Formalizing and Implementing Multi-Result Supercompilation

Ilya G. Klyuchnikov Sergei A. Romanenko

Keldysh Institute of Applied Mathematics
Russian Academy of Sciences

2012-07 / Meta 2012

Outline

1. Different types of supercompilation
   - SC: Deterministic/traditional SC (a function)
   - NDSC: Non-deterministic SC (a relation)
   - MRSC: Multi-result SC (a multi-valued function)

2. Nice features of multi-result supercompilation
   - Finiteness of trees of completed graphs
   - Decoupling whistle and generalization

3. The core of the MRSC Toolkit
   - Two representations for graphs of configurations
   - Operations on S-graphs

4. Conclusions

Outline

1. Different types of supercompilation
   - SC: Deterministic/traditional SC (a function)
   - NDSC: Non-deterministic SC (a relation)
   - MRSC: Multi-result SC (a multi-valued function)
2. Nice features of multi-result supercompilation
   - Finiteness of trees of completed graphs
   - Decoupling whistle and generalization
3. The core of the MRSC Toolkit
   - Two representations for graphs of configurations
   - Operations on S-graphs
4. Conclusions

Outline

1. Different types of supercompilation
   - SC: Deterministic/traditional SC (a function)
   - NDSC: Non-deterministic SC (a relation)
   - MRSC: Multi-result SC (a multi-valued function)
2. Nice features of multi-result supercompilation
   - Finiteness of trees of completed graphs
   - Decoupling whistle and generalization
3. The core of the MRSC Toolkit
   - Two representations for graphs of configurations
   - Operations on S-graphs
4. Conclusions

Outline

1. Different types of supercompilation
   - SC: Deterministic/traditional SC (a function)
   - NDSC: Non-deterministic SC (a relation)
   - MRSC: Multi-result SC (a multi-valued function)
2. Nice features of multi-result supercompilation
   - Finiteness of trees of completed graphs
   - Decoupling whistle and generalization
3. The core of the MRSC Toolkit
   - Two representations for graphs of configurations
   - Operations on S-graphs
4. Conclusions

Outline

SC: Deterministic/traditional SC (a function)

(Fold)
$$\frac{\exists\alpha : \mathit{foldable}(g, \beta, \alpha)}{g \to \mathit{fold}(g, \beta, \alpha)}$$

(Drive)
$$\frac{\nexists\alpha : \mathit{foldable}(g, \beta, \alpha) \quad \neg\mathit{dangerous}(g, \beta) \quad \mathit{cs} = \mathit{driveStep}(c)}{g \to \mathit{addChildren}(g, \beta, \mathit{cs})}$$

(Rebuild)
$$\frac{\nexists\alpha : \mathit{foldable}(g, \beta, \alpha) \quad \mathit{dangerous}(g, \beta) \quad c' = \mathit{rebuilding}(g, c)}{g \to \mathit{rebuild}(g, \beta, c')}$$

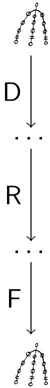
$\mathit{dangerous}(g, \beta)$ appears in (Drive) and (Rebuild)!

(Drive) and (Rebuild) are mutually exclusive.

(Rebuild) is used as a last resort if (Drive) is blocked by $\mathit{dangerous}(g, \beta)$...

SC: Deterministic/traditional SC (a function)

A finite sequence of graphs:



NDSC: Non-deterministic SC (a relation)

(Fold) $$\frac{\exists \alpha : \mathit{foldable}(g,\, \beta,\, \alpha)}{g \rightarrow \mathit{fold}(g,\, \beta,\, \alpha)}$$

(Drive) $$\frac{\nexists \alpha : \mathit{foldable}(g,\, \beta,\, \alpha) \quad \mathit{cs} = \mathit{driveStep}(c)}{g \rightarrow \mathit{addChildren}(g,\, \beta,\, \mathit{cs})}$$

(Rebuild) $$\frac{\nexists \alpha : \mathit{foldable}(g,\, \beta,\, \alpha) \quad c' \in \mathit{rebuildings}(c)}{g \rightarrow \mathit{rebuild}(g,\, \beta,\, c')}$$

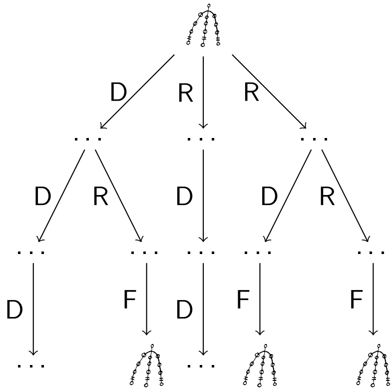
$\mathit{dangerous}(g,\, \beta)$ has disappeard from (Drive) and (Rebuild)!

