Generalising Tree Traversals to DAGs

Exploiting Sharing without the Pain

Patrick Bahr[1]   Emil Axelsson[2]

[1]University of Copenhagen
paba@diku.dk

[2]Chalmers University of Technology
emax@chalmers.se

PEPM 2015

Motivation

Goal

Do stuff on acyclic graphs, but pretend they are only trees.

Motivation

Goal

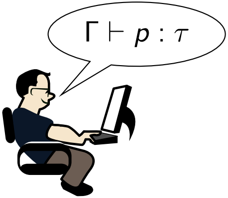
Do stuff on acyclic graphs, but pretend they are only trees.

Primary Application

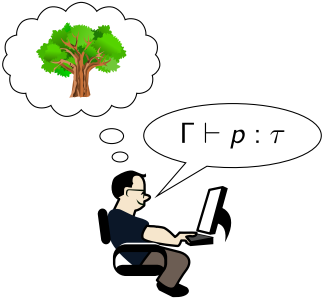
Abstract Syntax Graphs/Trees:

- type inference
- program analyses
- program transformations
- ...

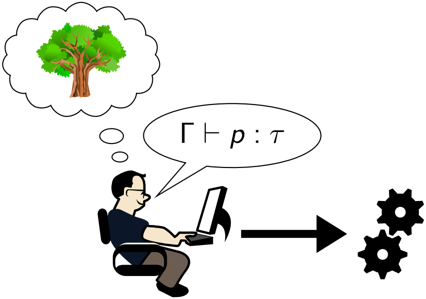
The Idea



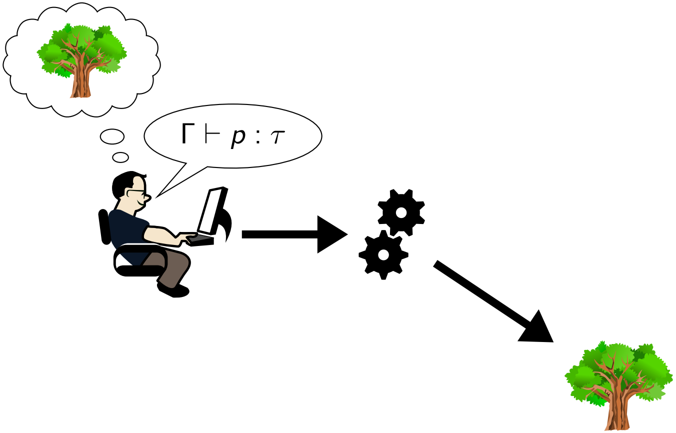
The Idea



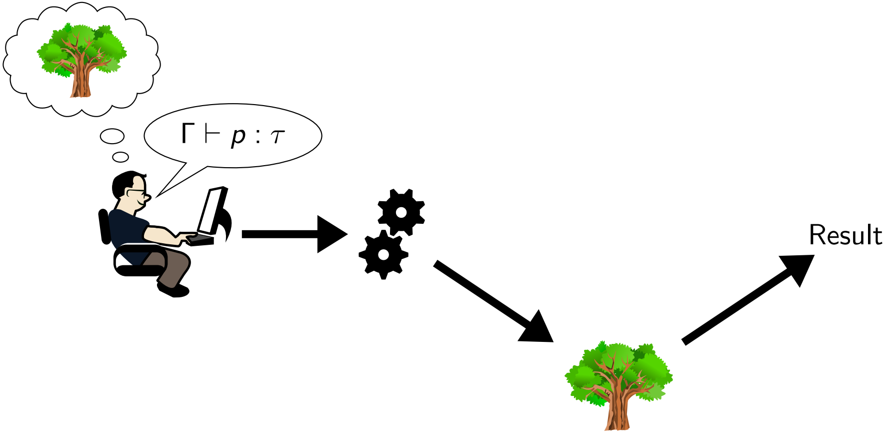
The Idea



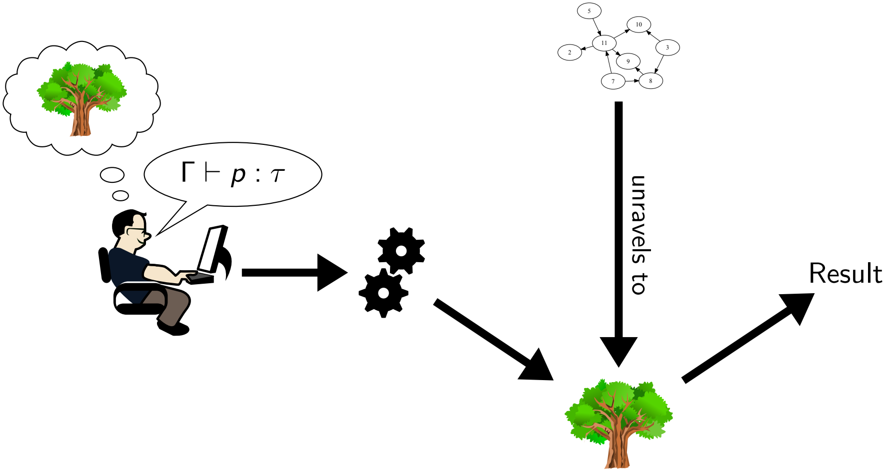
The Idea



