Rule-Based Graph Programming: Linear-Time Graph Algorithms in GP 2

Graham Campbell Brian Courtehoute Detlef Plump

Department of Computer Science
University of York

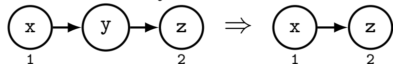
BCTCS, April 2019

Introduction

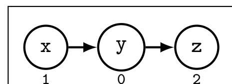
The graph programming language GP 2 is

- a high-level language
- rule-based for easier reasoning
- non-deterministic
- computationally complete
- equipped with a compiler generating efficient C code

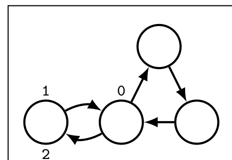
Rule Application in GP 2



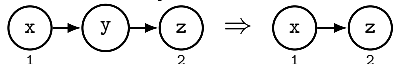
uniqueness of output guaranteed by this double-pushout structure:



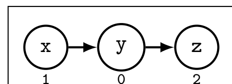
injective



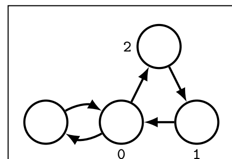
Rule Application in GP 2



uniqueness of output guaranteed by this double-pushout structure:

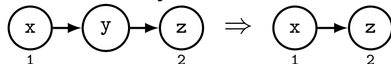


injective dangling condition



Rule Application in GP 2

```
a_rule (x,y,z:list)
```



uniqueness of output guaranteed by this double-pushout structure:

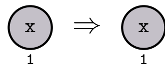
The GP 2 Program is-tree

Proposition (Correctness)

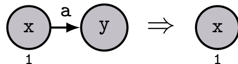
The program fails iff the input is not a non-empty tree.

```
Main = not_empty; prune!; if Check then fail
Check = {two_nodes, has_loop}
```

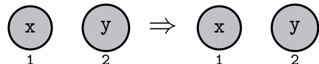
`not_empty(a,x,y:list)`



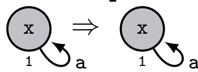
`prune(a,x,y:list)`



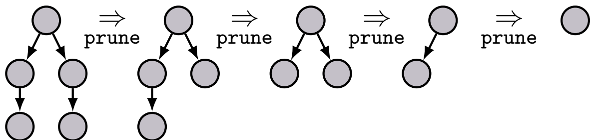
`two_nodes(x,y:list)`

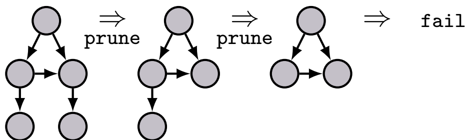


`has_loop(a,x:list)`



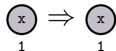
Execution Examples of is-tree



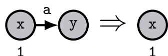



```
Main = not_empty; prune!; if Check then fail
Check = {two_nodes, has_loop}
```

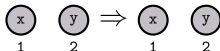
not_empty(a,x,y:list)



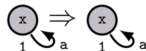
prune(a,x,y:list)



two_nodes(x,y:list)



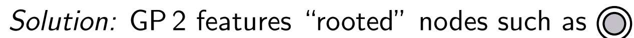
has_loop(a,x:list)



Time Complexity

For an input graph G with n nodes,

- `prune!` terminates after at most n applications
- `prune` is applied in $O(n)$ time due to matching
- `is-tree` requires $\mathrm{O}(n^2)$ time

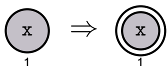
Solution: GP 2 features “rooted” nodes such as 

A rooted node can be accessed in constant time.

The GP 2 Program `is-tree-rooted`

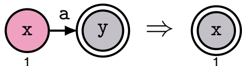
```
Main = init; Reduce!; if Check then fail
Reduce = {prune, push}
Check = {two_nodes, has_loop}
```

`init(x:list)`

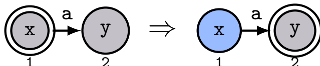


Pink denotes the "any" mark. It can match any colour.

