UNIVERSIDADE FEDERAL DE SANTA CATARINA

Uma Ferramenta Protótipo para Síntese de Software para Sistemas Embutidos a partir de SystemC

Roberto Hartke Neto

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

Uma Ferramenta Protótipo para Síntese de Software para Sistemas Embutidos a partir de SystemC

Roberto Hartke Neto
Autor

Luiz Cláudio Villar dos Santos
Orientador

Luís Fernando Friedrich
José Mazzuco Jr.
Banca examinadora

Palavras-chave
Sistemas Embutidos
Síntese de Software
SystemC

Florianópolis, fevereiro de 2004

Sumário

Resumo

O objetivo deste trabalho é traçar um modelo unificado para síntese de software para sistemas embutidos. O projeto é especificado em SystemC, uma linguagem unificada que descreve tanto o software quanto o hardware, que vem se tornando um novo padrão na área de desenvolvimento de sistemas embutidos. Este trabalho propõe um fluxo de síntese de software que usa grafos de fluxo como representação intermediária. Como grafos têm uma base matemática sólida, muitas otimizações de sistema podem ser feitas com algoritmos bem conhecidos baseados em grafos.

Este trabalho também mostra como código de montagem pode ser gerado a partir da representação baseada em grafos, traduzido para código de máquina e finalmente executado.

Palavras-chave: SystemC, grafos de fluxo, linguagem de montagem, ArchC, síntese de software.

Abstract

The goal of this work is to pave the way to an unified model for embedded software synthesis. The system is specified with SystemC, an unifying language describing both hardware and software, which is becoming a new standard in the field of embedded system design. This work proposes a software synthesis flow that uses flow graphs as intermediate representation. Since graphs have a solid mathematical background, many system optimizations can be cast into well-known graph-based algorithms.

This work also shows how assembly code can be generated from the graph-based representation, then translated into machine code and finally executed.

Keywords: SystemC, flow graphs, assembly, ArchC, software synthesis

1 Introdução

1.1 Sistemas Embutidos

Antes de tudo, o que seria um sistema embutido? Em uma definição superficial, é qualquer dispositivo que inclui um computador programável mas não é direcionado para ser um computador de propósito geral [10]. Então, um computador pessoal não é um sistema embutido, mas um telefone ou microondas com um microprocessador é um sistema computacional embutido.

1.2 Requisitos de Sistemas Embutidos

Requisitos são uma descrição das necessidades ou dos desejos para um produto. O objetivo básico dos requisitos é identificar e documentar o que é realmente necessário, em uma forma que comunica claramente essa informação ao cliente e aos membros da equipe de desenvolvimento. O desafio é definir os requisitos de maneira não-ambígua, de modo que os riscos sejam identificados e não aconteçam surpresas quando o produto for finalmente liberado [6].

A separação da análise de requisitos da especificação do sistema é geralmente necessária por causa da grande lacuna entre o que os clientes podem descrever sobre o sistema que eles querem e o que os arquitetos precisam para desenvolver o sistema. Clientes de sistemas embutidos geralmente não são projetistas de sistemas embutidos. O entendimento deles sobre o sistema é baseado nas interações do usuário com o sistema. Eles podem ter expectativas irreais sobre o que pode ser feito com o seu orçamento; e também expressar suas vontades em uma linguagem muito diferente do jargão usado pelos arquitetos do sistema. Capturar um conjunto consistente de requisitos e então transformá-los em uma especificação mais formal é uma maneira estruturada de gerenciar o processo de tradução da linguagem do cliente para a linguagem dos projetistas.

Requisitos podem ser funcionais ou não-funcionais. É claro que é preciso capturar as funções básicas do sistema embutido, mas a descrição funcional geralmente não é suficiente. Requisitos não funcionais típicos incluem:

- Consumo de energia: energia, é claro, é importante em sistemas alimentados por bateria e também importante em outras aplicações. Energia pode ser especificada nos requisitos como tempo de duração da bateria — o cliente não será capaz de descrever coisas como voltagem.
- Custo: o custo do produto ou preço de venda é quase sempre uma consideração importante. Os custos têm dois maiores componentes: a manufatura, que inclui os componentes e sua montagem; e a engenharia, que incluem custos com pessoal e outros custos do desenvolvimento.
- Desempenho: a velocidade do sistema é freqüentemente a maior consideração para a usabilidade do sistema e para o seu custo final.
- Tamanho e peso: os aspectos físicos do sistema final podem variar muito dependendo da aplicação. Um sistema para controle industrial para uma linha de montagem pode ser desenhado para ajustar-se em uma prateleira com tamanho padrão e sem limite de peso. Já um dispositivo de mão tipicamente tem requisitos de tamanho e peso que podem se espalhar por todo o projeto do sistema.

Programas para sistemas embutidos são em muitas maneiras mais restritos que os programas feitos para computadores de uso geral. Funcionalidades são importantes em ambos, mas aplicações embutidas enfrentam obstáculos diferentes. Por outro lado, sistemas computacionais embutidos têm que fornecer funcionalidades sofisticadas:

- Algoritmos complexos: as operações executadas pelo microprocessador podem ser muito sofisticadas. Por exemplo, o microprocessador que controla o motor de um automóvel precisa executar complicadas funções de filtragem para otimizar o desempenho do carro enquanto precisa minimizar a poluição e consumo de combustível.
- Interface com usuário: microprocessadores são freqüentemente usados para controlar interfaces com o usuário que podem incluir múltiplos menus e muitas opções.

Para tornar as coisas mais difíceis, operações computacionais às vezes apresentam restrições:

- Memória disponível: em sistemas embutidos a quantidade de memória disponível é muitas vezes menor do que a de um computador de uso geral. As aplicações construídas para estes sistemas têm que levar em consideração esta restrição.

- Tempo real: muitas aplicações embutidas precisam executar em tempo real — se o dado não estiver disponível até um determinado prazo, o sistema cai. Em alguns casos, a falha de não cumprir o prazo é insegura e pode pôr em risco vidas. Em outros casos, o não cumprimento do prazo não cria problemas de segurança, mas gera clientes insatisfeitos.

1.3 Contexto do trabalho

A abordagem predominante para o projeto de um sistema digital no nível de arquitetura é descrevê-lo através de uma linguagem de descrição de hardware. Tal descrição, além de servir como documentação, permite a simulação da arquitetura do sistema e a síntese automática do circuito lógico que implementa a arquitetura descrita. As linguagens de descrição de hardware mais populares atualmente são VHDL e Verilog.

A indústria de EDA (*Electronic Design Automation*) continua reunindo esforços para um movimento coletivo de desenvolvimento em níveis mais altos de abstração do que o nível de transferência de registradores (*Register Transfer Level* ou RTL). É necessário uma única linguagem comum e fundações de modelagem para criar um mercado de interoperabilidade de ferramentas de desenvolvimento, serviços e fazer IP (*Intellectual Property*) realidade.

Recentemente, a metodologia de projeto reforça a descrição em níveis mais abstratos de descrição. A idéia é descrever-se não somente o componente de hardware como também sua interação com o software. Este tipo de descrição é referenciada como nível de sistema. Há um esforço da comunidade de automação de projetos para consolidar sua linguagem de desenvolvimento de sistemas, conhecida como SystemC [8].

1.4 Características e Funcionalidades de SystemC

O padrão SystemC é controlado por um grupo composto pelas treze maiores companhias das indústrias de EDA e eletrônicos.

SystemC é uma biblioteca de classes em C++ e uma metodologia que pode-se utilizar

para criar precisos modelos baseados em ciclos de algoritmos de software, arquiteturas de hardware, interfaces em um SoC (*System On a Chip*) e projetos no nível de sistemas (*system-level designs*). Pode-se usar ferramentas de desenvolvimento de SystemC e de C++ padrão para criar um modelo de sistema, rapidamente simular para validar e otimizar o projeto, explorar vários algoritmos, e conceder a equipe de desenvolvimento de software e de hardware uma especificação executável do sistema. Uma especificação executável é essencialmente um programa em C++ que exibe o mesmo comportamento do sistema quando executado.

C ou C++ são as linguagens escolhidas para algoritmos de software e especificações de interface porque elas fornecem o controle e abstrações de dados necessários para desenvolver descrições de sistema compactas e eficientes. A maioria dos projetistas estão familiarizados com estas linguagens e a um grande número de ferramentas de desenvolvimento associadas a elas.

A biblioteca de classes de SystemC fornece o necessário para construir modelos da arquitetura de sistemas incluindo temporização do hardware, paralelismo, e comportamento reativo que falta em C++ padrão. Adicionar estas construções a C exigiria extensões proprietárias a linguagem, o que não é uma solução aceitável para a indústria. A linguagem C++ orientada a objetos provê a habilidade de estender a linguagem através de classes, sem adicionar novas contruções sintáticas. SystemC fornece essas classes necessárias e permite que projetistas continuem a usar a linguagem C++ e as ferramentas de desenvolvimento.

SystemC possui entre diversas outras características, a possibilidade de múltiplos níveis de abstração: suporta modelos sem temporização em diferentes níveis de abstração, abrangendo de modelos alto-nível até modelos detalhados em RTL. Suporta refinamento iterativo de modelos de níveis mais altos para níveis de abstração mais baixos.

2 Projeto da Representação

Representações intermediárias são o ponto de partida do que o programador escreveu para o que a máquina entende. Durante a tradução de uma linguagem de alto-nível para código de máquina, um compilador otimizador analisa e transforma a representação intermediária várias vezes. Os usuários do compilador querem que essas análises e transformações sejam rápidas e corretas. Os programadores do compilador querem que as otimizações sejam simples de escrever, fáceis de compreender, e fáceis de estender. O objetivo então é uma representação que seja simples e leve enquanto possibilita fácil expressão e otimizações rápidas.

2.1 Linguagem de Montagem Como Representaçao Intermediária

Um código escrito em linguagem de montagem (*assembly language*), usado como representação intermediária entre a linguagem de alto-nível e o código de máquina, tem diversas desvantagens e limitações:

- É dependente do conjunto de instruções, não pode ser utilizado em outra máquina com um conjunto de instruções diferente.
- Utiliza os recursos da arquitetura para qual foi gerado. Mesmo que o código seja compatível com outra máquina, o programa não utilizaria recursos adicionais que esta máquina poderia ter (maior número de registradores por exemplo).

2.1.1 Dependências de dados

Suponha duas instruções, I e J, com I ocorrendo antes que J. São possíveis de ocorrer os *hazards*:

- RAW (*read after write*) - J tenta ler a fonte antes que I escreva, então J lê incorretamente o valor antigo. Este é o tipo mais comum de *hazard*.

- WAW (*write after write*) - J tenta escrever em um operando antes de I escrever. A escrita acaba sendo feita na ordem errada, deixando o valor escrito por I no destino em vez do valor escrito por J. Este *hazard* está presente apenas em pipelines que permitem escrita em mais de um estágio (ou permite que uma instrução continue a executar mesmo quando alguma anterior está parada).

- WAR (*write after read*) - J tenta escrever no destino antes que ele seja lido por I, então I lê o novo valor o que é incorreto. Este hazard ocorre quando existem algumas instruções que escrevem seu resultado cedo no pipeline de instruções, e outras instruções que lêem tarde no pipeline.

Quando o conjunto de instruções possui instruções multiciclo, como multiplicação, divisão ou operações com ponto flutuante, a ocorrência de hazards (principalmente WAW) aumenta. Isso porque as instruções não atingem mais os estágios na ordem que foram iniciadas. Por exemplo, se a instrução I for uma divisão e J uma simples soma, onde I precede J e ambos escrevem no mesmo registrador, J chegará antes que I no estágio de escrita e causará um *hazard* WAW.

2.1.2 Dependências de nome

Uma dependência de nome ocorre quando duas instruções usam o mesmo registrador ou local de memória, chamado de nome, mas não há nenhum fluxo de dados entre as instruções associado com o nome. Há dois tipos de dependências entre uma instrução I que precede uma outra instrução J na ordem do programa:

- Uma **antidependência** entre a instrução I e a instrução J ocorre quando a instrução J escreve em uma registrador ou posição de memória que a instrução I lê e a instrução I é executada antes. Uma antidependência corresponde a um *hazard* WAR.

- Uma **dependência de saída** ocorre quando a instrução I e a instrução J escrevem no mesmo registrador ou posição de memória. A ordem das instruções precisa ser preservada. Dependências de saída correspondem a *hazards* WAW.

Antidependências e dependências de saída são dependências de nome, ao contrário de dependências de dados reais, porque não há valor sendo transmitido entre as instruções.

Isto significa que as instruções envolvidas numa dependência de nome podem executar simultaneamente ou serem reordenadas, se o nome (registrador ou posição de memória) usado nas instruções for mudado de tal forma que não exista mais conflito. Este renomeamento pode ser feito facilmente para registradores e é chamado de renomeamento de registradores (*register renaming*). Pode ser feito estaticamente por um compilador ou dinamicamente pelo hardware.

2.2 Grafos Como Representaçao Intermediária

Como vantagem de utilizar grafos como linguagem intermediária, pode-se citar:

- Independente de plataforma.
- Possibilidade de realizar otimizações, utilizando algoritmos básicos sobre grafos.
- Deixam explícito o paralelismo das operações.
- Grafos são velhos conhecidos do mundo da computação, tem uma base matemática sólida, e têm disponíveis uma grande variedade de algoritmos.

2.2.1 Grafo de fluxo de dados

Um grafo de fluxo de dados é um modelo de um programa sem condicionais. Em uma linguagem de programação de alto-nível, um segmento de código sem condicionais — mais precisamente, com um único ponto de entrada e saída — é conhecido como bloco básico. Antes que seja possível desenhar o grafo de fluxo de dados para o código, é necessário modificá-lo um pouco. É necessário reescrever o código no formato de atribuição única (*single-assignment form*), onde uma variável aparece apenas uma vez no lado esquerdo de uma atribuição (leia [10]). O formato de atribuição única é importante porque permite identificar uma localização única no código onde uma variável é computada. Este formato de atribuição única também elimina as dependências de nome das variáveis.

O formato de atribuição única significa que o grafo de fluxo de dados é acíclico. Manter o grafo de fluxo de dados acíclico é importante em vários tipos de análises possíveis de serem feitas no grafo. Há mais detalhes sobre os grafos de fluxo de dados em seções futuras. A tabela 1 mostra um exemplo de código e seu equivalente no formato de atribuição única.

Tabela 1: Formato de atribuição única

Código original	**Codigo reescrito**
int x = 1;	int x = 1;
int y = 2;	int y = 2;
int z = 3;	int z = 3;
x = x ∗ y;	int _x1_ = x ∗ y;
y = y ∗ z;	int _y1_ = y ∗ z;
z = x ∗ y;	int _z1_ = _x1_ ∗ _y1_;

2.3 Um modelo unificado para síntese de hardware e software

Sistemas embutidos são implementados como uma mistura de software e hardware. Geralmente, software é usado para proporcionar várias funcionalidades e flexibilidade, enquanto hardware é usado para prover desempenho.

A metodologia atual de projeto de sistemas embutidos requer separação da especificação e desenvolvimento de hardware e software. Esta metodologia traz problemas quando há necessidade de modificar o projeto do hardware ou software, às vezes é preciso até reprojetar o sistema. Esta metodologia ainda tem outros problemas:

- Falta de uma representação unificada do modelo de hardware e software, o que leva a dificuldades na verificação do sistema como um todo, e consequentemente incompatibilidades na integração software/hardware.
- Uma definição antecipada das partições, o que pode levar a projetos pouco otimizados.
- Falta de um fluxo de projeto bem definido, o que torna a revisão da especificação difícil.

Há várias abordagens diferentes para resolver o problema do projeto de sistemas embutidos. Nesse trabalho será desenvolvida e mostrada uma metodologia para especificação do sistema, e síntese e validação do software para o sistema embutido. O projeto é feito com uma linguagem unificada que descreve tanto o software quanto o hardware, SystemC, e com uma única abordagem de representação intermediária de software, grafo, e tendo como produto final código de montagem e consequente código de máquina.

2.3.1 Um modelo de comportamento baseado em grafos

A representação intermediária neste projeto é então uma estrutura baseada em grafos e orientada a objetos, mais precisamente grafos polares[1] de fluxo de dados.

Os vértices do DFG denotam operações, parâmetros, chamadas de função, valores de retorno, constantes ou são usados para polarizar o grafo. As arestas podem ser de apenas dois tipos: dados ou sequenciamento (não transportam dados, tem custo zero).

O grafo de fluxo de dados de um código torna a ordem em que as operações são executadas menos óbvia. Esta é uma das vantagens do grafo de fluxo de dados. Pode-se usá-lo para determinar reordenamentos possíveis das operações, o que pode ajudar a reduzir conflitos do pipeline e da *cache*. Pode-se utilizá-lo também quando a ordem exata das operações não importa. O grafo de fluxo de dados define uma ordem parcial das operações no bloco básico. É necessário garantir que um valor é computado antes que seja utilizado, mas geralmente há diversos ordenamentos possíveis de avaliar as expressões para satisfazer esse requisito.

2.4 Técnicas de otimização

Otimizações são um conjunto de transformações que preservam a semântica e que minimizam a quantidade de informação e instruções necessárias para executar um programa.

Otimizações podem ser implementadas de diversas maneiras. Podem ser realizadas durante a etapa de análise da compilação, durante a geração da representação intermediária, ou mesmo na representação intermediária.

Nas próximas seções serão explanadas algumas otimizações possíveis de serem feitas em grafos, a representação intermediária alvo deste trabalho. Elas são transformações baseadas no fluxo de dados, e podem ser consideradas otimizações "clássicas". As otimizações implementadas neste trabalho são locais, isto é, se aplicam apenas ao escopo do bloco básico.

2.4.1 Propagação de constantes

Propagação de constantes consiste em detectar operandos constantes e pré-computar o valor da operação. Sendo o resultado outra constante, a nova constante pode ser pro-

[1] Veja o anexo D para a definição de grafos polares

pagada para as operações que utilizavam o resultado da operação como entrada. A figura 1 ilustra esta otimização.

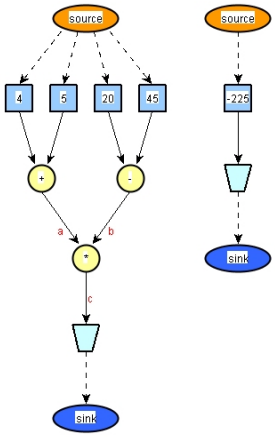

Figura 1: Propagação de constantes

2.4.2 Propagação de variáveis

Propagação de variáveis consiste em detectar cópias de variáveis, por exemplo, as atribuições do tipo x = y. Este tipo de otimização só pode ser feito se o valor da variável do lado esquerdo da atribuição (ou seja, a "cópia") não for alterado em alguma atribuição futura, o que é verdade no formato de atribuição única. Veja a tabela 2 como exemplo.

Tabela 2: Propagação de variáveis

Código original	**Código equivalente**

int a = 3 + 4;
int b = a;
int c = b * 3;
	int a = 3 + 4;
	int c = a * 3;

2.4.3 Eliminação de subexpressões comuns

A procura por subexpressões comuns tenta encontrar padrões isomórficos no grafo. Este passo é simplificado se as expressões aritméticas tiverem apenas dois operandos, que é o caso da representação intermediária adotada no trabalho. Então, esta transformação consiste em selecionar um alvo, por exemplo, uma operação qualquer, e procurar outra operação do mesmo tipo e que tenha os mesmos operandos de entrada. Veja a figura 2 como exemplo.

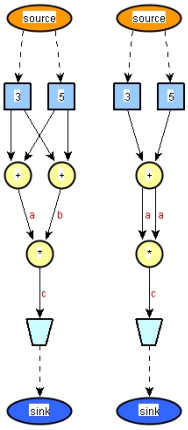

Figura 2: Eliminação de subexpressões comuns

2.4.4 Eliminação de código morto

Código morto consiste em todas as operações que não podem ser alcançadas, e aquelas que o resultado não é referenciado em nenhum outro lugar. Casos óbvios são comandos após um comando de retorno de um método. Outros casos involvem operações que precedem o comando de retorno mas que não tem seus resultados usados em nenhum lugar. A figura 3 mostra um exemplo aplicado a representação intermediária usada no trabalho.

