Construção dinâmica de um sistema interactivo para visualização do código intermédio do processo de compilação

Paulo Jorge Matos
Instituto Politécnico de Bragança
Campus de Santa Apolónia, 5300 Bragança
pmatos@ipb.pt

Pedro Rangel Henriques
Universidade do Minho
Gualtar, 4700 Braga
prh@di.uminho.pt

Resumo

O DOLPHIN é uma framework que suporta o desenvolvimento de compiladores de alta performance, multi-linguagem e multi-arquitectura. Inclui diversas componentes, uma das quais designada por DOLPHIN-Front-End for the Web, que reúne as componentes relacionadas com a web, nomeadamente: um ambiente integrado de desenvolvimento, que entre outras funcionalidades permite visualizar, sob a forma de hiper-texto, a informação ao longo do processo de compilação. Neste artigo é apresentada uma nova solução desenvolvida com o mesmo objectivo, o de visualizar a informação, mas agora recorrendo a uma solução gráfica, interactiva e que é construída dinamicamente a partir de uma descrição da informação feita em XML.

Palavras chave

Visualização gráfica, sistemas interactivos, XML, compiladores.

Abstract

O DOLPHIN is a framework conceived to help the development of high-performance, multi-language and multi-architecture compilers. It includes several components, one of them is designated by DOLPHIN – Front-End for the Web. This one is responsible for all DOLPHIN items related with the web, namely an integrated development environment that allows, between other things, to observe and analyze, as hiper-text, the information along the compilation process. In this article is introduced a new solution developed with this intent (visualize and analyze the information of the compilation process), with an interactive graphical interface, built dynamically from a description of the information done at XML.

Key words

Graphical visualization, interactive systems, XML, compilers.

1. INTRODUÇÃO

O DOLPHIN é uma *framework* que suporta o desenvolvimento de compiladores de alta performance, multi-linguagem e multi-arquitectura [Matos99, Matos02]. É formada por diversas componentes, mas a principal consiste num conjunto de ferramentas e rotinas, que associadas a um modelo de representação de código, permitem o desenvolvimento de compiladores. Dado que a maioria dessas rotinas utiliza soluções algorítmicas de elevada complexidade, cedo surgiu a necessidade de dotar o DOLPHIN de ferramentas para verificar e optimizar o seu funcionamento. O objectivo era permitir monitorizar e observar os efeitos das rotinas sobre o código submetido ao compilador, à semelhança do que é feito no VCG [Sander95].

O normal processo de *debbuging* do código não é suficiente nem eficaz nestas situações, isto porque a compilação envolve um conjunto muito vasto de tarefas e lida com grandes quantidades de informação.

A primeira solução implementada para monitorizar e observar os efeitos das rotinas, consistiu em dotar os compiladores desenvolvidos com o DOLPHIN de mecanismos para gerar a informação sobre o processo de compilação [Matos03]. Informação essa que era formatada através de uma linguagem hiper-textual, mais especificamente HTML. Esta solução não só possui os requisitos necessários como permite aceder à informação utilizando um simples browser. O que era uma operação dantesca, passou a ser facilmente exequível, permitindo por exemplo: navegar na estrutura do programa; relacionar diferentes formas de representação (ex. vincular expressões com os respectivos vértices do GFC); aceder a informação anexa associada a determinados elementos da RI (ex. vincular variáveis com a respectiva informação da tabela de identificadores); relacionar elementos de fases distintas do processo de compilação (fundamental para verificar o funcionamento das rotinas); e visualizar informação temporária gerada pelos processos de análise.

No sentido de facilitar o acesso e potenciar a utilização, a solução foi colocada disponível via Web, permitindo através de um browser fazer o *upload* do código fonte, o qual é remetido para uma *Common Gateway Interface* (CGI), que trata de invocar o compilador para gerar os ficheiros HTML. Estes são posteriormente redireccionados para o utilizador.

O resultado foi tão bom que deu origem ao *DOLPHIN -- Front-End for the Web* (DOLPHIN-FEW), o qual inclui entre os seus objectivos, a implementação de um ambiente integrado de desenvolvimento (IDE), para editar e compilar programas, com suporte multi-linguagem, multi-arquitectura e uma forte componente de optimização (características do DOLPHIN), disponível via Web. Serve também para verificar e optimizar o funcionamento das rotinas implementadas no DOLPHIN e é um auxiliar precioso no ensino de tópicos relacionados com o desenvolvimento de compiladores.