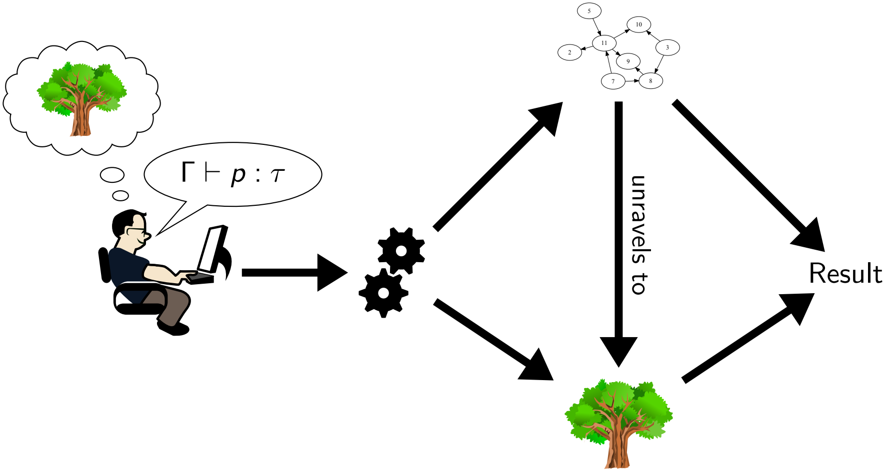
The Idea



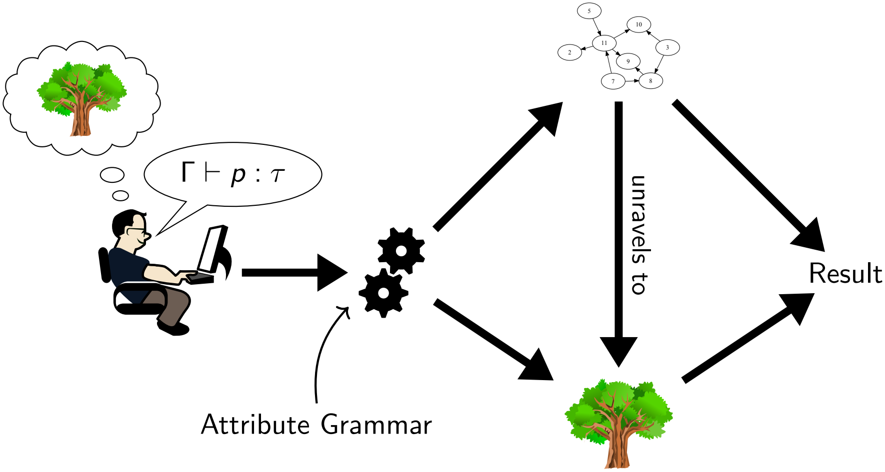
The Idea



The Idea

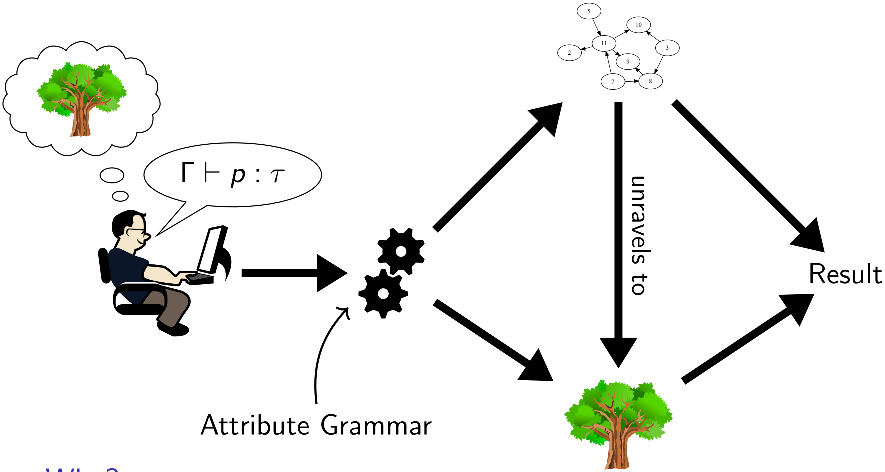


The Idea



$\Gamma \vdash p : \tau$
5
10
11
2
3
9
7
8
unravels to
Result
Attribute Grammar

The Idea



Why?

- It's more difficult to get a traversal on graphs right.
- But: it's more efficient to traverse the graph.

What's the Catch?

What's the Catch?

It doesn't work

What's the Catch?

It doesn't work in general.

What's the Catch?

It doesn't work in general.

But: it does work for many cases.

What's the Catch?

It doesn't work in general.

But: it does work for many cases.

Our Contribution

- Identify classes of AGs for which this approach works.
- Prototype implementation in Haskell.
- Case studies and benchmarks.

