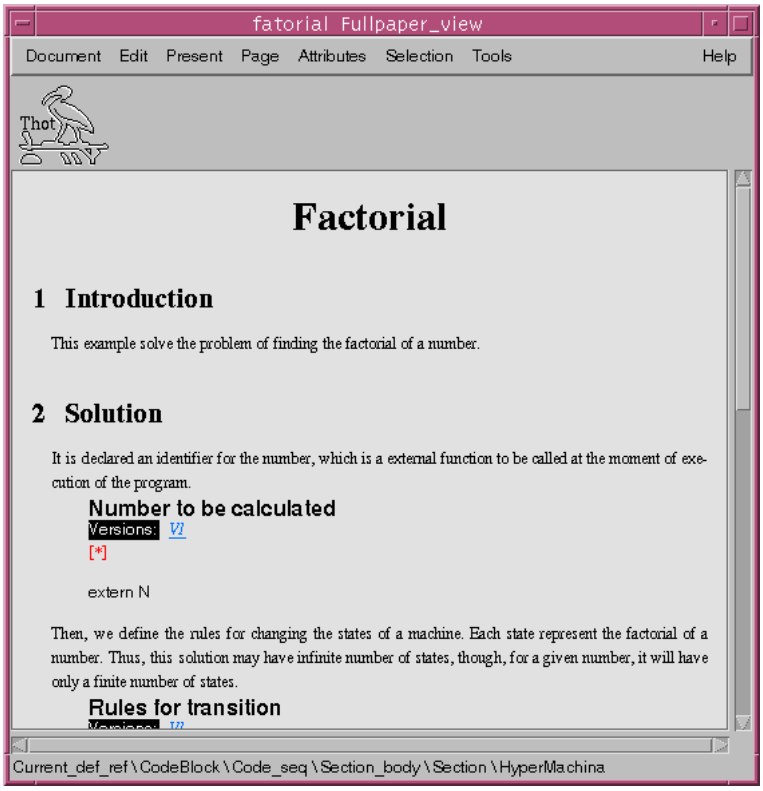


Figura 2: Programa Fatorial

### 5.3 Apresentação de um Documento HyperMachĭna

A apresentação de um documento HyperMachĭna serve para destacar os elementos estruturais do documento. Dentre as propriedades que a linguagem P do *Thot* define para a apresentação dos elementos de um documento podemos citar o tamanho, estilo e nome da fonte, largura e altura da caixa que engloba o elemento e a visibilidade do elemento.

Para ilustrar algumas dessas propriedades associadas aos elementos estruturais de um documento HyperMachĭna, mostramos a seguir alguns exemplos de documentos.

A Figura 2 mostra o título, as primeiras seções, parágrafos e definições de trechos de um programa em Machĭna para calcular o fatorial de um número. Todas as propriedades para diferenciar as fontes de cada elemento estrutural, bem como o alinhamento são definidos em um programa P. O Exemplo 7 mostra a definição de algumas propriedades específicas para o título.

```
    Document_title:
       BEGIN
#ifdef PAGE
          VertPos : Top = Enclosing . Top + 80 pt;
#else
          VertPos : Top = Enclosing . Top;
#endif
          Style : Bold;
          Size : 30 pt;
```

