fUML Activity Diagrams in *RACR*[1]

A *RACR* Solution of *The TTC 2015* *Model Execution Case*

Christoff Bürger

Department of Computer Science, Faculty of Engineering, LTH

Lund University

Lund, Sweden

christoff.burger@cs.lth.se

[1] https://github.com/christoff-buerger/racr

Background

Reference attribute grammar controlled rewriting

- RAG-controlled rewriting = RAGs + rewriting
  - RAGs for declarative, cached analyses
  - graph rewriting for state changes
  - seamless combination:
    - use of analyses to deduce rewrites
    - rewrites automatically update analyses

    } controlled

    >> incremental

Graph rewriting in terms of RAGs

- AG = abstract syntax tree (AST)
- RAG = abstract syntax graph (ASG)
  - reference attributes induce semantic overlay graph on top of AST
  - enables deduction *and* analyses of graph structure (e.g., transitive closure analyses like control-flow)
  - \>> rewrites change AST *but*
    - patterns are graphs (reference attributes)
    - reference edges change with AST

RACR

- reference implementation of RAG-controlled rewriting in *Scheme*

- *R6RS* library; API for:
  - ASG schema definition (AST schema + attribution)
  - ASG querying (AST + attributes)
  - rewriting (imperative/RAG-controlled/fixpoint; primitive/pattern-based; or combination of all)
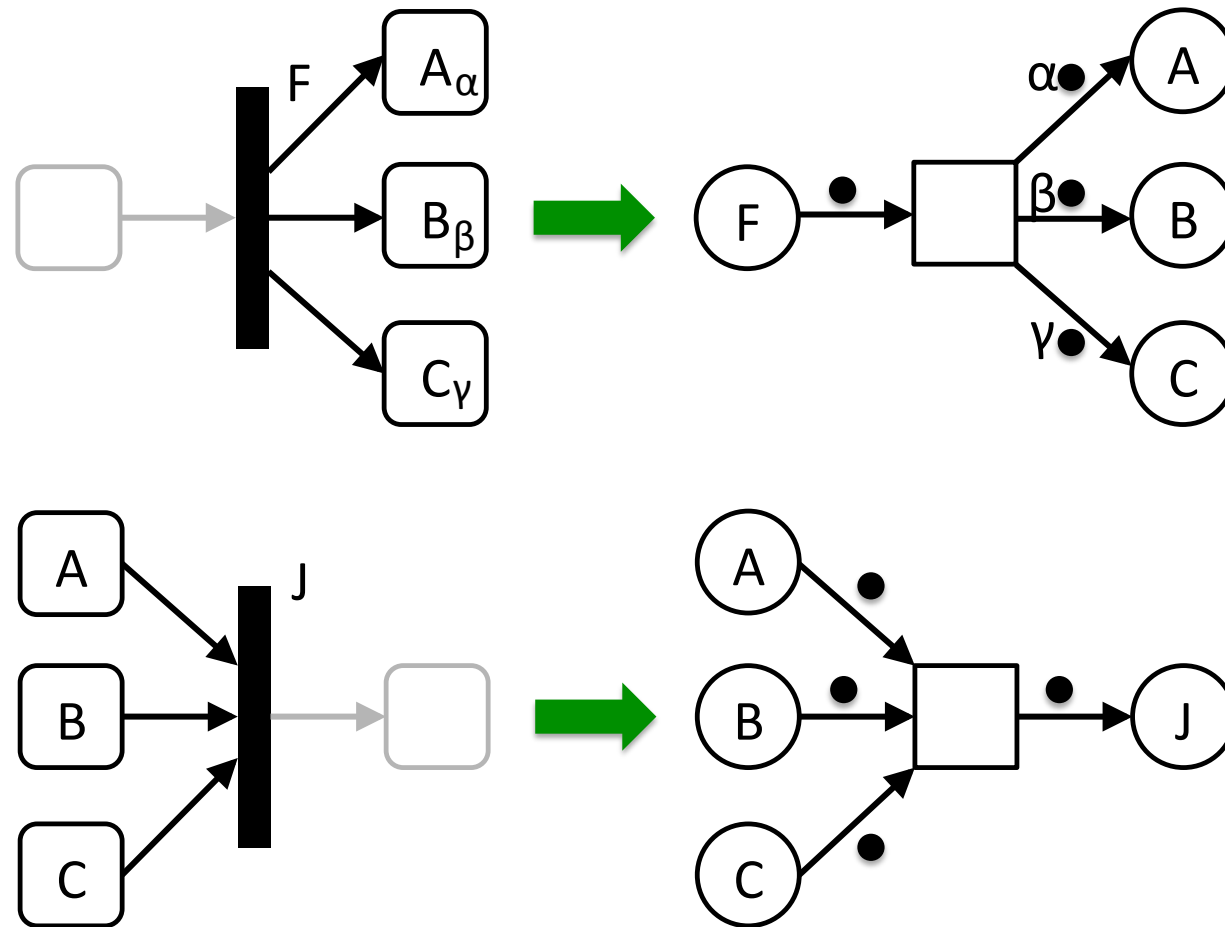
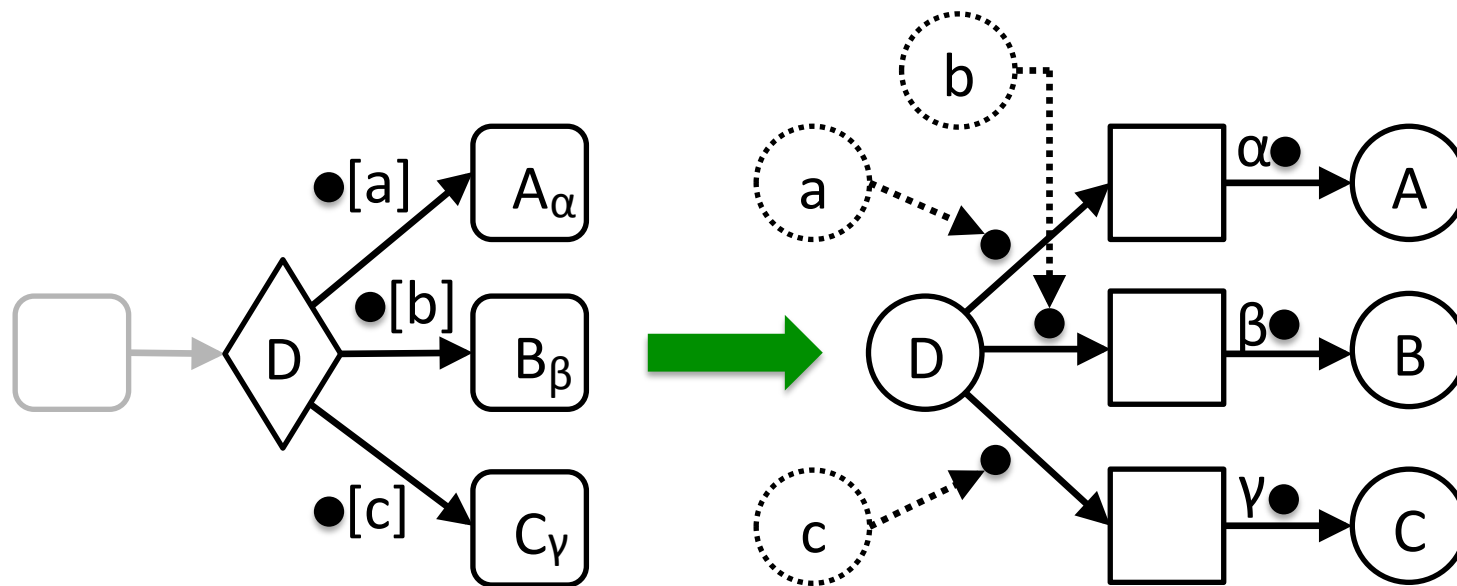 https://github.com/christoff-buerger/racr