A Toy Example

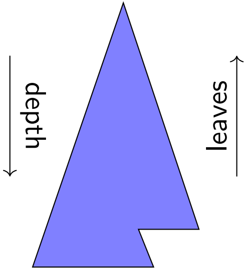
$$\begin{aligned}
\textbf{data}\ \mathit{IntTree} &= \mathit{Leaf}\ \mathit{Int} \\
&\mid \mathit{Node}\ \mathit{IntTree}\ \mathit{IntTree}
\end{aligned}$$

$$\begin{aligned}
&\mathit{leavesBelow} :: \mathit{Int} \to \mathit{IntTree} \to \mathit{Set}\ \mathit{Int} \\
&\mathit{leavesBelow}\ d\ (\mathit{Leaf}\ i) \\
&\quad \mid d \leqslant 0 \qquad\ = \mathit{Set.singleton}\ i \\
&\quad \mid \mathit{otherwise} = \mathit{Set.empty} \\
&\mathit{leavesBelow}\ d\ (\mathit{Node}\ t_1\ t_2) = \\
&\qquad\quad\ \ \mathit{leavesBelow}\ (d-1)\ t_1 \\
&\qquad \cup\ \ \mathit{leavesBelow}\ (d-1)\ t_2
\end{aligned}$$

A Toy Example

$$
\begin{aligned}
\textbf{data}\ \mathit{IntTree} &= \mathit{Leaf}\ \mathit{Int} \\
&\mid \mathit{Node}\ \mathit{IntTree}\ \mathit{IntTree}
\end{aligned}
$$

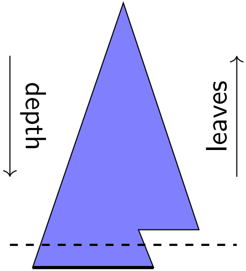
$$
\begin{aligned}
&\mathit{leavesBelow} :: \mathit{Int} \to \mathit{IntTree} \to \mathit{Set}\ \mathit{Int} \\
&\mathit{leavesBelow}\ d\ (\mathit{Leaf}\ i) \\
&\quad \mid d \leqslant 0 \qquad\ \ = \mathit{Set.singleton}\ i \\
&\quad \mid \mathit{otherwise} = \mathit{Set.empty} \\
&\mathit{leavesBelow}\ d\ (\mathit{Node}\ t_1\ t_2) = \\
&\qquad\quad\ \mathit{leavesBelow}\ (d-1)\ t_1 \\
&\quad\ \cup\ \ \mathit{leavesBelow}\ (d-1)\ t_2
\end{aligned}
$$



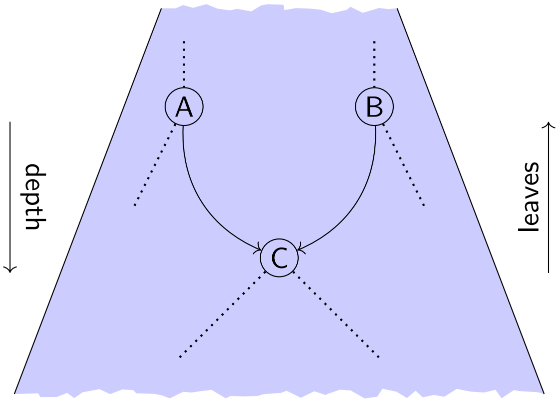
A Toy Example

$$
\begin{aligned}
&\textbf{data}\ \mathit{IntTree} = \mathit{Leaf}\ \mathit{Int}\\
&\qquad\qquad\quad\ \mid\ \mathit{Node}\ \mathit{IntTree}\ \mathit{IntTree}
\end{aligned}
$$

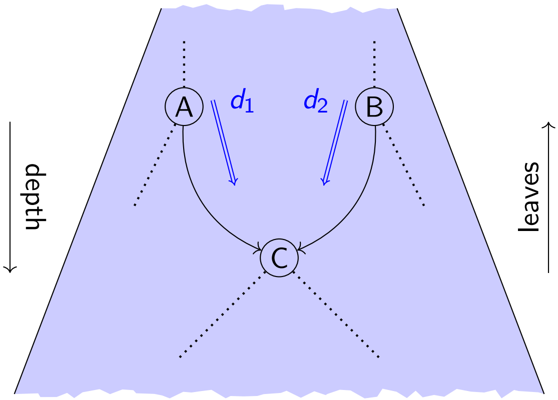
$$
\begin{aligned}
&\mathit{leavesBelow} :: \mathit{Int} \to \mathit{IntTree} \to \mathit{Set}\ \mathit{Int}\\
&\mathit{leavesBelow}\ d\ (\mathit{Leaf}\ i)\\
&\quad \mid d \leqslant 0 \qquad\quad = \mathit{Set.singleton}\ i\\
&\quad \mid \mathit{otherwise} = \mathit{Set.empty}\\
&\mathit{leavesBelow}\ d\ (\mathit{Node}\ t_1\ t_2) =\\
&\qquad\quad\ \ \mathit{leavesBelow}\ (d-1)\ t_1\\
&\qquad \cup\ \ \mathit{leavesBelow}\ (d-1)\ t_2
\end{aligned}
$$



