fUML Activity Diagrams with RAG-controlled rewriting

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

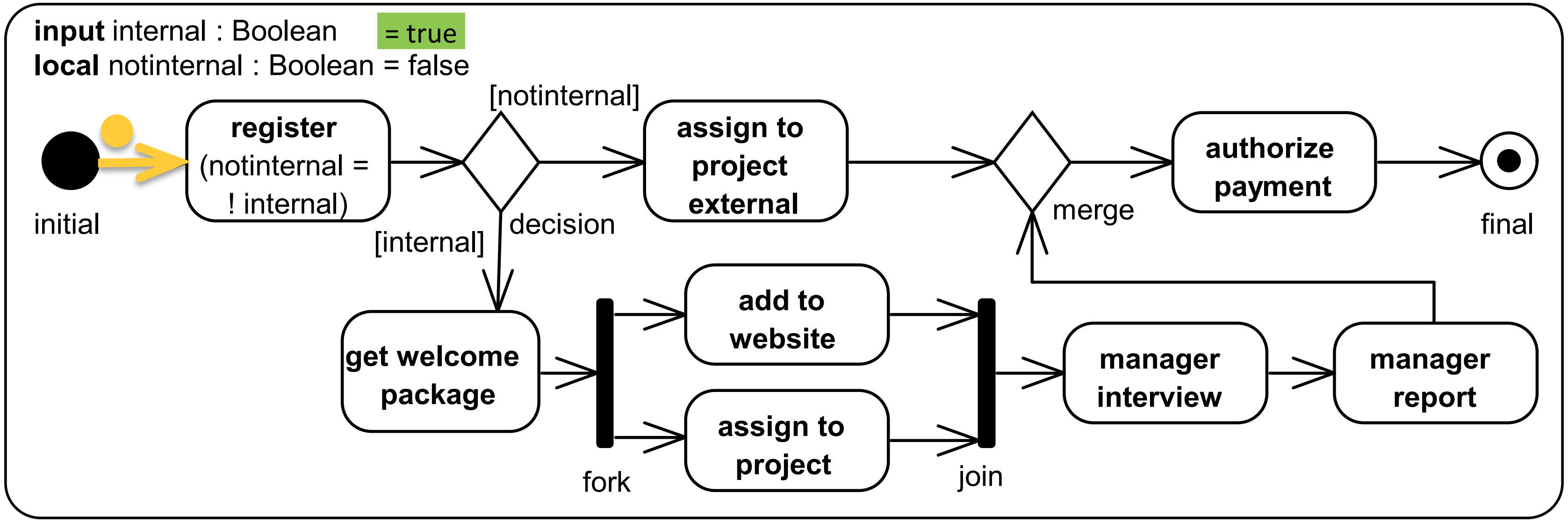
Christoff Bürger
christoff.buerger@gmail.com