No entanto a experiência da utilização do IDE levou-nos a concluir que mesmo esta forma de representação hiper-textual não é muito apelativa, nomeadamente quando utilizada para fins pedagógicos. O principal motivo estava relacionado com a quantidade de informação gerada e com a forma como esta é exposta ao utilizador. Tornava-se cada vez mais claro a falta de mecanismos que melhorassem a interactividade com o utilizador. Surgiu assim o trabalho apresentado neste artigo. Trata-se de um interface gráfico, interactivo, que é construído dinamicamente. É uma solução nova que utiliza diversas tecnologias, designadamente: *Schemas*, eXtended Markup Language (XML) e Macromedia Flash[1].

Apesar de especificamente desenvolvida para representar a informação do processo de compilação, esta solução pode também ser utilizada, sem alterações, para visualizar outras fontes de informação capazes de gerar XML.

Na secção seguinte descreve-se a implementação desta solução e na terceira secção, as conclusões.

2. O SISTEMA DE VISUALIZAÇÃO

Dada a quantidade de informação envolvida e à interactividade que requerida para a nova solução, era importante que o esforço de computação fosse transferido o máximo possível para o lado do cliente (browser). Este receberia a informação e tratava de gerir a visualização e a interactividade com o utilizador.

Após a análise de várias soluções, a opção recaiu sobre a utilização conjunta de XML com Macromedia Flash. Esta solução permite transferir toda informação para o cliente, em formato XML, depois utilizando a API disponibilizada através do Flash, do *Document Object Model* proposto pelo W3C (http://www.w3.org/TR/REC-DOM-Level-1/), constrói-se um objecto que representa o documento XML. Esse objecto pode ser manipulado pelo *Action Script* (a linguagem do Macromedia Flash) para construir a representação gráfica.

[1] Macromedia Flash é uma marca registada da Macromedia, Inc.

A Figura 1 representa a arquitectura da nova solução. O utilizador submete, através do IDE, o ficheiro de código fonte a uma CGI. Esta invoca o compilador para gerar os ficheiros em XML e, posteriormente, redirecciona para o browser a aplicação em Flash, com o endereço do ficheiro principal. A aplicação, ao ser executada no browser, faz o download dos ficheiros XML e constrói a representação gráfica.

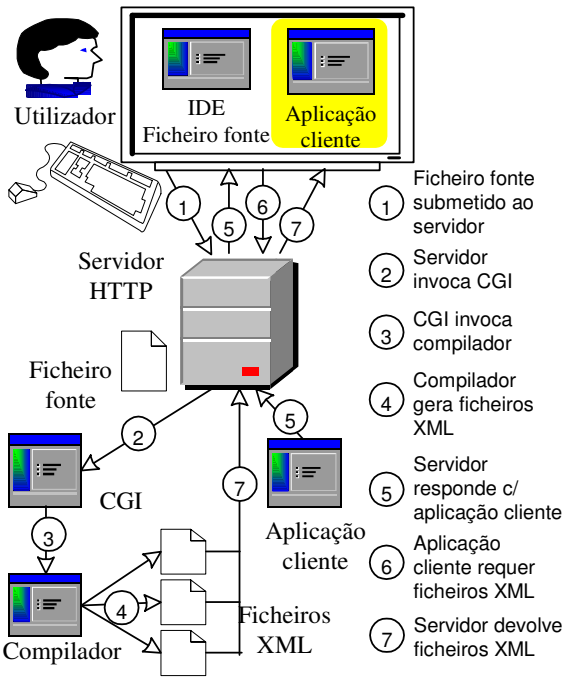

Figura 1. Arquitectura da nova solução.

2.1 *Schema* para a RI do DOLPHIN

A primeira tarefa foi definir o formato dos documentos XML. Dado que o DOLPHIN utiliza objectos (classes em C++) para construir a RI, cada um desses objectos deu origem a um elemento XML, que é representado pelo identificador do objecto. Cada elemento contém outros elementos, um por cada uma das variáveis do objecto. Esses elementos internos são representados pelo mesmo processo, isto é, a designação do elemento corresponde ao identificador da variável e o conteúdo corresponde ao respectivo valor. Quando as variáveis são compostas (arrays, estruturas, classes, etc), então o conteúdo de cada elemento consiste num conjunto de outros elementos.