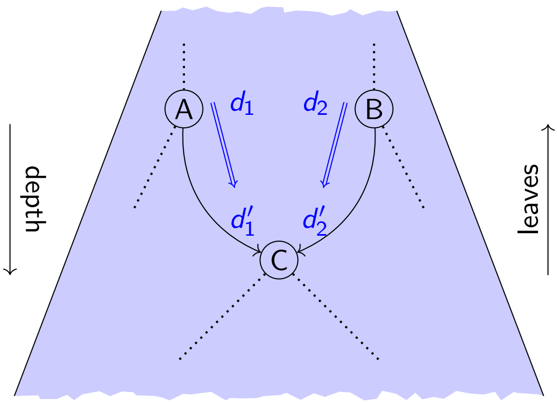
Traversal on Graphs



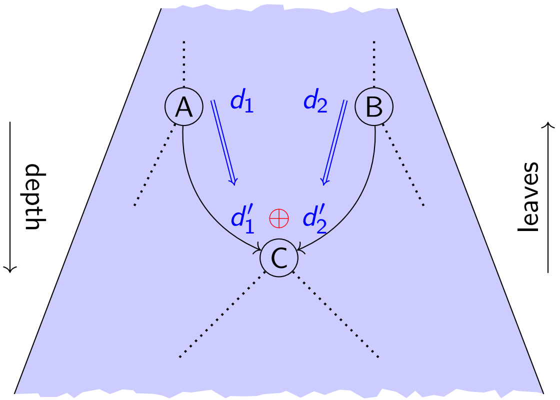
Traversal on Graphs



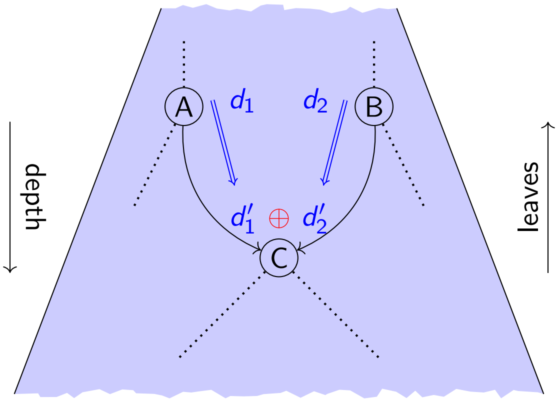
Traversal on Graphs



Traversal on Graphs



Traversal on Graphs



For which traversals is this correct?

But before that, let's implement it!

$$\begin{aligned}\textbf{data}\ \mathit{IntTree} \quad &= \mathit{Leaf}\ \mathit{Int} \\ &\mid\ \mathit{Node}\ \mathit{IntTree}\ \mathit{IntTree}\end{aligned}$$

But before that, let's implement it!

$$
\begin{aligned}
\textbf{data}\ \mathit{IntTree}\quad a &= \mathit{Leaf}\ \mathit{Int} \\
&\mid \mathit{Node}\ a\ a
\end{aligned}
$$

But before that, let's implement it!

$$
\begin{aligned}
\textbf{data}\ \mathit{IntTreeF}\ a = {} & \mathit{Leaf}\ \mathit{Int} \\
\mid {} & \mathit{Node}\ a\ a
\end{aligned}
$$

But before that, let's implement it!

$$\begin{aligned} \textbf{data}\ \mathit{IntTreeF}\ a &= \mathit{Leaf}\ \mathit{Int} \\ &\mid \mathit{Node}\ a\ a \end{aligned}$$

$$\begin{aligned} &\mathit{leavesBelow} :: \mathit{Int} \to \mathit{Tree}\ \mathit{IntTreeF} \to \mathit{Set}\ \mathit{Int} \\ &\mathit{leavesBelow} = \mathit{runAG}\ \mathit{leavesBelow}_S\ \mathit{leavesBelow}_I \end{aligned}$$

But before that, let's implement it!

$$\textbf{data}\ \mathit{IntTreeF}\ a = \mathit{Leaf}\ \mathit{Int} \mid \mathit{Node}\ a\ a$$

$$\begin{aligned}&\mathit{leavesBelow} :: \mathit{Int} \to \mathit{Tree}\ \mathit{IntTreeF} \to \mathit{Set}\ \mathit{Int}\\&\mathit{leavesBelow} = \mathit{runAG}\ \mathit{leavesBelow}_S\ \mathit{leavesBelow}_I\end{aligned}$$

$$\begin{aligned}&\mathit{leavesBelow}_G :: \mathit{Int} \to \mathit{Dag}\ \mathit{IntTreeF} \to \mathit{Set}\ \mathit{Int}\\&\mathit{leavesBelow}_G = \mathit{runAGDag}\ \mathit{min}\ \mathit{leavesBelow}_S\ \mathit{leavesBelow}_I\end{aligned}$$