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

TTC 2015 background

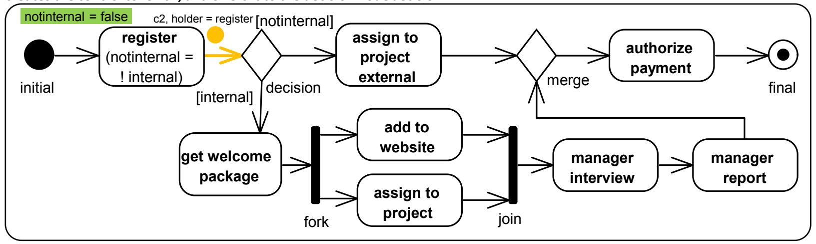
8^{th} *Transformation Tool Contest*

Task: execution of *fUML Activity Diagrams.*

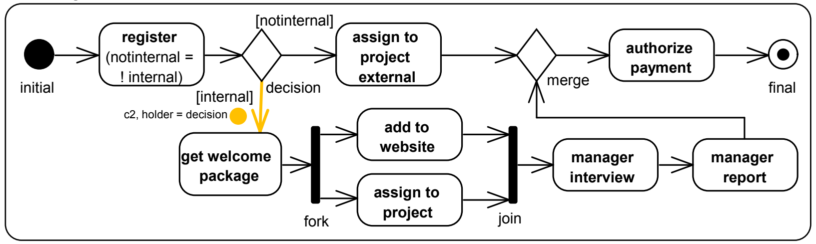


8th Transformation Tool Contest

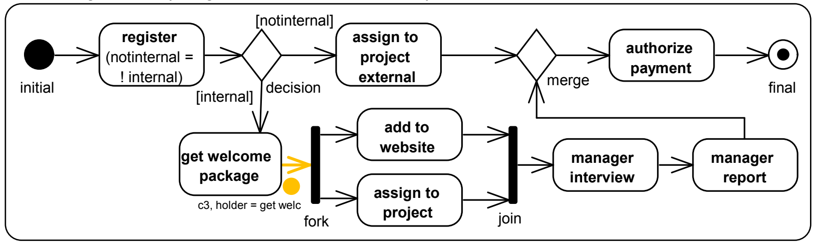
2. The action *register* consumes the token *c1*, executes the defined expression leading to an update of the variable *non-internal*, creates the control token *c2*, and offers it to the decision node *decision*.



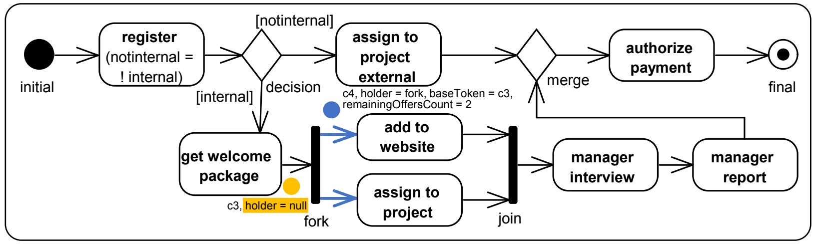
3. The decision node *decision* offers the control token *c2* to the opaque action *get welcome package*, because the variable *internal* defined as guard condition has the current value *true*.



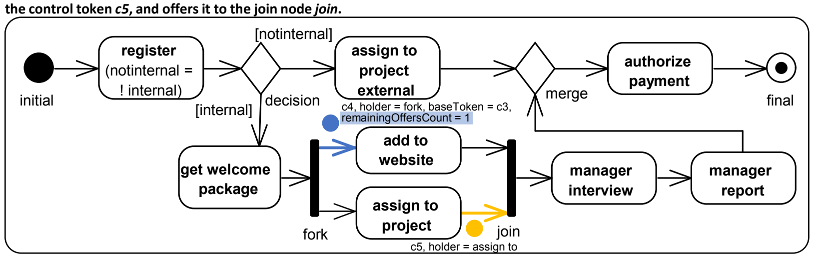
4. The action *get welcome package* consumes the control token *c2*, produces the control token *c3*, and offers it to the fork node.



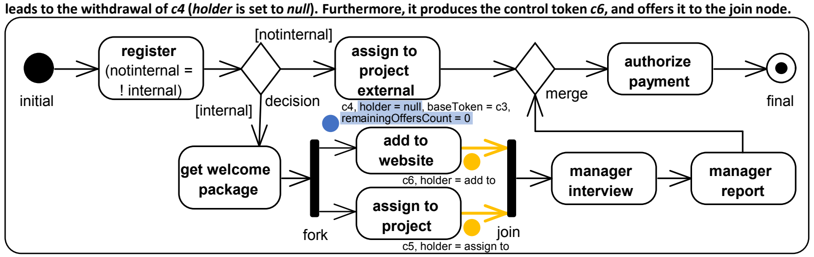
5. The fork node *fork* produces the forked token *c4* for the incoming control token *c3* (i.e., the forked token's base token). The remaining offers count is set to 2, because the fork node has two outgoing control flow edges. The forked token c4 is offered to the successor actions via two distinct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



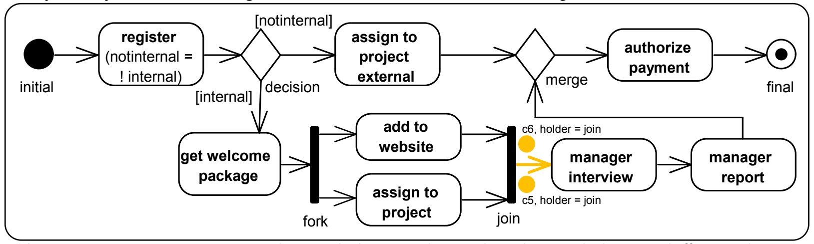
6. The action *assign to project* consumes its token offer for *c4* leading to an update of *c4*'s remaining offers count to 1, produces the control token *c5*, and offers it to the join node *join*.