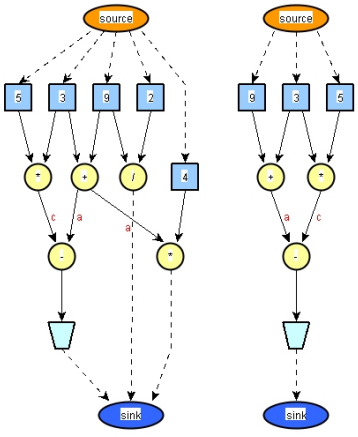

Figura 3: Eliminação de código morto

2.5 Linguagem de montagem

A linguagem de montagem é equivalente a linguagem de máquina, que tem instruções e variáveis totalmente codificadas em binário, mas em vez da notação puramente binária

a linguagem usa mnemónicas para especificar as operações pretendidas, bem como os valores ou localizações dos operandos. Embora seja melhor manuseado por seres humanos do que a linguagem de máquina, ele ainda é inteiramente dependente do conjunto de instruções dum dado processador, isto é, não é portátil entre processadores de famílias diferentes.

Para converter de código escrito em linguagem de montagem para código de máquina utiliza-se programas chamados de montadores (*assemblers*).

2.5.1 Alocação de registradores

Um dos problemas quando deseja-se escrever código de montagem é mapear variáveis e valores temporários para um conjunto fixo de registradores (juntamente com o espaço disponível na memória).

Para resolver o problema da alocação de registradores utiliza-se coloração de grafos. Cria-se um grafo de intervalo[2], e após a coloração, garante-se que um valor (representado por um nodo) não vai alterar o valor de outro porque não usam o mesmo registrador caso haja intersecção no seu tempo de vida.

Há vários algoritmos de coloração de grafos. Neste trabalho foi usado um algoritmo simples que colore os vértices sequencialmente, um de cada vez. Este algoritmo não garante que a coloraçao seja mínima. O algoritmo e exemplos podem ser vistos em [3].

Há programas onde não é possível converter todas as variáveis e temporários em registradores. Nestes casos, os valores nos registradores precisam ser salvos na memória (técnica chamada de "memory spilling"), e quando necessário, colocados em algum registrador novamente.

[2]Veja o anexo D para as definições de coloração de grafos e grafos de intervalo

3 Fluxo de Síntese de Software

O fluxo de síntese de software pode ser visto na figura 4. Os retângulos denotam ferramentas. Entradas, saídas e representações intermediárias são representadas por elipses.

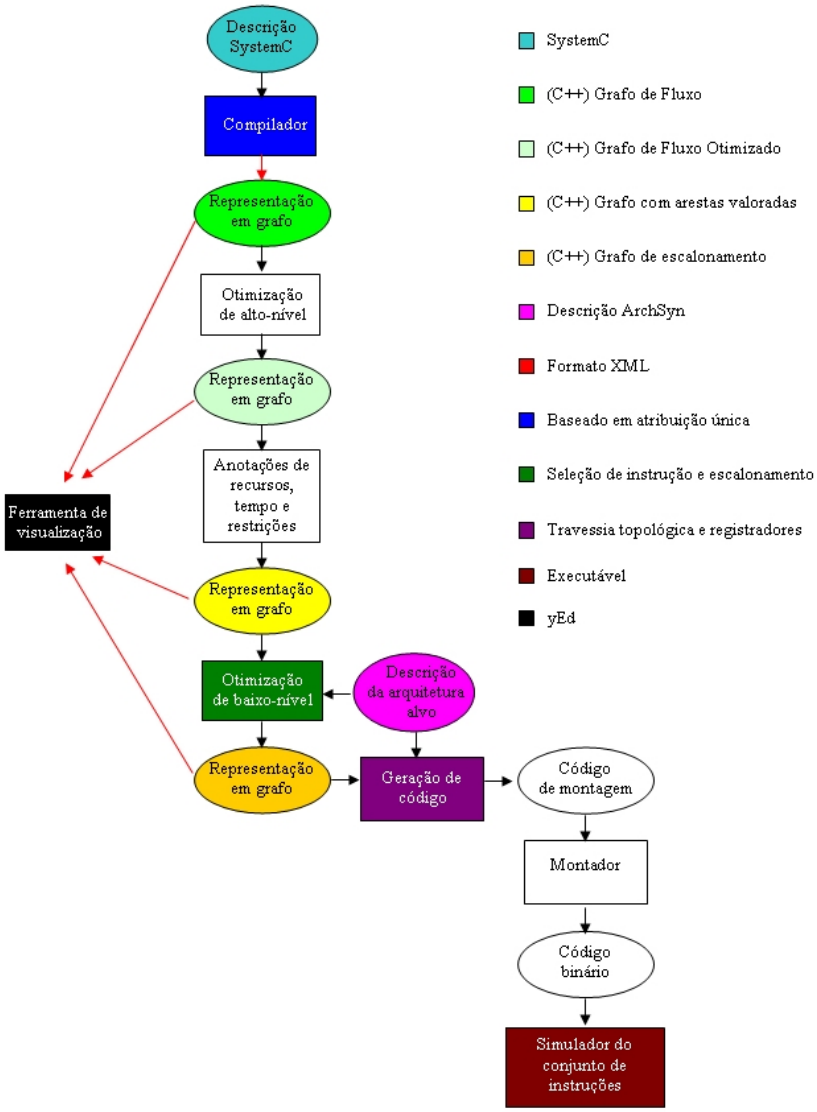

Figura 4: Panorama de um fluxo completo de síntese de software

O fluxo começa a partir de uma descrição em SystemC do sistema. Esta descrição

precisa estar no formato de atribuição única (veja a tabela 1). Neste formato a descrição está pronta para ser compilada em um grafo de fluxo de dados, que é a representação intermediária do fluxo e onde serão feitas todas as otimizações desejadas. O grafo (otimizado ou não) serve de entrada para o gerador de código, que tem como saída um arquivo escrito em código de montagem para o processador MIPS. Com um montador é possível gerar código de máquina (código binário) e executar o programa em uma máquina com processador MIPS ou em um simulador da arquitetura MIPS (ArchC [2], por exemplo).

3.1 Um protótipo para o Fluxo de Síntese de Software

Primeiramente, é necessário especificar quais funcionalidades serão implementadas. O fluxo de síntese foi mostrado e comentado na seção anterior. Na próxima, serão mostradas as restrições impostas ao protótipo para viabilizar sua implementação em tempo hábil.

3.1.1 Restrições

As ferramentas implementadas ou utilizadas no trabalho possuem algumas restrições ou simplificações. Nas próximas seções serão listadas algumas características do compilador implementado, das otimizações realizadas, do código de montagem gerado, e do simulador de arquitetura utilizado.

3.1.1.1 Subconjunto de SystemC

O subconjunto de SystemC implementado é apresentado nessa seção. Algumas funcionalidades presentes em SystemC não fazem sentido no escopo de sistemas embutidos, outras foram retiradas para simplificar a construção das ferramentas.

A tabela 3 mostra as construções de SystemC/C++ implementadas no subconjunto. A primeira coluna diz qual a categoria da construção, a segunda coluna mostra um exemplo ou palavra-chave, e a terceira coluna traz comentários ou observações sobre a implementação da funcionalidade, se necessário.

Apenas dois tipos de dados primitivos são implementados. Eles são mostrados na tabela 4.

A tabela 5 mostra os operadores aceitos. Eles podem ser usados com o tipos de dados

Tabela 3: Construções aceitas pela linguagem

Categoria	**Construção**	**Observações**
Comentários	Bloco /* */ e linha //	
Identificadores	Ex.: var, _var, var3	Começando com letra ou sublinhado e seguido por letras, números ou sublinhado
Classes	SC_MODULE e class	
Funções	tipo nome() { }	
Expressões	Ex.: a + b	
Arquivos incluídos	#include arquivo	Ignorados, como os comentários.
Declarações	tipo nome;	
Retorno de função	return	
Atribuições	Ex.: a = b + c	
Construtor	SC_MODULE ou classe()	
Modificadores de acesso	public, protected, private	Não fazem sentido na abordagem de grafos, mas são necessários em C++

Tabela 4: Tipos de dados aceitos

Categoria	**Construção**	**Comentários**
Inteiros	int	Decimais
Booleanos	bool	true ou false

mostrados na tabela 4.

Tabela 5: Operadores aceitos

Categoria	**Construção**
Aritméticos	/, *, +, -, %
Relacionais	!=, ==, >, <, >=, <=, !

Outras coisas importantes sobre o subconjunto da linguagem:

- Aceita instanciação de objetos das classes já declaradas.
- Aceita passagem de parâmetros nos métodos, mas somente variáveis ou constantes, não aceita expressões ou chamada de função como parâmetro.
- Não aceita declaração de variáveis globais. Isso porque todos os dados disponíveis

em um DFG entram por um único nodo (chamado de *source*), então os dados que o método necessita precisam ser passados como parâmetro.

- Não aceita parênteses em expressões.
- Não há precedência das operações. As expressões são avaliadas da esquerda para a direita.

Abaixo está um exemplo de código escrito com o subconjunto implementado.

```
#include "systemc.h"

int some(int x, int y) {
    return x + y;
}

class MinhaClasse {
    public:

    int divide(int x, int y) {
        return x / y;
    }

    MinhaClasse () { }

    int multiply() {
        int x = 1;
        int y = 2;
        return x * y;
    }

};

int main() {

    MinhaClasse m;
    int c = 9 + 4;
    int a = m.divide(c, 10);
    int a1 = -a * m.multiply();
    int a2 = a1 + some(0, 5);
    int b = 8 % c;
    return a2 - b;
```

```
}
```

3.1.1.2 Otimizações

A otimização feita nos grafos será limitada, serão implementados apenas alguns algoritmos simples. Também serão restritas ao bloco básico.

Não é possível fazer otimizações de baixo nível porque o simulador da arquitetura não suporta a descrição do *pipeline*, ou seja, não é *cycle-accurate*.

3.1.1.3 Geração de código

A alocação de registradores é estática, feita a partir da coloração do grafo de intervalo das variáveis. Não está previsto escalonamento de código.

Apenas a arquitetura MIPS será alvo deste trabalho.

3.1.1.4 Simulador da arquitetura MIPS

O simulador da arquitetura MIPS utilizado no trabalho (ArchC) não é *cycle-accurate*, o que retira uma métrica importante para medir o desempenho do código final.

4 Implementação

Todas as ferramentas foram implementadas em C++ e funcionam no sistema operacional Linux. Nenhuma tem interface gráfica, funcionam com comandos executados a partir do console.

Foram implementados alguns *shell scripts*, que chamam as ferramentas na seqüência certa, e somente se a anterior foi executada com sucesso.

Nas próximas seções serão descritas as ferramentas implementadas neste trabalho. A figura 5 mostra o diagrama UML correspondente à implementação.

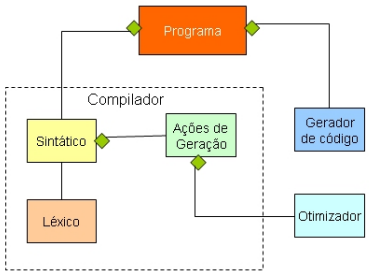

Figura 5: Diagrama UML das ferramentas

4.1 Pacote IOO

Para realizar a implementação das ferramentas do protótipo, foram utilizadas classes e estruturas de dados já disponíveis, implementadas por Felipe Vieira Klein (veja [4]). Foram feitas algumas modificações e inclusões no código original para suprir algumas necessidades da implementação do protótipo. Estão incluídas no pacote classes para grafos, tabela de hash, listas, vetores, nodos, arestas, entre outros.

4.2 Compilador

Um compilador tradicionalmente é dividido em duas partes ou fases: análise e geração de código. A fase de análise pode ser divida em outras três: análise léxica, sintática e semântica.

O analisador léxico e o analisador semântico foram gerados pela ferramenta ANTLR [1] a partir de gramáticas LL(k) (livres de contexto) tanto para a parte léxica (tradicionalmente implementada com gramáticas regulares) quanto para a parte sintática.

O analisador semântico não foi implementado. Isso porque uma descrição em SystemC precisa incluir o arquivo de cabeçalho "systemc.h", e certamente o compilador geraria erros caso fosse avaliar este arquivo porque ele implementa apenas um subconjunto de SystemC/C++.

Para resolver o problema, um *shell script* chama o compilador g++ antes do compilador. Ele verifica se a descrição SystemC está correta do ponto de vista léxico, sintático e semântico de SystemC/C++.

Se estiver tudo correto, o *script* chama o compilador implementado que faz a análise léxica e sintática do arquivo para verificar se as construções utilizadas estão dentro do subconjunto que a ferramenta implementa.

Se ocorrer algum erro léxico ou sintático o compilador mostra uma mensagem de erro e aborta.

Junto com a análise sintática é feita a geração dos grafos de fluxo de dados (geração orientada à sintaxe). Cada procedimento gera um DFG. Vértices são criados para operações (aritméticas e lógicas), chamadas de função, parâmetros, comandos de retorno e constantes. Arestas são criadas quando há fluxo de dados entre dois vértices. Quando o método chega ao fim, são criados os vértices *source* e *sink*, usados para polarizar o grafo. Arestas são criadas do vértice *source* para todos os outros vértices que não tenham nenhuma aresta entrante (com exceção do vértice *sink*). Da mesma forma, são adicionadas arestas de todos os vértices que não tenham nenhuma aresta saindo dele para o vértice *sink*.

O compilador chama o otimizador de código, caso a opção de otimização estiver habilitada. Também gera o grafo de intervalo de variáveis, que será utilizado pelo gerador de código para a alocação de registradores. O otimizador é chamado antes da geração do grafo de intervalo das variáveis, porque durante a otimização variáveis podem ser eliminadas.

A figura 6 mostra o DFG equivalente ao código da função *"main"* exemplo mostrado no capítulo anterior. Mais detalhes sobre a visualização podem ser vistas nos anexos.

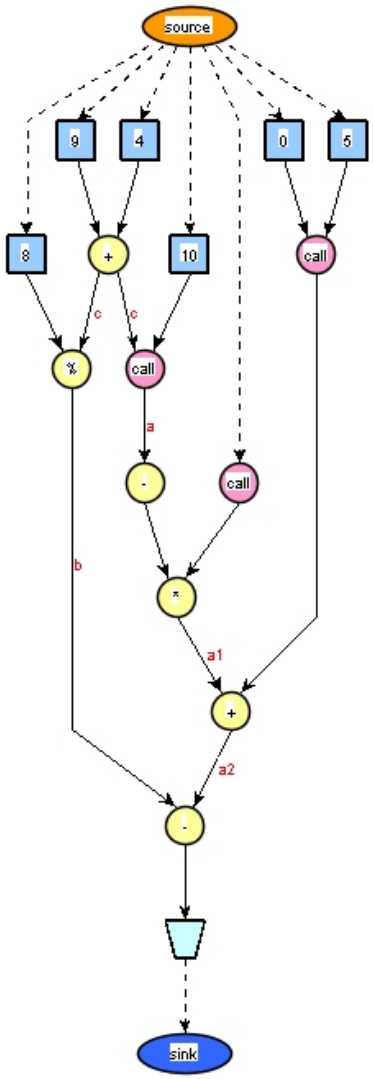

Figura 6: DFG correspondente a função *main* do código presente na seção 3.1.1.1

4.3 Otimizador

As otimizações realizadas se aplicam apenas a um DFG correspondente a um procedimento e não trocam informações com outros DFGs (outros procedimentos). São portanto otimizações locais, se aplicam ao bloco básico.

Foram implementadas as seguintes otimizações:

- Eliminação de código morto.

- Eliminação de subexpressões comuns.
- Propagação de constantes.
- Propagação de variáveis.

Pode-se citar outros tipos de otimização como:

- Multiplicação por 0 retorna 0 e por 1 retorna o multiplicador diferente de 1. Essas multiplicações são detectadas na etapa de otimização e eliminadas.
- Soma ou subtração por 0 retorna o próprio valor.
- A constante 0 já está presente no conjunto de registradores, o registrador $0 (ou $zero) do MIPS sempre retorna a constante.

As otimizações estão ativadas por padrão, mas podem ser desligadas editando-se um arquivo de configuração.

Mais detalhes e ilustrações das otimizações podem ser vistas na seção 2.4.

4.4 Gerador de código

O gerador de código transforma os grafos de fluxo em código de montagem do MIPS.

Como discutido anteriormente, é necessário que um valor seja computado antes de ser utilizado. Para garantir isso, é feita uma travessia em profundidade pós-ordem no grafo. Durante a travessia, os nodos são marcados se forem folhas (não tem nenhuma aresta saindo dele), ou se todos os nodos alcançáveis a partir dele já foram marcados. Esta travessia define então, uma ordem em que os nodos precisam ser computados. Essa ordem é inversa (o último nodo marcado será o primeiro a ser computado e o primeiro nodo marcado será o último) à ordem que os nodos foram marcados. Como exemplo, veja a figura 7.

A alocação de registradores para as variáveis é estática, obtida a partir do grafo de intervalos de vida gerado pelo compilador. A alocação de registradores temporários é controlada: procura-se um registrador vazio e aloca-se ele para algum valor. E sempre após uma operação verifica-se se os registradores utilizados como operandos podem ser liberados e marcados como livres.

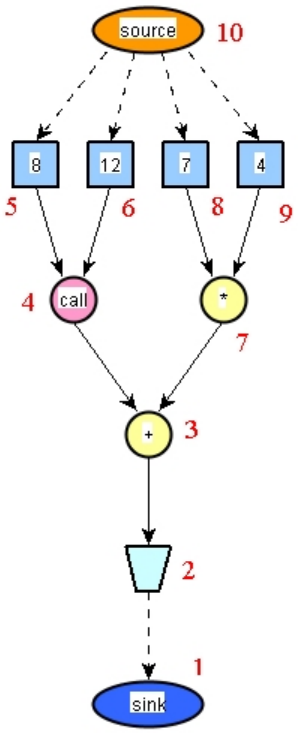

Figura 7: Possível ordem de avaliação dos nodos (inversa, de 10 até 1)

Quando acontecer de não haver registradores livres, usa-se *memory spilling*, o valor de um registrador é salvo na memória, e então ele é liberado para ser usado.

A tabela 6 mostra as intruções utilizadas no trabalho[1].

Para mais detalhes sobre a implementação de código de montagem (salvamento e recuperação de contexto, passagem de parâmetros, etc), veja o anexo B.

[1] $ denota um registrador

Tabela 6: Instruções do MIPS utilizadas no trabalho

Operação	**Instruções**	**Comentários**
Adição (+)	addu $, $, $	Soma quando os operandos estão em registradores
Adição (+)	addi $, $, constante	Soma com um dos operandos constante
Subtração (-)	subu $, $, $	
Subtração (-)	subi $, $, constante	
Multiplicação (*)	mult $, $ mflo $	O resultado de uma multiplicação é armazenado no registrador Lo, e acessado com a instrução mfhi
Divisão (/)	div $, $ mflo $	O quociente de uma divisão é armazenado no registrador Lo, e acessado com a instrução mflo
Módulo (%)	div $, $ mfhi $	O resto de uma divisão é armazenado no registrador Lo, e acessado com a instrução mfhi
Alterar sinal (-)	negu $, $	
E (&&)	and $, $, $	
Ou (||)	or $, $, $	
Igual (==)	seq $, $, $	
Diferente (!=)	sne $, $, $	
Negação (!)	not $, $	
Maior que (>)	sgt $, $, $	
Menor que (<)	slt $, $, $	
Maior igual que (>=)	sge $, $, $	
Menor igual que (<=)	sle $, $, $	
Chamada de procedimento	jal rótulo	
Retorno de função	jr $ra	
Carregar constante	li $, constante	Carregar a constante no registrador
Carregar variável	lw $, variável	Carregar a variável (disponível no segmento de dados) no registrador
Carregar da memória	lw $, constante ($)	Carregar no registrador o valor armazenado no endereço de memória apontado por $ somado com a constante
Salvar na memória	sw $, constante ($)	Salvar o valor do registrador no endereço apontado por $ somado com a constante

5 Validação do Protótipo

A validação do protótipo consiste em verificar se o sistema cumpriu com os objetivos traçados. Experimentos testam se as funcionalidades que o sistema implementa funcionam corretamente. A validação do sistema será mostrada nas próximas seções.