A Figura 2 representa várias classes utilizadas na RI (*IdentTable*, *FlowNode* e *JumpNode*) e o XML de dois objectos: um instanciado da classe *IdentTable* (*table*) e outro da classe *JumpNode* (*jump*). Este exemplo contém dois casos que merecem um tratamento especial: o *in* e o *table*. O primeiro caso é um *Set* de apontadores para objectos do tipo *FlowNode* (*template* em C++). A conversão desta estrutura segunda as regras anteriores, resulta na representação dos endereços dos objectos *FlowNode*, em vez dos próprios objectos, pelo que na geração do XML o que é utilizado é o objecto apontado e não o apontador. O segundo caso é o da variável *table*, uma tabela de Hash com chaves do tipo *string* e valores

do tipo *Cell* (apontador). Neste caso, a tabela de Hash é percorrida linearmente e por cada entrada da tabela são gerados dois elementos: o <*key*> e o <*val*>, que representam, respectivamente, a chave e o valor de cada entrada.

Houve ainda a necessidade de obter uma solução para resolver o problema derivado da herança entre classes. Nestes casos, a representação XML inclui não só as variáveis da classe do objecto, mas também as variáveis das classes ascendentes. A Figura 2 ilustra um destes exemplos (entre a classe *JumpNode* e a *FlowNode*).

C++

```
class IdentTable{
  THash<char*,Cell*> th;
  …
};
IdentTable table;

class FlowNode{
  TNode tnode;
  Set<FlowNode *> in;
  …
};
class JumpNode
:public FlowNode{
  Jump jp;
  …
};
JumpNode jump;
```

XML

```
<table><th>
      <key>…</key>
      <val>…<val>…
</th>…</table>

<jump>
  <tnode>JNODE</tnode>
  <in>
    <jnode>…</jnode>
    …
  </in>…
  <jp>…</jp>
  …
</jump>
```

Figura 2. Tratamento da herança entre classes.

Devido à quantidade de informação e ao número considerável de classes utilizadas na RI, surgiram algumas complicações (valores incorrectos, etiquetas não concluídas, etc), pelo que se optou por desenhar um *Schema* para definir e validar a estrutura dos documentos.

```
<complexType name="FlowNode">
  <sequence>
    <element name="tnode" type="TNode"/>
  </sequence>
</complexType>
<complexType name="JumpNode">
  <complexContent>
    <extension base="FlowNode">
      <sequence>
        <element name="jp" type="Jump"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Figura 3. *Schemas* das classes *FlowNode* e *JumpNode*.

No desenho do *Schema* cada classe deu origem a um tipo complexo (*ComplexType*). A herança entre classes foi tratada através dos próprios mecanismos de herança dos *Schemas* (através do *restriction base=”..”*). A Figura 3 ilustra este procedimento para as classes *JumpNode* e *FlowNode* da Figura 2. É apenas de realçar a utilização de tipos simples (*simpleType*) para representar enumerações, como é o caso do *TNode*, restringindo assim os valores que podem ser utilizados; e do *choice* para representar uniões (*union*’s) e elementos que podem assumir diferentes tipos (polimorfismo). A Figura 4 mostra exemplos destes casos.

Apenas para concluir esta secção, falta acrescentar que o *Schema* definido representa toda a informação sob a forma de elementos. Os atributos, como se explica mais adiante, são apenas utilizados para fornecer informação sobre a forma e comportamento da representação gráfica.

```
<simpleType name="TNode"><restriction base="xsd:string">
   <enumeration value="JNODE"/>
   <enumeration value="CJNODE"/>
   …
</restriction></simpleType>