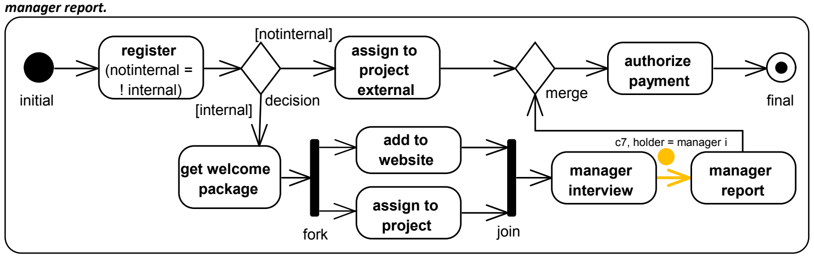
7. The action *add to website* consumes its token offer for *c4* leading to an update of *c4*'s remaining offers count to 0, which in turn leads to the withdrawal of *c4* (*holder* is set to *null*). Furthermore, it produces the control token *c6*, and offers it to the join node.



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) >> extend AST to ASG
    - 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

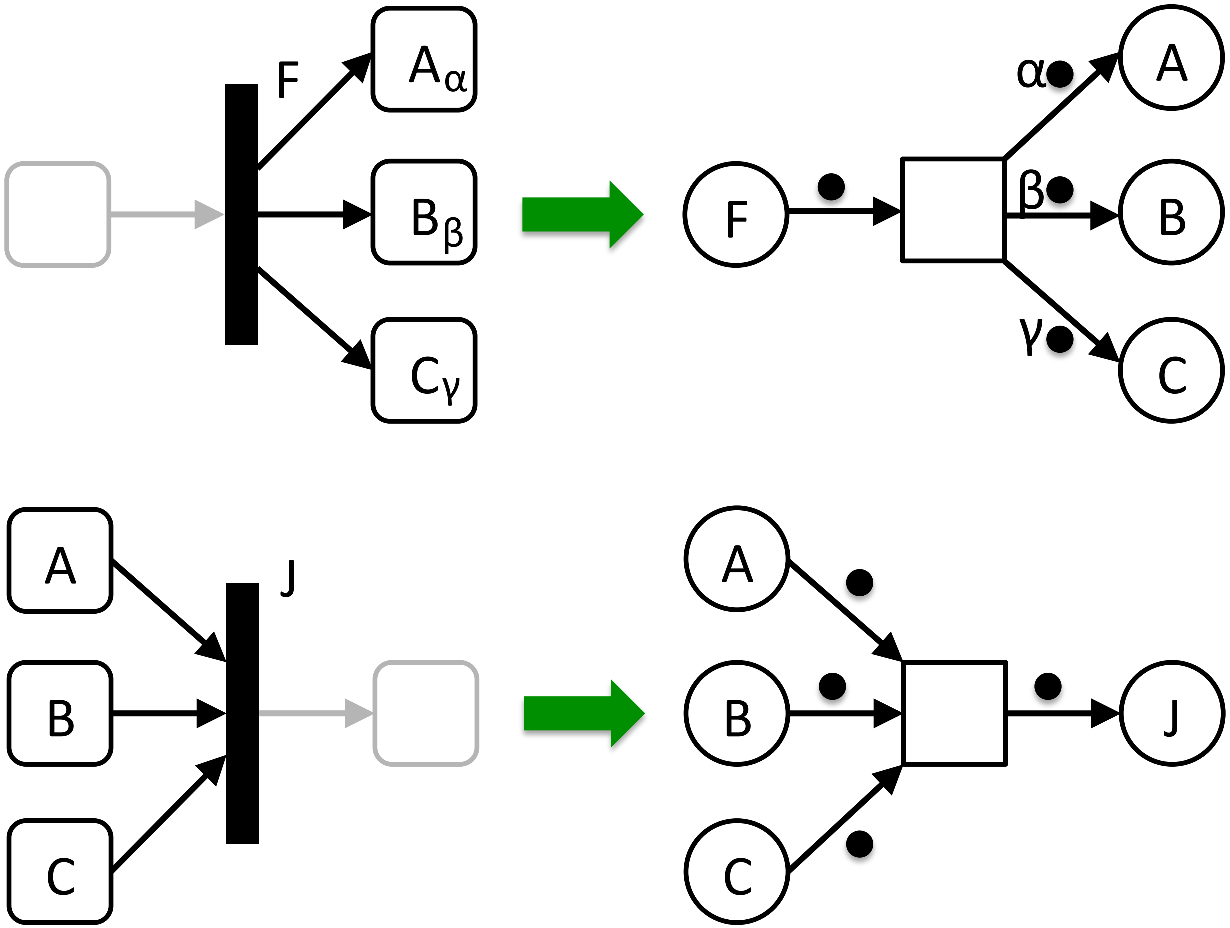
Solution

fUML Activity Diagram compiler

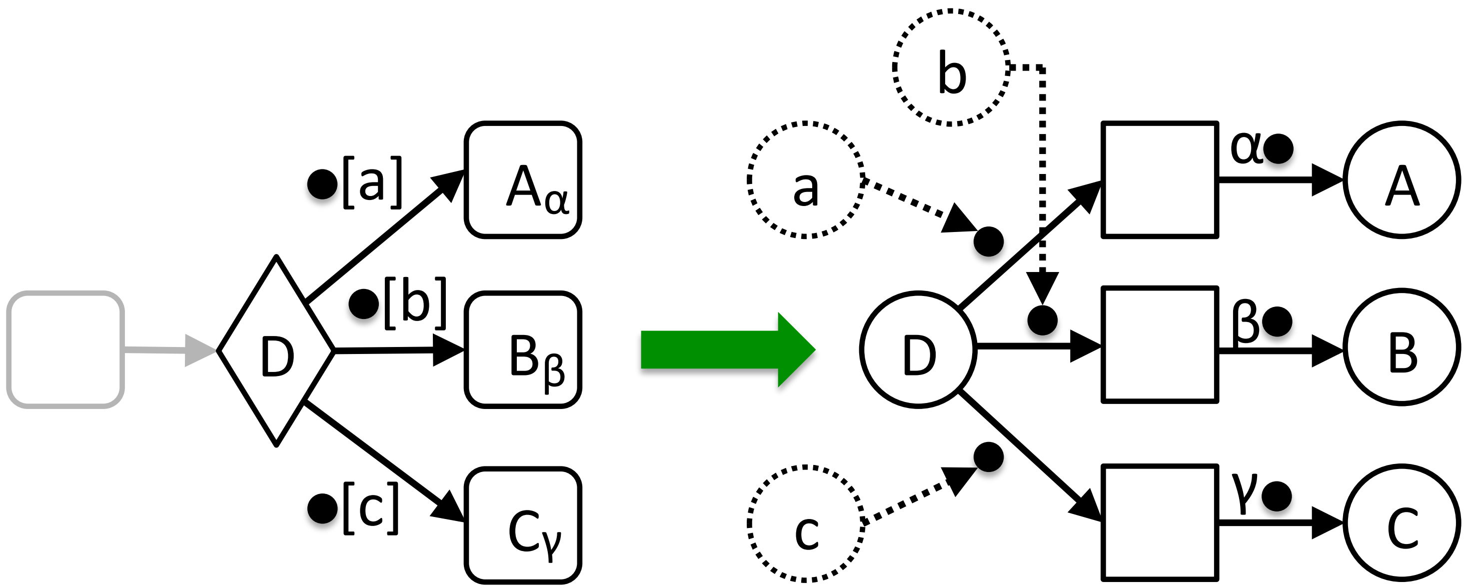
- attributes for:
  - name analysis (symbolic name resolution)
    - incoming & outgoing edges
    - variables

