UNIVERSITY of York

A Reference Interpreter for the Graph Programming Language GP 2

Christopher Bak[1], Glyn Faulkner[1], Colin Runciman and Detlef Plump

Department of Computer Science, The University of York

Graphs as Models, April 2015

[1]Supported by EPRSC Doctoral Training Grants.

Outline

- The GP 2 Language
- The GP 2 Reference Interpreter
  - Motivation & Requirements
  - Implementation
  - Performance
- Conclusions and Future Work

Graph Programs

- Domain-specific language for graph-based structures
- User supplies the input graph and the graph transformation rules
- Small set of imperative control constructs to organise rule applications
- Non-deterministic execution
- Simple syntax and semantics to facilitate formal reasoning

Semantics[1]

$$[\mathrm{Call}_1]\frac{G \Rightarrow_R H}{\langle R, G\rangle \to H} \qquad [\mathrm{Call}_2]\frac{G \not\Rightarrow_R}{\langle R, G\rangle \to \mathtt{fail}}$$

$$[\mathrm{Alap}_1]\frac{\langle P, G\rangle \to^+ H}{\langle P!, G\rangle \to \langle P!, H\rangle} \qquad [\mathrm{Alap}_2]\frac{\langle P, G\rangle \to^+ \mathtt{fail}}{\langle P!, G\rangle \to G}$$

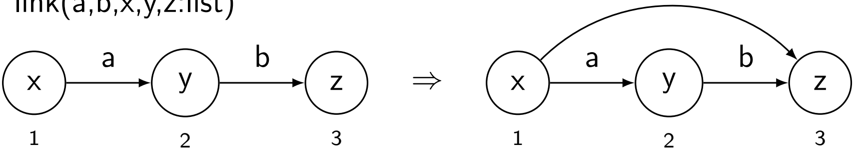
Graphs G and H are states. `fail` is the failure state. R is a rule. P is a program.

[1] *The Design of GP2*, Detlef Plump, EPTCS 82 (2012)

Transitive Closure

Main = link!

link(a,b,x,y,z:list)



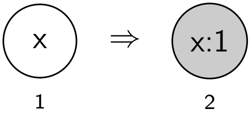
where not edge(1,3)

- Rule `link` applied as long as possible on the input graph.
- List labels used for generality.

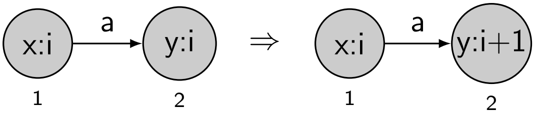
Vertex Colouring

Main = init!; inc!

init(x: list)



inc(a,x,y:list; i:int)

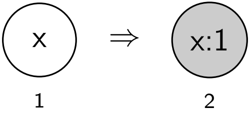


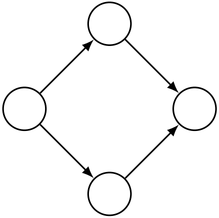
- Always outputs a valid colouring.
- Minimal colouring not guaranteed because of non-determinism.

Vertex Colouring

Main = **init!**; inc!

init(x: list)

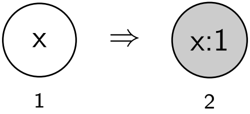


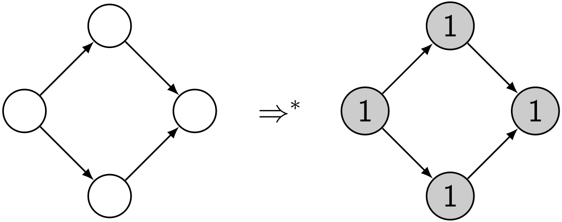


Vertex Colouring

Main = **init!**; inc!

init(x: list)

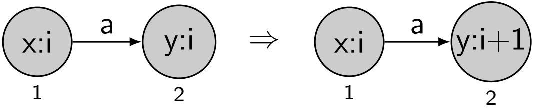


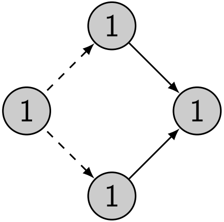


Vertex Colouring

Main = init!; **inc!**

inc(a,x,y:list; i:int)

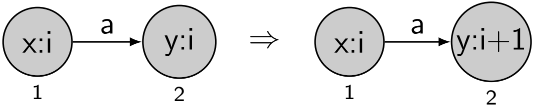


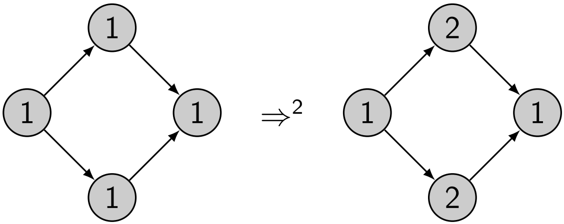


Vertex Colouring

Main = init!; **inc!**

inc(a,x,y:list; i:int)

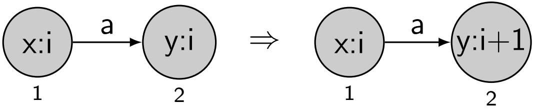


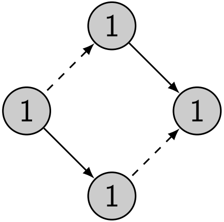


Vertex Colouring

Main = init!; **inc!**

inc(a,x,y:list; i:int)

