fUML Activity Diagrams with RAG-controlled Rewriting

A *RACR*[1] Solution of *The TTC 2015 Model Execution Case*

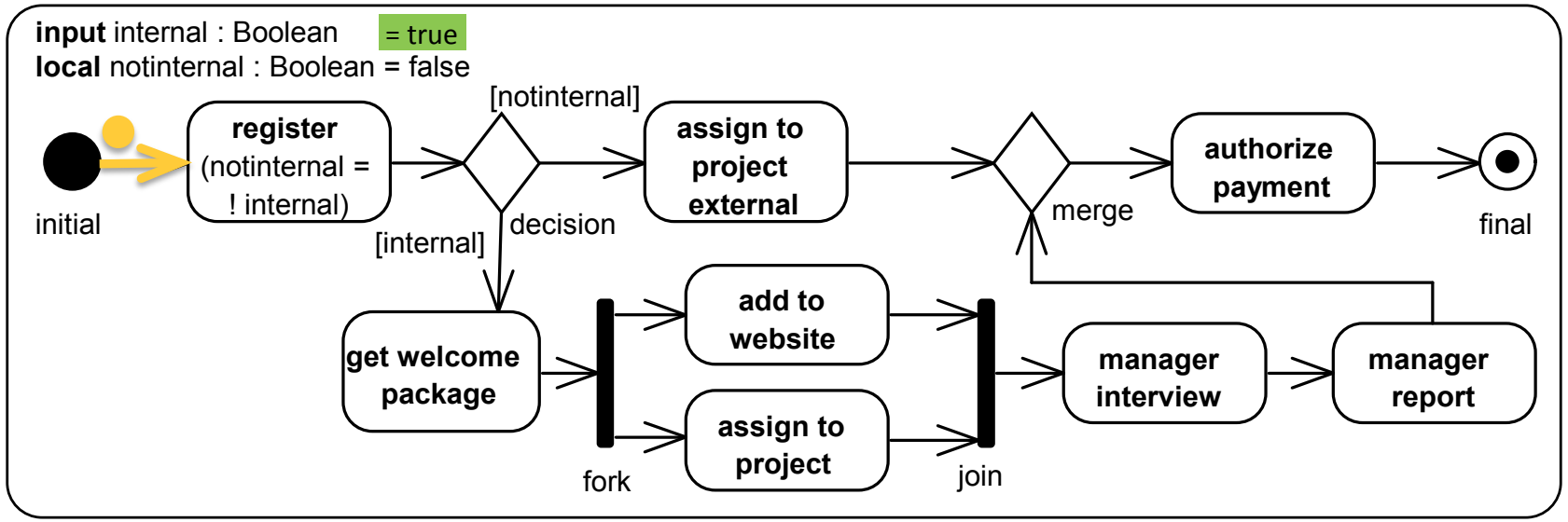
Christoff Bürger
Department of Computer Science, Faculty of Engineering, LTH
Lund University
Lund, Sweden
christoff.burger@cs.lth.se