$\oplus$

Implementing the semantic functions

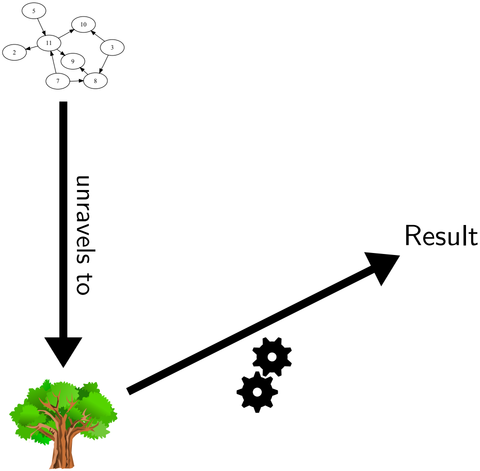
$$
\begin{aligned}
&\mathit{leavesBelow}_{\mathbf{I}} :: \mathit{Inh}\ \mathit{IntTreeF}\ \mathit{atts}\ \mathit{Int} \\
&\mathit{leavesBelow}_{\mathbf{I}}\ (\mathit{Leaf}\ i) \qquad\quad = \emptyset \\
&\mathit{leavesBelow}_{\mathbf{I}}\ (\mathit{Node}\ t_1\ t_2) = t_1 \mapsto d\ \&\ t_2 \mapsto d \\
&\quad \textbf{where}\ d = \mathit{above} - 1
\end{aligned}
$$

Implementing the semantic functions

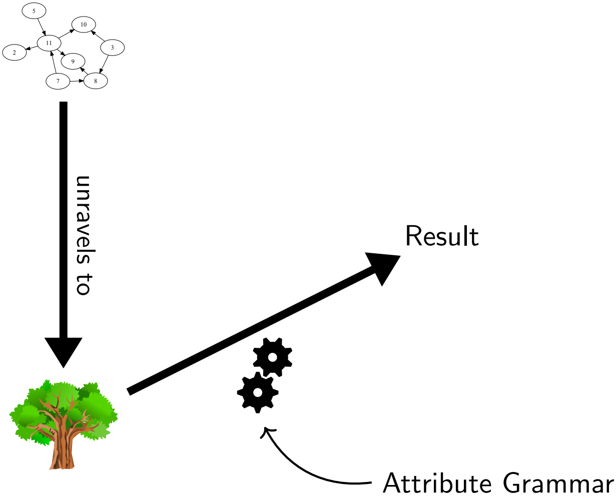
$$\begin{aligned}
&\mathit{leavesBelow}_I :: \mathit{Inh}\ \mathit{IntTreeF}\ \mathit{atts}\ \mathit{Int}\\
&\mathit{leavesBelow}_I\ (\mathit{Leaf}\ i) \qquad\ = \emptyset\\
&\mathit{leavesBelow}_I\ (\mathit{Node}\ t_1\ t_2) = t_1 \mapsto d\ \&\ t_2 \mapsto d\\
&\quad \mathbf{where}\ d = \mathit{above} - 1
\end{aligned}$$

$$\begin{aligned}
&\mathit{leavesBelow}_S :: (\mathit{Int} \in \mathit{atts}) \Rightarrow \mathit{Syn}\ \mathit{IntTreeF}\ \mathit{atts}\ (\mathit{Set}\ \mathit{Int})\\
&\mathit{leavesBelow}_S\ (\mathit{Leaf}\ i)\\
&\quad \mid (\mathit{above} :: \mathit{Int}) \leqslant 0 \qquad = \mathit{Set}.\mathit{singleton}\ i\\
&\quad \mid \mathit{otherwise} \qquad\qquad\quad\ = \mathit{Set}.\mathit{empty}\\
&\mathit{leavesBelow}_S\ (\mathit{Node}\ t_1\ t_2) = \mathit{below}\ t_1 \cup \mathit{below}\ t_2
\end{aligned}$$

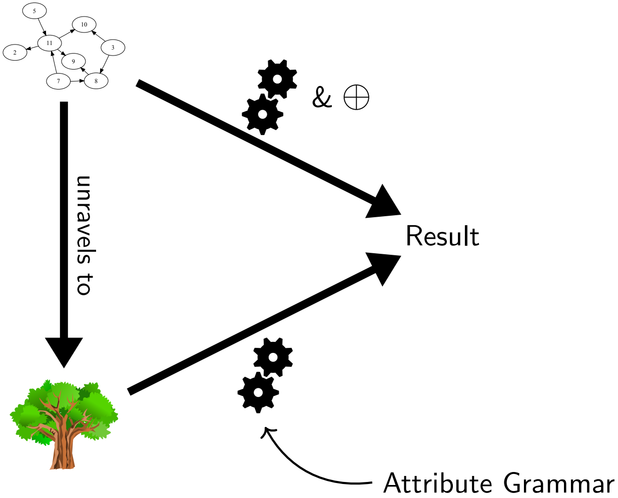
Correctness



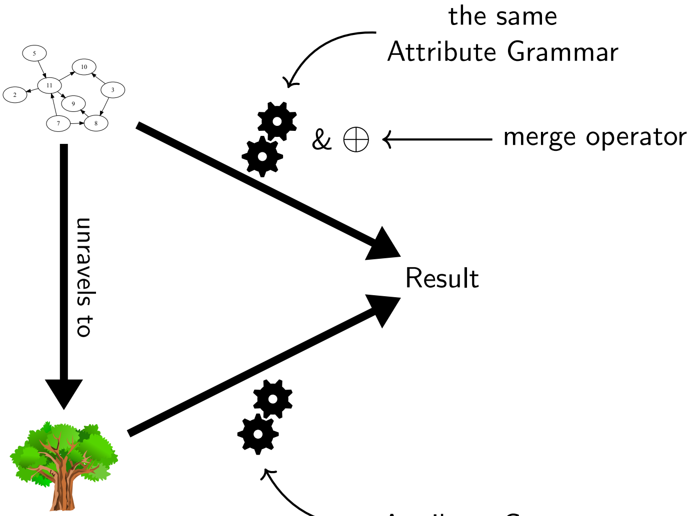
Correctness



Correctness



Correctness



Correspondence Theorems

Theorem (Monotone AGs)