    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

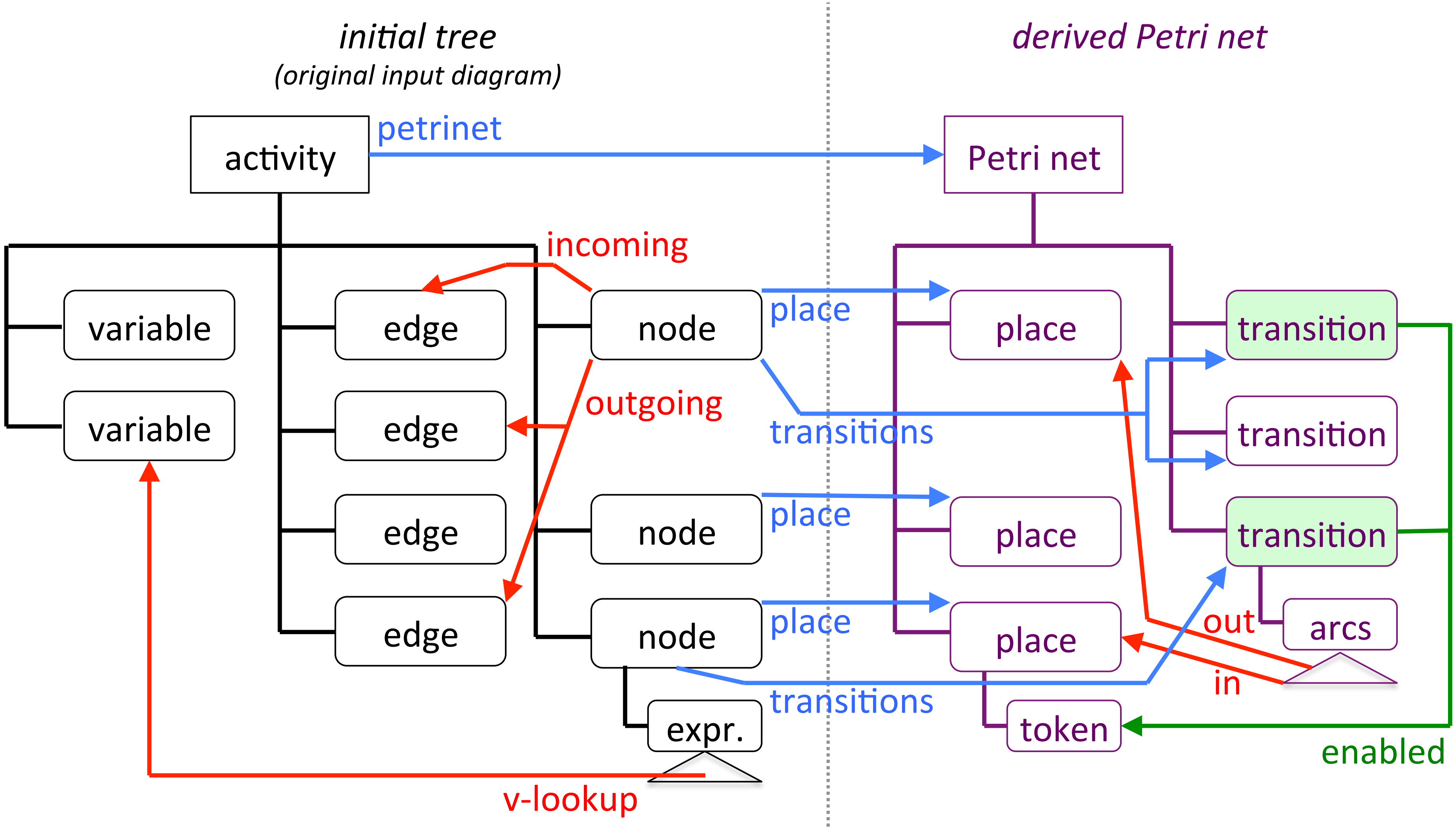


Petri net interpreter

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

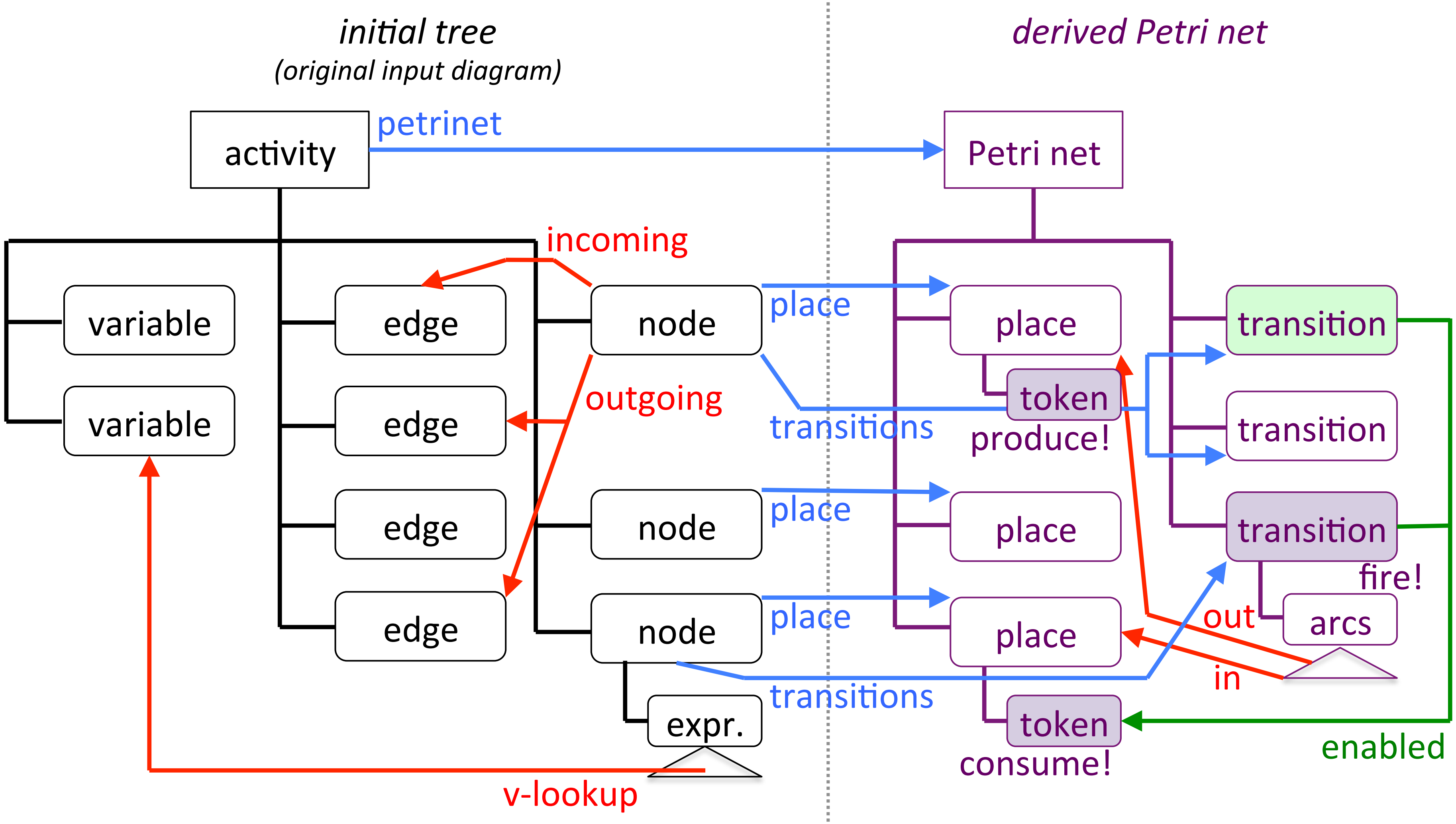
reuse

ASG



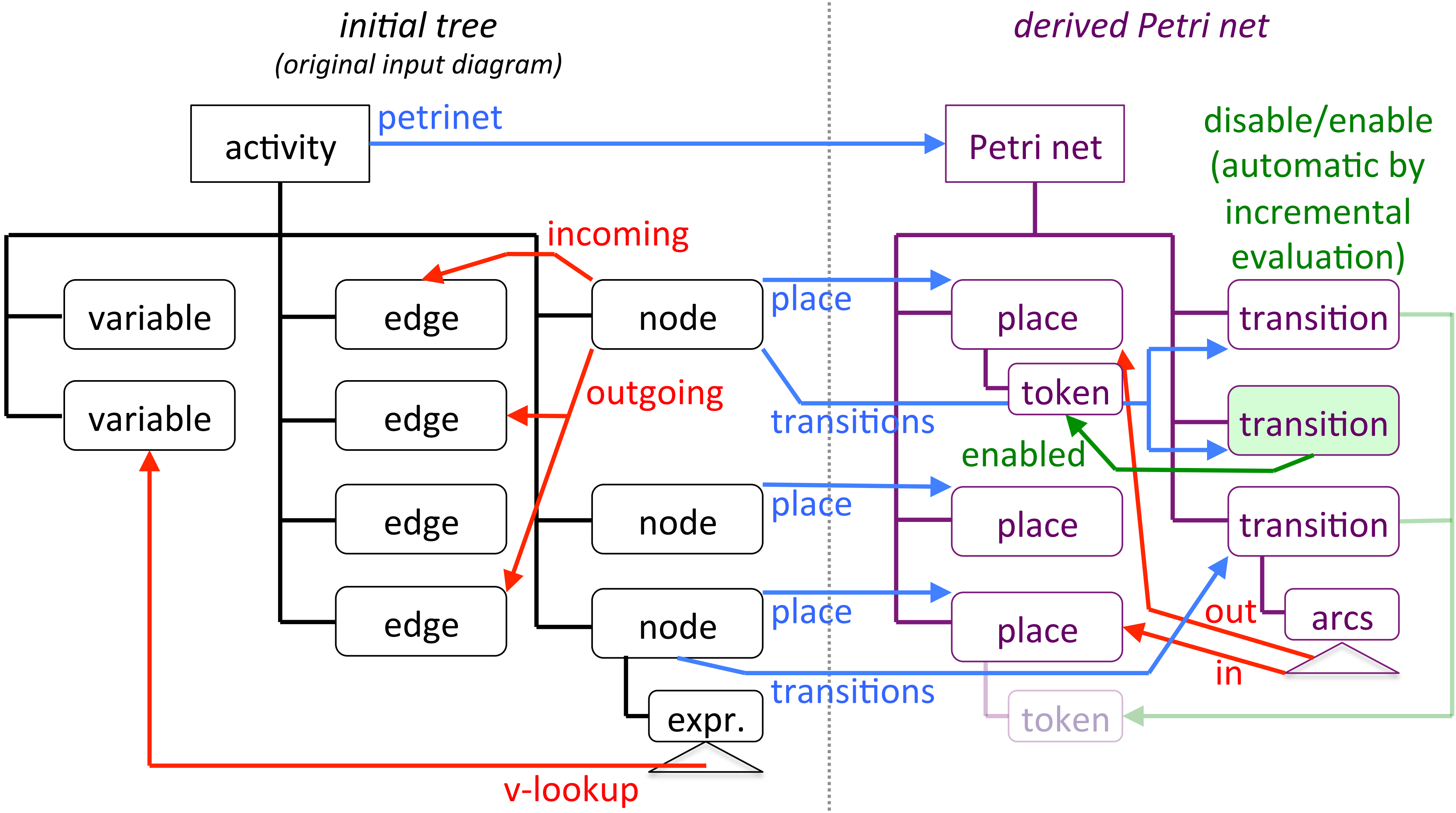
semantic overlay graph (excerpt): name analysis code generation enabled analysis

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

Implementation quality

- straightforward rewriting thanks to attribute-based analysis (rewrites leverage on analyses)
- focused rewriting (just actual state changes)
- efficient, although naïvely specified (incremental)
- declarative (automatic deduction of evaluation orders for intertwined analyses & rewriting)
- interactive (convenient runtime API for user-driven analyses & state changes)

Lines of code

Source code file	Solution part (language task)	LOC	
Activity Diagram interpreter (584):		*548*	
`analyses.scm`: *308*	AST specification	16	3%
	ASG accessors (constructors, child & attribute accessors)	89	16%
	Name analysis	36	7%
	Type analysis	21	4%
	Well-formedness	30	5%
	Petri net generation	94	17%
`parser.scm`: *234*	Parsing	229	42%
`user-interface.scm`: *42*	Initialisation & execution	33	6%
Petri net interpreter (243):		*222*	
`analyses.scm`: *134*	AST scheme	6	3%
	ASG accessors (constructors, child & attribute accessors)	34	15%
	Query support	12	5%
	Name analysis	19	9%
	Well-formedness	12	5%
	Enabled analysis	38	17%
`user-interface.scm`: *109*	Initialisation and Petri net syntax	32	15%
	Running and firing interface	14	6%
	Read-eval-print-loop interpreter	23	10%
	Testing (marking & enabled status)	32	15%