Solution

General idea

- two *RACR*-based language processors
  - *fUML Activity Diagram* → Petri net compiler
  - Petri net interpreter (coloured, weighted nets)

fUML Activity Diagram → Petri net

fUML Activity Diagram → Petri net

fUML Activity Diagram compiler

- attributes for:
  - name analysis (symbolic name resolution)
    - incoming & outgoing edges
    - variables

    reference attributes
  - type analysis (expression types)
  - well-formedness analysis (only *TTC* solution that rejects malformed diagrams)
  - Petri net generation

Petri net interpreter

- attributes for:
  - name analysis
  - well-formedness analysis
  - enabled analysis (kind of name analysis)
- rewrites for execution (firing)
  - delete consumed tokens
  - add produced tokens
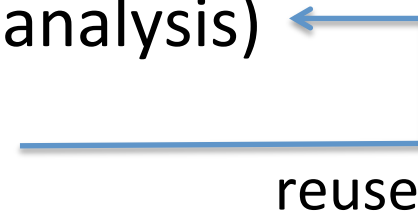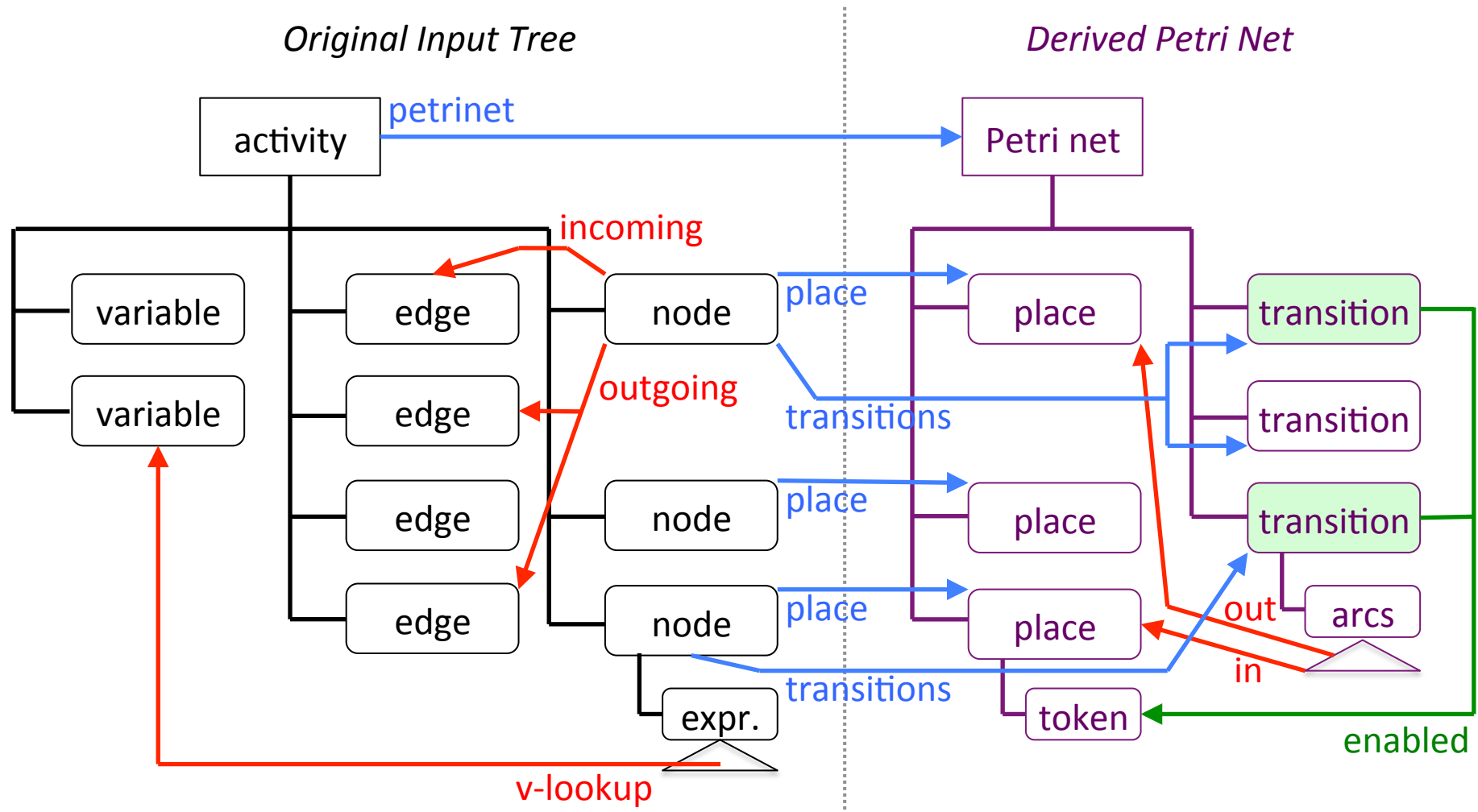
reuse