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

TTC 2015 Background

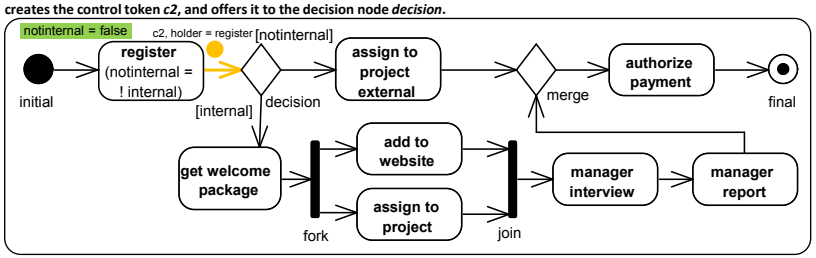
8th Transformation Tool Contest

Task: execution of *fUML Activity Diagrams.*

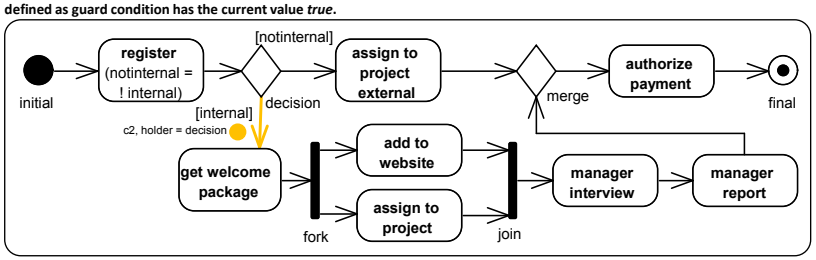


8th Transformation Tool Contest

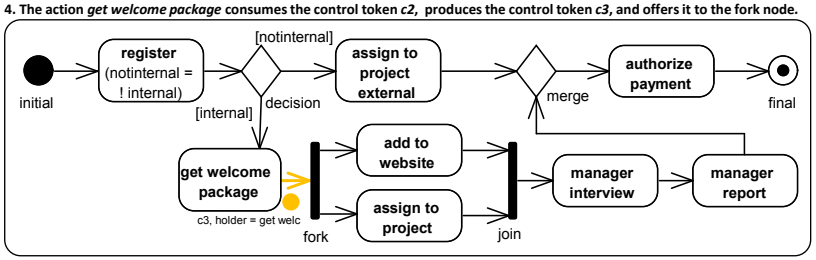
2. The action *register* consumes the token *c1*, executes the defined expression leading to an update of the variable *non-internal*, creates the control token *c2*, and offers it to the decision node *decision*.



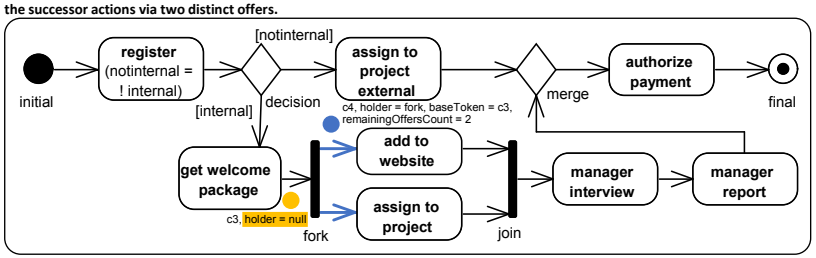
3. The decision node *decision* offers the control token *c2* to the opaque action *get welcome package*, because the variable *internal* defined as guard condition has the current value *true*.



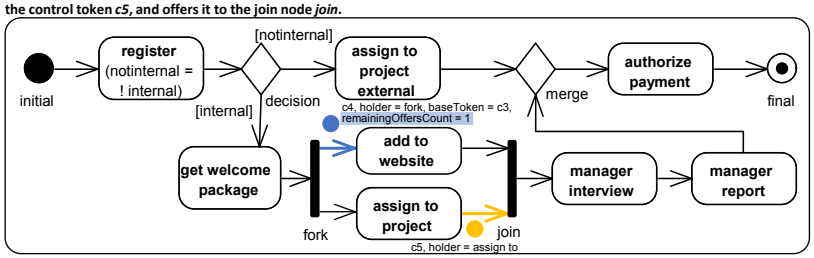
4. The action *get welcome package* consumes the control token *c2*, produces the control token *c3*, and offers it to the fork node.



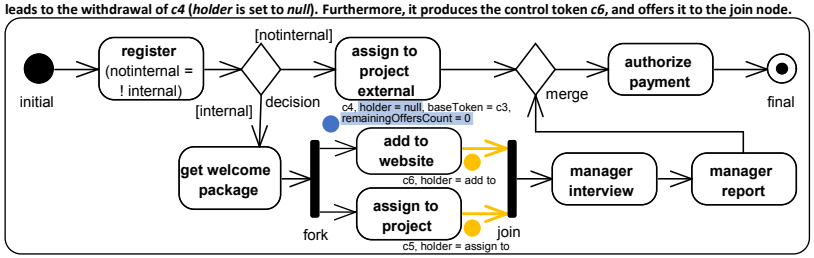
5. The fork node *fork* produces the forked token *c4* for the incoming control token *c3* (i.e., the forked token's base token). The remaining offers count is set to 2, because the fork node has two outgoing control flow edges. The forked token *c4* is offered to the successor actions via two distinct offers.