(Drive) and (Rebuild) are not mutually exclusive.

(Drive) is always applicable.

NDSC: Non-deterministic SC (a relation)

A (possibly) infinite tree of graphs:



MRSC: Multi-result SC (a multi-valued function)

(Fold) $$\frac{\exists \alpha : \mathit{foldable}(g, \beta, \alpha)}{g \to \mathit{fold}(g, \beta, \alpha)}$$

(Drive) $$\frac{\nexists \alpha : \mathit{foldable}(g, \beta, \alpha) \quad \neg \mathit{dangerous}(g, \beta) \quad \mathit{cs} = \mathit{driveStep}(c)}{g \to \mathit{addChildren}(g, \beta, \mathit{cs})}$$

(Rebuild) $$\frac{\nexists \alpha : \mathit{foldable}(g, \beta, \alpha) \quad c' \in \mathit{rebuildings}(c)}{g \to \mathit{rebuild}(g, \beta, c')}$$

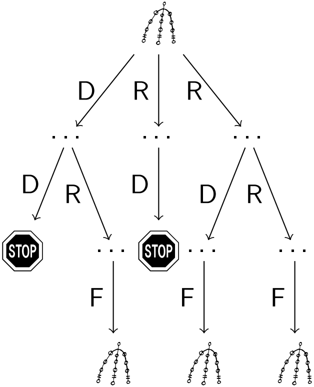
$\mathit{dangerous}(g, \beta)$ reappears in (Drive), but not in (Rebuild)!

(Drive) and (Rebuild) are not mutually exclusive.

(Drive) is not always applicable.
$\neg \mathit{dangerous}(g, \beta)$ ensures termination...

MRSC: Multi-result SC (a multi-valued function)

A (desirably) finite tree of graphs:



Outline

MRSC: Finiteness of trees of completed graphs

Theorem (Finiteness of sets of completed graphs)

If

1. *any infinite branch in a graph of configurations is detected by the predicate dangerous,*
2. *for any configuration c the set $rebuildings(c)$ is finite,*
3. *the number of successive rebuildings cannot be infinite (i.e. the chain $c_1, c_2, c_3, \ldots$, where $c_{k+1} \in rebuildings(c_k)$ is always finite),*

then the application of the MRSC-rules produces a finite set of completed graphs of configurations.

Proof.

Collapse all successive rebuildings into one rebuilding. Everything else follows from König lemma (using arguments similar to those in the Sørensen's proof. □

MRSC: Decoupling whistle and generalization

(Fold) $$\frac{\exists \alpha : \textit{foldable}(g,\, \beta,\, \alpha)}{g \to \textit{fold}(g,\, \beta,\, \alpha)}$$

(Drive) $$\frac{\nexists \alpha : \textit{foldable}(g,\, \beta,\, \alpha) \quad \neg \textit{dangerous}(g,\, \beta) \quad \textit{cs} = \textit{driveStep}(c)}{g \to \textit{addChildren}(g,\, \beta,\, \textit{cs})}$$

(Rebuild) $$\frac{\nexists \alpha : \textit{foldable}(g,\, \beta,\, \alpha) \quad c' \in \textit{rebuildings}(c)}{g \to \textit{rebuild}(g,\, \beta,\, c')}$$

Observation

$\textit{dangerous}(g,\, \beta)$ does not appear in (Rebuild).

Conclusion

The whistle does not have to know anything about rebuilding (generalization).

Outline

1. Different types of supercompilation
   - SC: Deterministic/traditional SC (a function)
   - NDSC: Non-deterministic SC (a relation)
   - MRSC: Multi-result SC (a multi-valued function)

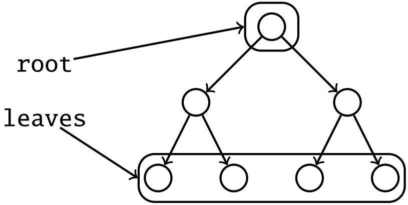
2. Nice features of multi-result supercompilation
   - Finiteness of trees of completed graphs
   - Decoupling whistle and generalization

3. The core of the MRSC Toolkit
   - Two representations for graphs of configurations
   - Operations on S-graphs

4. Conclusions

T-representation (= traditional, tree-based)

A T-graph:



- Good for top-down traversal of graphs.
- Convenient when transforming a graph into a residual program.
- However, when making additions to a T-graph in two different ways, we have to do some copying.
- However, a deterministic supercompiler deals with a single graph! Hence, no copying is required. . .