5.1 Validação do Modelo da Arquitetura Alvo

A validação do modelo da arquitetura alvo é um experimento que verifica se o código de máquina gerado pelo protótipo é equivalente ao código gerado por um compilador feito para a arquitetura, no caso do experimento será usado o compilador GNU g++.

A partir do confronto entre as respostas apresentadas pelo sistema implementado e pelo compilador GNU pode-se dizer se os códigos são equivalentes ou não. A figura 8 ilustra o experimento.

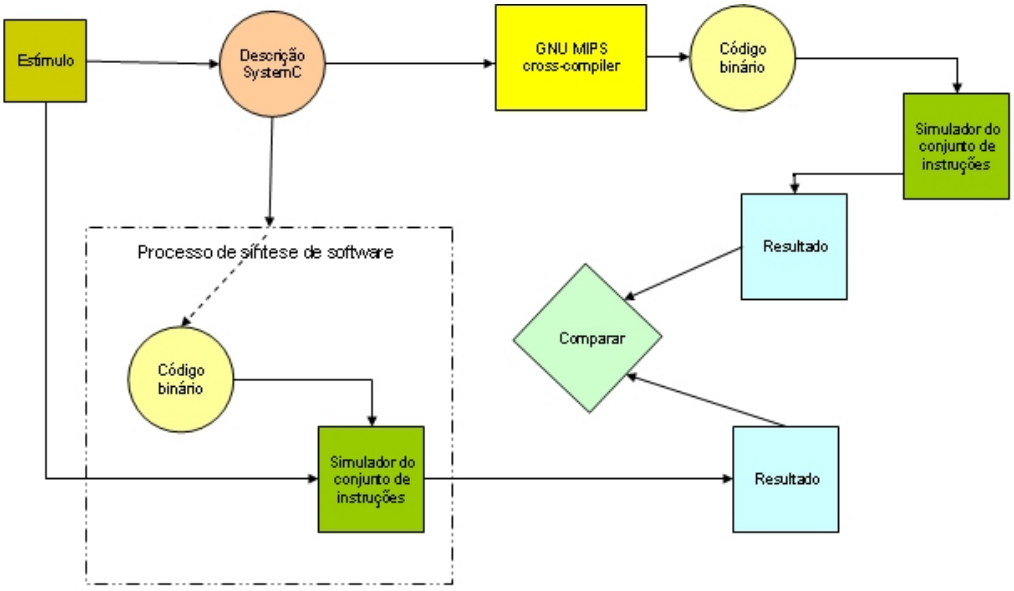

Figura 8: Validação funcional do protótipo de síntese de software

5.1.1 MIPS Cross-Toolchain

Toolchain é uma coleção de programas usados para desenvolvimento e construção de programas para uma determinada arquitetura alvo. Ela roda em uma arquitetura mas produz código objeto para a arquitetura alvo. *Cross-Toolchain* também é referenciado como *Cross-Compiler*.

Um dos usos é gerar programas quando não se tem uma máquina com a arquitetura alvo ou não é possível compilar o código nesta máquina (muitos dos casos quando trabalha-se com sistemas embutidos).

Agradecimentos a Sandro Rigo (autor do ArchC e do simulador da arquitetura MIPS), que cedeu os binários do *Cross-Toolchain* utilizados por ele na Unicamp.

5.1.2 Experimento

Foi feito um experimento onde o mesmo código fonte (escrito em SystemC/C++) foi submetido ao compilador GNU (presente no MIPS Cross-Toolchain apresentado anteriormente) e a ferramenta protótipo.

As duas ferramentas geraram como saída código de montagem, onde foi adicionado uma chamada ao procedimento "printf"(que imprime caracteres na saída padrão) presente na biblioteca de C, para que o resultado final fosse mostrado no tela.

Um shell script, que toma como parâmetros o arquivo assembly de entrada e o nome desejado para o executável como saída, chama o montador e o ligador (presentes no Cross-Toolchain) com as opções e bibliotecas necessárias, produzindo o executável.

O simulador do MIPS executou ambos os programas e o resultado impresso na tela foi o mesmo.

O experimento foi repetido várias vezes com diferentes códigos fonte de entrada (em SystemC/C++) e o resultado final do protótipo e do compilador GNU sempre foi igual.

5.2 Validação do Protótipo de Síntese de Software

A validação do protótipo de síntese de software tem como objetivo comparar o desempenho do código gerado pelo sistema com o código gerado pelo compilador GNU.

5.2.1 Experimento

Um dos experimentos feitos foi calcular o décimo primeiro número da série de Fibonacci. Veja o código fonte:

```
int main() {

    int a = 0;
    int b = 1;
    int c = a + b;
    int d = b + c;
    int e = c + d;
    int f = d + e;
    int g = e + f;
    int h = f + g;
    int i = g + h;
    int j = h + i;
    int k = i + j;

return k;

}
```

O código otimizado produzido pelo protótipo e pelo compilador GNU são equivalentes. O código do compilador GNU contém mais diretivas enquanto o código produzido pelo protótipo tem mais instruções porque faz salvamento e recuperação de contexto. A tabela 7 mostra os dois códigos.

5.2.2 Impacto das Otimizações

Em um programa como o anterior com apenas somas, o impacto das otimizações se mostra muito grande. No caso do experimento anterior, a única otimização aplicada pelo protótipo foi a propagação de constantes. O compilador GNU compilou o código fonte com a opção de otimizações locais ("-O1").

A tabela 8 mostra um comparação de quantas instruções são executadas com ou sem otimizações.

Tabela 7: Código assembly do experimento

Código do protótipo	**Código do compilador GNU**
.text	.file 1 "fibonacci.cpp"
.globl main	.section .mdebug.abi32
main: sw $fp, -4($sp)	.previous
sw $31, -8($sp)	.text
move $fp, $sp	.align 2
addu $sp, $sp, -8	.globl main
li $2, 55	$LFB1:
addu $sp, $sp, 8	.ent main
lw $fp, -4($sp)	main:
lw $31, -8($sp)	.frame $sp,0,$31
jr $31	.mask 0x00000000,0
.data	.fmask 0x00000000,0
	.set noreorder
	.set nomacro
	j $31
	li $2,55
	.set macro
	.set reorder
	.end main
	$LFE1:

Tabela 8: Comparação de código otimizado com não-otimizado

Número de instruções	**Compilador GNU**	**Protótipo**
Não-otimizado	43	21
Otimizado	2	9

A diferença no caso do código otimizado foi explicada na seção anterior. A diferença no caso do código não-otimizado acontece porque o compilador GNU usa alguns poucos registradores disponíveis na máquina apenas, e faz muito *memory spilling*. O protótipo trabalha somente com registradores, não há necessidade de fazer *memory spilling*.

6 Conclusões e Perspectivas

Neste trabalho foi apresentado um fluxo completo de síntese de software para sistemas embutidos. Foram apresentadas e justificadas as escolhas da entrada, representação intermediária e saída do fluxo, foram descritas as ferramentas implementadas para realizar o que foi proposto e mostrados os resultados obtidos.

As simplificações e restrições adotadas visaram viabilizar a implementação do protótipo, mas sem perda de generalidade.

6.1 Trabalhos futuros

Este trabalho pode ser complementado com a implementação de outras ferramentas, utilização de programas com mais funcionalidades ou com experimentos diferentes. Pode-se citar:

- Fazer uma implementação do gerador de código que leia a descrição da arquitetura de um arquivo de configuração e gere código.
- Incluir mais funcionalidades de SystemC.
- Um simulador de arquitetura *cycle-accurate* é necessário para melhor mostrar o desempenho do código gerado.
- Mais algoritmos de otimização poderiam resultar em um código mais eficiente.
- O experimento da seção 5.2 poderia ser repetido e uma comparação de desempenho poderia substituir a comparação de equivalência assim que o simulador *cycle-accurate* estiver disponível . Por exemplo, a latência do pior caso poderia ser medida em ambos os caminhos para estimar a aceleração.

6.2 Conclusões finais

Pode-se citar algumas conclusões obtidas ao fim do trabalho:

- A ferramenta facilita a elaboração e generalização do modelo baseado em grafos permitindo facilidades de visualização e procura por erros.
- É fornecida infra-estrutura para avaliações de desempenho de experimentos futuros.
- O sistema traça o caminho para o desenvolvimento futuro de um fluxo completo de síntese de software.
- O impacto de otimizações simples, como a propagação de constantes, pode ser grande no desempenho final dependendo do programa.
- O uso de grafos como representação intermediária mostrou-se uma opção viável no processo de compilação.

Referências Bibliográficas

[1] ANTLR. ANother Tool for Language Recognition. Disponível em <http://www.antlr.org>. Acesso em 27 nov. 2003.

[2] ArchC. The ArchC Architecture Description Language. Disponível em <http://www.archc.org>. Acesso em 05 fev. 2004.

[3] DE MICHELI, Giovanni. Synthesis and Optimization of Digital Circuits, McGraw-Hill, 1994.

[4] KLEIN, Felipe Vieira. Modelagem de Arquiteturas de Sistemas Digitais: Implementação e Ferramentas de Síntese Automática. Trabalho de Fim de Curso, UFSC, ago. de 2002.

[5] HENNESSY, John L., PATTERSON, David A. Computer Architecture: A Quantitative Approach. Morgan Kaufmann Publishers, second edition, 1995.

[6] LARMAN, Craig. Applying UML and Patterns - An Introduction to Object-Oriented Analysis and Design, Prentice-Hall, 1999.

[7] SWEETMAN, Dominic. See MIPS Run. Morgan Kaufmann Publishers, 1999.

[8] SYSTEMC. SystemC Community. Disponível em <http://www.systemc.org>. Acesso em 27 nov. 2003.

[9] SWAN, Stuart. An Introduction to System Level Modeling in SystemC 2.0. Disponível em <http://www.cadence.com/whitepapers/systemc_wp20.pdf>. Acesso em 27 nov. 2003.

[10] WOLF, Wayne. Computers as Components: Principles of Embedded Computing Systems Design. Morgan Kaufmann Publishers, 2000.

[11] yEd. yEd JavaTM Graph Editor. Disponível em <http://www.yworks.com/en/products_yed_about.htm>. Acesso em 05 dez. 2003.

ANEXO A – Ferramentas Auxiliares Implementadas

A.1 Grafo para XML

Para possibilitar a visualização do DFG graficamente foi implementada uma ferramenta que recebe um DFG como entrada e gera um arquivo de saída em um padrão XML aceito pela ferramenta de visualização de grafos yEd [11] (formato yGraphML).

A ferramenta cria nodos com cores e formatos diferentes dependendo do tipo de cada um (operação, constante, chamada de função, etc). Para as arestas existe apenas duas classificações: dados e sequenciamento.

Os nodos podem possuir rótulos indicando melhor qual a sua função. Para as operações (tanto lógicas quanto aritméticas) o rótulo indica qual das operações possíveis o nodo representa, por exemplo. Os nodos tem o seguinte significado:

- Quadrados cor azul: representam constantes.
- Triângulos cor verde claro: representam os argumentos do método (entradas).
- Trapézio cor azul claro: representam os valores de retorno (saídas).
- Círculos cor amarela: representam operações aritméticas.
- Círculos cor verde: representam operações lógicas.
- Círculos cor rosa: representam chamadas de procedimento.
- Elipse cor laranja: representa o nodo “source”do DFG.
- Elipse cor azul escuro: representa o nodo “sink”do DFG.

As arestas ligam os nodos, e seu significado está associado ao fluxo de dados entre eles. As arestas podem ter um rótulo associado, que representa a variável correspondente a aresta no código original (descrição SystemC).

- Arestas contínuas representam fluxo de dados entre dois nodos.
- Arestas hachuradas representam apenas sequenciamento do grafo (não transportam dados, não tem custo nenhum).

A figura 9 mostra um exemplo com todos os tipos de nodos e arestas possíveis de visualização.

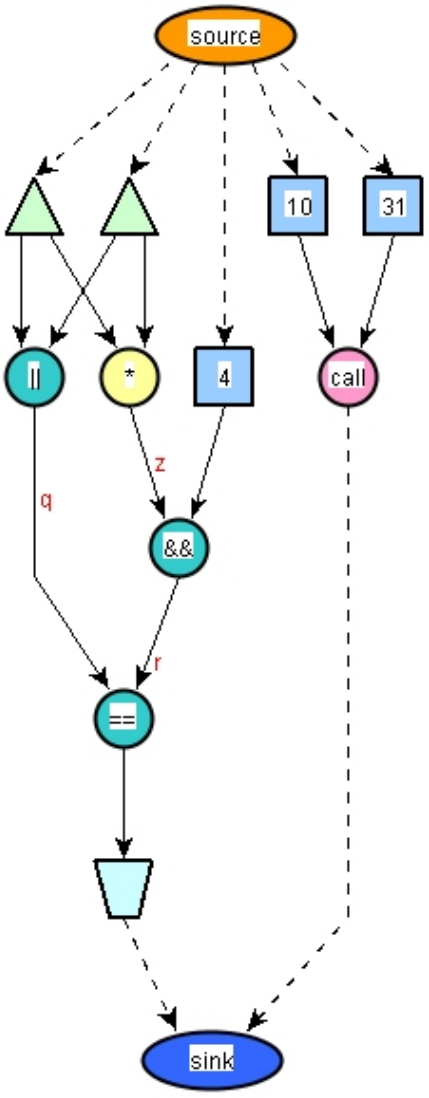

Figura 9: Grafo de fluxo de dados visualizado no yEd

ANEXO B – Visão geral da arquitetura MIPS e sua linguagem de montagem

Este anexo não se destina a ensinar a escrever código de montagem para o MIPS, ele apenas fala um pouco sobre os conceitos e funcionalidades usadas no trabalho. Para mais detalhes leia [7].

B.1 Processador

O processador tem 32 registradores de propósito geral, todos eles de 32 bits. Eles estão numerados de 0..31 e também podem ser referenciados por nomes simbólicos, que sugerem convenções para o uso dos registradores. Dos 32 registradores, o primeiro e o último são especiais:

- $0 é a constante 0, e não pode ser alterada.
- $31 armazena o endereço de retorno de uma chamada de função. É chamado de $ra.

A tabela 9 mostra o uso convencionado para alguns registradores.

B.2 Conjunto de instruções

Instruções podem ser agrupadas nas seguintes categorias:

Carregar e armazenar (*load and store*): carrega dados da memória em um registrador, ou armazena o conteúdo de um registrador na mémoria. Exemplos:

Tabela 9: Registradores do MIPS

Número do registrador	**Nome**	**Usado para**
0	zero	Sempre retorna 0
2-3	v0, v1	Valor retornado por uma subrotina
4-7	a0-a3	(argumentos) Primeiros quatro parâmetros de uma subrotina
8-15 24, 25	t0-t7 t8, t9	(temporarios) Subrotinas podem utilizá-los sem salvar
16-23	s0-s7	Uma subrotina que escreve em algum desses precisa salvar seu valor antigo e restaurar antes de retornar, de maneira que a rotina chamadora tenha seus valores preservados
29	sp	Stack pointer
30	fp	Frame pointer
31	ra	Endereço de retorno de uma subrotina

```
lw $t0, num1 # carrega palavra num1 em $t0
sw $t0, 0($sp) # armazena o conteúdo de $t0 na posição
               # de memória apontada por $sp + constante
```

Operações aritméticas e lógicas: realiza operações nos dados dos registradores e armazena o resultado em um outro registrador. Exemplo:

```
add $t0, $t3, $t4 # $t0 = $t3 + $t4
subi $t0, $t3, 4 # $t0 = $t3 - 4
```

Desvios: altera o endereço da próxima instrução a ser executada (armazenado em um registrador denominado *Program Counter*). Exemplo:

```
jal procedimento # executa procedimento
```

B.3 Estrutura de um programa

Um programa está dividido em alguns segmentos. Entre elas:

- Segmento de dados, precedido por pela diretiva **.data**. Declara variáveis usadas no programa e que serão salvas na memória.
- Segmento de código, precedido pela diretiva **.text**. Contém as instruções do programa. O ponto inicial de execução do programa deve ser rotulado de **main**.

B.4 Chamadas de função e retorno

Quando é feita uma chamada de função normalmente deseja-se retornar ao ponto de partida após o término da função chamada. O MIPS usa o registrador 31 ($ra) para armazenar o endereço de retorno. Há duas instruções envolvidas no processo. São elas:

- jal (jump and link): salva o endereço atual do PC no registrador $ra e pula para um procedimento.
- jr $ra coloca o valor de $ra no PC, fazendo a função retornar a função chamadora.

Como convenção, ainda temos que os parâmetros passados da função chamadora para a função chamada são salvos nos registradores $a0-$a3, e os valores de retorno da função chamada são retornados nos registradores $v0 e $v1.

B.5 Chamadas de função e a pilha

A seção anterior mostrou que para chamadas de função e retorno usamos jal e jr. Mas essa não é toda a história, pois uma chamada de função dentro da função chamada destruiria o endereço armazenado em $ra. Para superar este problema utiliza-se a pilha (*stack*).

B.5.1 *Stack e frame pointer*

O *stack pointer* aponta para o menor endereço (topo) da pilha. Ele pode aumentar e diminuir durante a execução de um procedimento.

O *frame pointer* fica fixo durante toda a execução do procedimento. Aponta para o endereço que o *stack pointer* tinha quando o procedimento foi iniciado.

B.5.2 Salvar e restaurar o contexto

Antes de fazer uma chamada de função, a função chamadora precisa:

- Salvar registradores de parâmetros utilizados na função chamadora ($a0-$a3) e os temporários utilizados ($t0-$t9).
- Salvar os parâmetros da função nos registradores $a0-$a3. Se houver mais de quatro, salvar os restantes na pilha.
- Executar a instrução jal.

Quando um procedimento é iniciado, ele precisa salvar o contexto:

- Salvar os registradores $s0-$s7 usados no método, assim como o registrador $fp e o $ra.
- Estabelecer o novo frame pointer.

Antes de retornar a função chamadora, salva-se o valor de retorno e restaura-se o contexto:

- Salvar valores de retorno em $v0 e $v1.
- Restaurar todos os registradores salvos na pilha no início do método.
- Executar a instrução jr.

Após retornar de uma chamada de função, restaura-se os registradores salvos na pilha (argumentos e temporários).

B.6 *Memory spilling*

Quando há necessidade de utilizar um registrador, mas não há mais registradores livres na máquina, é preciso salvar o conteúdo de algum registrador na memória para que ele possa ser utilizado, mas seu valor não seja perdido, e quando necessário, carregar

o valor da memória para o registrador novamente. O valor é salvo no topo da pilha, apontado pelo *stack pointer* que é variável. O acesso a esse valor será feito através do *frame pointer*, que fica fixo durante toda a execução do procedimento.

ANEXO C – Ferramentas utilizadas

C.1 ANTLR

ANTLR, *ANother Tool for Language Recognition*, (antigo PCCTS) é uma ferramenta que fornece um *framework* para construir reconhecedores, compiladores, e tradutores a partir de descrições gramaticais contendo ações Java, C#, ou C++. ANTLR possui uma sintaxe consistente para especificar analisadores léxicos, sintáticos e construir árvores sintáticas, e tem suporte para predicados sintáticos e semânticos, e *lookahead* com a quantidade desejada. ANTLR é um analisador do tipo LL(k). Para mais informações veja [1].

C.2 ArchC

ArchC é uma linguagem de descrição de arquitetura código aberto que está sendo desenvolvida no Laboratório de Sistemas Computacionais (LSC) do Instituto de Computação da Universidade de Campinas (IC-UNICAMP).

ArchC é baseado em SystemC. O objetivo de ArchC é fornecer uma ferramenta poderosa que permite avaliar rapidamente novas idéias em áreas como: desenvolvimento de processadores e ISA, hierarquia de memória e outros aspectos da pesquisa de arquitetura de computadores.