ASG

Original Input Tree   *Derived Petri Net*

semantic overlay graph (excerpt): ■ name analysis ■■ code generation ■ enabled analysis

Execution (RAG-controlled rewriting)

Original Input Tree

Derived Petri Net
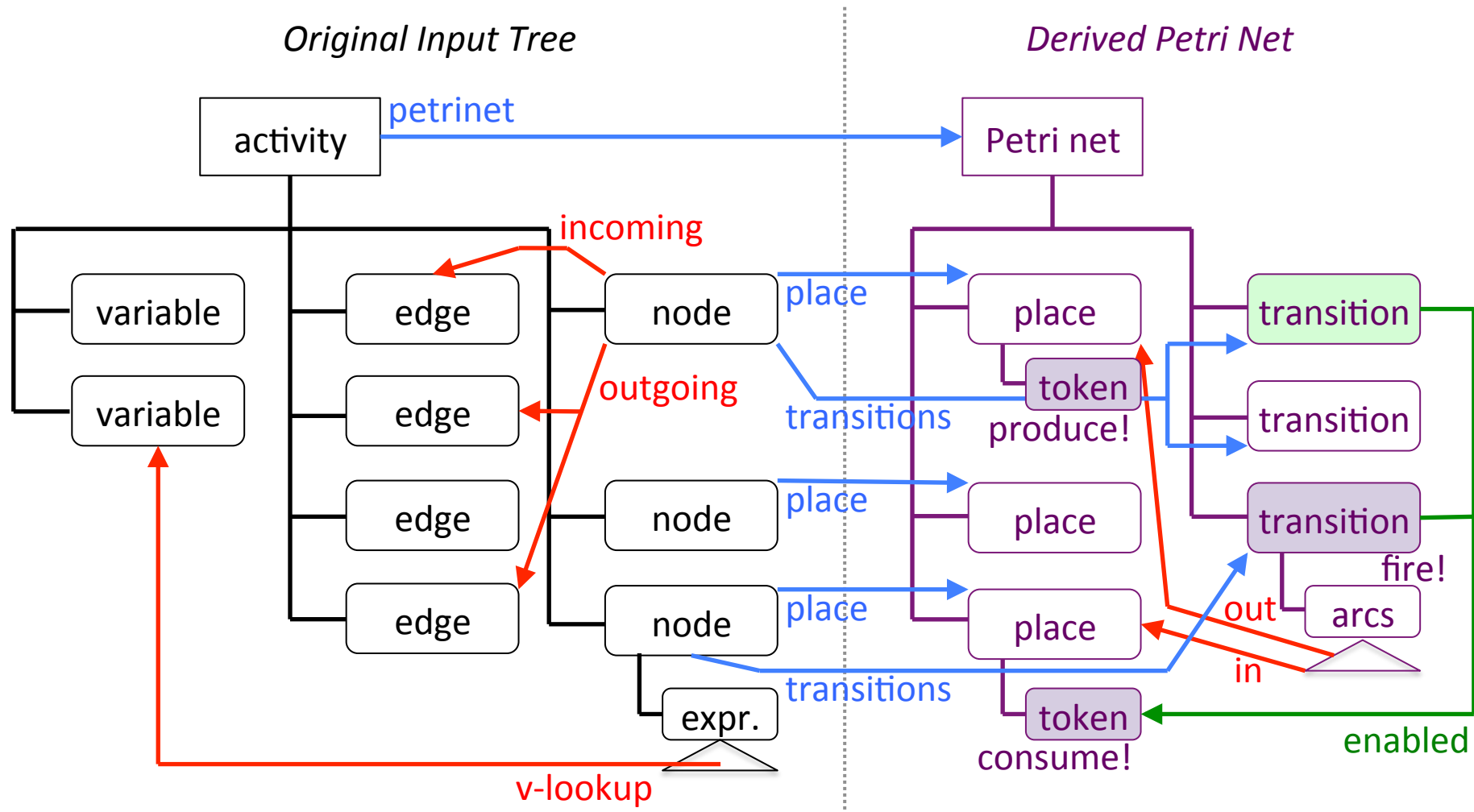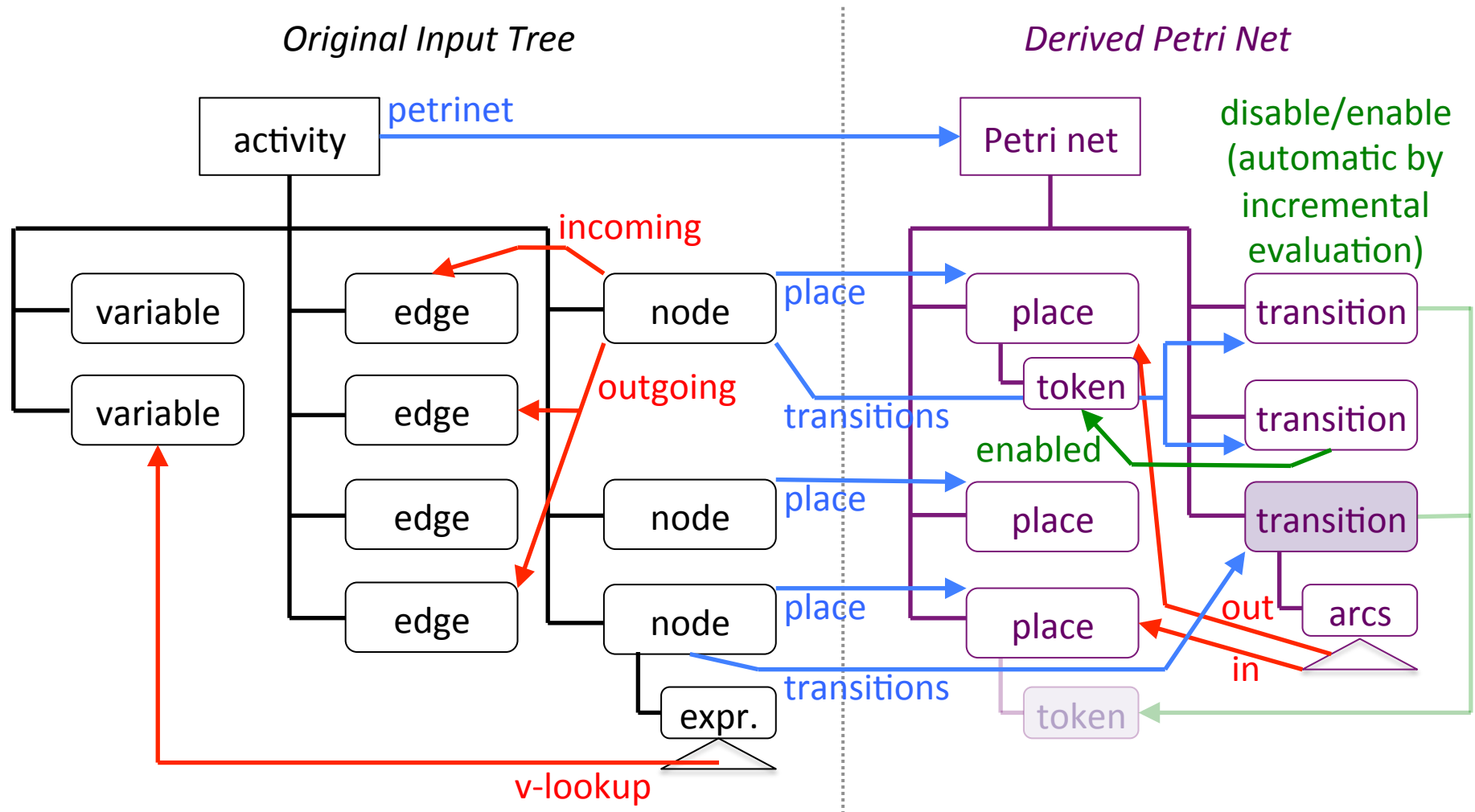