Why T-representation is not good for MRSC?

A problem

- MRSC is able to produce millions of graphs of configurations.
- Huge memory consumption, a lot of copying...

A simple solution

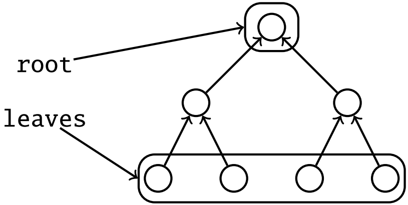
Sharing!

A sophisticated solution (Sergei Grechanik)

Hypergraphs, hyperedges.

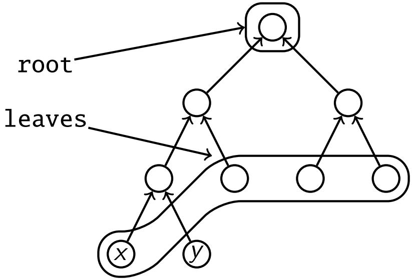
S-representation (= based on spaghetti-stacks)

An S-graph:

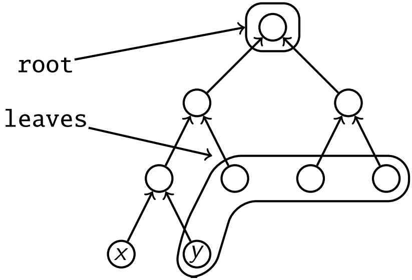


- Good, when making additions to an S-graph in two different ways, as no copying is required.
- Convenient for a multi-result supercompiler dealing with large collections of graph!

Reuse of nodes in S-graphs



(a) After adding x.



(b) After adding y

- S-graphs are immutable!
- "Modifying" an S-graph in different ways we create new S-graphs.
- The original S-graphs and derived S-graphs share common parts.

An implementation in Scala: T-graphs

```scala
type TPath = List[Int]

case class TNode[C, D](
  conf: C, outs: List[TEdge[C, D]],
  base: Option[TPath], tPath: TPath)

case class TEdge[C, D](
  node: TNode[C, D], driveInfo: D)

case class TGraph[C, D](
  root: TNode[C, D], leaves: List[TNode[C, D]])
```

- C is the type of configurations.
- D is the type of edge labels, produced by driving.
- TPath is the type of paths to nodes.

An implementation in Scala: S-graphs

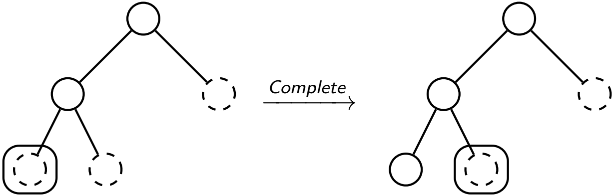
```scala
type SPath = List[Int]

case class SNode[C, D](
  conf: C, in: SEdge[C, D],
  base: Option[SPath], sPath: SPath)

case class SEdge[C, D](
  node: SNode[C, D], driveInfo: D)

case class SGraph[C, D](
  incompleteLeaves: List[SNode[C, D]],
  completeLeaves: List[SNode[C, D]],
  completeNodes: List[SNode[C, D]]) {

  val isComplete = incompleteLeaves.isEmpty
  val current = if (isComplete) null else incompleteLeaves.head
}
```

Rewrite steps for S-graphs

```
sealed trait GraphRewriteStep[C, D]

case class CompleteCurrentNodeStep[C, D]
  extends GraphRewriteStep[C, D]

case class AddChildNodesStep[C, D](ns: List[(C, D)])
  extends GraphRewriteStep[C, D]

case class FoldStep[C, D](to: SPath)
  extends GraphRewriteStep[C, D]

case class RebuildStep[C, D](c: C)
  extends GraphRewriteStep[C, D]
```

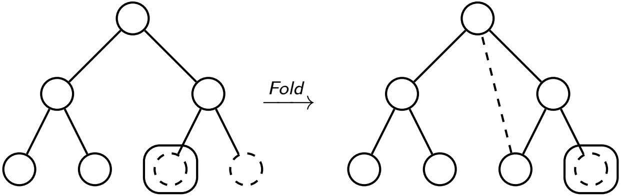
These rewriting operations form a “basis” sufficient for building S-graphs during multi-result supercompilation. (Unlike deterministic supercompiation, there are no roll-backs!)