Uma descrição de arquitetura em ArchC é dividida em duas partes: a arquitetura do conjunto de instruções (AC_ISA) e os elementos da arquitetura (AC_ARCH). Na descrição AC_ISA o projetista especifica detalhes sobre o conjunto de instruções como: nome das instruções, formatos, tamanhos, assim como cada comportamento de uma instrução e informações necessárias para decodificá-la. Na descrição AC_ARCH o projetista lista os recursos da arquitetura como módulos de armazenamento, a estrutura do *pipeline*, etc.

O ArchC ainda não está disponível ao público geral.

ANEXO D – Definições de grafos

Neste anexo serão apresentadas algumas definições e teorias sobre grafos citadas durante o trabalho. Para mais detalhes, leia [3].

D.1 Grafos polares

Um grafo polar de fluxo de dados DFG(V, A) é um grafo orientado, onde cada vértice vi ∈ V representa uma operação, e cada aresta definida por (vi, vj) ∈ A caracteriza uma dependência de dados entre as operações vi e vj. Os pólos são os vértices v0 e vn, e são chamados de fonte ("source") e sumidouro ("sink").

D.2 Coloração de grafos

Colorar os vértices de um grafo não-dirigido G(V,E) é rotular os vértices de forma que nenhuma aresta em E tenha dois extremos com o mesmo rótulo. O problema de decisão de coloração de vértices é determinar se há um número cromático menor (ou igual) a um determinado inteiro. O problema de otimização correspondente é a procura por uma coloração com o mínimo número de cores. Este é um problema NP completo.

D.3 Grafo de intervalo

Um grafo de intervalo é um grafo cujos vértices podem ser postos em uma correspondência de um para um com um conjunto de intervalos, de maneira que dois vértices sejam adjacentes se e somente se os intervalos correspondentes tem intersecção.

ANEXO E – Código fonte

E.1 Pacote IOO com suporte a hierarquia

Para mais detalhes sobre o pacote IOO com suporte a hierarquia utilizado no trabalho, leia [4].

E.2 Compilador

Gramática do analisador léxico.

```
/*
 * Make sure to run antlr.Tool on the SystemCLexer.g file first!
 */
options {
mangleLiteralPrefix = "TK_";
language="Cpp";
}

class SystemCLexer extends Lexer;
options {
   k = 2;
   exportVocab=SystemC; // exports SystemCTokenTypes.txt
   charVocabulary = '\3'..'\377';
}

tokens {
PRIVATE = "private";
PROTECTED = "protected";
PUBLIC = "public";
CLASS = "class";
INT = "int" ;
RETURN = "return";
```

```
VOID = "void";
SC_MODULE = "SC_MODULE";
BOOL = "bool";
SC_CTOR  = "SC_CTOR";
BOOLTRUE = "true";
BOOLFALSE = "false";
}

WS_ : (' '
| '\t'
| '\n'{newline();}
| '\r')
{ _ttype = antlr::Token::SKIP; }
;


SL_COMMENT :
"//"
(~'\n')* '\n'
{ _ttype = antlr::Token::SKIP; newline(); }
;

PRE_PROCESSOR :
"#"
(~'\n')* '\n'
{ _ttype = antlr::Token::SKIP; newline(); }
;

ML_COMMENT
: "/*"
( { LA(2)!='/' }? '*'
| '\n' { newline(); }
| ~('*'|'\n')
)*
"*/"
{ $setType(antlr::Token::SKIP); }
;

LPAREN
options {
paraphrase="'('";
}
: '('
```

```
;

RPAREN
options {
paraphrase="')'";
}
: ')'
;

LCURLY
options {
paraphrase="'{'";
}
: '{'
;

RCURLY
options {
paraphrase="'}'";
}
: '}'
;

STAR
options {
paraphrase="'*'";
}
: '*'
;

PLUS
options {
paraphrase="'+'";
}
: '+'
;

DIV
options {
paraphrase="'/'";
}
: '/'
;
```

```
MINUS
options {
paraphrase="'-'";
}
: '-'
;

MOD
options {
paraphrase="'%'";
}
: '%'
;

ASSIGN
options {
paraphrase="'='";
}
: '='
;

SEMI
options {
paraphrase="';'";
}
: ';'
;

COMMA
options {
paraphrase="',''";
}
: ','
;

COLON
options {
paraphrase="':'";
}
: ':'
;
```

```
NOT
options {
paraphrase="'!'";
}
: '!'
;

LESSTHAN
options {
paraphrase="'<'";
}
: '<'
;

GREATERTHAN
options {
paraphrase="'>'";
}
: '>'
;

LESSTHANOREQUALTO
options {
paraphrase="'<='";
}
: "<="
;

GREATERTHANOREQUALTO
options {
paraphrase="'>='";
}
: ">="
;

OR
options {
paraphrase="'||'";
}
: "||"
;

AND
```

```
options {
paraphrase="'&&'";
}
: "&&"
;

NOTEQUAL
options {
paraphrase="'!='";
}
: "!="
;

EQUAL
options {
paraphrase="'=='";
}
: "=="
;

DOT
options {
paraphrase="'.'";
}
: '.'
;

protected
DIGIT
options {
paraphrase="'number'";
}
: '0'..'9'
;

INTLIT
options {
paraphrase="'integer'";
}
: (DIGIT)+
;

ID
```

```
options {
testLiterals = true;
paraphrase = "an identifier";
}
: ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'_'|'0'..'9')*
;
```

Gramática do analisador sintático.

```
header {

ANTLR_USING_NAMESPACE(std)
ANTLR_USING_NAMESPACE(antlr)

#include "SystemCToGraph.h"


}

options {
language="Cpp";
}

{
// My code section


}

class SystemCParser extends Parser;
options {
importVocab=SystemC; // use vocab generated by lexer
k = 2;

defaultErrorHandler=false ;

}

{
scgraph::SystemCToGraph scg;


public :
```

```
inline scgraph::SystemCToGraph & getSCG() {
return scg;
}

}

initial_symbol :
( class_head | func_declaration )+
;

class_head
:
SC_MODULE LPAREN declarator RPAREN LCURLY { scg.classBegin16(); }
class_body RCURLY SEMI { scg.classEnd19(); }
| CLASS declarator LCURLY { scg.classBegin16(); }
class_body RCURLY SEMI { scg.classEnd19(); }
;

class_body :
( func_declaration | constructor | modifiers )*
;

modifiers :
( PUBLIC | PROTECTED | PRIVATE ) COLON
;

constructor :
SC_CTOR LPAREN declarator RPAREN { scg.constructorBegin17(); }
block { scg.constructorEnd18(); }
| declarator LPAREN RPAREN { scg.constructorBegin17(); }
block { scg.constructorEnd18(); }
;

func_declaration :
type_specifier declarator LPAREN { scg.function03(); }
(listparams)? RPAREN
{ scg.beginFunction04(); } block { scg.endFunction07(); }
;

var_declaration :
type_specifier declarator { scg.varDeclaration11(); }
(ASSIGN { scg.assign12(); } expression { scg.assignEnd13(); } )?
SEMI
```

```
;

instance_declaration
{ std::string s; }
: ident:ID
{ s = ident->getText(); scg.classIdentifier20(((char *)s.c_str())); }
declarator SEMI { scg.classInstantiation21(); }
;

declarator
{ std::string s; }
: id:ID { s = id->getText(); scg.identifier02(((char *)s.c_str())); }
;

listparams  :
type_specifier declarator { scg.argument09(); }
( COMMA type_specifier declarator { scg.argument09(); } )*
;

block :
LCURLY (listcmds)? RCURLY
;


listcmds :
(ID declarator)=> instance_declaration (listcmds)*
| var_declaration (listcmds)*
| cmd (listcmds)*
;

cmd :
cmdatrib SEMI
| cmdreturn SEMI
;


cmdatrib :
declarator ASSIGN { scg.assign12(); } expression { scg.assignEnd13(); }
;

cmdreturn :
RETURN expression { scg.return06(); }
;
```

```
function_call :
LPAREN { scg.functionCall15(); } ( factor { scg.functionParameter24(); }
(COMMA factor { scg.functionParameter24(); } )* )*  RPAREN
;

function_call2 :
LPAREN ( expression { scg.functionParameter24(); }
(COMMA expression { scg.functionParameter24(); } )* )*  RPAREN
;

functions :
DOT {scg.instDotFunc22(); } declarator { scg.methodIdentifier26(); }
{ scg.instCallFunc23(); } function_call2 { scg.functionCallEnd27(); }
| function_call { scg.functionCallEnd27(); }
;

type_specifier
: INT  { scg.typeSpecifier01(TYPE_INT); }
| BOOL  { scg.typeSpecifier01(TYPE_BOOL); }
;

factor
{ int up; std::string s; }
:
( up = unary_operator { scg.unary14(up); } )?
(
constant:INTLIT { s = constant->getText();
scg.constant05(atoi(((char *)s.c_str()))); }
| (declarator DOT)=> factor_func
| (declarator LPAREN)=> factor_func
| BOOLTRUE { scg.constant05(1); }
| BOOLFALSE { scg.constant05(0); }
| declarator { scg.identifierExpr10(); }
)
;

factor_func :
declarator functions
;


//////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////
//////////////////////////////  EXPRESSIONS ////////////
////////////////////////////////////////////////////////
////////////////////////////////////////////////////////

expression
: assignment_expression
;

/* right-to-left for assignment op */
assignment_expression
: conditional_expression
;

conditional_expression
:
logical_or_expression
;

constant_expression
:
conditional_expression
;

logical_or_expression
:
logical_and_expression (OR { scg.logical25(OR_OP); }
logical_and_expression)*
;

logical_and_expression
:
equality_expression (AND { scg.logical25(AND_OP); }
equality_expression)*
;

equality_expression
:
relational_expression
((NOTEQUAL { scg.logical25(NOTEQUAL_OP); }
| EQUAL { scg.logical25(EQUAL_OP); }) relational_expression)*
;
```

```
relational_expression
: additive_expression
(options {warnWhenFollowAmbig = false;}:
( LESSTHAN { scg.logical25(LESSTHAN_OP); }
| GREATERTHAN { scg.logical25(GREATERTHAN_OP); }
| LESSTHANOREQUALTO { scg.logical25(LESSTHANOREQUALTO_OP); }
| GREATERTHANOREQUALTO { scg.logical25(GREATERTHANOREQUALTO_OP); }
)
 additive_expression
)*
;


additive_expression :
    multiplicative_expression
(options{warnWhenFollowAmbig = false;}:
(PLUS { scg.operation08(PLUS_OP); }
| MINUS { scg.operation08(MINUS_OP); }  ) multiplicative_expression
)*
;


multiplicative_expression
: unary_expression
(options{warnWhenFollowAmbig = false;}:
(STAR { scg.operation08(STAR_OP); }
| DIV { scg.operation08(DIV_OP); }
| MOD { scg.operation08(MOD_OP); }  ) unary_expression
)*
;


unary_expression
:
( postfix_expression
)
;

postfix_expression
:
(
primary_expression
```

```
)
;

primary_expression
: factor
;


unary_operator returns [int ret]
: ret = decimal_unary
| ret = bool_unary
;

decimal_unary returns [int ret]
: PLUS { ret = PLUS_OP; }
| MINUS { ret = MINUS_OP; }
;

bool_unary returns [int ret]
: NOT { ret = NOT_OP; }
;
```

Código do programa principal

```
#include <iostream>
#include <fstream>


#include "SystemCLexer.hpp"
#include "SystemCParser.hpp"
#include "SystemCToGraph.h"
#include "GraphToXML.h"
#include "CodeGen.h"



ANTLR_USING_NAMESPACE(std)
ANTLR_USING_NAMESPACE(antlr)



// Here's where we do the real work...
static int parseFile(const string& input, const string& output)
```

```
{

try
{
ifstream s(input.c_str());

// Create a scanner that reads from the input stream
SystemCLexer lexer(s);
lexer.setFilename(input);

// Create a parser that reads from the scanner
SystemCParser parser(lexer);
parser.setFilename(input);

// start parsing at the compilationUnit rule
parser.initial_symbol();


scgraph::SystemCToGraph *scg = &(parser.getSCG());

Vector< DFGPtr > & dfgs = scg->getDFGs();


Vector < DFGPtr> & cgrs = scg->getCGraphs();


for (int i = 0; i < scg->getNumDFGs(); i++) {
std::stringstream ss ;
ss << output.c_str() << i << ".ygraphml";
graphxml::GraphToXML gxml((ss.str()).c_str());

gxml.generateDFGXML(*dfgs[i]);
}

codegen::CodeGen cdg;

cdg.setVariablesGraph( cgrs, scg->getNumCGraphs() );

cdg.generateCodeDFGs( dfgs, scg->getNumDFGs() );

std::stringstream ss ;
ss << output.c_str()  << ".s";
cdg.write((ss.str()).c_str());
```

```
} catch (RecognitionException& re) {
cerr << re.toString();
return 1;
} catch (ANTLRException& e) {
cerr << "parser exception: " << e.toString() << endl;
return 1;
}
catch (exception& e) {
cerr << "exception: " << e.what() << endl;
return 1;
}

return 0;
}

int main(int argc,char* argv[])
{
// Use a try/catch block for parser exceptions
try
{
// if we have at least one command-line argument
if (argc == 3)
{
cout << "Wait..." << endl;

int result = parseFile(argv[1], argv[2]);
if (result == 1) {
cout << "\nCorrect the errors and try again." << endl;
return 1;
}

cout << "Done." << endl;

}
else {
cerr << "Usage: " << argv[0]
<< " <SystemCFile> <yedFile>" << endl;
}
}
catch(exception& e) {
cerr << "exception: " << e.what() << endl;
```

```
return 1;
}
}
```

E.3 Ações de geração

Código do arquivo SystemCToGraph.h.

```
#ifndef __SystemCToGraph_h__
#define __SystemCToGraph_h__

#include "dfg.h"
#include "dfn.h"
#include "dfe.h"
#include "Hash.h"
#include "Vector.h"

#include <sstream>
#include <string>

#include "defines.h"
#include "Otimizador.h"

namespace scgraph {

struct TableItem {

int value;
int type;
DFN *node;
String name;

int last_used;
int defined;

TableItem() { }
TableItem( const String & Nm ) : name( Nm ) { }

int operator==( const TableItem & Rhs ) const
        { return name == Rhs.name; }
```

```
    int operator!=( const TableItem & Rhs ) const
        { return name != Rhs.name; }

};

struct MethodItem {

int num_params;
int type;
DFN *source;
DFG *grafo;
String name;

MethodItem() { }
MethodItem( const String & Nm ) : name( Nm ) { }

int operator==( const MethodItem & Rhs ) const
        { return name == Rhs.name; }
    int operator!=( const MethodItem & Rhs ) const
        { return name != Rhs.name; }

};

struct ClassItem {

HashTable< MethodItem > method_table;
String name;

ClassItem() { }
ClassItem( const String & Nm ) : name( Nm ) { }

int operator==( const ClassItem & Rhs ) const
        { return name == Rhs.name; }
    int operator!=( const ClassItem & Rhs ) const
        { return name != Rhs.name; }

};

struct InstanceItem {

ClassItem *class_table; //pointer to its variable table
String name;
```

```
InstanceItem() { }
InstanceItem( const String & Nm ) : name( Nm ) { }

int operator==( const InstanceItem & Rhs ) const
        { return name == Rhs.name; }
    int operator!=( const InstanceItem & Rhs ) const
        { return name != Rhs.name; }
};
// Standard String hash function
static unsigned int HashString( const String & Key, int TableSize )
{
    unsigned int HashVal = 0;
    int i = 0;

    while( ( i < Key.length() ) && (Key[ i ] != '\0' ) )
        HashVal = ( HashVal << 5 ) ^ HashVal ^ Key[ i++ ];

    return ( HashVal % TableSize );
}

static unsigned int HashInt( const int & Key, int TableSize ) {
    return ( Key % TableSize );
}

static unsigned int Hash( const TableItem & Key, int TableSize ) {
return HashString(Key.name, TableSize );
}

static unsigned int Hash( const MethodItem & Key, int TableSize ) {
return HashString(Key.name, TableSize );
}

static unsigned int Hash( const ClassItem & Key, int TableSize ) {
return HashString(Key.name, TableSize );
}

static unsigned int Hash( const InstanceItem & Key, int TableSize ) {
return HashString(Key.name, TableSize );
}


class SystemCToGraph {
```

```
private:
Vector< DFGPtr > dfg_vector; // dfgs representing the methods
int num_dfgs;

Vector< DFNPtr > assigns;
int num_ass;

Vector < DFGPtr > color_graph;
int num_cgraph;

Vector< String > assigns_labels;


DFG *dfg; // pointer to dfg representing method in scope

HashTable< TableItem > *var_table;
// variable table to method in scope

HashTable< MethodItem > *method_table;
// method table for class (or global) is scope

HashTable< MethodItem > global_method_table;
// methods out of any class scope

HashTable< ClassItem > class_table; // classes table

HashTable< InstanceItem > instance_table; // instances

InstanceItem *pinst; // pointer to instance calling a method

ClassItem *pclass; // pointer to class in scope

MethodItem *pmet; // pointer to method in scope



DFN *lastNode; // lastNode created
DFN *call; // pointer to call node
DFN *openOp; // pointer to operation
DFN *opFunc;

int rotulo; // label for the DFNs... to distinguish them
int num_params;
```

```
// number of parameters int the method declaration or call

char *sid; // last identifier read
char *mid; // last method identifier read
char *cid; // last class identifier read
char *iid; // last instance identifier read
char *strassign;
int type_sp; // last variable/method type read

int line_count;

protected:

inline void resetAll() {
line_count = 0;

num_dfgs = 0;

num_cgraph = 0;

num_params = 0;
rotulo = 0;
type_sp = 0;
num_ass = 0;

dfg = NULL;
pmet = NULL;
pclass = NULL;
pinst = NULL;
lastNode = NULL;
openOp = NULL;
opFunc = NULL;
call = NULL;
sid = NULL;
iid = NULL;
cid = NULL;
mid = NULL;
strassign = NULL;
var_table = NULL;


method_table = &global_method_table;
}
```

```
inline int ehVariavel(DFN & nodo) {
for (int i = 0; i < num_ass; i++) {
if ((&nodo) == assigns[ i ]) {
return i;
}
}
return -1;
}

void createVarGraph();

public:

SystemCToGraph() {
resetAll();
}

~SystemCToGraph();

void typeSpecifier01(int newtype);
void identifier02(char *s);
void function03();
void beginFunction04();
void constant05(int c);
void return06();
void endFunction07();
void operation08(int op_type);
void argument09();
void identifierExpr10();
void varDeclaration11();
void assign12();
void assignEnd13();
void unary14(int u_type);
void functionCall15();
void classBegin16();
void constructorBegin17();
void constructorEnd18();
void classEnd19();
void classIdentifier20(char *s);
void classInstantiation21();
```

```
void instDotFunc22();
void instCallFunc23();
void functionParameter24();
void logical25(int log_type);
void methodIdentifier26();
void functionCallEnd27();



inline Vector< DFGPtr > & getDFGs() { return ( dfg_vector ); }

inline int getNumDFGs()  { return num_dfgs; }

inline Vector< DFGPtr > & getCGraphs() {
return ( color_graph ); }

inline int getNumCGraphs() { return num_cgraph; }




};


} // end namespace

#endif
```