Let

(1) G *be a non-circular AG,*

(2) $\oplus$ *an assoc., comm. operator on inherited attributes, and*

(3) $\lesssim$ *such that* G *is monotone and* $\oplus$ *is decreasing w.r.t.* $\lesssim$*.*

If $(G, \oplus)$ *terminates on a DAG* g *with result* r*,*
then G *terminates on* $\mathcal{U}(g)$ *with result* r' *such that* $r \lesssim r'$*.*

Correspondence Theorems

Theorem (Monotone AGs)

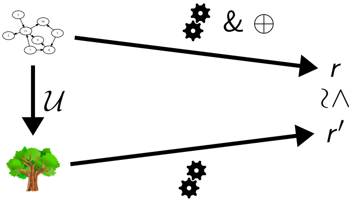
Let

(1) G *be a non-circular AG,*

(2) $\oplus$ *an assoc., comm. operator on inherited attributes, and*

(3) $\lesssim$ *such that* G *is monotone and* $\oplus$ *is decreasing w.r.t.* $\lesssim$.

If $(G, \oplus)$ *terminates on a DAG* g *with result* r,
then G *terminates on* $\mathcal{U}(g)$ *with result* r' *such that* $r \lesssim r'$.



Correspondence Theorems

Theorem (Monotone AGs)

Let

(1) *G be a non-circular AG,*

(2) *$\oplus$ an assoc., comm. operator on inherited attributes, and*

(3) *$\lesssim$ such that G is monotone and $\oplus$ is decreasing w.r.t. $\lesssim$.*

If $(G, \oplus)$ terminates on a DAG g with result r,
then G terminates on $\mathcal{U}(g)$ with result r' such that $r \lesssim r'$.

Example

For the *leavesBelow* AG, define $\lesssim$ as follows:

- on *Int*: $x \lesssim y \iff x \leq y$
- on *Set Int*: $S \lesssim T \iff S \supseteq T$

Correspondence Theorems

Theorem (Monotone AGs)

Let

(1) *G be a non-circular AG,*

(2) *$\oplus$ an assoc., comm. operator on inherited attributes, and*

(3) *$\lesssim$ such that G is monotone and $\oplus$ is decreasing w.r.t. $\lesssim$.*

If $(G, \oplus)$ terminates on a DAG g with result r,
then G terminates on $\mathcal{U}(g)$ with result r' such that $r \lesssim r'$.

Example

For the *leavesBelow* AG, define $\lesssim$ as follows:

- on *Int*: $x \lesssim y \iff x \leq y$
- on *Set Int*: $S \lesssim T \iff S \supseteq T$

$$\implies \quad \mathit{leavesBelow}_G\ d\ g \supseteq \mathit{leavesBelow}\ d\ (\mathcal{U}(g))$$

Correspondence Theorems

Theorem (Monotone AGs)

Let

(1) *G be a non-circular AG,*

(2) *$\oplus$ an assoc., comm. operator on inherited attributes, and*

(3) *$\lesssim$ such that G is monotone and $\oplus$ is decreasing w.r.t. $\lesssim$.*

If $(G, \oplus)$ terminates on a DAG g with result r,
then G terminates on $\mathcal{U}(g)$ with result r' such that $r \lesssim r'$.

Example

For the *leavesBelow* AG, define $\lesssim$ as follows:

- on *Int*: $x \lesssim y \iff x \leq y$
- on *Set Int*: $S \lesssim T \iff S \supseteq T$

$$\implies \quad leavesBelow_G \; d \; g \supseteq leavesBelow \; d \; (\mathcal{U}(g))$$

for $leavesBelow_G \; d \; g \subseteq leavesBelow \; d \; (\mathcal{U}(g))$ see paper

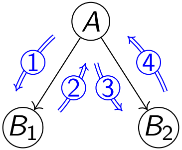
Termination

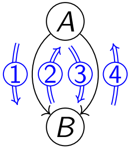
- We know: non-circular AGs terminate on any tree.
- But: non-circular AGs may diverge on DAGs.

Termination

- We know: non-circular AGs terminate on any tree.
- But: non-circular AGs may diverge on DAGs.

Example

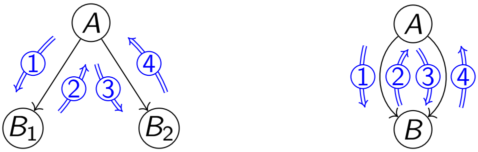




Termination

- We know: non-circular AGs terminate on any tree.
- But: non-circular AGs may diverge on DAGs.

Example



Theorem (termination)

Let G, $\oplus$, and $\lesssim$ be as before.