<complexType name="Function">
<sequence>…
  <element name="setNodes"><complexType>
    <sequence minOccurs="0" maxOccurs="unbounded">
      <choice>
        <element name="jNode" type="JumpNode"/>
        <element name="rNode" type="ReturnNode"/>
        …
      </choice></sequence></complexType>
</element></sequence></complexType>
```

Figura 4. Enumerações e variáveis polimórficas.

2.2 Os componentes Flash

A aplicação Flash é formada por um corpo principal e um conjunto de componentes. No corpo principal realiza-se o *setup* da aplicação, que consiste em fazer: o download dos ficheiros XML, construir o *objecto* XML e instanciar um objecto do tipo *DefaultObject* (uma das componentes Flash desenvolvidas), o qual inicia a construção da representação gráfica.

Os componentes desenvolvidos estão preparados para interpretar alguns atributos, que condicionam a representação e o comportamento da visualização. Os principais atributos são:

__ID - *String* que identifica o objecto;

__type - *String* que define o tipo de componente. Por omissão é utilizado o *DefaultObject*;

__vtype - *String* que define o estado visual do objecto. Existem três estados possíveis:

- *popup* - Assinala objectos que apenas surgem mediante um evento que ocorre num outro objecto (*onOver*, *onClick*, etc). Estes objectos podem ser convertidos para o estado *standalone*;
- *standalone* - Assinala objectos cuja visualização não depende exclusivamente de outros objectos. Podem ser escondidos, redimensionados, deslocados e convertidos para o tipo *popup*;
- *root* - Assinala o objecto inicial da representação. Está sempre visível e apenas pode ser redimensionado ou deslocado;

__label - *String* que define a *label* do objecto ou o objecto ao qual o elemento deve ficar associado;

__x, *__y* - Atributos do tipo real, que representam a posição horizontal e vertical do objecto;

__visible - *Booleano* que indica se o objecto está ou não visível.

2.2.1 Componentes genéricos

A aplicação desenvolvida possui componentes de uso genérico, que podem ser utilizados para visualizar um qualquer documento XML, e componentes construídos especificamente para o DOLPHIN. Ambos têm a capacidade de ler os atributos *__x*, *__y*, *__visible*, *__vtype* e *__label*, para definirem respectivamente a: sua posição, se estão ou não visíveis, como se devem comportar (*popup*, *standalone*, *root*), e se devem utilizar alguma *label* específica na identificação do objecto.

O mais genérico dos componentes é o *DefaultObject*, que corresponde sempre ao objecto inicial da representação, e a partir do qual os restantes são construídos. A Figura 5 permite ver o aspecto deste objecto no formato *standalone* (o formato *popup* pode ser visualizado na Figura 6). De notar a existência de dois botões no canto superior direito: um para redimensionar e o outro para fechar o objecto; e de uma *Label* no canto superior esquerdo, normalmente com a designação do elemento raiz do objecto XML, ou então com o valor do atributo *__label* (caso este esteja definido). À frente desta *label* pode ainda existir um *link* para o objecto ascendente (na hierarquia do objecto XML). A restante informação representada diz respeito aos elementos descendentes.

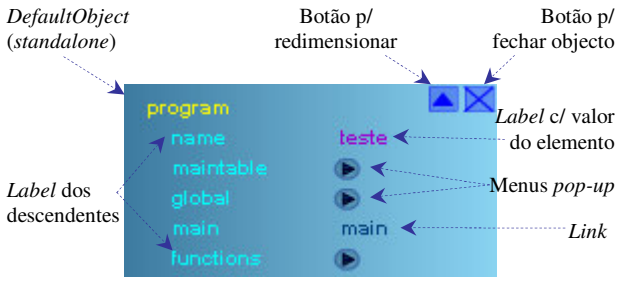

Figura 5. *DefaultObject* no formato *standalone*.