Código do arquivo SystemCToGraph.cpp.

```
#include "SystemCToGraph.h"

namespace scgraph {

SystemCToGraph::~SystemCToGraph() {

if (sid != NULL)
delete [ ] sid;

if (iid != NULL)
```

```
delete [ ] iid;

if (mid != NULL)
delete [ ] mid;

if (cid != NULL)
delete [ ] cid;


//deletar color_graph
for (int i = 0; i < num_cgraph; i++) {
delete (color_graph[i]);
}



}


void SystemCToGraph::typeSpecifier01(int newtype) {
std::printf("### 01 ###\n");
type_sp = newtype;
}

void SystemCToGraph::identifier02(char *s) {
std::printf("### 02 ###\n");
if (sid != NULL)
delete [ ] sid;

if( s == NULL )
s = "";

int strLength = std::strlen( s );
    sid = new char[ strLength + 1 ];
    std::strcpy( sid, s );

}

void SystemCToGraph::function03() {
std::printf("### 03 ###\n");

var_table = new HashTable< TableItem >;
```

```
String msid(sid);

// new method
pmet = new MethodItem(msid);
pmet->type = type_sp;



if( num_dfgs == dfg_vector.Length( ) ) {
         dfg_vector.Double( );
for ( int i = dfg_vector.Length() / 2;
i < dfg_vector.Length(); i++ )
dfg_vector[ i ] = NULL;
}

// new dfg
     dfg_vector[ num_dfgs ] = new DFG();
     dfg = dfg_vector[ num_dfgs ];
     num_dfgs++;

dfg->setLabel(msid);

// source and sink will be created at the end of method scope
// with DFG::NOPChecker() function

pmet->grafo = dfg;

num_params = 0;

}

void SystemCToGraph::beginFunction04() {
std::printf("### 04 ###\n");
pmet->num_params = num_params;
num_params = 0;
}

void SystemCToGraph::constant05(int c) {
std::printf("### 05 ###\n");

std::stringstream sstr;
sstr << "constant" << c;
```

```
String tid( ( sstr.str() ).c_str() );

TableItem ti(tid);

  if (!var_table->IsFound(ti)) {

std::stringstream ss;
ss << rotulo++;

String sst((ss.str()).c_str());

DFN & d = dfg->addNode(sst);
d.setType(CONS_NODE);
d.setCT(CT_NORMAL);
d.setValue(c);

ti.node = &d;
lastNode = &d;

ti.defined = -2;
ti.last_used = -2;

var_table->Insert(ti);

// edge from source to constant will be insert by DFG::NOPChecker()

if (openOp != NULL) {
dfg->addEdge(d, (*openOp), OP_COST, 1);
lastNode = openOp;

openOp = NULL;
}

} else {

TableItem & tti = var_table->Find(ti);

lastNode = tti.node;


if (openOp != NULL) {
dfg->addEdge(*(tti.node), (*openOp), OP_COST, 1);
lastNode = openOp;
```

```
openOp = NULL;
}

}

}

void SystemCToGraph::return06() {
std::printf("### 06 ###\n");

std::stringstream ss ;
ss << rotulo++;

String sst((ss.str()).c_str());

DFN & d = dfg->addNode(sst);
d.setType(OUT_NODE);
d.setCT(CT_NORMAL);

     DFE & e = dfg->addEdge(*lastNode, d, RET_COST, 1);

int v = ehVariavel(*lastNode);

if (v >= 0)
e.setLabel(assigns_labels[ v ]);

}

void SystemCToGraph::endFunction07() {
std::printf("### 07 ###\n");

dfg->NOPChecker();

DFN & un = dfg->getUpNOP();
pmet->source = & dfg->getUpNOP();
DFN & bn = dfg->getBottomNOP();

bn.setType(SINK_NODE);
un.setType(SOURCE_NODE);
un.setValue(pmet->num_params);

method_table->Insert(*pmet);
```

```
createVarGraph();

pmet = NULL;

delete var_table;

var_table = NULL;

dfg = NULL;

line_count = 0;
}

void SystemCToGraph::operation08(int op_type) {
std::printf("### 08 ###\n");

std::stringstream ss ;
ss << rotulo++;

String sst((ss.str()).c_str());

DFN & d = dfg->addNode(sst);
d.setType(op_type);
d.setCT(CT_NORMAL);

openOp = &d;

DFE & e = dfg->addEdge((*lastNode), d, OP_COST, 1);

int v = ehVariavel(*lastNode);

if (v >= 0)
e.setLabel(assigns_labels[ v ]);
}

void SystemCToGraph::argument09() {
std::printf("### 09 ###\n");



TableItem ti(sid);
```

```
ti.type = type_sp;
ti.value = 0;

std::stringstream ss ;
ss << rotulo++;

String sst((ss.str()).c_str());

DFN & d = dfg->addNode(sst);
d.setType(IN_NODE);
d.setCT(CT_NORMAL);
d.setValue(num_params);

ti.node = &d;

ti.defined = -1;
ti.last_used = -1;

var_table->Insert(ti);

String s1(sid);

if( num_ass == assigns.Length( ) ) {
         assigns_labels.Double( );
}

assigns_labels[ num_ass ] = s1;

if( num_ass == assigns.Length( ) ) {
         assigns.Double( );
}

assigns[ num_ass++ ] = &d;

num_params++;
// edge from source to argument will be insert by DFG::NOPChecker()
}

void SystemCToGraph::identifierExpr10() {
std::printf("### 10 ###\n");

TableItem ti(sid);
ti.value = 0;
```

```
TableItem & tti = var_table->Find(ti);

lastNode = tti.node;


tti.last_used = line_count;

if (openOp != NULL) {

DFN & d = dfg->getNode((tti.node)->getName());

DFE & e =
dfg->addEdge(d.getName(), openOp->getName(), OP_COST, 1);
lastNode = openOp;

openOp = NULL;

int v = ehVariavel(*(tti.node));

if (v >= 0)
e.setLabel(assigns_labels[ v ]);
}
}

void SystemCToGraph::varDeclaration11() {
std::printf("### 11 ###\n");

TableItem ti(sid);
ti.type = type_sp;
ti.value = 0;
ti.defined = line_count;
ti.last_used = -1;

var_table->Insert(ti);
}

void SystemCToGraph::assign12() {
std::printf("### 12 ###\n");

String s1(sid);

if( num_ass == assigns.Length( ) ) {
```

```
         assigns_labels.Double( );
}

assigns_labels[ num_ass ] = s1;

int strLength = std::strlen( sid );
     strassign = new char[ strLength + 1 ];
        std::strcpy( strassign, sid );


}

void SystemCToGraph::assignEnd13() {
std::printf("### 13 ###\n");

TableItem ti(strassign);
ti.value = 0;

TableItem & tti = var_table->Find(ti);

tti.node = lastNode;

if( num_ass == assigns.Length( ) ) {
         assigns.Double( );
for ( int i = assigns.Length() / 2; i < assigns.Length(); i++ )
assigns[ i ] = NULL;
}

assigns[ num_ass++ ] = lastNode;

delete [ ] strassign;
strassign = NULL;

line_count++;
}

void SystemCToGraph::unary14(int u_type) {
std::printf("### 14 ###\n");
std::stringstream ss ;
ss << rotulo++;

String sst((ss.str()).c_str());
```

```
DFN & d = dfg->addNode(sst);
d.setType(u_type);
d.setCT(CT_NORMAL);

openOp = &d;
}

void SystemCToGraph::functionCall15() {
std::printf("### 15 ###\n");

num_params = 0;

MethodItem mi(sid);

MethodItem & mmi = method_table->Find(mi);

std::stringstream ss ;
ss << rotulo++;

String sst((ss.str()).c_str());

DFN & d = dfg->addNode(sst);
d.setType(CALL_NODE);
d.setCT(CT_CALL, (mmi.grafo) );


call = &d;

if (openOp != NULL) {
DFE &e = dfg->addEdge(d, (*openOp), OP_COST, 1);
lastNode = openOp;

opFunc = openOp;
openOp = NULL;

int v = ehVariavel(d);

if (v >= 0)
e.setLabel(assigns_labels[ v ]);
}

}
```

```
void SystemCToGraph::classBegin16() {
std::printf("### 16 ###\n");

pclass = new ClassItem(sid);

method_table = &(pclass->method_table);
}

void SystemCToGraph::constructorBegin17() {
std::printf("### 17 ###\n");
}

void SystemCToGraph::constructorEnd18() {
std::printf("### 18 ###\n");
}

void SystemCToGraph::classEnd19() {
std::printf("### 19 ###\n");

class_table.Insert(*pclass);



pclass = NULL;

method_table = &global_method_table;
}

void SystemCToGraph::classIdentifier20(char *s) {
std::printf("### 20 ###\n");
if (cid != NULL)
delete [ ] cid;

if( s == NULL )
s = "";

int strLength = std::strlen( s );
    cid = new char[ strLength + 1 ];
    std::strcpy( cid, s );

}

void SystemCToGraph::classInstantiation21() {
```

```
std::printf("### 21 ###\n");

ClassItem ci(cid);

ClassItem & cci = class_table.Find(ci);

InstanceItem ii(sid);

ii.class_table = &cci;

instance_table.Insert(ii);
}

void SystemCToGraph::instDotFunc22() {
std::printf("### 22 ###\n");
if (iid != NULL)
delete [ ] iid;

if( sid == NULL )
sid = "";

int strLength = std::strlen( sid );
    iid = new char[ strLength + 1 ];
    std::strcpy( iid, sid );

    num_params = 0;

}

void SystemCToGraph::instCallFunc23() {
std::printf("### 23 ###\n");

InstanceItem ii(iid);
InstanceItem & i = instance_table.Find(ii);

MethodItem mi(mid);

MethodItem & mmi = (i.class_table)->method_table.Find(mi);

std::stringstream ss ;
ss << rotulo++;

String sst((ss.str()).c_str());
```

```
DFN & d = dfg->addNode(sst);
d.setType(CALL_NODE);
d.setCT(CT_CALL, (mmi.grafo) );

call = &d;

if (openOp != NULL) {
DFE & e = dfg->addEdge(d, (*openOp), OP_COST, 1);
lastNode = openOp;

opFunc = openOp;
openOp = NULL;

int v = ehVariavel(d);

if (v >= 0)
e.setLabel(assigns_labels[ v ]);
}

}

void SystemCToGraph::functionParameter24() {
std::printf("### 24 ###\n");

num_params++;

DFE & e = dfg->addEdge(*(lastNode), *call, PAR_COST, 1);

int v = ehVariavel(*lastNode);

if (v >= 0)
e.setLabel(assigns_labels[ v ]);
}

void SystemCToGraph::logical25(int log_type) {
std::printf("### 25 ###\n");

std::stringstream ss ;
ss << rotulo++;

String sst((ss.str()).c_str());
```

```
DFN & d = dfg->addNode(sst);
d.setType(log_type);
openOp = &d;

DFE & e = dfg->addEdge((*lastNode), d, LOG_COST, 1);

int v = ehVariavel(*lastNode);

if (v >= 0)
e.setLabel(assigns_labels[ v ]);

}

void SystemCToGraph::methodIdentifier26() {
std::printf("### 26 ###\n");

if (mid != NULL)
delete [ ] mid;

if( sid == NULL )
sid = "";

int strLength = std::strlen( sid );
    mid = new char[ strLength + 1 ];
    std::strcpy( mid, sid );

}

void SystemCToGraph::functionCallEnd27() {
std::printf("### 27 ###\n");

call->setValue(num_params);

lastNode = call;

if (opFunc != NULL) {
lastNode = opFunc;

opFunc = NULL;
}


call = NULL;
```

```
}

void SystemCToGraph::createVarGraph() {


if (var_table == NULL)
return;

TableItem *ti;
TableItem *tj;

#ifdef OPTIMIZATIONS
graphopt::Otimizador opt;
opt.allOpt(dfg);
#endif

DFG *vargraph = new DFG();

for (int i = 0; i < var_table->getArraySize(); i++) {

ti = & var_table->getPosition(i);

if (var_table->WasFound()) {
if (ti->defined >= 0) {
vargraph->addNode(ti->name);

for (int j = 0; j < i; j++) {
tj = & var_table->getPosition(j);

if (var_table->WasFound()) {
if (tj->defined >= 0) {

if (ti->defined < tj->last_used) {
vargraph->addEdge(ti->name, tj->name, 0);

}
}
}
}
}
}
}
```

```
if( num_cgraph == color_graph.Length( ) ) {
          color_graph.Double( );
}

color_graph[ num_cgraph++ ] = vargraph;

}



} // end namespace
```

Código do arquivo defines.h.

```
#ifndef __defines_h__
#define __defines_h__

/* #define OPTIMIZATIONS -> faz otimizações nos grafos */

#define OPTIMIZATIONS 1

/* não alterar nada abaixo desta linha */



#define RET_COST 1
#define OP_COST 1
#define PAR_COST 1
#define LOG_COST 1

#define CT_NORMAL 1
#define CT_CALL 2

// defines for DFN::type
#define OP_NODE 20
#define CONS_NODE 21
#define SOURCE_NODE 22
#define SINK_NODE 23
#define CALL_NODE 24
#define IN_NODE 25
```

```
#define OUT_NODE 26
#define MAIN_IN_NODE 27
#define MAIN_OUT_NODE 28
#define LOGIC_NODE 29

#define TYPE_INT 51
#define TYPE_BOOL 52
#define TYPE_INST 53

#ifndef NULL
#define NULL 0
#endif


#ifndef MINUS_OP
#define MINUS_OP 2
#endif

#ifndef STAR_OP
#define STAR_OP 0
#endif

#ifndef PLUS_OP
#define PLUS_OP 1
#endif

#ifndef MOD_OP
#define MOD_OP 4
#endif

#ifndef DIV_OP
#define DIV_OP 3
#endif

#ifndef AND_OP
#define AND_OP 5
#endif

#ifndef OR_OP
#define OR_OP 6
#endif

#ifndef NOT_OP
```

```
#define NOT_OP 7
#endif

#ifndef NOTEQUAL_OP
#define NOTEQUAL_OP 8
#endif

#ifndef EQUAL_OP
#define EQUAL_OP 9
#endif

#ifndef LESSTHAN_OP
#define LESSTHAN_OP 10
#endif

#ifndef GREATERTHAN_OP
#define GREATERTHAN_OP 11
#endif

#ifndef LESSTHANOREQUALTO_OP
#define LESSTHANOREQUALTO_OP 12
#endif

#ifndef GREATERTHANOREQUALTO_OP
#define GREATERTHANOREQUALTO_OP 13
#endif

#define _ADD 1
#define _ADDI 2

#define RSAV 1
#define RTEMP 2
#define RARG 3

#define SBASE 16
#define TBASE 8
#define ABASE 4
#define VBASE 2

#endif
```

E.4 Gerador de código

Código do arquivo Codegen.h.

```
#ifndef __CodeGen_h__
#define __CodeGen_h__

#include "dfg.h"
#include "dfn.h"
#include "graph.h"
#include "list.h"
#include "Vector.h"
#include "Hash.h"

#include "defines.h"

#include <string>
#include <sstream>


namespace codegen {


struct VarRegItem {

String name;
int reg;
int offset;
int used;
int ehVariavel;

VarRegItem() { used = 0; ehVariavel = 0; offset = -1; reg = -1; }
VarRegItem( const String & Nm ) : name( Nm ) {
used = 0; ehVariavel = 0; offset = -1; reg = -1; }

int operator==( const VarRegItem & Rhs ) const
        { return name == Rhs.name; }
     int operator!=( const VarRegItem & Rhs ) const
        { return name != Rhs.name; }
};
```

```
static unsigned int HashStr( const String & Key, int TableSize )
{
    unsigned int HashVal = 0;
    int i = 0;

    while( ( i < Key.length() ) && (Key[ i ] != '\0' ) )
        HashVal = ( HashVal << 5 ) ^ HashVal ^ Key[ i++ ];

    return ( HashVal % TableSize );
}

static unsigned int Hash( const VarRegItem & Key, int TableSize ) {
return HashStr(Key.name, TableSize );
}



class CodeGen {

private:

Vector< DFNPtr > nodes_visited;
int counter;

Vector< DFGPtr > color_graph;
int num_cgraph;

Vector< int > max_color;

Vector< int > deslocamento;
int func_index;

Vector< bool > sx;
Vector< bool > tx;

std::stringstream code;


std::stringstream data;


HashTable< VarRegItem > *var_table;
```

```
int args_used;
int rets_used;
int sav_used;
int temp_used;

int prox_pos;

int aargs;
int ttemp;
int ssaved;
int rrets;

int lock;
int lock2;


void postVisit(DFG & grafo, DFN & nodo);


void generateCode(DFG & grafo, int index);

void saveRegistersOnStack(int saved);

void restoreRegistersFromStack(int saved);

void restoreAfterJal(int tempor, int args);

void saveBeforeJal(int tempor, int args);

int colorateGraph(DFG & grafo);

void callCode(DFG & grafo, int index, DFN & nodo);

void PlusSubCode(DFG & grafo, int index, DFN & nodo, int operation);

void inCode(DFG & grafo, int index, DFN & nodo);

void sourceCode(DFG & grafo, int index, DFN & nodo);

void consCode(DFG & grafo, int index, DFN & nodo);

void sinkCode(DFG & grafo, int index, DFN & nodo);
```

```
void outCode(DFG & grafo, int index, DFN & nodo);

void StarDivCode(DFG & grafo, int index, DFN & nodo, int operation);

int getUnusedSx();

int getUnusedTx();

void freeRegister(int reg);

void tryToFreeRegister(VarRegItem & vri, DFG & grafo);

public:

CodeGen() {
counter = 0;

prox_pos = 0;
args_used = 0;
rets_used = 0;
sav_used = 0;
temp_used = 0;

lock = -1;
lock2 = -1;

aargs = 0;
ttemp = 0;
ssaved = 0;
rrets = 0;

num_cgraph = 0;

var_table = NULL;


for (int i = 0; i < sx.Length(); i++) {
sx[i] = false;
tx[i] = false;
}

}
~CodeGen();
```

```
void generateCodeDFG(DFG & grafo, int index);

void generateCodeDFGs(Vector< DFGPtr > & vetor, int size);

void setVariablesGraph(Vector< DFGPtr > & vetor, int size);

void write(const char* filename);



};


} // end namespace


#endif
```