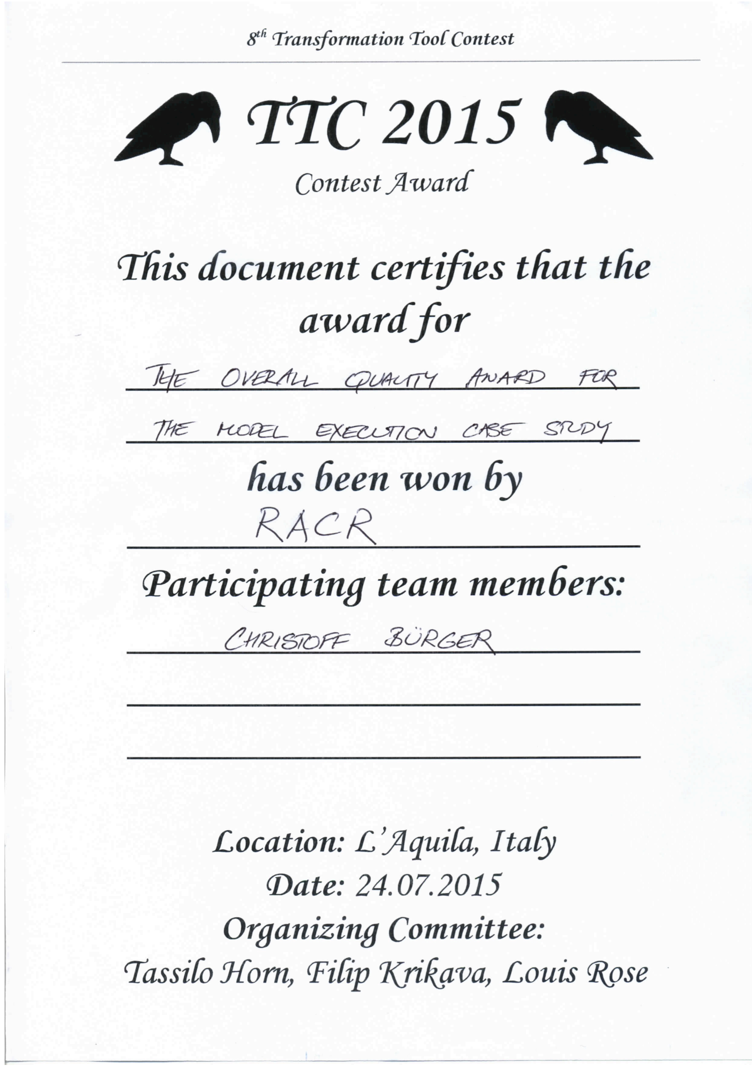
no further software artefacts

Performance

Tasks performed *(later include previous)*	Test case 1	2	3_1	3_2	Time spent *(low / high / average)*
Activity diagram parsing	762 / 762	763 / 763	797 / 797	641 / 641	45% / 92% / 53%
Activity diagram well-formedness	859 / 97	869 / 106	983 / 186	643 / 2	0% / 11% / 7%
Petri net generation	973 / 114	989 / 120	1125 / 142	647 / 4	1% / 8% / 7%
Petri net well-formedness	1141 / 168	1158 / 169	1296 / 171	655 / 8	1% / 11% / 9%
Petri net enabled	1167 / 26	1185 / 27	1376 / 80	656 / 1	0% / 5% / 2%
Petri net execution...	1617 / 450	1555 / 370	1768 / 392	699 / 43	6% / 28% / 22%
...using enabled passes	2274 / 1107	1229 / 44	1462 / 86	718 / 62	4% / 49% / 23%
Incremental savings (enabled analyses not cached)					*(low / high / average)*
Petri net execution...	9894 / 8727	8171 / 6986	8707 / 7331	916 / 260	83% / 95% / 95%
...using enabled passes	18889 / 17722	1536 / 351	1818 / 442	1057 / 401	81% / 94% / 93%

execution times in ms
(cf. solution description)

Conclusion



8[th] Transformation Tool Contest

TTC 2015

Contest Award

This document certifies that the award for

THE OVERALL QUALITY AWARD FOR

THE MODEL EXECUTION CASE STUDY

has been won by

RACR

Participating team members:

CHRISTOFF BÜRGER

Location: L'Aquila, Italy

Date: 24.07.2015

Organizing Committee:

Tassilo Horn, Filip Krikava, Louis Rose

Benefits of RAG-controlled rewriting