O próprio *DefaultObject* percorre a lista dos elementos descendentes e constrói dinamicamente a representação interna. Cada descendente é representado por um par formado por uma *label*, com a designação do respectivo elemento, e por um objecto do tipo *label*, *link* ou *pop-up* menu. A *label* é utilizada para representar o valor de elementos do tipo simples. O *pop-up* menu e o *link* são ambos utilizados para ligações a elementos compostos. Há no entanto diferenças entre eles, para além do aspecto gráfico (um é representado por um botão animado e o outro por texto), o *pop-up* é utilizado nos descendentes do tipo composto e força a criação dos objectos para esses elementos (passando a respectiva sub árvore do objecto XML). Nestas condições o atributo *__type* normalmente não está definido, ou então está definido com o valor "*sequence*".

O *link* está sempre associado a elementos do tipo simples, mas que representam ligações a elementos do tipo composto. Não cria nenhum objecto novo, espera-se no entanto que o objecto alvo seja criado num outro contexto, para poder ser correctamente referenciado pelo *link*. O atributo *__type* é definido com o valor "*link*" e o objecto a referenciar é definido através do atributo *__label* (ex: *<main __type="link" __label="ID1200">main</main>*).

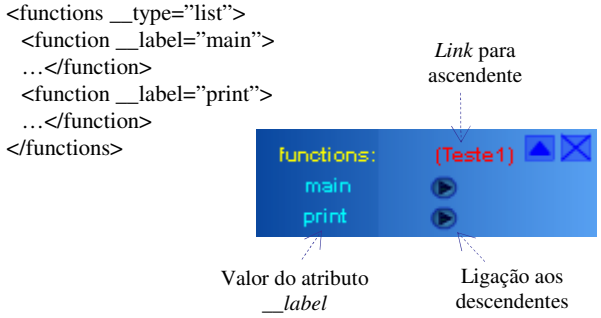

Figura 6. *DefaultObject* para uma lista.

O *link* e o *pop-up* menu são semelhantes no funcionamento, isto é, quando o apontador passa sobre eles o objecto que representam fica em foco (se estava invisível passa a visível). Caso o utilizador click na tecla esquerda do rato, o objecto passa do estado *popup* para *standalone* ou vice-versa (depende do estado inicial). Ao sair de cima do botão *pop-up* ou do *link* o objecto volta ao estado inicial, ou ao estado que ficou após se pressionar a tecla do rato.

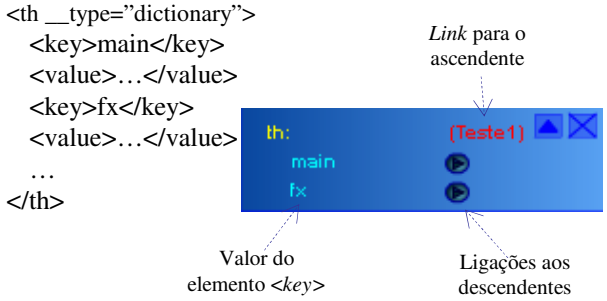

Figura 7. *DefaultObject* para um dicionário.

O atributo *__type* serve para modificar a forma como a informação é processado no *DefaultObject*. Colocando *__type* com o valor "*lsuper*", significa que deve ser colocado um *link*, à frente da *label* do objecto, com ligação ao elemento ascendente. Isto é importante para associar o objecto ao seu ascendente e para distinguir objectos do mesmo tipo.

Para elementos cujos descendentes são todos do mesmo tipo, o *__type* deve ser definido com o valor "*list*", o que permite representar esses descendentes, não pela própria designação (que é igual para todos os descendentes), mas pelo valor do atributo *__label*. A Figura 6 mostra um destes casos.

O atributo *__type* pode ainda assumir o valor de "*dictionary*". Neste caso os descendentes são processados dois a dois, em que o valor do primeiro é utilizado como *label* e o segundo, como objecto que surge do lado direito

(*label*, botão *pop-up* ou *link*). A Figura 7 representa uma destas situações.

2.2.2 Componentes específicos

Os componentes específicos visam dar uma interpretação mais intuitiva da informação gerada pelo DOLPHIN. Existem cinco componentes específicos: o *Node* e o *CNode*, que representam vértices do GFC; o *Expression*, que representa nodos das árvores de expressões; e o *DTlist* e *DT* que representam, respectivamente, listas de *DataTransfer* e *DataTransfer*'s (elementos da RI que visam caracterizar as trocas de dados). A Figura 8 mostra um exemplo que inclui vários destes objectos.

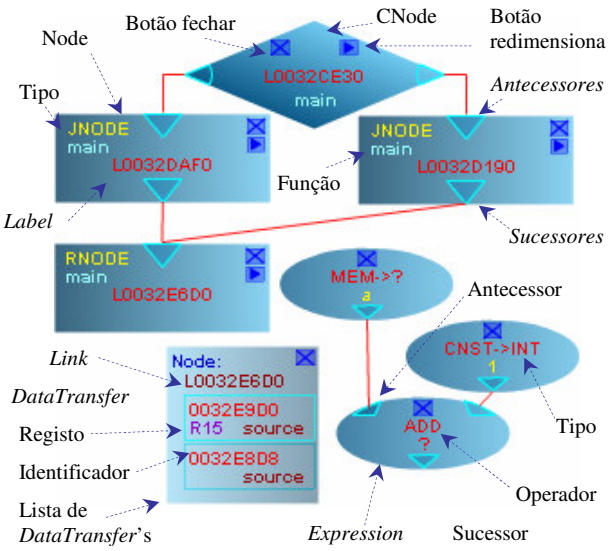

Figura 8. Visualização do GFC, dos *DataTransfer*'s e das árvores de expressões.