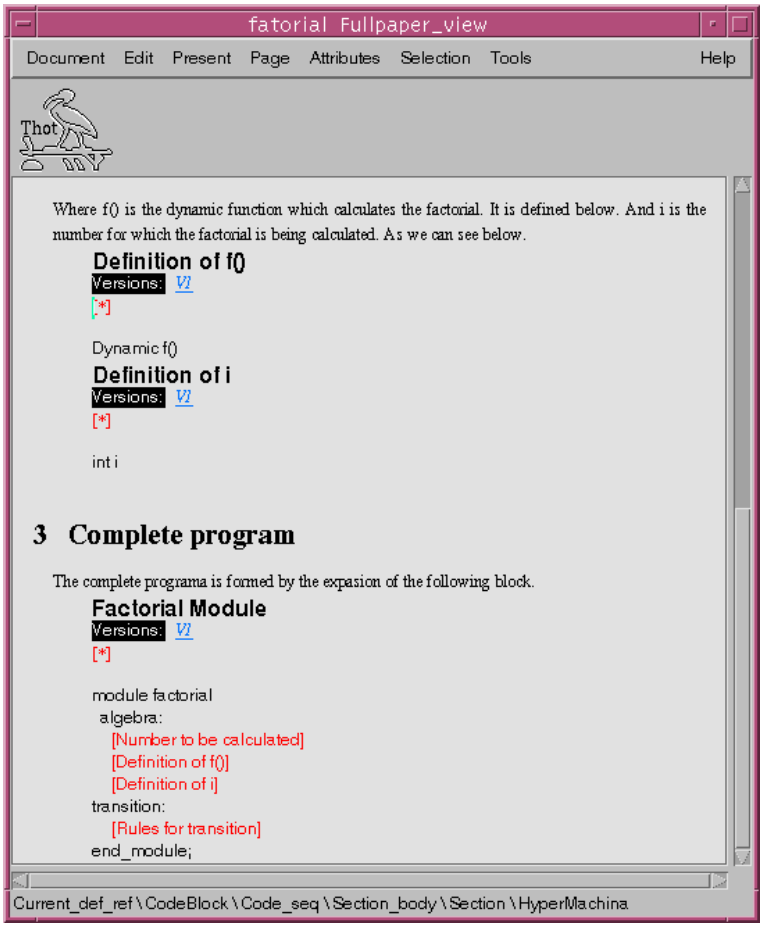


Figura 3: Um Módulo em Machĭna Completo

```
   Adjust : VMiddle;
   IN Table_of_contents
      BEGIN
         Visibility: 8;
         Size : Enclosing + 3;
      END;
END;
```

Exemplo 7

A Figura 3 mostra um módulo em Machĭna com duas partes, *algebra* e *transition*. Os elementos entre colchetes são macros para as definições que foram feitas anteriormente no programa.

Finalmente, a Figura 4 apresenta a visão da tabela de conteúdos do documento. Vale lembrar que essa visão não é um novo documento, mas uma perspectiva tirada do documento principal.

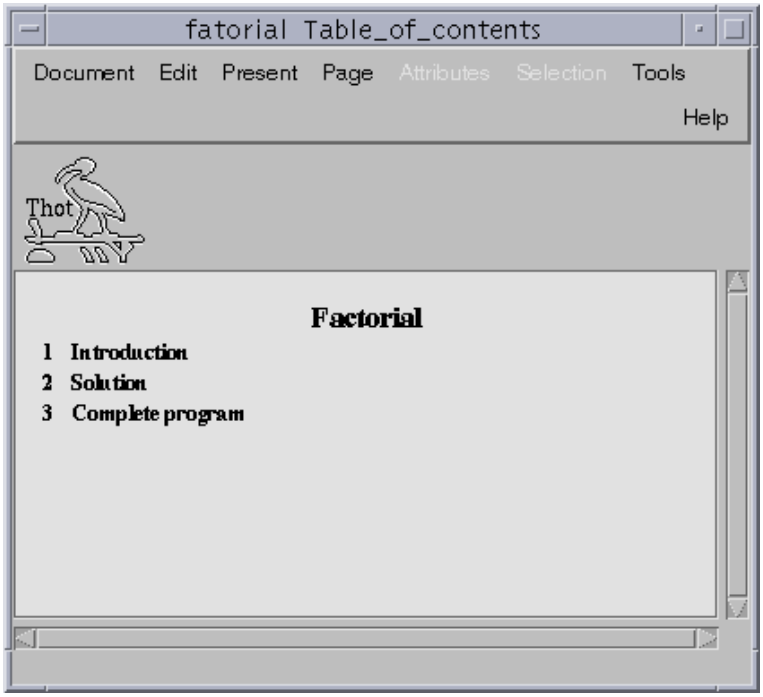


Figura 4: Visão da Tabela de Conteúdos

### 5.4 Funções Disponíveis em um Documento HyperMachĭna

As funções que podem ser usadas em um documento HyperMachĭna são descritas pela linguagem A do *Thot*. Abaixo, citamos as funções que estão disponíveis e portanto podem ser usadas com um documento HyperMachĭna. Esta seção finaliza com as funções que já estão previstas, mas que ainda não foram implementadas.