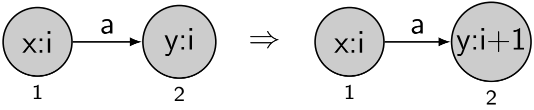


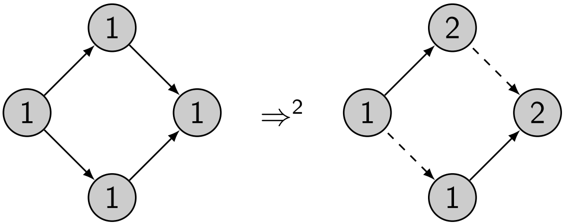


Vertex Colouring

Main = init!; **inc!**

inc(a,x,y:list; i:int)

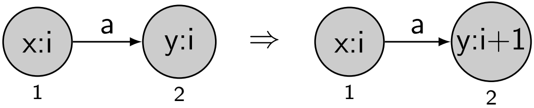


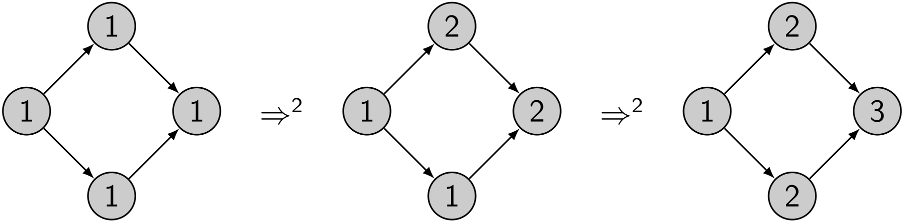


Vertex Colouring

Main = init!; **inc!**

inc(a,x,y:list; i:int)





Motivation

- Test correctness of later compiled implementations
- Fully implement non-determinacy
- Familiarise language implementers with the semantics of GP 2
- Identify any gaps or ambiguities in the semantics

Motivation

- Test correctness of later compiled implementations
- Fully implement non-determinacy
- Familiarise language implementers with the semantics of GP 2
- Identify any gaps or ambiguities in the semantics

Simplicity is an over-riding aim

- Speed and memory use are secondary concerns
- Sophistication is to be actively avoided if it complicates the implementation!

Requirements

General requirements:

- Quick to develop
- Easy to maintain and reason about
- Must be fast enough to do “useful work”

Requirements

General requirements:

- Quick to develop
- Easy to maintain and reason about
- Must be fast enough to do “useful work”

For a given program/host-graph pair…

- Generate all possible output graphs
- Produce all distinct output graphs up to isomorphism
- Output a single result

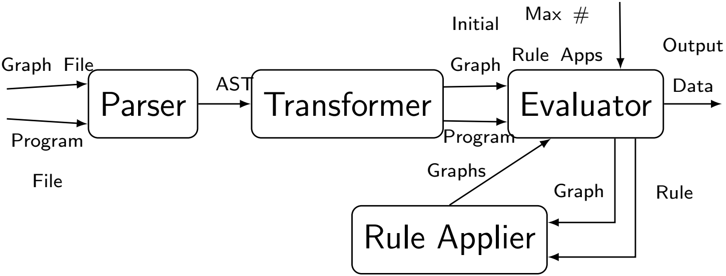
Requirements

Also, stand-alone tools:

- Isomorphism checker
- Host-graph generator
- Graph viewer

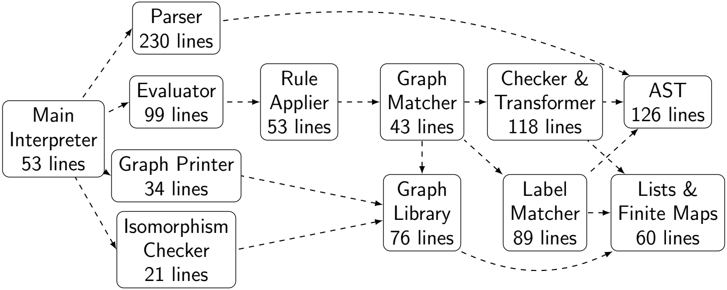
Implementation

- Based on the GP 2 semantics
- Written in Haskell[2]



[2] https://www.haskell.org/

Implementation



- Approx. 1000 SLOC
- Exploits distinctive features of Haskell to achieve conciseness:
    - list-comprehensions
    - lazy evaluation

Performance

- Produce a fourth-generation Sierpinski triangle in 6.5 seconds
- A cyclic graph of 1000 nodes fails an acyclicity test in 1.8 seconds
- Transitive closure of a linear graph of 50 nodes takes 66 seconds
- Vertex colouring a 9x9 grid in one-result mode takes less than a second. . .
- . . . but in all-result mode exceeds 5 minutes with only a 3x3 grid

A more detailed discussion of performance can be found in the paper

Conclusions

- We have developed a useful reference tool for our ongoing research
- Also useful ancillary tools:
    - GraphViz-based graph visualiser
    - Stand-alone isomorphism checker
    - Host-graph generator, based on hypergraph grammars
- Gained a clear understanding of the GP 2 semantics
- Become aware of some ‘edge-cases’ that might trip us up in our compiler work

Further work

- Better error reporting
- A performant compiler
- GUI program editor
- Formal verification against GP 2 semantics.