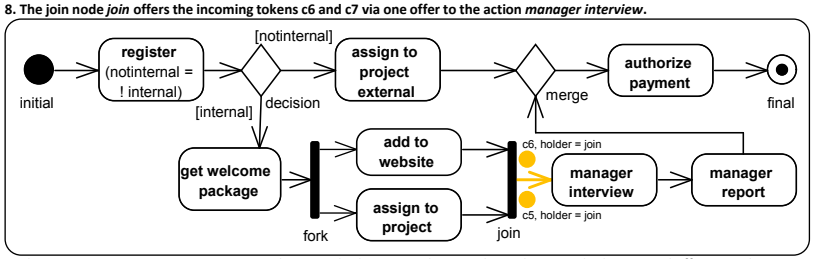
6. The action *assign to project* consumes its token offer for *c4* leading to an update of *c4*'s remaining offers count to 1, produces the control token *c5*, and offers it to the join node *join*.



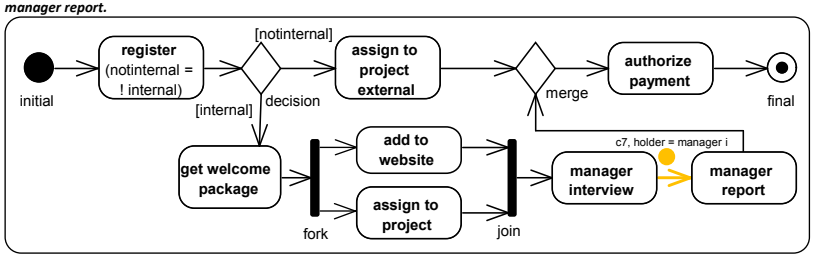
7. The action *add to website* consumes its token offer for *c4* leading to an update of *c4*'s remaining offers count to 0, which in turn leads to the withdrawal of *c4* (*holder* is set to *null*). Furthermore, it produces the control token *c6*, and offers it to the join node.



8. The join node *join* offers the incoming tokens c6 and c7 via one offer to the action *manager interview*.



9. The action *manager interview* consumes the control tokens *c5 and c6*, produces the control token *c7*, and offers it to the action *manager report*.



RACR Solution Background

General solution idea

Interpreter consisting of two parts …

- *Activity Diagram* → Petri net compiler (analyses)
- Petri net interpreter (state transformations)

… implemented using RAG-controlled rewriting.

RAG-controlled rewriting

- RAG-controlled rewriting = RAGs + graph rewriting
  - reference attribute grammar for declarative analyses
    - reference attributes induce semantic overlay graph on top of abstract syntax tree (AST)
    - enables deduction *and* analyses of graph structure

    >> deduced, memoized abstract syntax graph (ASG)
  - graph rewriting for ASG transformations
    - left hand: ASG pattern (ASTs connected via reference attributes)
    - right hand: manipulations on matched, underlying AST

    >> ASG changes with AST (updated by RAG)
  - seamless combination:
    - use of analyses to deduce rewrites
    - rewrites automatically update analyses