**Vistas:**

O documento pode ser visto por meio de perspectivas chamadas vistas, que são especificadas na apresentação. Essas vistas são formas de visualização de partes específicas do programa.

O documento completo é visualizado em uma única vista, denominada vista integral do documento. As vistas podem ser abertas simultaneamente e são automaticamente sincronizadas. Ao invés de escrever na vista integral, o usuário pode editar em uma vista específica, e a informação aparece nas demais vistas abertas. Apenas dois tipos de vistas foram especificadas até o momento: *Fullpaper_view* e *Table_of_contents*.

**Programas e Versões**

Um programa é formado por um conjunto de blocos de código. Um documento HyperMachĭna pode conter diferentes versões de um programa. De fato, versões são programas que se diferenciam em pelo menos um bloco de código. Uma versão possui um ou mais blocos iniciais, que são os blocos que não são referenciados por nenhum outro. A partir daí, os

blocos referenciados pelos blocos iniciais são incluídos no programa para formar a versão final, e assim sucessivamente, até que se alcance os blocos que não possuam referências para outros blocos.

### Processos Automáticos

O HyperMachĭna permitirá que o usuário compile, execute e verifique seu programa Machĭna automaticamente.

As ferramentas para compilação, execução e verificação de programas Machĭna podem ser facilmente adaptadas ao HyperMachĭna. No entanto, elas não foram incluídas porque ainda estavam em desenvolvimento na época de conclusão deste trabalho.

### Índices

O HyperMachĭna possuirá os seguintes tipo de índice: índice de referência cruzada, que indica onde um módulo aparece no documento, onde a sua definição é encontrada e onde o módulo é usado em outros programas ou versões no documento; e o índice de programas e versões que mostra onde o programa e suas versões foram primeiramente definidos.

### Conversões e Exportações

O *Thot* produz documentos em um nível abstrato chamado de forma canônica. Pode ser definida uma série de regras de tradução de documentos. Para o HyperMachĭna serão definidos dois esquemas diferentes de tradução: exportação de documentos para HTML e exportação de documentos para ASCII.

### Projeções

Projeções são como as vistas. Porém, aquelas não são obtidas da mesma forma automática com o *Thot* como estas. As projeções são dinâmicas, são perspectivas complexas que podem ser obtidas de um documento HyperMachĭna. As projeções que estarão disponíveis para um documento HyperMachĭna permitirão visualizar as partes específicas de um módulo em Machĭna, bem como cada um dos tipos de definições da álgebra de um programa. São elas:

**Programa** visualizar um programa completo. Da forma como seria enviado para um compilador ou verificador de programa automáticos.

***Import*** perspectiva que descreve as inter-relações de dependência dos módulos de um programa em Machĭna.

***Algebra*** perspectiva das definições da álgebra de um programa Machĭna.

**Definição de Tipos** visualizar as definições de tipos.

**Definição de Funções Estáticas** visualizar as definições das funções estáticas.

**Definição de Funções Dinâmicas** visualizar as definições das funções dinâmicas.

**Definição de Funções Externas** visualizar as definições das funções externas.

**Ações** visualizar as definições das ações.

**Inicializações** visualizar as definições das inicializações.

***Transition*** vista que exibe as regras de transição de cada módulo em Machĭna.

***Invariant*** vista que exibe os invariantes de um programa Machĭna.

# 6 Conclusão

Durante o desenvolvimento deste projeto, pode-se verificar a flexibilidade do editor *Thot*. As linguagens de especificação de estrutura, apresentação, tradução e aplicação permitem a criação de ferramentas de edição de texto poderosas com aplicações específicas, como por exemplo o HyperMachĭna, baseadas no editor *Thot*.

