Presentation of TC-7

Assistants 2009

May 6, 2014

Presentation of TC-7

1. Overview of the tarball
2. Monoburg
3. Instruction representation

Overview of the tarball

1. Overview of the tarball
2. Monoburg
3. Instruction representation

The tree structure of TC-7

- New directories:
  - ‘`src/assem`’: Yet another intermediate language (last one!).
  - ‘`src/target`’: Classes describing the target architecture (`Mips` and `Ia32`). Given in full.
  - ‘`src/target`’: Instruction selection.

The tree structure of TC-7

- New directories:
  - ‘`src/assem`’: Yet another intermediate language (last one!).
  - ‘`src/target`’: Classes describing the target architecture (`Mips` and `Ia32`). Given in full.
  - ‘`src/target`’: Instruction selection.

The tree structure of TC-7

- New directories:
  - '`src/assem`': Yet another intermediate language (last one!).
  - '`src/target`': Classes describing the target architecture (`Mips` and `Ia32`). Given in full.
  - '`src/target`': Instruction selection.

The tree structure of TC-7

- New directories:
  - '`src/assem`': Yet another intermediate language (last one!).
  - '`src/target`': Classes describing the target architecture (`Mips` and `Ia32`). Given in full.
  - '`src/target`': Instruction selection.

Monoburg

1. Overview of the tarball
2. Monoburg
   - Introduction
   - Presentation
3. Instruction representation

Introduction

1. Overview of the tarball
2. Monoburg
   - Introduction
   - Presentation
3. Instruction representation

What is Monoburg?

- Implementation of `IBURG`, developed in the context of Mono Novell (2004)
- Created in order to generate the code-generator for the Mono Virtual Machine, which uses JIT (Just-In-Time) compilation.
- Simple, and maintainable compared to the older '`codegen.cc`' of `Tiger`.

What is Monoburg?

- Implementation of `IBURG`, developed in the context of Mono Novell (2004)
- Created in order to generate the code-generator for the Mono Virtual Machine, which uses JIT (Just-In-Time) compilation.
- Simple, and maintainable compared to the older '`codegen.cc`' of `Tiger`.

What is Monoburg?

- Implementation of `IBURG`, developed in the context of Mono Novell (2004)
- Created in order to generate the code-generator for the Mono Virtual Machine, which uses JIT (Just-In-Time) compilation.
- Simple, and maintainable compared to the older '`codegen.cc`' of `Tiger`.

Presentation

1. Overview of the tarball
2. Monoburg
   - Introduction
   - Presentation
3. Instruction representation

Principle

- Give all nodes of the tree.
- Do pattern matching on tree to select the best rewrite: bottom up algorithm (BURG: Bottom Up Rewrite System).
- Each rewrite can have an associated cost.

Principle

- Give all nodes of the tree.
- Do pattern matching on tree to select the best rewrite: bottom up algorithm (`BURG`: Bottom Up Rewrite System).
- Each rewrite can have an associated cost.

Principle

- Give all nodes of the tree.
- Do pattern matching on tree to select the best rewrite: bottom up algorithm (`BURG`: Bottom Up Rewrite System).
- Each rewrite can have an associated cost.

Example (Excerpt from move.brg)

```
move: Move(Mem(e1 : exp), Mem(e2 : exp))
{
  temp::Temp rval;

  rExp exp = e2.cast<Exp> ();
  assertion (exp);
  EMIT (MIPS_ASSEMBLY.load_build (exp->asm_get (), rval));

  exp = e1.cast<Exp> ();
  assertion (exp);
  EMIT (MIPS_ASSEMBLY.store_build (rval, exp->asm_get ()));
}
```

Instruction representation

1. Overview of the tarball
2. Monoburg
3. Instruction representation
   - Runtime

Constraints

- Represent a final assembly instruction: a label, an instruction or a move.
- Used for intermediate language and final assembly: different registers depending on register allocation.

Constraints

- Represent a final assembly instruction: a label, an instruction or a move.
- Used for intermediate language and final assembly: different registers depending on register allocation.

Implementation

- Labels and instructions are represented by a printf-style string.
- Registers and labels are stored in separated lists.
- Replacement is done at display of asm.

Implementation

- Labels and instructions are represented by a printf-style string.
- Registers and labels are stored in separated lists.
- Replacement is done at display of asm.

Implementation

- Labels and instructions are represented by a printf-style string.
- Registers and labels are stored in separated lists.
- Replacement is done at display of asm.

Example

```
// First list: used temporaries list.
// Second list: defined temporaries list.
// Third list: labels list.
res.push_back (new assem::Oper ("j\t'j", L (), L (), jump_list));
```

Runtime

1. Overview of the tarball
2. Monoburg
3. Instruction representation
   - Runtime

Principle

- The Tiger language provides primitives.
- Primitives can't be written in Tiger!
- Primitives are written in assembly language, then included in the output.

Principle

- The Tiger language provides primitives.
- Primitives can't be written in Tiger!
- Primitives are written in assembly language, then included in the output.

Principle

- The Tiger language provides primitives.
- Primitives can't be written in Tiger!
- Primitives are written in assembly language, then included in the output.

Example (Excerpt from the runtime.s)

```
## Routine: print ----------------------------------------
# Print the string $a0
.text
tc_print:
lw $a2, ($a0)
addi $a1, $a0, 4
li $a0, 1
li $v0, 0x03
syscall ; write
## Content of $v0 is undetermined
jr $ra
```