Rewrite steps for S-graphs: Complete



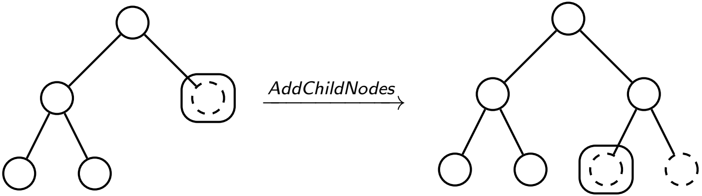
- CompleteCurrentNodeStep — marks the current leaf as a completed one. Used in driving.

Rewrite steps for S-graphs: Fold



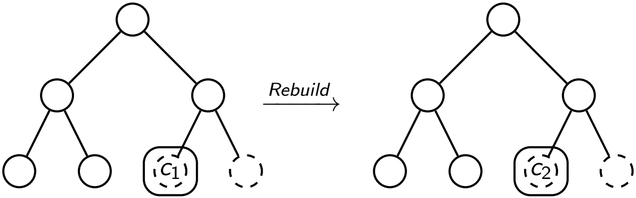
- FoldStep — performs a folding.

Rewrite steps for S-graphs: AddChildNodes



- AddChildNodesStep — adds child nodes to the current node. Used in driving.

Rewrite steps for S-graphs: Rebuild



- RebuildStep — performs a lower rebuilding of the graph (by replacing the configuration in the current node).

MRSC "middleware" for supercompiler construction

```
trait GraphRewriteRules[C, D] {
  type N = SNode[C, D]
  type G = SGraph[C, D]
  type S = GraphRewriteStep[C, D]
  def steps(g: G): List[S]
}

case class GraphGenerator[C, D]
  (rules: GraphRewriteRules[C, D], conf: C)
  extends Iterator[SGraph[C, D]] { ... }
```

- A concrete supercompiler is required to provide an implementation for the method `steps`.
- `steps` does not rewrite graphs: it only generates "commands" to be executed by the MRSC Toolkit.
- The main loop of supercompilation is implemented as an iterator that produces graphs in a lazy way, by demand.

The MRSC Toolkit: publications

- Ilya Klyuchnikov and Sergei Romanenko. Multi-Result Supercompilation as Branching Growth of the Penultimate Level in Metasystem Transitions. *Ershov Informatics Conference 2011*. (Revised version is in *LNCS 7162, pp. 210—226, 2012*).
- Ilya Klyuchnikov and Sergei Romanenko. MRSC: a toolkit for building multi-result supercompilers. *Preprint 77. Keldysh Institute of Applied Mathematics, Moscow*. 2011. `http://library.keldysh.ru/preprint.asp?lg=e&id=2011-77`
- Andrei V. Klimov, Ilya G. Klyuchnikov, Sergei A. Romanenko. Automatic verification of counter systems via domain-specific multi-result supercompilation. *Preprint 19. Keldysh Institute of Applied Mathematics, Moscow*. 2012 `http://library.keldysh.ru/preprint.asp?lg=e&id=2012-19`
- Andrei V. Klimov, Ilya G. Klyuchnikov, Sergei A. Romanenko Implementing a domain-specific multi-result supercompiler by means of the MRSC toolkit. *Preprint 24. Keldysh Institute of Applied Mathematics, Moscow*. 2012. `http://library.keldysh.ru/preprint.asp?lg=e&id=2012-24`

The MRSC Toolkit: a public repository at GitHub

`https://github.com/ilya-klyuchnikov/mrsc`

There one can find:

- The Core of the MRSC Toolkit.
- A domain-specific supercompiler for counter systems.
- The results of verification of a number of communication protocols.
- PFP: a toolkit for implementing multi-result supercompilers for functional languages.
- ...

Outline

1. Different types of supercompilation
   - SC: Deterministic/traditional SC (a function)
   - NDSC: Non-deterministic SC (a relation)
   - MRSC: Multi-result SC (a multi-valued function)
2. Nice features of multi-result supercompilation
   - Finiteness of trees of completed graphs
   - Decoupling whistle and generalization
3. The core of the MRSC Toolkit
   - Two representations for graphs of configurations
   - Operations on S-graphs
4. Conclusions

Conclusions

- 3 kinds of supercompilation (deterministic, non-deterministic and multi-result one) can be specified in a uniform way by graph rewriting rules.
- Under certain conditions, a multi-result supercompiler produces a finite number of residual programs and terminates.
- Conceptually, multi-result supercompilation is simpler than deterministic, single-result supercompilation, since the whistle and the generalization algorithm can be completely decoupled.
- The use of immutable data-structures (S-graphs) and data sharing in the implementation of multi-result supercompilation, makes it possible to generate thousands of graphs, while still keeping memory consumption within reasonable limits.