If $\lesssim$ is well-founded on inherited attributes, then $(G, \oplus)$ terminates on any DAG.

Correspondence Theorem for Copying AGs

Copying AGs

- inherited attributes are just propagated, not changed
- Example: Bird's repmin problem.

Correspondence Theorem for Copying AGs

Copying AGs

- inherited attributes are just propagated, not changed
- Example: Bird's repmin problem.

Theorem (copying AGs)

Let

(1) *G be a copying, non-circular AG, and*

(2) *$x \oplus y \in \{x, y\}$ for all x, y.*

Then

(i) *$(G, \oplus)$ terminates on any DAG, and*

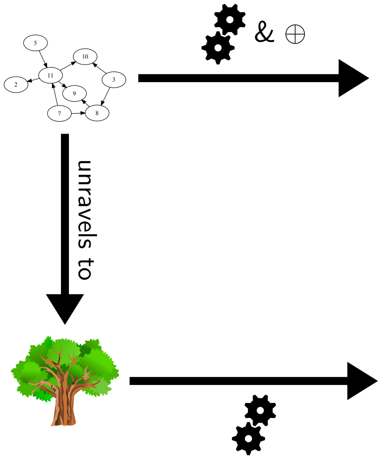
(ii) *$(\!|G, \oplus|\!)(g) = [\![G]\!](\mathcal{U}(g))$.*

Graph Transformations

- Our framework generalises to tree/DAG-transformations
- Idea: attributes may contain trees/DAGs.

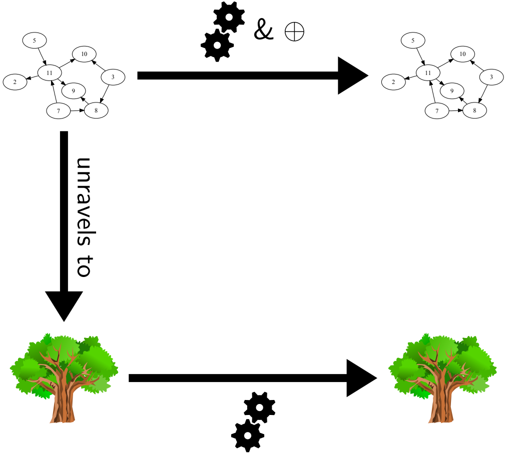
Graph Transformations

- Our framework generalises to tree/DAG-transformations
- Idea: attributes may contain trees/DAGs.



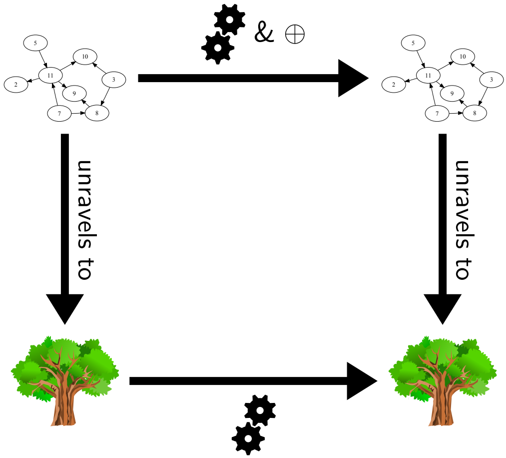
Graph Transformations

- Our framework generalises to tree/DAG-transformations
- Idea: attributes may contain trees/DAGs.



Graph Transformations

- Our framework generalises to tree/DAG-transformations
- Idea: attributes may contain trees/DAGs.



Example: Bird's Repmin Problem

$\textbf{newtype}\ Min_S = Min_S\ Int;$

$\textbf{newtype}\ Min_I = Min_I\ Int$

$$
\begin{aligned}
&min_S :: Syn\ IntTreeF\ atts\ Min_S\\
&min_S\ (Leaf\ i) \quad\ = Min_S\ i\\
&min_S\ (Node\ a\ b) = min\ (below\ a)\ (below\ b)
\end{aligned}
$$

$$
\begin{aligned}
&min_I :: Inh\ IntTreeF\ atts\ Min_I\\
&min_I\ \_ = \emptyset
\end{aligned}
$$

$$
\begin{aligned}
&rep :: (Min_I \in atts) \Rightarrow Rewrite\ IntTreeF\ atts\ IntTreeF\\
&rep\ (Leaf\ i) \quad\ = \textbf{let}\ Min_I\ i' = above\\
&\qquad\qquad\qquad\quad\ \textbf{in}\ Leaf\ i'\\
&rep\ (Node\ a\ b) = Node\ a\ b
\end{aligned}
$$