    } mutual control

    >> incremental

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

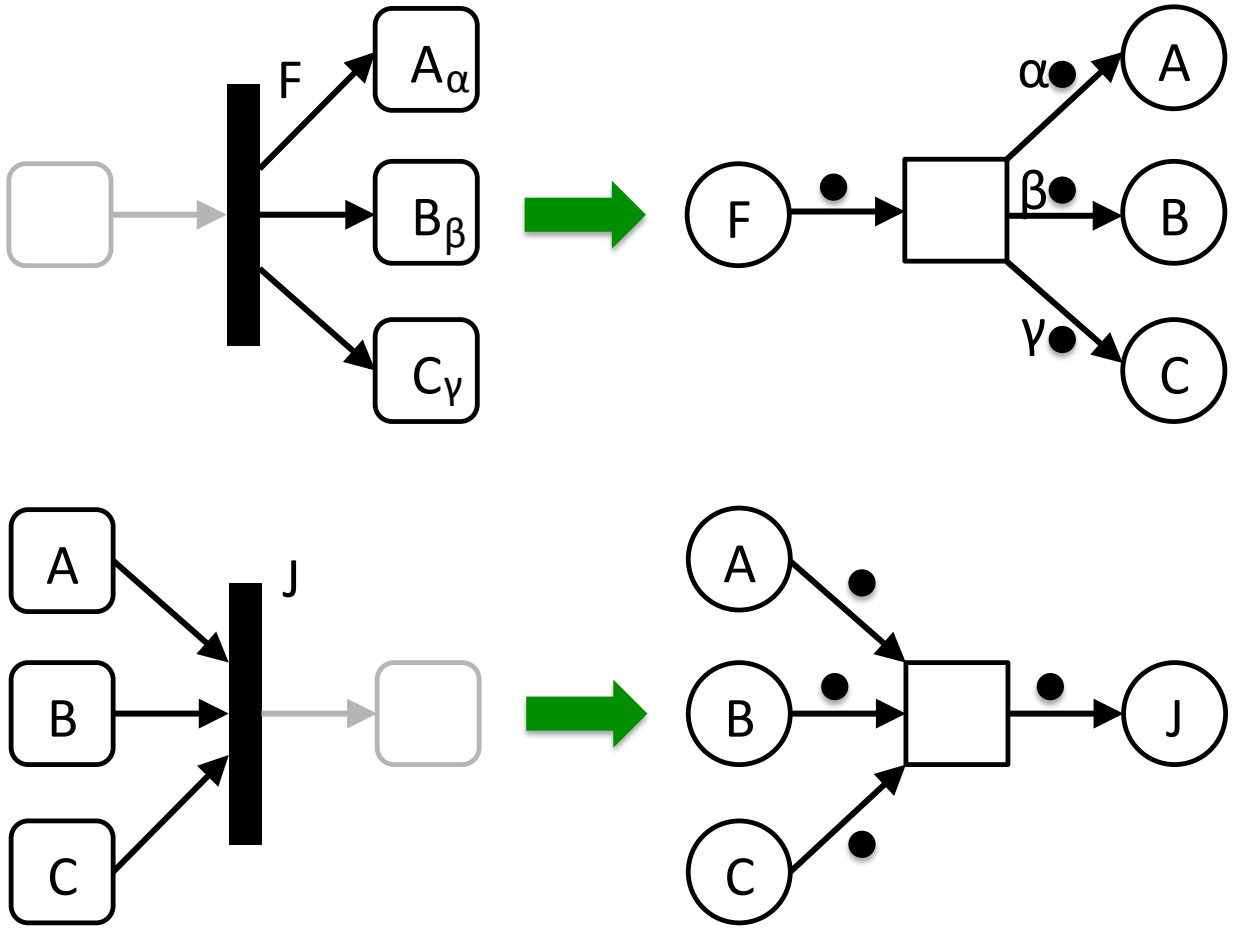
Solution

fUML Activity Diagram compiler

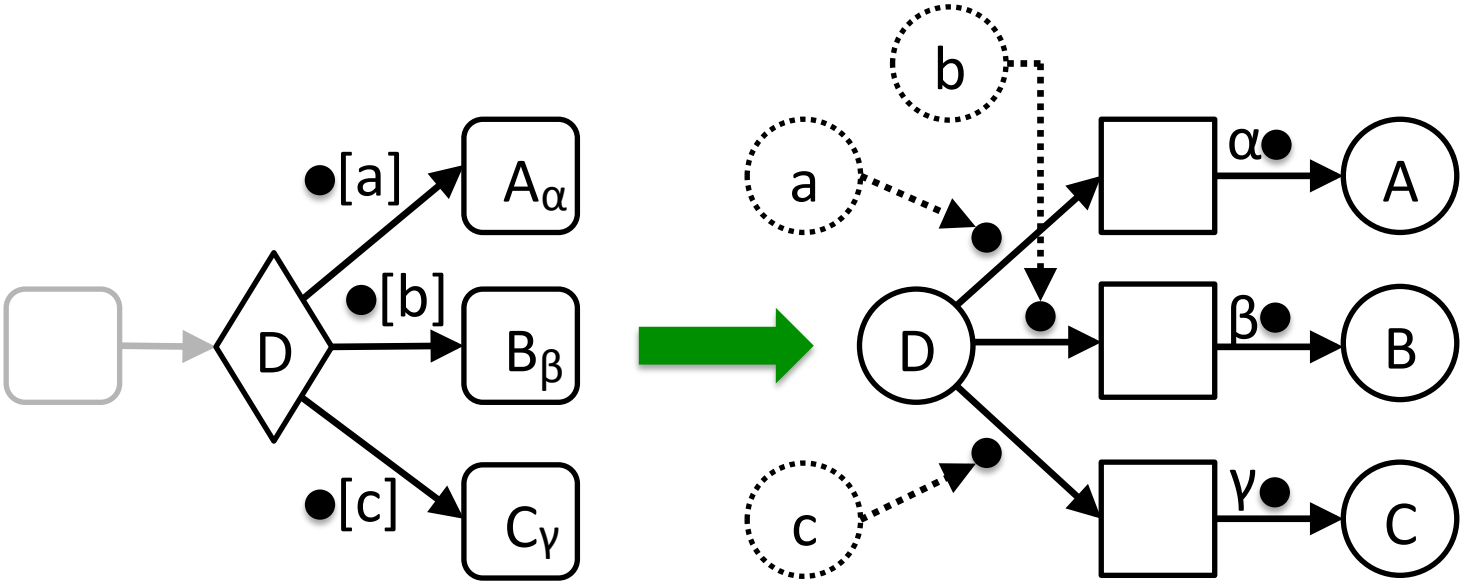
- attributes for:
  - name analysis (symbolic name resolution)
    - incoming & outgoing edges
    - variables

    } reference attributes
  - type analysis (expression types)
  - well-formedness analysis (only *TTC* solution that rejects malformed diagrams)
  - code generation (i.e., Petri net generation)