O estilo de programação literário se adapta bem com a metodologia de especificação de sistema ASM, adicionando à formalidade desta, a possibilidade de uma descrição mais legível para o leitor humano. A ferramenta HyperMachĭna, à medida que proporciona um ambiente de desenvolvimento integrado, facilita esse trabalho de especificação e documentação em um só documento.

Neste projeto, desempenhamos as seguintes atividaes:

- finalização da ferramenta HyperPro,
- definição da estrutura de um documento HyperMachĭna, de forma que as funções do editor *Thot* pudessem ser exploradas da melhor forma possível,
- implementação do programa .S que especifica a estrutura de um documento HyperMachĭna,
- implementação do programa .P que especifica a forma de apresentação dos elementos de um documento HyperMachĭna,
- implementação do programa .A que especifica as funcionalidades que são acessíveis por meio de menus e botões da ferramenta HyperMachĭna,
- a conclusão deste relatório técnico que descreve as funcionalidades da ferramenta HyperMachĭna.

Como continuação deste trabalho podemos destacar:

- Implementação das projeções definidas para um programa Machĭna.
- Definir esquemas de tradução de documentos escritos no HyperMachĭna para outros formatos, como por exemplo, LaTex e HTML.
- Incorporação de ferramentas para executar e verificar um sistema Machĭna.

# Referências

[1] Deransart, P., Bigonha, Roberto S, Ed-Dbali, AbdelAli, Siqueira, Jose, Bigonha, Mariza A S, *Basic HyperPro Functionalities and Utilities*, Relatório Técnico $n^o$ 23, Departamento de Ciência da Computação, ICEx, UFMG, 1997, 17 páginas.

[2] Deransart, P., Bigonha R.S., Parot, P., Bigonha, M.A.S., Siqueira, J., *A Hypertext Based Environment to Write Literate Logic Programms*, Relatório Técnico $n^o$ 15, Departamento de Ciência da Computação, ICEx, UFMG, 1996.

[3] Parot, Patrick, *The HyperPro Experimental System*, Relatório Técnico $n^o$ 16/96, Departamento de Ciência da Computação, ICEx, UFMG, 1996.

[4] Deransart, P., Bigonha Roberto S., Parot, P., Bigonha, Mariza A.S., Siqueira, J., *A Literate Logic Programming System*, *Anais do I Simpósio Brasileiro de Linguagens de Programação da* SBC, Belo Horizonte, 1996, páginas:1-16.

[5] Deransart, P., Bigonha Roberto S., Parot, P., Bigonha, Mariza A.S., Siqueira, J., *A Hypertext Based Environment to Write Literate Logic Programms*, *Anais do* JICSLP'96, Bonn, Germany, setembro/1996, páginas:247-252.

[6] Quint, V., *The Thot user manual*, 1995

[7] Quint, V. and Vatton, I., *Hypertext aspects of the Grif structured editor: design and applications*, INRIA Rocquencourt, 1992, July.

[8] Quint, V., *The Languages of Thot*, Internal report, INRIA-CNRS, 1992, May.

[9] Quint, V. and Vatton, I., *The Thot Application Generation Language*, May 7, 1997.

[10] Knuth, Donald D., *Literate Programming*, The Computer Journal, Vol. 27, No. 2, 1984, pp. 97-111.

[11] Ramsey Norman, *Literate-Programming Tools Can be Simple and Extensible*, Departament of Computer Science, Princeton University, November 1993.

[12] Ramsey Norman, *Literate Programming Simplified*, IEEE Software, V.11(5), 97-105, September 1994.

[13] Ramsey Norman, *The noweb Hacker's Guide*, Departament of Computer Science, Princeton University, September 1992 (Revised August 1994).

[14] Leler, William, *Constraint Progamming Languages: their specification and generation*

[15] Y. Gurevich. Evolving Algebras. A Tutorial Introduction. *Bulletin of EATCS*, 43:264-284, 1991.

[16] Y. Gurevich. Evolving Algebras 1993: Lipari Guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9-36. Oxford University Press, 1995.