Example: Bird's Repmin Problem

$\textbf{newtype}\ Min_S = Min_S\ Int;$

$\textbf{newtype}\ Min_I = Min_I\ Int$

$$
\begin{aligned}
&min_S :: Syn\ IntTreeF\ atts\ Min_S \\
&min_S\ (Leaf\ i) \quad\ \ = Min_S\ i \\
&min_S\ (Node\ a\ b) = min\ (below\ a)\ (below\ b)
\end{aligned}
$$

$$
\begin{aligned}
&min_I :: Inh\ IntTreeF\ atts\ Min_I \\
&min_I\ \_ = \emptyset
\end{aligned}
$$

$$
\begin{aligned}
&rep :: (Min_I \in atts) \Rightarrow Rewrite\ IntTreeF\ atts\ IntTreeF \\
&rep\ (Leaf\ i) \quad\ \ = \textbf{let}\ Min_I\ i' = above \\
&\qquad\qquad\qquad\ \ \ \textbf{in}\ Leaf\ i' \\
&rep\ (Node\ a\ b) = Node\ a\ b
\end{aligned}
$$

$$
\begin{aligned}
&repmin :: Tree\ IntTreeF \to Tree\ IntTreeF \\
&repmin = runRewrite\ min_S\ min_I\ rep\ init \\
&\quad \textbf{where}\ init\ (Min_S\ i) = Min_I\ i
\end{aligned}
$$

Example: Bird's Repmin Problem

$\textbf{newtype}\ \mathit{Min}_S = \mathit{Min}_S\ \mathit{Int};$ $\qquad \textbf{newtype}\ \mathit{Min}_I = \mathit{Min}_I\ \mathit{Int}$

$$
\begin{aligned}
&\mathit{min}_S :: \mathit{Syn}\ \mathit{IntTreeF}\ \mathit{atts}\ \mathit{Min}_S \\
&\mathit{min}_S\ (\mathit{Leaf}\ i) \quad = \mathit{Min}_S\ i \\
&\mathit{min}_S\ (\mathit{Node}\ a\ b) = \mathit{min}\ (\mathit{below}\ a)\ (\mathit{below}\ b)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{min}_I :: \mathit{Inh}\ \mathit{IntTreeF}\ \mathit{atts}\ \mathit{Min}_I \\
&\mathit{min}_I\ \_ = \emptyset
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{rep} :: (\mathit{Min}_I \in \mathit{atts}) \Rightarrow \mathit{Rewrite}\ \mathit{IntTreeF}\ \mathit{atts}\ \mathit{IntTreeF} \\
&\mathit{rep}\ (\mathit{Leaf}\ i) \quad = \textbf{let}\ \mathit{Min}_I\ i' = \mathit{above} \\
&\qquad\qquad\qquad\ \ \textbf{in}\ \mathit{Leaf}\ i' \\
&\mathit{rep}\ (\mathit{Node}\ a\ b) = \mathit{Node}\ a\ b
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{repmin} :: \mathit{Tree}\ \mathit{IntTreeF} \rightarrow \mathit{Tree}\ \mathit{IntTreeF} \\
&\mathit{repmin} = \mathit{runRewrite}\ \mathit{min}_S\ \mathit{min}_I\ \mathit{rep}\ \mathit{init} \\
&\quad \textbf{where}\ \mathit{init}\ (\mathit{Min}_S\ i) = \mathit{Min}_I\ i
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{repmin}_G :: \mathit{Dag}\ \mathit{IntTreeF} \rightarrow \mathit{Dag}\ \mathit{IntTreeF} \\
&\mathit{repmin}_G = \mathit{runRewriteDag}\ \mathit{const}\ \mathit{min}_S\ \mathit{min}_I\ \mathit{rep}\ \mathit{init} \\
&\quad \textbf{where}\ \mathit{init}\ (\mathit{Min}_S\ i) = \mathit{Min}_I\ i
\end{aligned}
$$

Summary

Our Contributions

- Haskell library to run AGs on DAGs
- Correspondence & termination theorems to prove correctness

Summary

Our Contributions

- Haskell library to run AGs on DAGs
- Correspondence & termination theorems to prove correctness

More in the paper

- Examples: type inference; circuits
- full theory & proofs
- parametric AGs ($\rightarrow$ tech report)
- Benchmarks ($\rightarrow$ tech report)

Conclusion

Future and Ongoing Work

- AGs with fixpoint iteration $\rightsquigarrow$ cyclic graphs
- mutually recursive data types and GADTs
- deep pattern matching in AGs
- corresponding notion of non-circularity for AGs on DAGs

Conclusion

Future and Ongoing Work

- AGs with fixpoint iteration $\rightsquigarrow$ cyclic graphs
- mutually recursive data types and GADTs
- deep pattern matching in AGs
- corresponding notion of non-circularity for AGs on DAGs

Implementation

Available from `http://j.mp/AG-DAG`.

- Haskell library source code
- more examples
- benchmarks

Conclusion

Future and Ongoing Work

- AGs with fixpoint iteration $\rightsquigarrow$ cyclic graphs
- mutually recursive data types and GADTs
- deep pattern matching in AGs
- corresponding notion of non-circularity for AGs on DAGs

Implementation

Available from `http://j.mp/AG-DAG`.

- Haskell library source code
- more examples
- benchmarks

Try the compositional datatypes library

```
> cabal install compdata-dags
```

Generalising Tree Traversals to DAGs

Exploiting Sharing without the Pain

Patrick Bahr[1] Emil Axelsson[2]

[1]University of Copenhagen
`paba@diku.dk`

[2]Chalmers University of Technology
`emax@chalmers.se`

Source Code Repository

`http://j.mp/AG-DAG`

Haskell Library

```
> cabal install compdata-dags
```