fUML Activity Diagram → Petri net

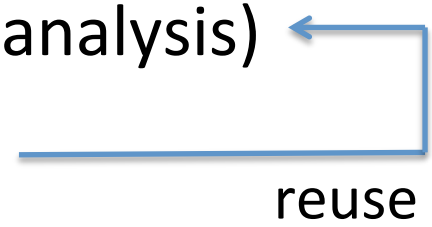


fUML Activity Diagram → Petri net

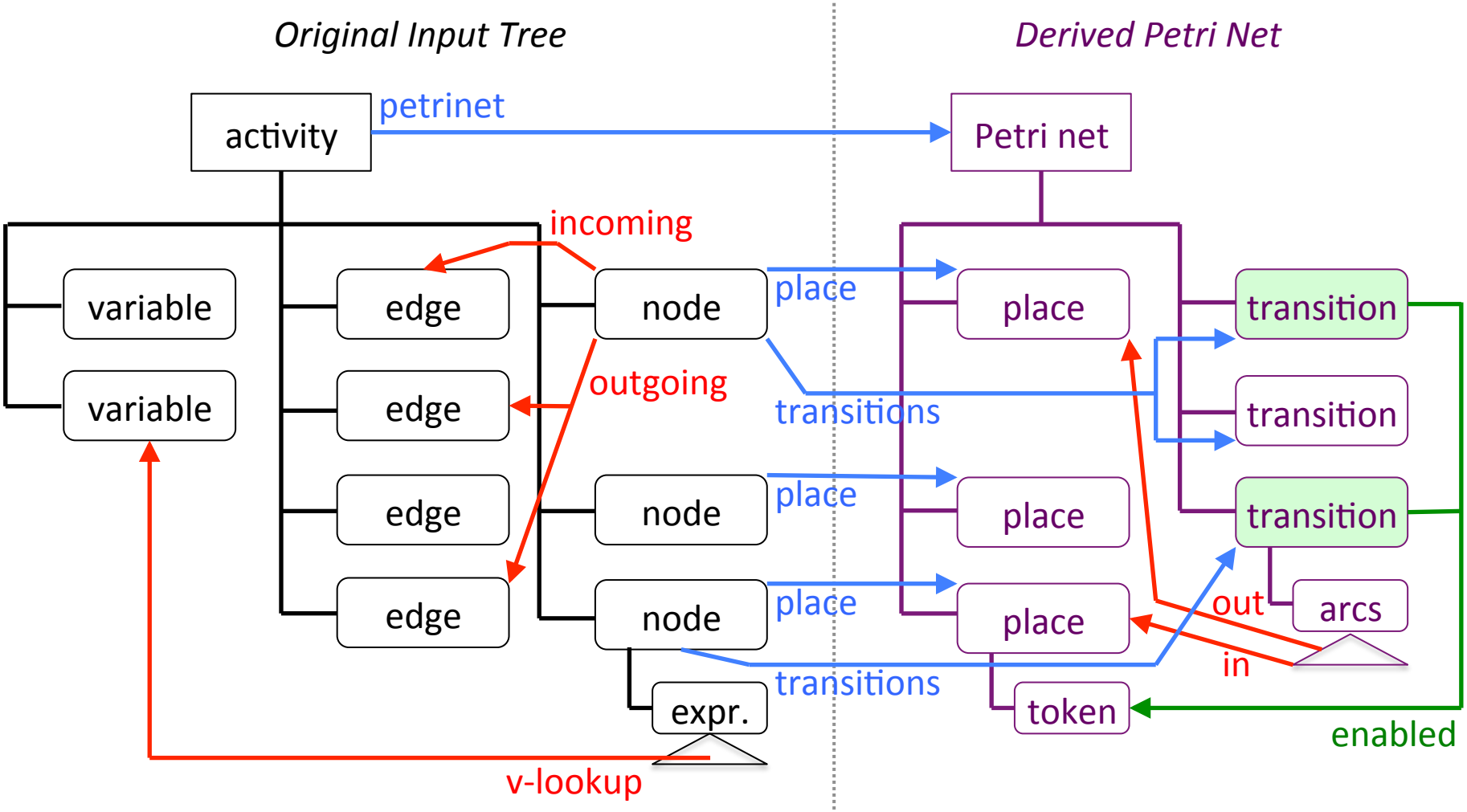


Petri net interpreter

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

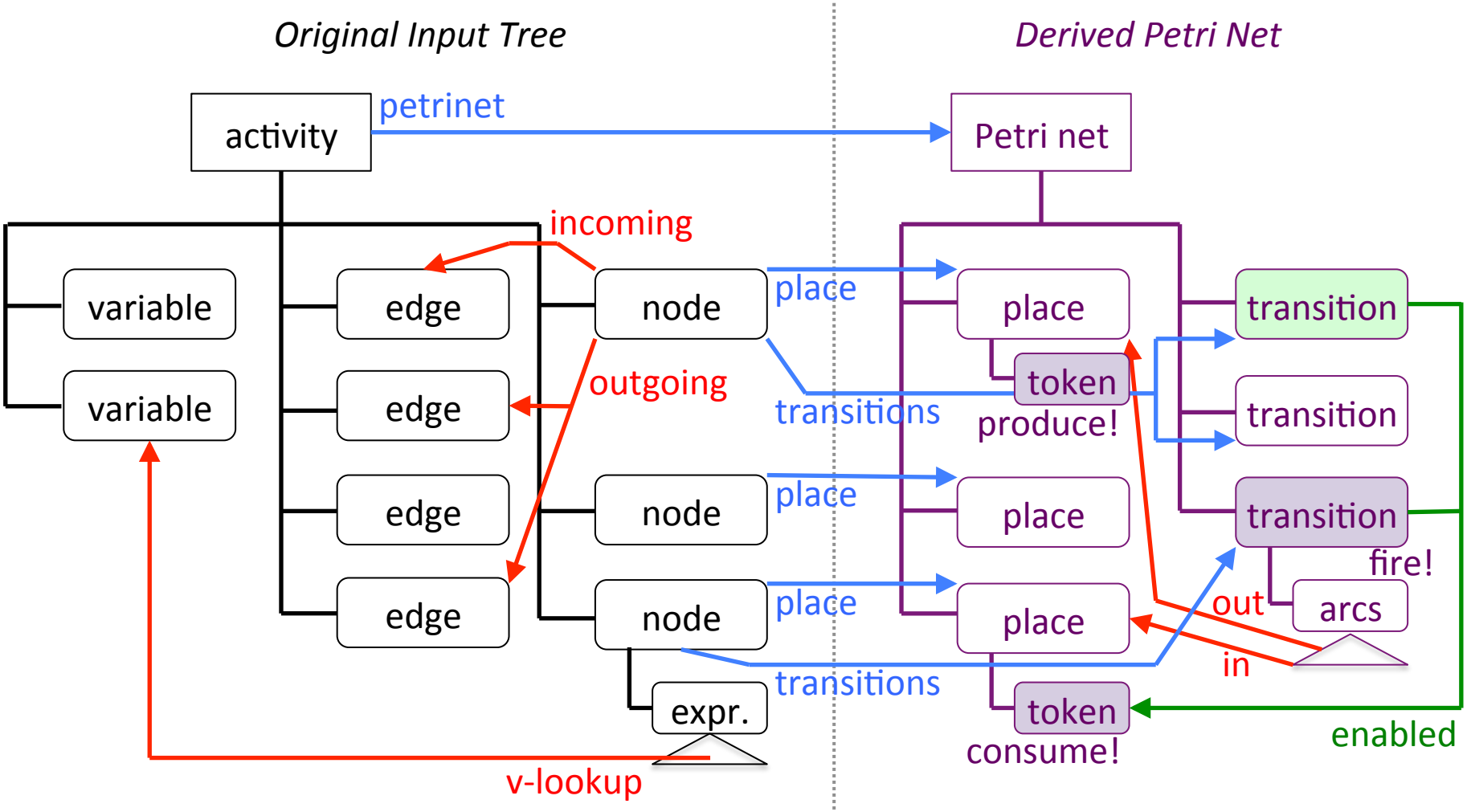


ASG



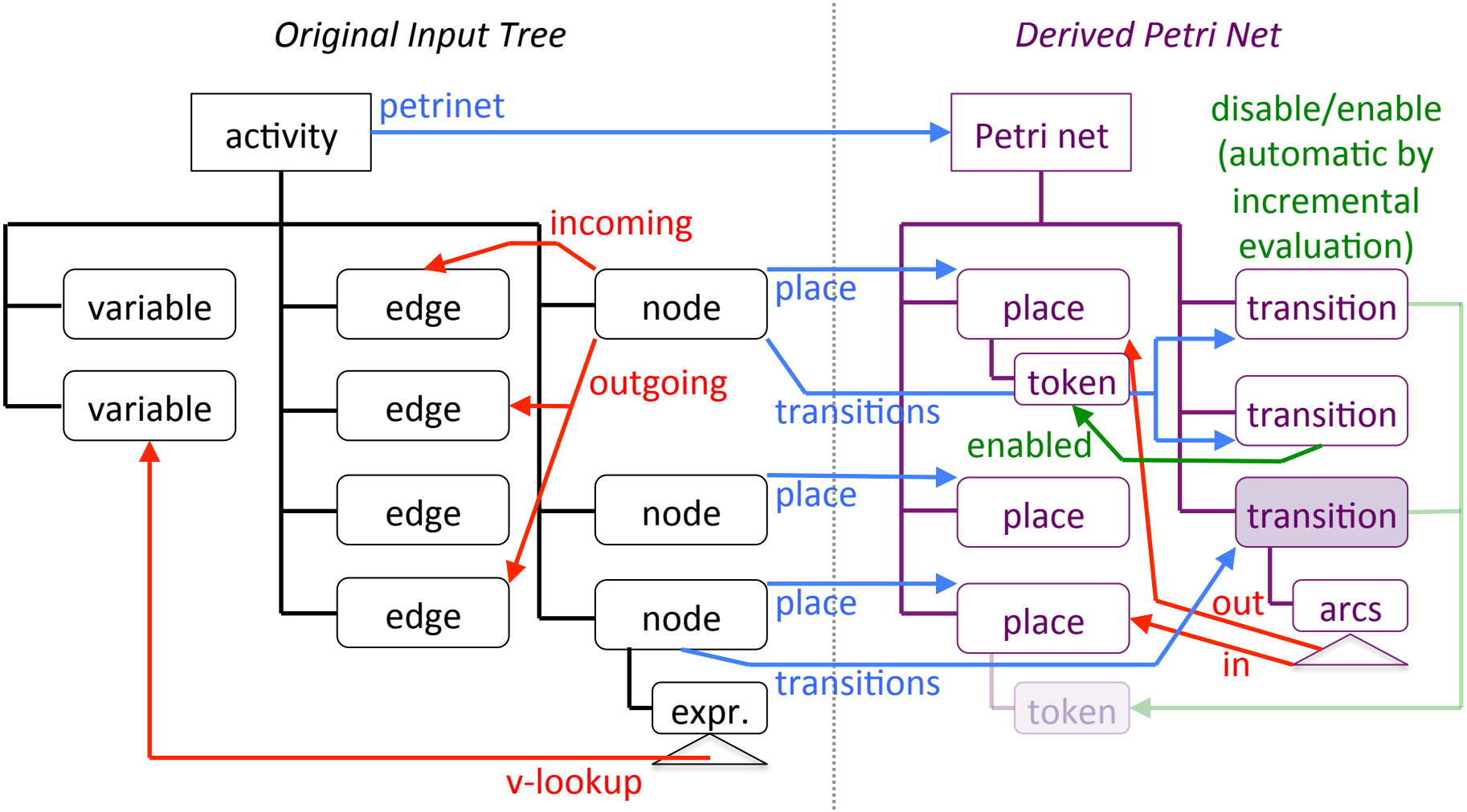