[17] Fábio Tirelo, Roberto S. Bigonha, Marcelo A. Maia e Vladimir O. Iório. *Machĭna: a Linguagem de Especificação de ASM*, RT LLP 008/99.

[18] D. Bèauquier and A. Slissenko. On Semantics of Algorithms with Continuos Time. Technical report 97-15, Dept. of Informatics, Université Paris-12, October 1997.

[19] D. Bèauquier and A. Slissenko. The Railroad Crossing Problem: Towards Semantics of Timed Algorithms and their Model-Checking in High-Level Languages. In M. Bidoit and M. Dauchet, editors, *TAPSOFT'97: Theory and Practice of Software Development, 7th International Joint Conference CAAP/FASE*, volume 1214 of LNCS, pages 201-212. Springer, 1997.

[20] E. Borger. Why Use Evolving Algebras for Hardware and Software Engineering? In M. Bartosek, J. Staudek, and J. Wiederman, editors, *Proceedings of SOFSEM'95, 22nd Seminar on Current Trends in Theory and Practice of Informatics*, volume 1012 of LNCS, pages 236-271. Springer, 1995.

[21] E. Borger and U. Glasser. Modelling and Analysis of Distributed and Reactive Systems using Evolving Algebras. In Y. Gurevich and E. Borger, editors, *Evolving Algebras - Mini-Course, BRICS Technical Report (BRICS-NS-95-4)*, pages 128-153. University of Aarhus, Denmark, July 1995.

[22] E. Borger, Y. Gurevich, and D. Rosenzweig. The Bakery Algorithm: Yet Another Specification and Verification. In E. Borger, Editor, *Specification and Validadtion Methods*, pages 231-243. Oxford University Press, 1995.

[23] E. Borger and J. Huggins. Abstract State Machines 1988-1998: Commented ASM Bibliography. *Bulletin of EATCS*, 64:105-127, February 1998.

[24] E. Borger and S. Mazzanti. A Pratical Method for Rigorously Controllable Hardware Design. In J.P. Bowen, M.B. Hinchey, and D. Till, editors, *ZUM'97: The Z Formal Specification Notation*, volume 1212 of LNCS, pages 151-187.

[25] E. Borger and D. Rosenzweig. A Mathematical Definition of Full Prolog. In *Science of Computer Programming*, volume 24, pages 249-286. North-Holland, 1994.

[26] E. Borger and W. Schulte. Definig the Java Virtual Machine as Platform for Provably Correct Java Compilation. In L. Brim and J. Gruska and J. Zlatuska, editor, *Mathematical Foundations of Computer Science 1998, 23rd International Symposium, MFCS'98, Brno, Czech Republic*, number 1450 in LNCS. Springer, August 1998.

[27] E. Borger and W. Schulte. Programmer Friendly Modular Definition of the Semantics of Java. In J. Alves-Foss, editor, *Formal Syntax and Semantics of Java, LNCS*. Springer, 1998.

[28] U. Glasser and R. Karges. Abstract State Machine Semantics of SDL. *Journal of Universal Computer Science*, 3(12):1382-1414, 1997.

[29] Y. Gurevich and J. Huggins. The Semantics of the C Programming Language. In E. Borger, H. Kleine Buning, G. Jager, S. Martini, and M. M. Richter, editors, *Computer Science Logic*, volume 702 of LNCS, pages 274-309. Springer, 1993.

[30] Y. Gurevich and J. Huggins. The Railroad Crossing Problem: An Experiemnt with Instantaneous Actions and Immediate Reactions. In *Proceedings of CSL'95 (Computer Science Logic)*, volume 1092 of of LNCS, pages 266-290. Springer, 1996.

[31] Y. Gurevich and R. Mani. Group Membership Protocol: Specification and Verification. In E. Borger, editor, *Specification and Validation Methods*, pages 295-328. Oxford University Press, 1995.

[32] C. Wallace. The Semantics of the C++ Programming Language. In E. Borger, editor, *Specification and Validadtion Methods*, pages 131-164. Oxford University Press, 1995.