O *Node* é representado por um rectângulo que contém: duas *label*'s, uma com a identificação do vértice e a outra com o tipo (RNODE/JNODE); um *link* para a função à qual o vértice pertence; um botão para fechar o vértice e outro para aceder à respectiva lista de *DataTransfer*'s. Existem ainda áreas sensíveis que se activam à passagem do rato. No caso do *Node* permitem aceder aos vértices antecessores e sucessores do GFC. O comportamento das zonas sensíveis é muito semelhante ao do *link* e do botão *pop-up*, a diferença está no facto de desenharem as ligações entre vértices. O componente *CNode* apenas difere do *Node* na representação gráfica (losango). Ambos podem ser deslocados pelo ecrã.

O componente *DTlist* possui um *link*, para o respectivo vértice do GFC, e uma lista de instâncias derivadas do componente *DT*. Cada uma contém: a identificação do *DataTransfer*; o registo utilizado na operação; e um *link* para uma instancia do componente *Expression*. O *DTlist* pode ser fechado e deslocado, já os *DT* são apenas elementos estáticos dentro do *DTlist*.

O componente *Expression*, representado por uma forma oval, contém duas *label*'s: uma que identifica o tipo de operação; e outra que identifica o tipo de dado (inteiro, real, etc), e se necessário mais alguma informação relevante. Contém ainda um botão para fechar o objecto e algumas zonas sensíveis: duas para aceder aos descendentes da expressão (esquerdo e direito) e uma terceira para aceder aos sucessores.

3. CONCLUSÕES

A utilização de componentes Flash juntamente com XML é cada vez mais uma fórmula de sucesso para animar e representar informação. Julgamos nós que esta fórmula foi aqui utilizada racionalmente, para resolver um problema real com necessidades específicas. Os ganhos inerentes da implementação desta aplicação ainda estão por avaliar, dado que por um lado ainda está em desenvolvimento e por outro ainda não foi disponibilizada aos potenciais utilizadores. No entanto possui já algumas características importantes, a saber: é simples de utilizar; em nossa opinião está bem desenhada; interactua bastante bem com o utilizador; permite satisfazer determinadas necessidades que de outra forma seriam inacessíveis; e disponibiliza a informação de uma forma bastante amigável.

O desenvolvimento desta aplicação serviu também para desenhar um *Schema* para a RI do DOLPHIN, o qual prevemos utilizar em outros projectos, alguns dos quais bastante inovadores.

É ainda de salientar que existem planos para dar continuidade ao desenvolvimento desta aplicação, nomeadamente no sentido de: construir novos componentes para representar informação que ficou de fora na actual solução (ex. informação temporária sobre os processos de análise); e construir uma solução para simular o comportamento das rotinas de análise e optimização, a qual passa pela implementação de novos componentes, não para representar mais informação, mas para processar os actuais, permitindo assim simular o comportamento das soluções implementadas ao nível do DOLPHIN.

Não fica de fora a possibilidade de evoluir os componentes genéricos para lidar com outro tipo de dados, como por exemplo: imagens, som, filmes, páginas web, etc.

Para concluir, é de realçar novamente que esta aplicação pode ser utilizada para animar e visualizar outras fontes de informação, sejam capazes de gerar XML.

4. REFERÊNCIAS

[Matos99] P. Matos. *Estudo e desenvolvimento de sistemas de geração de back-ends para compiladores*. Master Thesis, Universidade do Minho, 1999.

[Matos02] P. Matos. *The DOLPHIN framework*. Technical Report, Universidade do Minho 2002.

[Matos03] P. Matos, P.Henriques. DOLPHIN-FEW – An example of a web system to analyze and study compiler behavior. Proceedings of the E-Society Conference, 2003.

[Sander95] G. Sander. VCG Visualization of Compiler Graphs. Technical Report A01-95, Universität des Saarlandes, 1995.