semantic overlay graph (excerpt): ■ name analysis ■■ code generation ■ enabled analysis

Execution (RAG-controlled rewriting)



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

Execution (RAG-controlled rewriting)



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

Evaluation

Performance

Tasks Performed *(later tasks include previous ones)*	Test Cases (`testperformance_variant`) 1	2	3_1	3_2	Time Spend *(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

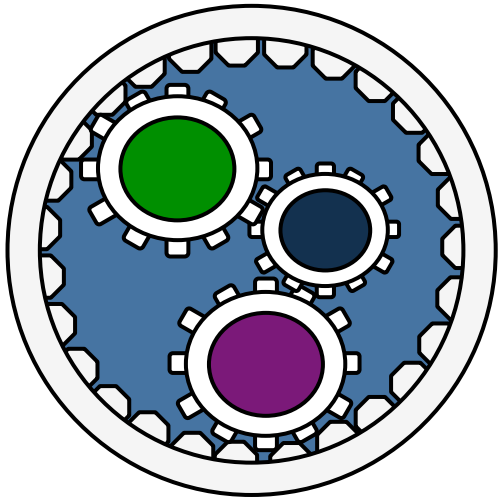
Benefits of RAG-controlled rewriting

interactive

...

incremental
meta programming

model transformation



runtime models

IDEs

incremental reasoning

...

Efficient Analyses

Efficient Rewriting

Programmed /
RAG Controlled Rewriting

RACR