# Presentation of TC-8

Assistants 2009

May 6, 2014

# Presentation of TC-8

# Overview of the tarball

# The tree structure of TC-8

- New directory:
  - ‘`src/liveness`’: Where you will work. Graphs describing the flow of control, the liveness and the interferences.
- New files:
  - ‘`lib/miscellaneous/graphs.h`’: Generic implementation of graphs.

# The tree structure of TC-8

- New directory:
  - '`src/liveness`': Where you will work. Graphs describing the flow of control, the liveness and the interferences.
- New files:
  - '`lib/misc/graph*`': Generic implementation of graph.

# The tree structure of TC-8

- New directory:
  - '`src/liveness`': Where you will work. Graphs describing the flow of control, the liveness and the interferences.
- New files:
  - '`lib/misc/graph*`': Generic implementation of graph.

# The tree structure of TC-8

- New directory:
  - '`src/liveness`': Where you will work. Graphs describing the flow of control, the liveness and the interferences.
- New files:
  - '`lib/misc/graph*`': Generic implementation of graph.

# Graph

# Quick overview of Graph

- Rely on boost.
- Implemented as an adjacency graph.
- Take a look at methods and free functions for the class `adjacency_list`.

# Quick overview of Graph

- Rely on boost.
- Implemented as an adjacency graph.
- Take a look at methods and free functions for the class `adjacency_list`.

# Quick overview of Graph

- Rely on boost.
- Implemented as an adjacency graph.
- Take a look at methods and free functions for the class `adjacency_list`.

# Flowgraph

# Quick overview of Flowgraph

- Nodes of the graph are instructions.
- If an instruction *a* can be followed by another *b*, then an edge is created between *a* and *b*.
- The flowgraph is an input for the creation of the liveness graph.

# Quick overview of Flowgraph

- Nodes of the graph are instructions.
- If an instruction *a* can be followed by another *b*, then an edge is created between *a* and *b*.
- The flowgraph is an input for the creation of the liveness graph.

# Quick overview of Flowgraph

- Nodes of the graph are instructions.
- If an instruction *a* can be followed by another *b*, then an edge is created between *a* and *b*.
- The flowgraph is an input for the creation of the liveness graph.

# Liveness

# Quick overview of Liveness

- Answer to the question: who is alive just before the instruction, and who just afterwards?
- Rely on an iterative algorithm with a fixed point (even the least fixed point!).

# Quick overview of Liveness

- Answer to the question: who is alive just before the instruction, and who just afterwards?
- Rely on an iterative algorithm with a fixed point (even the least fixed point!).

# Interference

# Quick overview of Interference

- Nodes of the graph are temporaries.
- If temporary *a* is alive at the same time than *b*, then they interfere, and an edge is created.
- The Interference graph is used for the register allocation.
- What about MOVE?
  - If one move *a* in *b*, an interference edge should be created.
  - But, in fact we want *a* and *b* to be in the same register.
  - No edge, but remember it was a MOVE!

# Quick overview of Interference

- Nodes of the graph are temporaries.
- If temporary *a* is alive at the same time than *b*, then they interfere, and an edge is created.
- The Interference graph is used for the register allocation.
- What about MOVE?
  - If one move *a* in *b*, an interference edge should be created.
  - But, in fact we want *a* and *b* to be in the same register.
  - No edge, but remember it was a MOVE!

# Quick overview of Interference

- Nodes of the graph are temporaries.
- If temporary *a* is alive at the same time than *b*, then they interfere, and an edge is created.
- The Interference graph is used for the register allocation.
- What about MOVE?
  - If one move *a* in *b*, an interference edge should be created.
  - But, in fact we want *a* and *b* to be in the same register.
  - No edge, but remember it was a MOVE!

# Quick overview of Interference

- Nodes of the graph are temporaries.
- If temporary *a* is alive at the same time than *b*, then they interfere, and an edge is created.
- The Interference graph is used for the register allocation.
- What about MOVE?
  - If one move *a* in *b*, an interference edge should be created.
  - But, in fact we want *a* and *b* to be in the same register.
  - No edge, but remember it was a MOVE!

# Quick overview of Interference

- Nodes of the graph are temporaries.
- If temporary *a* is alive at the same time than *b*, then they interfere, and an edge is created.
- The Interference graph is used for the register allocation.
- What about MOVE?
  - If one move *a* in *b*, an interference edge should be created.
  - But, in fact we want *a* and *b* to be in the same register.
  - No edge, but remember it was a MOVE!

# Quick overview of Interference

- Nodes of the graph are temporaries.
- If temporary *a* is alive at the same time than *b*, then they interfere, and an edge is created.
- The Interference graph is used for the register allocation.
- What about MOVE?
  - If one move *a* in *b*, an interference edge should be created.
  - But, in fact we want *a* and *b* to be in the same register.
  - No edge, but remember it was a MOVE!

# Quick overview of Interference

- Nodes of the graph are temporaries.
- If temporary *a* is alive at the same time than *b*, then they interfere, and an edge is created.
- The Interference graph is used for the register allocation.
- What about MOVE?
  - If one move *a* in *b*, an interference edge should be created.
  - But, in fact we want *a* and *b* to be in the same register.
  - No edge, but remember it was a MOVE!

# Testing

# Testing TC-8

- No automated way to do it!
- A few unit tests, but principally:
- By looking at the generated graphs with dotty.

# Testing TC-8

- No automated way to do it!
- A few unit tests, but principally:
- By looking at the generated graphs with dotty.

# Testing TC-8

- No automated way to do it!
- A few unit tests, but principally:
- By looking at the generated graphs with dotty.