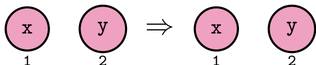
`prune(a,x,y:list)`



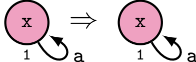
`push(a,x,y:list)`



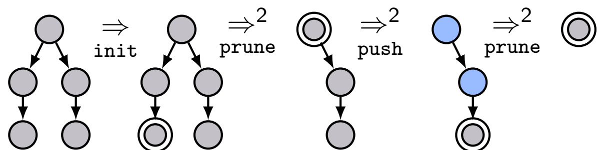
`two_nodes(x,y:list)`

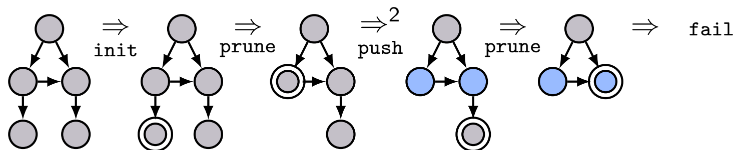


`has_loop(a,x:list)`



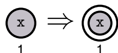
Execution Examples of `is-tree-rooted`



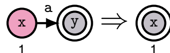



```
Main = init; {prune, push}!; if {two_nodes, has_loop} then fail
```

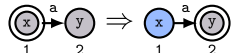
init(x:list)



prune(a,x,y:list)



push(a,x,y:list)



Complexity

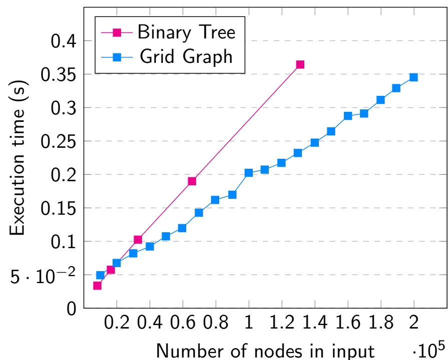
A rule $L \Rightarrow R$ with condition c is *fast* if

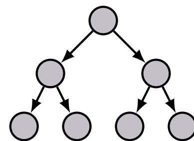
- Every component of L has a root.
- Neither L nor R contain repeated list, atom or string variables.
- The condition c contains neither the edge predicate, nor a test e_1=e_2 or e_1!=e_2 where both e_1 and e_2 contain a list, string or atom variable.

Theorem (Complexity of Matching Fast Rules, Bak-Plump 2012)

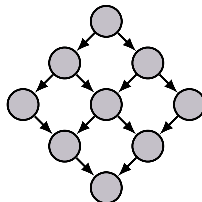
Rooted graph matching can be implemented to run in constant time for fast rules, provided there are upper bounds on the maximal node degree and the number of roots in host graphs.

Performance of `is-tree-rooted`





A complete binary tree



A grid graph

Topological Sorting

A *topological sorting* is a linear order $\leq$ (antisymmetric, transitive, connex binary relation) such that for each edge from u to v, $u \leq v$.

Idea of the program:

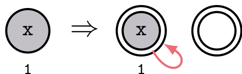
- Encoding the topological sorting as a stack of nodes
- Using depth-first-search (DFS) for graph traversal
- Pushing nodes onto the stack during the back step of a directed DFS
- If the program gets stuck, using an undirected DFS to find an unsorted node

The GP 2 Program top-sort

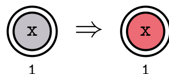
```
Main = init; SearchUnsortedNodes

SearchUnsortedNodes = ((try unsorted then SortNodes;
search_forward)!; try search_back else break)!
```

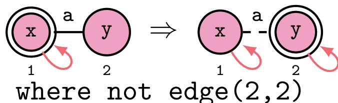
init (x:list)



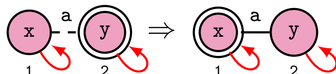
unsorted (x:list)



search_forward (a,x,y:list)



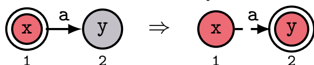
search_back (a,x,y:list)