Código do arquivo Codegen.cpp.

```
#include "CodeGen.h"

namespace codegen {



CodeGen::~CodeGen() { }

void CodeGen::generateCodeDFG(DFG & grafo, int index) {

counter = 0;

DFN & source = grafo.getUpNOP();

postVisit(grafo, source);

generateCode(grafo, index);


}
```

```
void CodeGen::postVisit(DFG & grafo, DFN & nodo) {

if (nodo.getKey() == 1)
return;


DFEPtr current;
DFNPtr SuccessorNode;

List< DFEPtr > *out = nodo.getOutgoingEdges();



for ( List<DFEPtr>::iterator itr = out->begin();
itr != out->end(); ++itr ) {
current = *itr;

SuccessorNode = & grafo.getNode( current->getDestiny() );

postVisit(grafo, *SuccessorNode);
}

if( counter == nodes_visited.Length( ) ) {
nodes_visited.Double( );
}

nodo.setKey(1);
nodes_visited[ counter ] = & nodo;

counter++;

}

void CodeGen::generateCodeDFGs(Vector< DFGPtr > & vetor,
int size) {
counter = 0;

for (int i = 0; i < size; i++) {
generateCodeDFG(*(vetor[ i ]), i);
}

}
```

```
void CodeGen::generateCode(DFG & grafo, int index) {


for (int i = (counter - 1); i >= 0 ; i--) {

switch ( (nodes_visited[ i ])->getType() ) {
case SOURCE_NODE: // source
std::printf("### Source ###\n");
sourceCode(grafo, index, *(nodes_visited[i]));
break;
case SINK_NODE: // sink
std::printf("### Sink ###\n");
sinkCode(grafo, index, *(nodes_visited[i]));
break;
case CALL_NODE:
std::printf("### Call ###\n");
callCode(grafo, index, *(nodes_visited[i]));
break;
case CONS_NODE:
std::printf("### Constant ###\n");
consCode(grafo, index, *(nodes_visited[i]));
break;
case IN_NODE:
std::printf("### Argument ###\n");
inCode(grafo, index, *(nodes_visited[i]));
break;
case OUT_NODE:
std::printf("### Return ###\n");
outCode(grafo, index, *(nodes_visited[i]));
break;
case STAR_OP:
std::printf("### Multiply ###\n");
StarDivCode(grafo, index, *(nodes_visited[i]), STAR_OP);
break;
case PLUS_OP:
std::printf("### Add ###\n");
PlusSubCode(grafo, index, *(nodes_visited[i]), PLUS_OP);
break;
case DIV_OP:
std::printf("### Divide ###\n");
```

```
StarDivCode(grafo, index, *(nodes_visited[i]), DIV_OP);
break;
case MOD_OP:
std::printf("### Module ###\n");
StarDivCode(grafo, index, *(nodes_visited[i]), MOD_OP);
break;
case MINUS_OP:
std::printf("### Subtract ###\n");
PlusSubCode(grafo, index, *(nodes_visited[i]), MINUS_OP);
break;
case LESSTHANOREQUALTO_OP:
std::printf("### Less or Equal ###\n");
StarDivCode(grafo, index, *(nodes_visited[i]),
LESSTHANOREQUALTO_OP);
 break;
case GREATERTHANOREQUALTO_OP:
std::printf("### Greater or Equal ###\n");
StarDivCode(grafo, index, *(nodes_visited[i]),
GREATERTHANOREQUALTO_OP);
break;
case GREATERTHAN_OP:
std::printf("### Greater ###\n");
StarDivCode(grafo, index, *(nodes_visited[i]), GREATERTHAN_OP);
break;
case LESSTHAN_OP:
std::printf("### Less ###\n");
StarDivCode(grafo, index, *(nodes_visited[i]), LESSTHAN_OP);
break;
case EQUAL_OP:
std::printf("### Equal ###\n");
StarDivCode(grafo, index, *(nodes_visited[i]), EQUAL_OP);
break;
case NOTEQUAL_OP:
std::printf("### Not Equal ###\n");
StarDivCode(grafo, index, *(nodes_visited[i]), NOTEQUAL_OP);
break;
case NOT_OP:
std::printf("### Not ###\n");
StarDivCode(grafo, index, *(nodes_visited[i]), NOT_OP);
break;
case AND_OP:
std::printf("### And ###\n");
PlusSubCode(grafo, index, *(nodes_visited[i]), AND_OP);
```

```
break;
case OR_OP:
std::printf("### Or ###\n");
PlusSubCode(grafo, index, *(nodes_visited[i]), OR_OP);
break;
default:
break;
}// end switch

} //end for

}


void CodeGen::saveRegistersOnStack(int saved) {




int spoffset = 4;

code << "\tsw \t$fp, -" << spoffset << "($sp)\n";
spoffset += 4;
code << "\tsw \t$31, -" << spoffset << "($sp)\n";
spoffset += 4;
for (int i = 0; i < saved; i++) {
code << "\tsw \t$" << (SBASE + i) << ", -" << spoffset << "($sp)\n";
spoffset += 4;
}

code << "\tmove \t$fp, $sp\n";

// Make room for -(arguments + returns + saved) words on stack
code << "\taddu \t$sp, $sp, -" <<
( (saved + 2) * 4 ) << "\n";


}

void CodeGen::restoreRegistersFromStack(int saved) {


// Make room for (arguments + returns + saved) words on stack
```

```
code << "\taddu \t$sp, $sp, " <<
( -deslocamento[func_index] ) << "\n";

int spoffset = 4;

code << "\tlw \t$fp, -" << spoffset << "($sp)\n";
spoffset += 4;
code << "\tlw \t$31, -" << spoffset << "($sp)\n";
spoffset += 4;
for (int i = 0; i < saved; i++) {
code << "\tlw \t$" << (SBASE + i) << ", -" <<
spoffset << "($sp)\n";
spoffset += 4;
}


}

void CodeGen::saveBeforeJal(int tempor, int args) {
if (args > 4)
args = 4;
if (tempor > 7)
tempor = 7;

// Make room for -(arguments + returns + saved) words on stack
if ((tempor + args) > 0)
code << "\taddu \t$sp, $sp, -" <<
( (tempor + args) * 4 ) << "\n";

int spoffset = 0;
for (int i = 0; (i < tempor); i++) {
code << "\tsw \t$" << (TBASE + i) <<
", " << spoffset << "($sp)\n";
spoffset += 4;
}

for (int i = 0; (i < args); i++) {
code << "\tsw \t$" << (ABASE + i) <<
", " << spoffset << "($sp)\n";
spoffset += 4;
}
}
```

```
void CodeGen::restoreAfterJal(int tempor, int args) {
if (args > 4)
args = 4;
if (tempor > 7)
tempor = 7;

int spoffset = 0;

for (int i = 0; i < tempor; i++) {
code << "\tlw \t$" << (TBASE + i) <<
", " << spoffset << "($sp)\n";
spoffset += 4;
}

for (int i = 0; i < args; i++) {
code << "\tlw \t$" << (ABASE + i) <<
", " << spoffset << "($sp)\n";
spoffset += 4;
}


if ( (tempor + args) > 0)
code << "\taddu \t$sp, $sp, " << ( (tempor + args) * 4 ) << "\n";
}

void CodeGen::setVariablesGraph(Vector< DFGPtr > & vetor, int size) {

num_cgraph = size;

for (int i = 0; i < size; i++) {
color_graph[i] = vetor[i];
max_color[i] = colorateGraph(*(color_graph[i]));

}
}

int CodeGen::getUnusedSx() {
for (int i = 0; i < 7; i++) {

if (! sx[i]) {
sx[i] = true;
return SBASE + i;
}
```

```
}
return -1;
}

int CodeGen::getUnusedTx() {
for (int i = 0; i < 9; i++) {

if (! tx[i]) {
if (i == 7) { tx[i] = true; return 24; }
else if (i == 8) { tx[i] = true; return 25; }
else { tx[i] = true; return TBASE + i; }

}
}

VarRegItem *vri;
int reg = -1;

for (int i = prox_pos; i < var_table->getArraySize(); i++) {
vri = & var_table->getPosition(i);

if (var_table->WasFound()) {
if ( ( (vri->reg > 7) && (vri->reg < 16) ) || (vri->reg == 24)
|| (vri->reg == 25) ) { // temporario
if ( (vri->reg == lock) || (vri->reg == lock2) )
continue;

reg = vri->reg;
if (vri->offset > -1) {
code << "\taddu \t$sp, $sp, -4\n";
code << "\tsw \t$" << reg << ", 0($sp)\n";
deslocamento[func_index] += (-4);

vri->offset = deslocamento[func_index];
}
vri->reg = -1;
break;
}

}

}
```

```
prox_pos = ++prox_pos % var_table->getArraySize();

return reg;
}


void CodeGen::freeRegister(int reg) {

if (reg >= TBASE && reg < 16) { // tx 0-7
tx[reg - TBASE] = false;
} else if (reg >= SBASE && reg < 24) {
sx[reg - SBASE] = false;
} else if (reg == 24) {
tx[7] = false;
} else if (reg == 25) {
tx[8] = false;
}

}

void CodeGen::tryToFreeRegister(VarRegItem & vri, DFG & grafo) {



vri.used = vri.used + 1;

if (vri.ehVariavel)
return;

DFNPtr pnode = & grafo.getNode(vri.name);

int num = pnode->numberOfSuccessors();

if (num == vri.used) {
freeRegister(vri.reg);
}
}

void CodeGen::write(const char* filename) {
std::ofstream out;

out.open(filename);
```

```
out << ".text\n";
out << ".globl main\n";
// .align ??
out << code.str();
out << ".data\n";
out << data.str();

out.close();
}

int CodeGen::colorateGraph(DFG & grafo) {

DFNPtr current;
DFEPtr currentEdge;
DFNPtr otherNode;
List< DFEPtr > *inout;
int c = -1;
int maxc = -1;

  Vector< DFNPtr > & nodes = grafo.getNodes();

for ( int i = 0; i < nodes.Length(); i++) {
if ( nodes[ i ] == NULL )
    continue;
         current = nodes[ i ];

c = 0;

inout = current->getOutgoingEdges();

for ( List<DFEPtr>::iterator itr = inout->begin();
itr != inout->end(); ++itr ) {
currentEdge = *itr;

otherNode = & grafo.getNode( currentEdge->getDestiny() );

if (otherNode->getValue() == c)
c++;
}

inout = current->getIngoingEdges();

for ( List<DFEPtr>::iterator itr = inout->begin();
```

```
itr != inout->end(); ++itr ) {
currentEdge = *itr;

otherNode = & grafo.getNode( currentEdge->getDestiny() );

if (otherNode->getValue() == c)
c++;
}

current->setValue(c);

if (c > maxc)
maxc = c;

}


return (maxc + 1);

}

void CodeGen::callCode(DFG & grafo, int index, DFN & nodo) {


DFEPtr current;
DFNPtr pnode, cnode;
DFGPtr dgraph;
List< DFEPtr >*edge_list;

dgraph  = (DFGPtr) nodo.getCalled();

saveBeforeJal(temp_used, nodo.getValue());

args_used = 0;

edge_list = nodo.getIngoingEdges();

// se tiver mais que 4 parametros, eh necessario salvar na pilha

int parameters = nodo.getValue();

if (parameters > 4) {
code << "\taddu \t$sp, $sp, -" << ( (parameters - 4) * 4 ) << "\n";
```

```
}

// for each predecessor of "node", store parameter on $aX
for ( List<DFEPtr>::iterator itr = edge_list->begin();
itr != edge_list->end(); ++itr ) {
current = *itr;
pnode = & grafo.getNode( current->getSource() );


//se a aresta tiver label é uma variável, procurar na tabela
if (current->getLabel() != "") {

if (pnode->getType() != IN_NODE)
cnode = & ( color_graph[ index ] )->getNode( current->getLabel() );


VarRegItem vri(current->getLabel());
VarRegItem & vvri = var_table->Find(vri);
if (var_table->WasFound()) {
if (args_used < 4) {
if (vvri.reg != -1) {
code << "\tmove \t$" << (ABASE + args_used++) <<
", $" << vvri.reg << "\n";
} else {
int tr = getUnusedTx();
code << "\tlw \t$" << tr << ", " << vvri.offset << "($fp)\n";
code << "\tmove \t$" << (ABASE + args_used++) << ", $" <<
tr << "\n";
freeRegister(tr);
}
tryToFreeRegister(vvri, grafo);
}
else {
if (vvri.reg != -1) {
code << "\tsw \t$" << (vvri.reg) << ", " <<
((args_used++ - 4) * 4) << "($sp)\n";
tryToFreeRegister(vvri, grafo);
} else {
int tr = getUnusedTx();
code << "\tlw \t$" << tr << ", " << vvri.offset <<
"($fp)\n";
code << "\tsw \t$" << (vvri.reg) << ", " <<
((args_used++ - 4) * 4) << "($sp)\n";
```

```
freeRegister(tr);
}
}
continue;
} else { // variavel nao encontrada na tabela
if (pnode->getType() == CONS_NODE) {
if (cnode->getValue() > 6) {
int ttx = getUnusedTx();
vri.reg = ttx;
} else {
vri.reg = SBASE + cnode->getValue();
vri.ehVariavel = 1;
}
code << "\tlw \t$" << (vri.reg) << ", " <<
( current->getLabel() ).c_str() << "\n";
if (args_used < 4) {
code << "\tmove \t$" << (ABASE + args_used++) <<
", $" << vri.reg << "\n";
} else {
code << "\tsw \t$" << (vri.reg) << ", " <<
((args_used++ - 4) * 4) << "($sp)\n";
}
tryToFreeRegister(vri, grafo);
}

var_table->Insert(vri);
}
} else if ( pnode->getType() == CONS_NODE) { // constante como parametro
if (args_used < 4) {
if (pnode->getValue() == 0)
code << "\tmove \t$" << (ABASE + args_used++) << ", $0\n";
else
code << "\tli \t$" << (ABASE + args_used++) <<
", " << pnode->getValue() << "\n";
} else {
if (pnode->getValue() == 0) {
code << "\tsw \t$0, " << ((args_used++ - 4) * 4) << "($sp)\n";
} else {
int utx = getUnusedTx();
code << "\tli \t$" << (utx) << ", " << pnode->getValue() << "\n";
code << "\tsw \t$" << (utx) << ", " <<
((args_used++ - 4) * 4) << "($sp)\n";
freeRegister(utx);
```

```
}
}
} else { // expressao como parametro: nao aceita!!

}

} //fim for



code << "\tjal \t" << (dgraph->getLabel()).c_str() << "\n";

if (parameters > 4) {
code << "\taddu \t$sp, $sp, " << ( (parameters - 4) * 4 ) << "\n";
}

restoreAfterJal(temp_used, nodo.getValue());


// atualizar tabela de variaveis
edge_list = nodo.getOutgoingEdges();

for ( List<DFEPtr>::iterator itr = edge_list->begin();
itr != edge_list->end(); ++itr ) {
current = *itr;
if (current->getLabel() == "") {
VarRegItem uvri(nodo.getName());

if (var_table->IsFound(uvri))
continue;

int ttt = getUnusedTx();
uvri.reg = ttt;

var_table->Insert(uvri);
code << "\tmove \t$" << ttt << ", $" << VBASE << "\n";
continue;
}

VarRegItem vri(current->getLabel());
VarRegItem & vvri = var_table->Find(vri);

if (var_table->WasFound()) {
```

```
code << "\tmove \t$" << vvri.reg << ", $" << VBASE << "\n";

} else {
cnode = & ( color_graph[ index ] )->getNode( current->getLabel() );
if (cnode->getValue() > 6) {
int tx = getUnusedTx();
vri.reg = tx;
} else {
vri.reg = SBASE + cnode->getValue() ;
vri.ehVariavel = 1;
}

code << "\tmove \t$" << vri.reg << ", $" << VBASE << "\n";
var_table->Insert(vri);
}
break;
}









}

void CodeGen::PlusSubCode(DFG & grafo, int index, DFN & nodo,
int operation) {

int result_reg = 0;
DFEPtr current;
DFNPtr pnode, cnode;
String v1;
String v2;

int act_op = 0;
int act_reg1 = 0;
int act_reg2 = 0;
int act_oper_c = 0;


bool jafoi, jafoicons, cleancons;
List< DFEPtr >*edge_list;
```

```
edge_list = nodo.getIngoingEdges();

act_op = _ADD;
jafoi = false;
jafoicons = false;
cleancons = false;


for ( List<DFEPtr>::iterator itr = edge_list->begin();
itr != edge_list->end(); ++itr ) {
current = *itr;
pnode = & grafo.getNode( current->getSource() );



if (current->getLabel() != "") {

if (pnode->getType() != IN_NODE)
cnode = & ( color_graph[ index ] )->getNode( current->getLabel() );

VarRegItem vri(current->getLabel());

VarRegItem & vvri = var_table->Find(vri);

// se variavel ja esta na tabela...
if (var_table->WasFound()) {
if (!jafoi) {
if (vvri.reg != -1) {
act_reg1 = vvri.reg;
} else {
int tr = getUnusedTx();
code << "\tlw \t$" << tr << ", " << vvri.offset << "($fp)\n";
lock = tr;
act_reg1 = tr;
vvri.reg = tr;
}

v1 = vvri.name;
jafoi = true;
} else {
if (vvri.reg != -1) {
```

```
act_reg2 = vvri.reg;
} else {
int tr = getUnusedTx();
code << "\tlw \t$" << tr << ", " << vvri.offset << "($fp)\n";
lock2 = tr;
act_reg2 = tr;
vvri.reg = tr;
}
v2 = vvri.name;


}
continue;
} else {

if (pnode->getType() == CONS_NODE) { // constante

if (!jafoi) {
if (cnode->getValue() > 6) {
int tx = getUnusedTx();
vri.reg = tx;
} else {
vri.reg = SBASE + cnode->getValue();
vri.ehVariavel = 1;
}

jafoi = true;
act_reg1 = vri.reg;
v1 = vri.name;
lock = act_reg1;

} else {
if (cnode->getValue() > 6) {
int tx = getUnusedTx();
vri.reg = tx;
} else {
vri.reg = SBASE + cnode->getValue();
vri.ehVariavel = 1;
}

act_reg2 = vri.reg;