semantic overlay graph (excerpt): ■ name analysis ■■ code generation ■ enabled analysis

Execution (RAG-controlled rewriting)

Original Input Tree

Derived Petri Net

activity

petrinet

Petri net

incoming

disable/enable
(automatic by
incremental
evaluation)

variable

edge

node

place

place

transition

outgoing

token

transitions

transition

variable

edge

enabled

node

place

place

transition

edge

node

place

place

out

arcs

edge

transitions

in

expr.

token

v-lookup

semantic overlay graph (excerpt): ■ name analysis ■■ code generation ■ enabled analysis

Evaluation

Performance

Tasks Performed	Test Cases (`testperformance_variant`)				Time Spend
(later tasks include previous ones)	1	2	3_1	3_2	*(lowest / highest / average)*
Activity diagram parsing	831 / 831	871 / 871	875 / 875	718 / 718	41% / 86% / 50%
Activity diagram well-formedness	926 / 95	1017 / 146	1079 / 204	739 / 21	3% / 11% / 7%
Petri net generation	1042 / 116	1061 / 44	1196 / 117	741 / 2	0% / 6% / 4%
Petri net well-formedness	1220 / 178	1230 / 169	1466 / 270	746 / 5	1% / 14% / 10%
Petri net execution	2026 / 806	1776 / 546	1912 / 446	831 / 85	10% / 40% / 29%
Petri net execution (enabled passes)	2618 / 1398	1344 / 114	1572 / 106	836 / 90	7% / 53% / 27%

execution times in ms
(cf. solution description)

Lines of code

Source Code File	Solution Part (language task)	LOC	
Activity diagram language (507):		*499*	
`analyses.scm:` *255*	AST specification	18	4%
	ASG accessors (constructors, child & attribute accessors)	65	13%
	Name analysis	32	6%
	Type analysis	23	5%
	Well-formedness	32	6%
	Petri net generation	90	18%
`parser.scm:` *219*	Parsing	214	43%
`user-interface.scm:` *33*	Initialisation & execution	25	5%
Petri net language (255):		*200*	
`analyses.scm:` *102*	AST specification	9	5%
	ASG accessors (constructors, child & attribute accessors)	32	16%
	Name analysis	13	7%
	Well-formedness	10	5%
	Enabled analysis	29	15%
`execution.scm:` *43*	Running and firing semantics	31	16%
`user-interface.scm:` *80*	Initialisation & Petri net syntax	33	17%
	Read-eval-print-loop interpreter	19	10%
	Testing nets (marking & enabled status)	24	12%

no further software artefacts