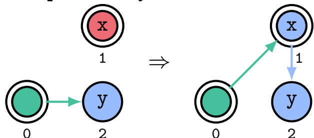
The Procedure SortNodes

```
SortNodes = (sort_forward!; try sort_back_push else (try sort_back_stack
else (try red_push else red_stack; break)))!
```

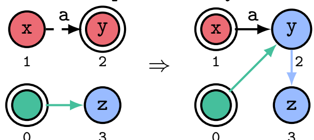
sort_forward (a,x,y:list)



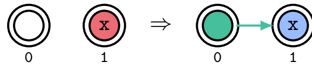
red_push (x,y:list)



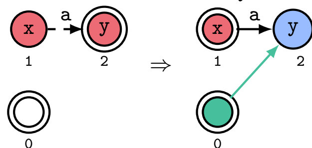
sort_back_push (a,x,y,z:list)



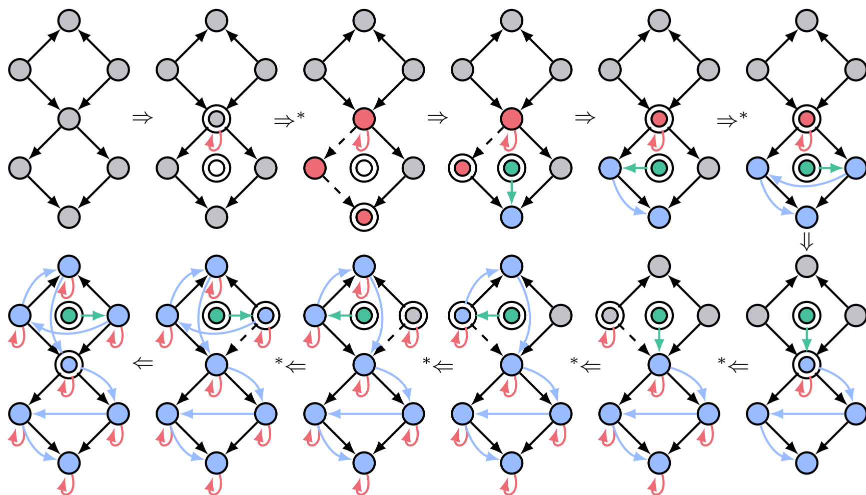
red_stack (x:list)



sort_back_stack (a,x,y:list)



Example Execution of `top-sort`



Correctness and Complexity

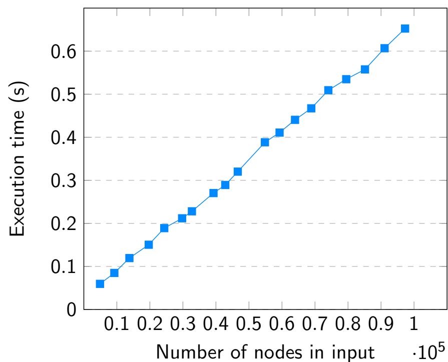
Proposition (Correctness)

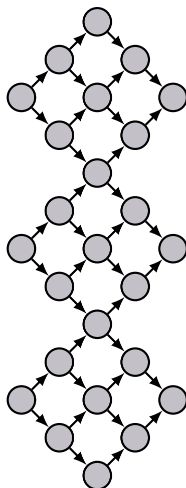
Given a connected graph with grey nodes and no directed cycles G, `top-sort` outputs G with all nodes marked blue and with additional blue edges that define a topological sorting of G. Furthermore, there's an additional green node, an additional green edge, and a red loop on each node.

Proposition (Complexity)

Given a connected graph with grey nodes and no directed cycles G, `top-sort` terminates in linear time.

Performance of top-sort





A grid chain graph

Conclusion

Rooted rules permit linear-time implementations of tree recognition and topological sorting in GP 2 for inputs of bounded degree.

Future work:

- Finding a way to implement linear-time algorithms for inputs of unbounded degree by modifying GP 2 and its implementation.
- Investigating how the implementation of data structures as part of the host graph can be used to implement more linear-time algorithms in GP 2.