v2 = vri.name;
lock2 = act_reg2;
```

```
}


code << "\tlw \t$" << (vri.reg) << ", " <<
( current->getLabel() ).c_str() << "\n";


var_table->Insert(vri);



} else { // call ou parametro
VarRegItem & uvri = var_table->Find(pnode->getName());
if (var_table->WasFound()) {
if (!jafoi) {
act_reg1 = uvri.reg;
v1 =  uvri.name;
jafoi = true;
lock = act_reg1;
} else {
act_reg2 = uvri.reg;
lock2 = act_reg2;
v2 = uvri.name;
}
} else {
if (!jafoi) {
if (cnode->getValue() > 6) {
int tx = getUnusedTx();
vri.reg = tx;

} else {
vri.reg = SBASE + cnode->getValue();
vri.ehVariavel = 1;
}
act_reg1 = vri.reg;
jafoi = true;
lock = act_reg1;
v1 = vri.name;
} else {
if (cnode->getValue() > 6) {
int tx = getUnusedTx();
vri.reg = tx;
```

```
} else {
vri.reg = SBASE + cnode->getValue();
vri.ehVariavel = 1;
}
act_reg2 = vri.reg;
lock2 = act_reg2;
v2 = vri.name;
}

var_table->Insert(vri);

}

}


}

} else if ( pnode->getType() == CONS_NODE) { // gera addu
act_op = _ADDI;
if (!jafoicons) {
act_oper_c = pnode->getValue();
jafoicons = true;
} else { // se for 2 constantes... tem que botar uma em um registrador
if (pnode->getValue() == 0) {
act_reg1 = 0;
} else {
int dreg = getUnusedTx();
act_reg1 = dreg;
code << "\tli \t$" << dreg << ", " << pnode->getValue() << "\n";
cleancons = true;
lock = act_reg1;
}

}

} else { // call ou outra op

if (!jafoi) {
VarRegItem & uvrii = var_table->Find(pnode->getName());
if (var_table->WasFound()) {
if (uvrii.reg != -1) {
```

```
act_reg1 = uvrii.reg;
} else {
int tr = getUnusedTx();
code << "\tlw \t$" << tr << ", " << uvrii.offset << "($fp)\n";
act_reg1 = tr;
uvrii.reg = tr;
lock = act_reg1;
}

jafoi = true;
v1 = uvrii.name;

} else {
std::printf("ih fudeu\n");
}
} else {
VarRegItem & uvrii = var_table->Find(pnode->getName());
if (var_table->WasFound()) {
if (uvrii.reg != -1) {
act_reg2 = uvrii.reg;
lock2 = act_reg2;
} else {
int tr = getUnusedTx();
code << "\tlw \t$" << tr << ", " << uvrii.offset << "($fp)\n";
act_reg2 = tr;
uvrii.reg = tr;
lock2 = act_reg2;
}

v2 = uvrii.name;

} else {
std::printf("ih fudeu2\n");
}
}
}
}



// atualizar tabela de variaveis...
//se é variavel armazenar em $sx .. senao em $tx
edge_list = nodo.getOutgoingEdges();
```

```
for ( List<DFEPtr>::iterator itr = edge_list->begin();
itr != edge_list->end(); ++itr ) {
current = *itr;
if (current->getLabel() == "") {
VarRegItem vri(nodo.getName());

if (! var_table->IsFound(vri)) {
result_reg = getUnusedTx();
vri.reg = result_reg;

var_table->Insert(vri);
} else {
std::printf("doh!\n");
}
break;
}

pnode = & grafo.getNode( current->getSource() );

cnode = & ( color_graph[ index ] )->getNode( current->getLabel() );



VarRegItem vri(current->getLabel());
VarRegItem & vvri = var_table->Find(vri);

if (var_table->WasFound()) {
result_reg = vvri.reg;
} else {
if (cnode->getValue() > 6) {
int tx = getUnusedTx();
vri.reg = tx;
} else {
vri.reg = SBASE + cnode->getValue();
vri.ehVariavel = 1;
}

result_reg = vri.reg;

var_table->Insert(vri);

}
```

```
break;
}



if (act_op == _ADD) {
bool negativo = false;

if (operation == PLUS_OP)
code << "\taddu \t$" << (result_reg) << ", $" <<
(act_reg1) << ", $" << (act_reg2) << "\n";
else if (operation == MINUS_OP) {
if (act_reg2 != 0)
code << "\tsubu \t$" << (result_reg) << ", $" <<
(act_reg1) << ", $" << (act_reg2) << "\n";
else {
negativo = true;
code << "\tnegu \t$" << (result_reg) << ", $" <<
(act_reg1) << "\n";
}
}
else if (operation == AND_OP)
code << "\tand \t$" << (result_reg) << ", $" <<
(act_reg1) << ", $" << (act_reg2) << "\n";
else if (operation == OR_OP)
code << "\tor \t$" << (result_reg) << ", $" <<
(act_reg1) << ", $" << (act_reg2) << "\n";



VarRegItem & vv1 = var_table->Find(v1);
tryToFreeRegister(vv1, grafo);

if (!negativo) {
VarRegItem & vv2 = var_table->Find(v2);
tryToFreeRegister(vv2, grafo);
}
} else {
if (operation == PLUS_OP)
code << "\taddu \t$" << (result_reg) << ", $" <<
(act_reg1) << ", " << act_oper_c << "\n";
else if (operation == MINUS_OP)
code << "\tsubu \t$" << (result_reg) << ", $" <<
```

```
(act_reg1) << ", " << act_oper_c << "\n";
else if (operation == AND_OP)
code << "\tand \t$" << (result_reg) << ", $" <<
(act_reg1) << ", " << act_oper_c << "\n";
else if (operation == OR_OP)
code << "\tor \t$" << (result_reg) << ", $" <<
(act_reg1) << ", " << act_oper_c << "\n";


if (cleancons) {
freeRegister(act_reg1);
} else {
VarRegItem & vv1 = var_table->Find(v1);
tryToFreeRegister(vv1, grafo);
}
}

lock = -1;
lock2 = -1;

}

void CodeGen::inCode(DFG & grafo, int index, DFN & nodo) {



DFEPtr current;

List< DFEPtr >*edge_list;

// atualizar tabela de variaveis
edge_list = nodo.getOutgoingEdges();

for ( List<DFEPtr>::iterator itr = edge_list->begin();
itr != edge_list->end(); ++itr ) {
current = *itr;
if (current->getLabel() == "") {
VarRegItem vri (nodo.getName());
if (nodo.getValue() < 4) {
vri.reg = ABASE + nodo.getValue();
} else {
vri.reg = -1;
vri.offset = (nodo.getValue() - 4) * 4;
```

```
}
var_table->Insert(vri);
break;

}

VarRegItem vri(current->getLabel());


if (! var_table->IsFound(vri)) {

if (nodo.getValue() < 4) {
vri.reg = ABASE + nodo.getValue();
} else {
vri.reg = -1;
vri.offset = (nodo.getValue() - 4) * 4;
}

vri.ehVariavel = 1;
var_table->Insert(vri);
}
break;
}

}

void CodeGen::sourceCode(DFG & grafo, int index, DFN & nodo) {


DFGPtr dgraph;

func_index = index;


var_table = new HashTable< VarRegItem >;

dgraph = (DFGPtr) nodo.getFather();

code << (dgraph->getLabel()).c_str() << ":";

aargs = (dgraph->getUpNOP()).getValue();
ttemp = temp_used;
rrets = 0;
```

```
ssaved = max_color[index];

for (int i = 0; i < 10 && i < ssaved; i++)
sx[i] = true;

args_used = 0;
rets_used = 0;
sav_used = 0;
temp_used = 0;

saveRegistersOnStack(ssaved);

deslocamento[index] = (ssaved + 2) * (-4);

}

void CodeGen::consCode(DFG & grafo, int index, DFN & nodo) {


DFEPtr current;
List< DFEPtr >*edge_list;


edge_list = nodo.getOutgoingEdges();

for ( List<DFEPtr>::iterator itr = edge_list->begin();
itr != edge_list->end(); ++itr ) {
current = *itr;

if (current->getLabel() == "")
continue;

data << ( current->getLabel() ).c_str() << ":\t.word \t" <<
nodo.getValue() << "\n";

break;
}
}

void CodeGen::sinkCode(DFG & grafo, int index, DFN & nodo) {


restoreRegistersFromStack(ssaved);
```

```
code << "\tjr \t$31\n";

if (var_table != NULL) {
var_table->MakeEmpty();
delete var_table;
var_table = NULL;
}

}

void CodeGen::outCode(DFG & grafo, int index, DFN & nodo) {

DFEPtr current;
DFNPtr pnode;


List< DFEPtr >*edge_list;

edge_list = nodo.getIngoingEdges();


// for each predecessor of "node"
for ( List<DFEPtr>::iterator itr = edge_list->begin();
itr != edge_list->end(); ++itr ) {
current = *itr;
pnode = & grafo.getNode( current->getSource() );

if (current->getLabel() != "") {
VarRegItem vri(current->getLabel());

VarRegItem & vvri = var_table->Find(vri);

if (var_table->WasFound()) {
code << "\tmove \t$2, $" << vvri.reg << "\n";
tryToFreeRegister(vvri, grafo);

return;
} else {
if (pnode->getType() == CONS_NODE) {
code << "\tli \t$2, " << pnode->getValue() << "\n";
return;
} else {
```

```
}
}
break;
} else if (pnode->getType() == CONS_NODE) {
code << "\tli \t$2, " << pnode->getValue() << "\n";
break;
} else { // call ou expressao
VarRegItem vri(pnode->getName());

VarRegItem & uvri = var_table->Find(vri);

if (var_table->WasFound()) {
code << "\tmove \t$2, $" << uvri.reg << "\n";
tryToFreeRegister(uvri, grafo);
}

break;
}




}



}

void CodeGen::StarDivCode(DFG & grafo, int index, DFN & nodo,
int operation) {

int result_reg = 0;
DFEPtr current;
DFNPtr pnode, cnode;
String v1;
String v2;
bool cleancons1, cleancons2, jafoi;


int act_reg1 = 0;
int act_reg2 = 0;
```

```
List< DFEPtr >*edge_list;


edge_list = nodo.getIngoingEdges();


jafoi = false;
cleancons1 = false;
cleancons2 = false;



for ( List<DFEPtr>::iterator itr = edge_list->begin();
itr != edge_list->end(); ++itr ) {
current = *itr;
pnode = & grafo.getNode( current->getSource() );



if (current->getLabel() != "") {

if (pnode->getType() != IN_NODE)
cnode = & ( color_graph[ index ] )->getNode( current->getLabel() );

VarRegItem vri(current->getLabel());

VarRegItem & vvri = var_table->Find(vri);

// se variavel ja esta na tabela...
if (var_table->WasFound()) {
if (!jafoi) {
if (vvri.reg != -1) {
act_reg1 = vvri.reg;
} else {
int tr = getUnusedTx();
code << "\tlw \t$" << tr << ", " << vvri.offset << "($fp)\n";
lock = tr;
act_reg1 = tr;
vvri.reg = tr;
}

v1 = vvri.name;
```

```
jafoi = true;
} else {
if (vvri.reg != -1) {
act_reg2 = vvri.reg;
} else {
int tr = getUnusedTx();
code << "\tlw \t$" << tr << ", " << vvri.offset << "($fp)\n";
lock2 = tr;
act_reg2 = tr;
vvri.reg = tr;
}
v2 = vvri.name;


}
continue;
} else {

if (pnode->getType() == CONS_NODE) { // constante

if (!jafoi) {
if (cnode->getValue() > 6) {
int tx = getUnusedTx();
vri.reg = tx;
} else {
vri.reg = SBASE + cnode->getValue();
vri.ehVariavel = 1;
}

jafoi = true;
act_reg1 = vri.reg;
v1 = vri.name;
lock = act_reg1;

} else {
if (cnode->getValue() > 6) {
int tx = getUnusedTx();
vri.reg = tx;
} else {
vri.reg = SBASE + cnode->getValue();
vri.ehVariavel = 1;
}
```

```
act_reg2 = vri.reg;
v2 = vri.name;
lock2 = act_reg2;

}


code << "\tlw \t$" << (vri.reg) << ", " << ( current->getLabel() ).c_str() << "\n";


var_table->Insert(vri);



} else { // call ou parametro
VarRegItem & uvri = var_table->Find(pnode->getName());
if (var_table->WasFound()) {
if (!jafoi) {
act_reg1 = uvri.reg;
v1 =  uvri.name;
jafoi = true;
lock = act_reg1;
} else {
act_reg2 = uvri.reg;
lock2 = act_reg2;
v2 = uvri.name;
}
} else {
if (!jafoi) {
if (cnode->getValue() > 6) {
int tx = getUnusedTx();
vri.reg = tx;

} else {
vri.reg = SBASE + cnode->getValue();
vri.ehVariavel = 1;
}
act_reg1 = vri.reg;
jafoi = true;
lock = act_reg1;
v1 = vri.name;
} else {
if (cnode->getValue() > 6) {
```

```
int tx = getUnusedTx();
vri.reg = tx;

} else {
vri.reg = SBASE + cnode->getValue();
vri.ehVariavel = 1;
}
act_reg2 = vri.reg;
lock2 = act_reg2;
v2 = vri.name;
}

var_table->Insert(vri);

}

}


}

} else if ( pnode->getType() == CONS_NODE) { // gera addu
if (!jafoi) {
if (pnode->getValue() == 0) {
act_reg1 = 0;
} else {
int dreg = getUnusedTx();
act_reg1 = dreg;
code << "\tli \t$" << dreg << ", " << pnode->getValue() << "\n";
cleancons1 = true;
lock = act_reg1;
jafoi = true;
}
} else {
if (pnode->getValue() == 0) {
act_reg2 = 0;
} else {
int dreg = getUnusedTx();
act_reg2 = dreg;
code << "\tli \t$" << dreg << ", " << pnode->getValue() << "\n";
cleancons2 = true;
lock = act_reg2;
}
```

```
}
} else { // call ou outra op

if (!jafoi) {
VarRegItem & uvrii = var_table->Find(pnode->getName());
if (var_table->WasFound()) {
if (uvrii.reg != -1) {
act_reg1 = uvrii.reg;
} else {
int tr = getUnusedTx();
code << "\tlw \t$" << tr << ", " << uvrii.offset << "($fp)\n";
act_reg1 = tr;
uvrii.reg = tr;
lock = act_reg1;
}

jafoi = true;
v1 = uvrii.name;

} else {
std::printf("ih fudeu\n");
}
} else {
VarRegItem & uvrii = var_table->Find(pnode->getName());
if (var_table->WasFound()) {
if (uvrii.reg != -1) {
act_reg2 = uvrii.reg;
lock2 = act_reg2;
} else {
int tr = getUnusedTx();
code << "\tlw \t$" << tr << ", " << uvrii.offset << "($fp)\n";
act_reg2 = tr;
uvrii.reg = tr;
lock2 = act_reg2;
}

v2 = uvrii.name;

} else {
std::printf("ih fudeu2\n");
}
}
}
```

```
}



// atualizar tabela de variaveis... se é variavel armazenar em $sx .. senao em $tx
edge_list = nodo.getOutgoingEdges();

for ( List<DFEPtr>::iterator itr = edge_list->begin();
itr != edge_list->end(); ++itr ) {
current = *itr;
if (current->getLabel() == "") {
VarRegItem vri(nodo.getName());

if (! var_table->IsFound(vri)) {
result_reg = getUnusedTx();
vri.reg = result_reg;

var_table->Insert(vri);
} else {
std::printf("doh!\n");
}
break;
}

pnode = & grafo.getNode( current->getSource() );

cnode = & ( color_graph[ index ] )->getNode( current->getLabel() );



VarRegItem vri(current->getLabel());
VarRegItem & vvri = var_table->Find(vri);

if (var_table->WasFound()) {
result_reg = vvri.reg;
} else {
if (cnode->getValue() > 6) {
int tx = getUnusedTx();
vri.reg = tx;
} else {
vri.reg = SBASE + cnode->getValue();
vri.ehVariavel = 1;
}
```

```
result_reg = vri.reg;

var_table->Insert(vri);

}
break;
}

bool notoper = false;

if (operation == STAR_OP) {
code << "\tmult \t" << "$" << (act_reg1) << ", $" <<
(act_reg2) << "\n";
code << "\tmflo \t$" << result_reg << "\n";
} else if (operation == DIV_OP) {
code << "\tdiv \t$0" << ", $" << (act_reg1) << ", $" <<
(act_reg2) << "\n";
code << "\tmflo \t$" << result_reg << "\n";
} else if (operation == MOD_OP) {
code << "\tdiv \t$0" << ", $" << (act_reg1) << ", $" <<
(act_reg2) << "\n";
code << "\tmfhi \t$" << result_reg << "\n";
} else if (operation == NOTEQUAL_OP)
code << "\tsne \t$" << (result_reg) << ", $" <<
(act_reg1) << ", $" << (act_reg2) << "\n";
else if (operation == EQUAL_OP)
code << "\tseq \t$" << (result_reg) << ", $" <<
(act_reg1) << ", $" << (act_reg2) << "\n";
else if (operation == LESSTHAN_OP)
code << "\tsltu \t$" << (result_reg) << ", $" <<
(act_reg1) << ", $" << (act_reg2) << "\n";
else if (operation == GREATERTHAN_OP)
code << "\tsgtu \t$" << (result_reg) << ", $" <<
(act_reg1) << ", $" << (act_reg2) << "\n";
else if (operation == LESSTHANOREQUALTO_OP)
code << "\tsleu \t$" << (result_reg) << ", $" <<
(act_reg1) << ", $" << (act_reg2) << "\n";
else if (operation == GREATERTHANOREQUALTO_OP)
code << "\tsgeu \t$" << (result_reg) << ", $" <<
(act_reg1) << ", $" << (act_reg2) << "\n";
else if (operation == NOT_OP) {
notoper = true;
```

```
code << "\tnot \t$" << (result_reg) << ", $" <<
(act_reg1) << "\n";
}




if (cleancons1) {
freeRegister(act_reg1);
} else {
VarRegItem & vv1 = var_table->Find(v1);
tryToFreeRegister(vv1, grafo);
}

if (cleancons2) {
freeRegister(act_reg2);
} else {
if (!notoper) {
VarRegItem & vv2 = var_table->Find(v2);
tryToFreeRegister(vv2, grafo);
}
}


lock = -1;
lock2 = -1;
}



} // end namespace
```

E.5 Otimizador

Código do arquivo Otimizador.h.

```
#ifndef __Otimizador_h__
#define __Otimizador_h__
```

```
#include "dfg.h"
#include "dfn.h"
#include "dfe.h"
#include "Hash.h"
#include "Vector.h"

#include <sstream>
#include <string>

#include "defines.h"

namespace graphopt {


class Otimizador {

private:

void DeadNodeElimination(DFGPtr grafo, DFNPtr nodo);

public:

Otimizador() {

}

~Otimizador();

void allOpt(DFGPtr grafo);

void TreeHeightReduction(DFGPtr grafo);

void ConstantPropagation(DFGPtr grafo);

void VariablePropagation(DFGPtr grafo);

void DeadCodeElimination(DFGPtr grafo);

void OperatorStrengthReduction(DFGPtr grafo);

void CommonSubexpressionElimination(DFGPtr grafo);
```

```
};

} //end namespace

#endif
```

Código do arquivo Otimizador.cpp.

```
#include "Otimizador.h"

namespace graphopt {

Otimizador::~Otimizador() { }

void Otimizador::allOpt(DFGPtr grafo) {

ConstantPropagation(grafo);
VariablePropagation(grafo);
OperatorStrengthReduction(grafo);
TreeHeightReduction(grafo);
CommonSubexpressionElimination(grafo);
DeadCodeElimination(grafo);
}

void Otimizador::TreeHeightReduction(DFGPtr grafo) {

}

void Otimizador::ConstantPropagation(DFGPtr grafo) {
DFEPtr current;
DFNPtr pnode, xnode, znode;
List< DFEPtr >*edge_list;
int soma, times;
bool flag;

Vector< DFNPtr > & nodes = grafo->getNodes();

for ( int i = 0; i < nodes.Length(); i++) {
```

```
if ( nodes[ i ] == NULL )
    continue;
         pnode = nodes[ i ];

if ( ( pnode->getType() == STAR_OP ) || ( pnode->getType() == PLUS_OP )
|| ( pnode->getType() == MINUS_OP ) ) {

edge_list = pnode->getIngoingEdges();
if ( pnode->getType() == STAR_OP )
soma = 1;
else
soma = 0;

flag = true;
times = 0;

for ( List<DFEPtr>::iterator itr = edge_list->begin();
itr != edge_list->end(); ++itr ) {
current = *itr;
xnode = & grafo->getNode( current->getSource() );

if (xnode->getType() != CONS_NODE) {
flag = false;
break;
} else {
switch (pnode->getType()) {
case STAR_OP:
soma *= xnode->getValue();
break;
case MINUS_OP:
if (times == 0)
soma = xnode->getValue();
else
soma -= xnode->getValue();
break;
case PLUS_OP:
soma += xnode->getValue();
break;
}
}
times++;
}
```

```
if (flag) {

std::stringstream sstr;
sstr << "constant" << soma;

String tid( ( sstr.str() ).c_str() );
xnode = & grafo->addNode(tid);
xnode->setType(CONS_NODE);
xnode->setCT(CT_NORMAL);
xnode->setValue(soma);

String upn(UP_NOP);
grafo->addEdge(upn, tid, 0);

edge_list = pnode->getOutgoingEdges();


for ( List<DFEPtr>::iterator itr = edge_list->begin();
itr != edge_list->end(); ++itr ) {
current = *itr;
znode = & grafo->getNode( current->getDestiny() );

grafo->addEdge(tid, znode->getName(), 1, 1);
}

edge_list = pnode->getIngoingEdges();

for ( List<DFEPtr>::iterator itr = edge_list->begin();
itr != edge_list->end(); ++itr ) {
current = *itr;
xnode = & grafo->getNode( current->getSource() );
if ( (xnode->numberOfSuccessors() < 2 ) || (current->getRepeats() > 1) ) {
String btp(BOTTOM_NOP);
grafo->addEdge(xnode->getName(), btp, 0);
}
}

grafo->removeNode(pnode->getName());

}

}
```

```
}


}

void Otimizador::VariablePropagation(DFGPtr grafo) {

}

void Otimizador::DeadCodeElimination(DFGPtr grafo) {


DFEPtr current;
DFNPtr pnode;
List< DFEPtr >*edge_list;

DFN & sink = grafo->getBottomNOP();

edge_list = sink.getIngoingEdges();

// for each predecessor of "node", store parameter on $aX
for ( List<DFEPtr>::iterator itr = edge_list->begin();
itr != edge_list->end(); ++itr ) {
current = *itr;
pnode = & grafo->getNode( current->getSource() );

if (pnode->getType() == OUT_NODE)
continue;
else
DeadNodeElimination(grafo, pnode);
}
}

void Otimizador::DeadNodeElimination(DFGPtr grafo, DFNPtr nodo) {
DFEPtr current;
DFNPtr pnode;
List< DFEPtr >*edge_list;

edge_list = nodo->getIngoingEdges();

// for each predecessor of "node", store parameter on $aX
for ( List<DFEPtr>::iterator itr = edge_list->begin();
itr != edge_list->end(); ++itr ) {
```

```
current = *itr;
pnode = & grafo->getNode( current->getSource() );

if (pnode->numberOfSuccessors() < 2) {

DeadNodeElimination(grafo, pnode);

}
}

if (nodo->numberOfSuccessors() < 2) {
grafo->removeNode(nodo->getName());
}

}

void Otimizador::OperatorStrengthReduction(DFGPtr grafo) {
DFEPtr current, current2, current3;
DFNPtr pnode, qnode, rnode, snode, bottomnode;
List< DFEPtr >*edge_list, *edge_list2, *edge_list3;


Vector< DFNPtr > & nodes = grafo->getNodes();

for ( int i = 0; i < nodes.Length(); i++) {
if ( nodes[ i ] == NULL )
    continue;
         pnode = nodes[ i ];
if (pnode->getType() == STAR_OP) {
edge_list = pnode->getIngoingEdges();

for ( List<DFEPtr>::iterator itr = edge_list->begin();
itr != edge_list->end(); ++itr ) {
current = *itr;
qnode = & grafo->getNode( current->getSource() );
if (qnode->getType() == CONS_NODE) {
if (qnode->getValue() == 1) {
if (qnode->numberOfSuccessors() == 1) {
bottomnode = & grafo->getBottomNOP();
grafo->addEdge(qnode->getName(), bottomnode->getName(), 0);
}
grafo->removeEdge(*qnode, *pnode);
edge_list2 = pnode->getIngoingEdges();
```

```
for ( List<DFEPtr>::iterator itr2 = edge_list2->begin();
itr2 != edge_list2->end(); ++itr2 ) {
current2 = *itr2;
rnode = & grafo->getNode( current2->getSource() );
edge_list3 = pnode->getOutgoingEdges();
for ( List<DFEPtr>::iterator itr3 = edge_list3->begin();
itr3 != edge_list3->end(); ++itr3 ) {
current3 = *itr3;
snode = & grafo->getNode( current3->getDestiny() );
grafo->addEdge(*rnode, *snode, 1 ,1);
}

}
grafo->removeNode(pnode->getName());
break;


} else if (qnode->getValue() == 0) {
grafo->removeEdge(*qnode, *pnode);

edge_list2 = pnode->getIngoingEdges();

for ( List<DFEPtr>::iterator itr2 = edge_list2->begin();
itr2 != edge_list2->end(); ++itr2 ) {
current2 = *itr2;
rnode = & grafo->getNode( current2->getSource() );
if (rnode->numberOfSuccessors() == 1) {
bottomnode = & grafo->getBottomNOP();
grafo->addEdge(rnode->getName(), bottomnode->getName(), 0);
}
grafo->removeEdge(*rnode, *pnode);
}

edge_list3 = pnode->getOutgoingEdges();

for ( List<DFEPtr>::iterator itr3 = edge_list3->begin();
itr3 != edge_list3->end(); ++itr3 ) {
current3 = *itr3;
snode = & grafo->getNode( current3->getDestiny() );
grafo->addEdge(qnode->getName(), snode->getName(), 1, 1);
}

grafo->removeNode(pnode->getName());
```

```
break;

}
}

}
} else if ( pnode->getType() == PLUS_OP || pnode->getType() == MINUS_OP) {
edge_list = pnode->getIngoingEdges();
bool passed_once = false;

for ( List<DFEPtr>::iterator itr = edge_list->begin();
itr != edge_list->end(); ++itr ) {
if (pnode->getType() == MINUS_OP && !passed_once) {
passed_once = true;
continue;
}

current = *itr;
qnode = & grafo->getNode( current->getSource() );


if (qnode->getValue() == 0) {
if (qnode->numberOfSuccessors() == 1) {
bottomnode = & grafo->getBottomNOP();
grafo->addEdge(qnode->getName(), bottomnode->getName(), 0);
}
grafo->removeEdge(*qnode, *pnode);
edge_list2 = pnode->getIngoingEdges();

for ( List<DFEPtr>::iterator itr2 = edge_list2->begin();
itr2 != edge_list2->end(); ++itr2 ) {
current2 = *itr2;
rnode = & grafo->getNode( current2->getSource() );
edge_list3 = pnode->getOutgoingEdges();
for ( List<DFEPtr>::iterator itr3 = edge_list3->begin();
itr3 != edge_list3->end(); ++itr3 ) {
current3 = *itr3;
snode = & grafo->getNode( current3->getDestiny() );
grafo->addEdge(rnode->getName(), snode->getName(), 1, 1);
}
}
grafo->removeNode(pnode->getName());
break;
```

```
}


}

}
}

}

void Otimizador::CommonSubexpressionElimination(DFGPtr grafo) {
DFEPtr current, current2, current3;
DFNPtr pnode, qnode, xnode, ynode, wnode;
List< DFEPtr >*edge_list, *edge_list2, *edge_list3;
bool flag = true;

Vector< DFNPtr > & nodes = grafo->getNodes();
Vector< DFNPtr > operations;
int opcount = 0;
int tipo = -1;

for ( int i = 0; i < nodes.Length(); i++) {
if ( nodes[ i ] == NULL )
    continue;
         pnode = nodes[ i ];
tipo = pnode->getType();
if ((tipo == STAR_OP) || (tipo == PLUS_OP) || (tipo == MINUS_OP)
|| (tipo == DIV_OP) || (tipo == MOD_OP) ) {

if( opcount == operations.Length( ) ) {
         operations.Double( );
for ( int j = operations.Length() / 2; j < operations.Length(); j++ )
operations[ j ] = NULL;
}

operations[ opcount++ ] = pnode;
}
}

for (int i = 0; i < opcount; i++) {
if ( operations[ i ] == NULL )
    continue;
```

```
pnode = operations[i];
for (int j = i + 1; j < opcount; j++) {
if ( operations[ i ] == NULL )
    continue;

qnode = operations[j];

edge_list = pnode->getIngoingEdges();

for ( List<DFEPtr>::iterator itr = edge_list->begin();
itr != edge_list->end(); ++itr ) {
current = *itr;
xnode = & grafo->getNode( current->getSource() );

edge_list2 = qnode->getIngoingEdges();

flag = true;

for ( List<DFEPtr>::iterator itr2 = edge_list2->begin();
itr2 != edge_list2->end(); ++itr2 ) {
current2 = *itr2;
ynode = & grafo->getNode( current2->getSource() );

if (ynode->getName() == xnode->getName())
flag = false;
}

if (flag) {
break;
}
}

if (!flag) {
edge_list2 = pnode->getOutgoingEdges();
String edge_label;

for ( List<DFEPtr>::iterator itr2 = edge_list2->begin();
itr2 != edge_list2->end(); ++itr2 ) {
current2 = *itr2;
if (current2->getLabel() != "") {
edge_label = current2->getLabel();
break;
}
```

```
}

edge_list3 = qnode->getOutgoingEdges();

for ( List<DFEPtr>::iterator itr3 = edge_list3->begin();
itr3 != edge_list3->end(); ++itr3 ) {
current3 = *itr3;
wnode = & grafo->getNode( current3->getDestiny() );

DFE & ed = grafo->addEdge(pnode->getName(), wnode->getName(), 1, 1);
if (edge_label != "")
ed.setLabel(edge_label);
}

grafo->removeNode(qnode->getName());
operations[j] = NULL;
}
}
}

}


} // end namespace
```

E.6 Grafo para XML

Código do arquivo GraphToXML.h.

```
/**
  * @author Roberto Hartke Neto
  * @date 03/10/2003
  */

#ifndef __GraphToXML_h__
#define __GraphToXML_h__

#include <fstream>
#include <string>
#include "Vector.h"
#include "graph.h"
```

```
#include "dfg.h"
#include "defines.h"


/**
  * \namespace graphxml namespace as c++ good programming.
  */
namespace graphxml {

/**
 * \class GraphToXML
 *  A class to convert a Graph instance to a ygraphml
 (yEd graphml extension) file. Can be viewed on yEd.
 */

class GraphToXML {


protected:
/**
     * Output stream.
     * Writes strings to a file.
     */
std::ofstream out;

/**
     * Writes the header of a ygraphl file. Method called in initDocument(...).
 * @see initDocument()
     */
void writeHeader();

/**
     * Writes the tail of a ygraphl file. Method called in endDocument(...).
 * @see endDocument()
     */
void writeTail();

public:
/**
       * An empty constructor.
       * No line codes inside.
       */
GraphToXML() { }
```

```
/**
       * An initializer constructor.
       * Calls initDocument(...) method.
   * @param filename File to be written.
   * @see initDocument()
       */
GraphToXML(const char* filename) {
initDocument(filename);
}

/**
       * An empty destructor.
       * No line codes inside.
       */
~GraphToXML();

/**
       * Writes node code to the file.
       * @param type Node type: 0 for source and sink,
       1 for constants and 2 for operations.
       * @param number Node number identifier.
       * @param level_h Horizontal level, needed to draw the node.
   * @param level_v Vertical level, needed to draw the node.
   * @param text Text to be shown inside node.
       */
void newNode(int type, int number, int level_h, int level_v,
const char *s = "");

/**
       * Writes edge code to the file.
       * @param source Number of the edge source node.
       * @param target Number of the edge target node.
       * @param number Edge identifier number.
   * @param text Text to be shown beside edge.
   */
void newEdge(int source, int target, int number, int type = -1, const char *s = "");

/**
       * Opens the file (or create it) and writes initial ygraphml code.
       * @param filename File to be written.
       */
void initDocument(const char* filename);
```

```
/**
       * Writes final ygraphml code and closes the file.
       */
void endDocument();

/**
       * Writes ygraphml code necessary to visualize a Graph instance.
   * @param grafo Graph instance which wants to be visualized.
       */
void generateGraphXML(Graph & grafo);

void generateDFGXML(DFG & grafo);

};

}

#endif
```

Código do arquivo GraphToXML.cpp.

```
#include "GraphToXML.h"
#include "edge.h"
#include "node.h"

namespace graphxml {

GraphToXML::~GraphToXML() { }

void GraphToXML::initDocument(const char* filename) {

out.open(filename);
}

void GraphToXML::endDocument() {

writeTail();

out.close();
}
```

```
void GraphToXML::writeHeader() {
out << "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n";
out << "<!-- This file was written by the GraphToXML application.-->\n";
out << "<graphml xmlns=\"http://graphml.graphdrawing.org/xmlns/graphml\" >\n";
    out << "\t<key id=\"yfiles.nodedata\" for=\"node\" yfiles.type=\"nodedata\"/>\n";
out << "\t<key id=\"yfiles.edgedata\" for=\"edge\" yfiles.type=\"edgedata\"/>\n";
out << "\t<graph id=\"G\" edgedefault=\"directed\">\n";
}

void GraphToXML::writeTail() {
out << "\t</graph>\n";
out << "</graphml>\n";
}


void GraphToXML::newEdge(int source, int target, int number, int type,
const char *s) {
char *line;
switch (type) {
case 1:
line = "line"; break;
case 0:
line = "dashed1"; break;
default:
line = "line1"; break;
}

out << "\t\t<edge id=\"G:e" << number << "\" source=\"G:n" <<
source <<"\" target=\"G:n"<< target <<"\">\n"; //
out << "\t\t\t<data key=\"yfiles.edgedata\" >\n";
out << "\t\t\t<EdgeLayout>\n";
out << "\t\t\t\t<SourcePort>\n";
out << "\t\t\t\t\t<Location>\n";
out << "\t\t\t\t\t\t<X>0.0</X>\n";
out << "\t\t\t\t\t\t<Y>0.0</Y>\n";
out << "\t\t\t\t\t</Location>\n";
out << "\t\t\t\t</SourcePort>\n";
out << "\t\t\t\t<Path>\n";
out << "\t\t\t\t</Path>\n";
out << "\t\t\t\t<TargetPort>\n";
out << "\t\t\t\t\t<Location>\n";
out << "\t\t\t\t\t\t<X>0.0</X>\n";
out << "\t\t\t\t\t\t<Y>0.0</Y>\n";
```

```
out << "\t\t\t\t\t</Location>\n";
out << "\t\t\t\t</TargetPort>\n";
out << "\t\t\t</EdgeLayout>\n";
out << "\t\t\t<Color r=\"0\" g=\"0\" b=\"0\" />\n";
out << "\t\t\t<LineType type=\"" << line << "\"/>\n";
out << "\t\t\t<CurveType type=\"Polyline\"/>\n";
out << "\t\t\t<SourceArrow>\n";
out << "\t\t\t\t<Arrow type=\"1\"/>\n";
out << "\t\t\t</SourceArrow>\n";
out << "\t\t\t<TargetArrow>\n";
out << "\t\t\t\t<Arrow type=\"2\"/>\n";
out << "\t\t\t</TargetArrow>\n";
out << "\t\t\t<Label>\n";
out << "\t\t\t\t<Text>"<< s << "</Text>\n"; //
out << "\t\t\t\t<Model value=\"2\"/>\n";
out << "\t\t\t\t<Position value=\"11\"/>\n";
out << "\t\t\t\t<Visible value=\"true\"/>\n";
out << "\t\t\t\t<Alignment value=\"1\"/>\n";
out << "\t\t\t\t<Color r=\"255\" g=\"0\" b=\"0\" />\n";
out << "\t\t\t\t<Background value=\"false\"/>\n";
out << "\t\t\t\t<FontName value=\"Dialog\"/>\n";
out << "\t\t\t\t<FontSize value=\"12\"/>\n";
out << "\t\t\t\t<FontStyle value=\"0\"/>\n";
out << "\t\t\t</Label>\n";
out << "\t\t\t</data>\n";
out << "\t\t</edge>\n";
}

void GraphToXML::newNode(int type, int number, int level_h,
int level_v, const char *s) {
//std::printf("Nodo: %d\n", type);
int *rgb = new int[3]; // red, green, blue
int *pos_size = new int[4]; //x, y, width, height
int shape;
pos_size[0] = (level_h + 1)* 80;
pos_size[1] = (level_v + 1)* (-80);
pos_size[2] = 30;
pos_size[3] = 30;

switch (type) {
case SOURCE_NODE: // source
rgb[0] = 255; rgb[1] = 153; rgb[2] = 0; shape = 2; pos_size[2] = 73;
s = "source"; break;
```

```
case SINK_NODE: // sink
rgb[0] = 51; rgb[1] = 102; rgb[2] = 255; shape = 2; pos_size[2] = 73;
s = "sink"; break;
case CONS_NODE: // constant
rgb[0] = 153; rgb[1] = 204; rgb[2] = 255; shape = 0; break;
case CALL_NODE: // call
rgb[0] = 255; rgb[1] = 153; rgb[2] = 204; shape = 2; s = "call";
break;
case IN_NODE: // parameter
rgb[0] = 204; rgb[1] = 255; rgb[2] = 205; shape = 5; break;
case OUT_NODE: // return value
rgb[0] = 204; rgb[1] = 254; rgb[2] = 255; shape = 10; break;
case STAR_OP: //operation
rgb[0] = 255; rgb[1] = 255; rgb[2] = 153; shape = 2; s = "*";
break;
case PLUS_OP: //operation
rgb[0] = 255; rgb[1] = 255; rgb[2] = 153; shape = 2; s = "+";
break;
case DIV_OP: //operation
rgb[0] = 255; rgb[1] = 255; rgb[2] = 153; shape = 2; s = "/";
break;
case MOD_OP: //operation
rgb[0] = 255; rgb[1] = 255; rgb[2] = 153; shape = 2; s = "%";
break;
case MINUS_OP: //operation
rgb[0] = 255; rgb[1] = 255; rgb[2] = 153; shape = 2; s = "-";
break;
case LESSTHANOREQUALTO_OP: //operation
rgb[0] = 51; rgb[1] = 204; rgb[2] = 204; shape = 2; s = "&lt;=";
break;
case GREATERTHANOREQUALTO_OP: //operation
rgb[0] = 51; rgb[1] = 204; rgb[2] = 204; shape = 2; s = "&gt;=";
break;
case GREATERTHAN_OP: //operation
rgb[0] = 51; rgb[1] = 204; rgb[2] = 204; shape = 2; s = "&gt;";
break;
case LESSTHAN_OP: //operation
rgb[0] = 51; rgb[1] = 204; rgb[2] = 204; shape = 2; s = "&lt;";
break;
case EQUAL_OP: //operation
rgb[0] = 51; rgb[1] = 204; rgb[2] = 204; shape = 2; s = "==";
break;
case NOTEQUAL_OP: //operation
```

```
rgb[0] = 51; rgb[1] = 204; rgb[2] = 204; shape = 2; s = "!=";
break;
case NOT_OP: //operation
rgb[0] = 51; rgb[1] = 204; rgb[2] = 204; shape = 2; s = "!";
break;
case AND_OP: //operation
rgb[0] = 51; rgb[1] = 204; rgb[2] = 204; shape = 2; s = "&amp;&amp;";
break;
case OR_OP: //operation
rgb[0] = 51; rgb[1] = 204; rgb[2] = 204; shape = 2; s = "||";
break;
default:
rgb[0] = 204; rgb[1] = 204; rgb[2] = 255; shape = 2;
break;
}



out << "\t\t<node id=\"G:n" << number << "\">\n"; //
    out << "\t\t\t<data key=\"yfiles.nodedata\" >\n";
out << "\t\t\t\t<NodeLayout>\n";
out << "\t\t\t\t\t<Location>\n";
out << "\t\t\t\t\t\t<X>" << pos_size[0] << "</X>\n"; //
out << "\t\t\t\t\t\t<Y>" << pos_size[1] << "</Y>\n"; //
out << "\t\t\t\t\t</Location>\n";
out << "\t\t\t\t\t<Size>\n";
out << "\t\t\t\t\t\t<Width>" << pos_size[2] << "</Width>\n"; //
out << "\t\t\t\t\t\t<Height>" << pos_size[3] << "</Height>\n"; //
out << "\t\t\t\t\t</Size>\n";
out << "\t\t\t\t</NodeLayout>\n";
out << "\t\t\t\t<FillColor>\n";
out << "\t\t\t\t\t<Color r=\"" << rgb[0] << "\" g=\"" <<
rgb[1] << "\" b=\"" << rgb[2] << "\" />\n"; //

out << "\t\t\t\t</FillColor>\n";
out << "\t\t\t\t<LineColor>\n";
out << "\t\t\t\t\t<Color r=\"0\" g=\"0\" b=\"0\" />\n";
out << "\t\t\t\t</LineColor>\n";
out << "\t\t\t\t<LineType type=\"line2\"/>\n";
out << "\t\t\t\t<Transparent>false</Transparent>\n";
out << "\t\t\t\t<Shape type=\"" << shape << "\"/>\n"; //
out << "\t\t\t\t<Label>\n";
out << "\t\t\t\t\t<Text>" << s << "</Text>\n"; //
```

```
out << "\t\t\t\t\t<Model value=\"1\"/>\n";
out << "\t\t\t\t\t<Position value=\"100\"/>\n";
out << "\t\t\t\t\t<Visible value=\"true\"/>\n";
out << "\t\t\t\t\t<Alignment value=\"1\"/>\n";
out << "\t\t\t\t\t<Color r=\"0\" g=\"0\" b=\"0\" />\n";
out << "\t\t\t\t\t<Background value=\"false\"/>\n";
out << "\t\t\t\t\t<FontName value=\"Dialog\"/>\n";
out << "\t\t\t\t\t<FontSize value=\"12\"/>\n";
out << "\t\t\t\t\t<FontStyle value=\"0\"/>\n";
out << "\t\t\t\t</Label>\n";
out << "\t\t\t</data>\n";
out << "\t\t</node>\n";

delete [ ] rgb;
delete [ ] pos_size;
}


void GraphToXML::generateGraphXML(Graph & grafo) {
writeHeader();

Vector< NODEPtr > & nodes = grafo.getNodes();
Vector< EDGEPtr > & edges = grafo.getEdges();

int j = 0;
int k = 0;
for ( int i = 0; i < nodes.Length(); i++) {
if ( nodes[ i ] == NULL )
    continue;
         //NODEPtr current = nodes[ i ];
newNode(0, i, j++, k) ;

if ((j % 8) == 0)
k++;
}

for ( int i = 0; i < edges.Length(); i++) {
     if ( edges[ i ] == NULL )
     continue;
         EDGEPtr current = edges[ i ];
newEdge(current->getSource(), current->getDestiny(), i);
     }
```

```
endDocument();

}

void GraphToXML::generateDFGXML(DFG & grafo) {
writeHeader();

Vector< DFNPtr > & nodes = grafo.getNodes();
Vector< DFEPtr > & edges = grafo.getEdges();
int j = 0;
int k = 0;
for ( int i = 0; i < nodes.Length(); i++) {
if ( nodes[ i ] == NULL )
    continue;
         DFN *current = nodes[ i ];
if (current->getType() == CONS_NODE) {
std::stringstream sstr;
sstr << current->getValue();
newNode(current->getType(), i, j++, k, (sstr.str()).c_str()) ;
} else
newNode(current->getType(), i, j++, k) ;

if ((j % 8) == 0)
k++;
}

for ( int i = 0; i < edges.Length(); i++) {
     if ( edges[ i ] == NULL )
     continue;

         DFEPtr current = edges[ i ];
newEdge(current->getSource(), current->getDestiny(), i,
(int) current->getCost(), (current->getLabel()).c_str());


     }

endDocument();

}

} //end namespace
```