A Semantic Approach to Modularizing SDN Software

Anduo Wang (adw@temple.edu), Temple University

motivation

today, the onus of coordinating SDN software falls on the admin to write modular programs

- modularization prefixed in specific programming constructs varies from one DSL to another
- manual (control flow) composition relies on the internalized knowledge of experienced admin

making modularization an architectural primitive of systems organized around data flow

- data-flow modularization architecture in pre-SDN era
  - x-Kernel, Click, XORP, layering, declarative networking …
- data-flow architecture for SDN?
  - SDN building blocks (modules) are extremely flexible and keep evolve — conflicts among modules?
  - interact in arbitrary ways — not terminate?

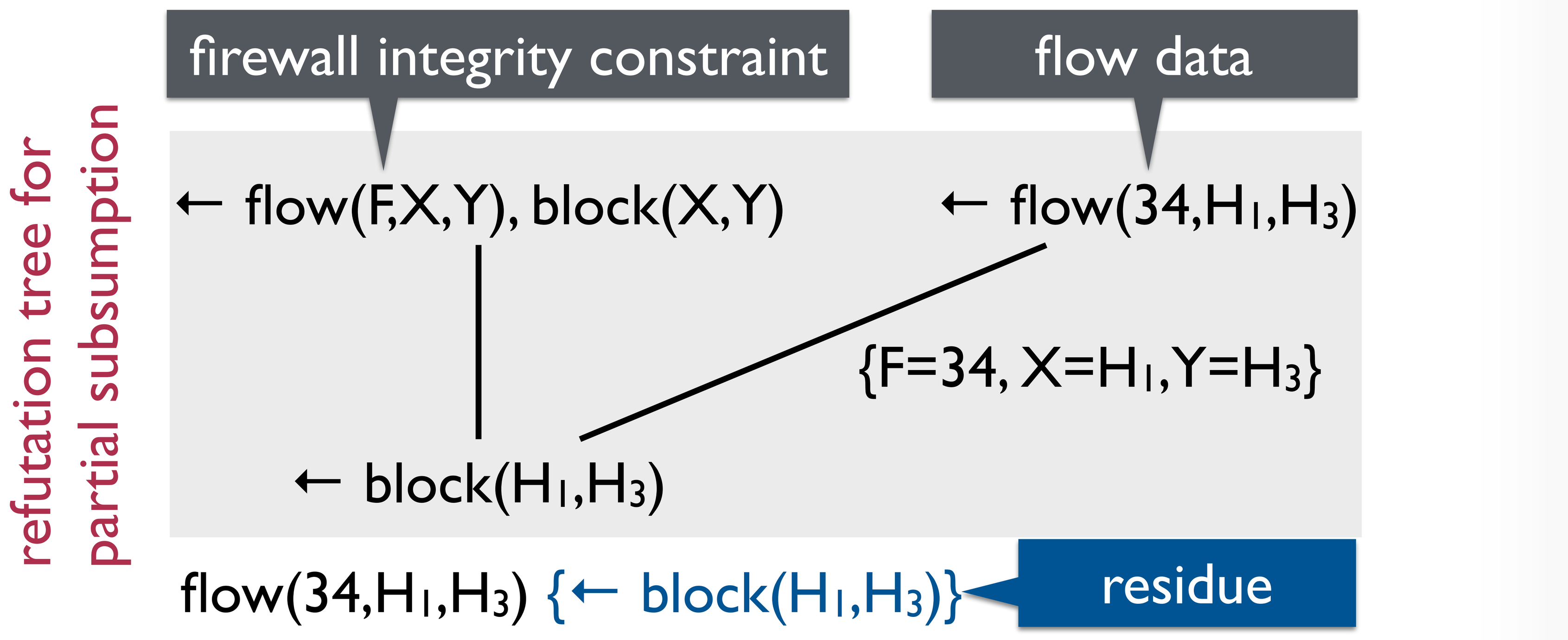
towards a modularized data-flow architecture

- software as semantic units that maintain properties
- consistency / termination by semantic analysis

background: residue method

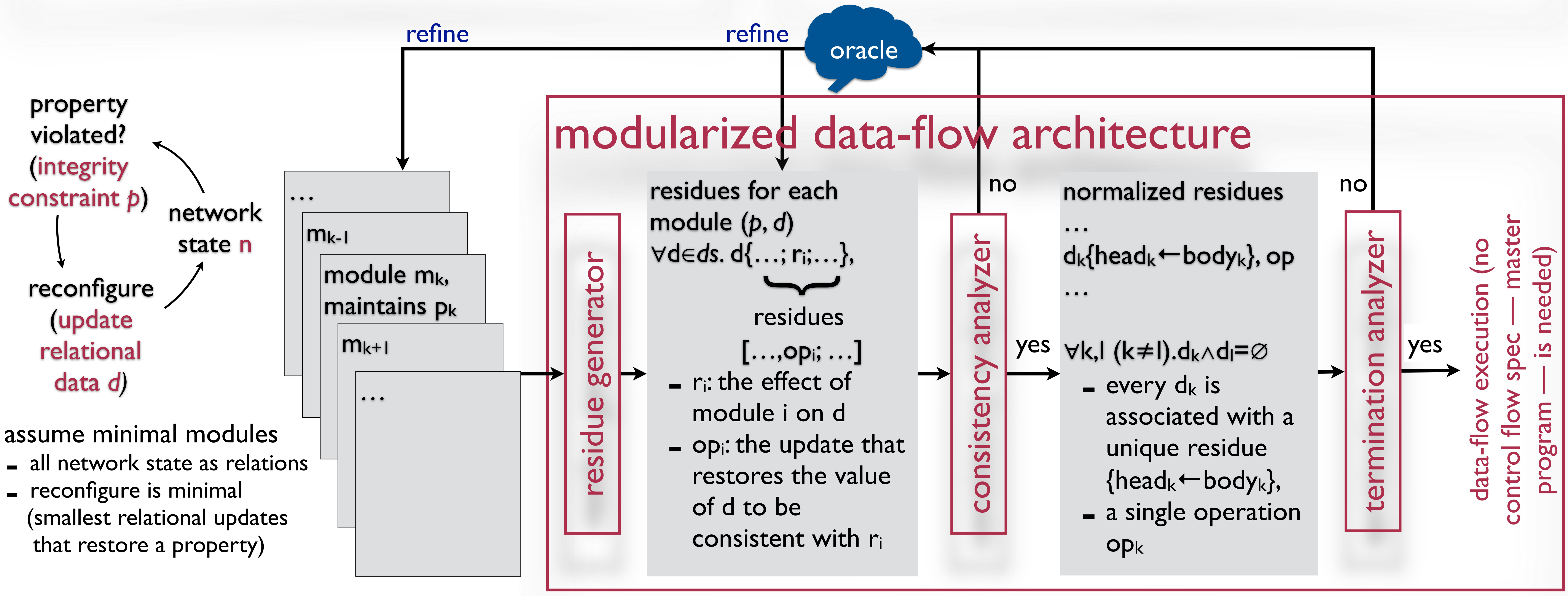
residue represents the integrity constraint's effect on the network data

- integrity constraint as the subsuming clause
- network state (negated relation) as the subsumed clause
- the fragment at the bottom of refutation tree



- residue represents the firewall's effect on network flow flow(34, H_1,H_3) is allowed only if ¬block(H_1,H_3) (when compliant)

modularized data-flow architecture



assume minimal modules

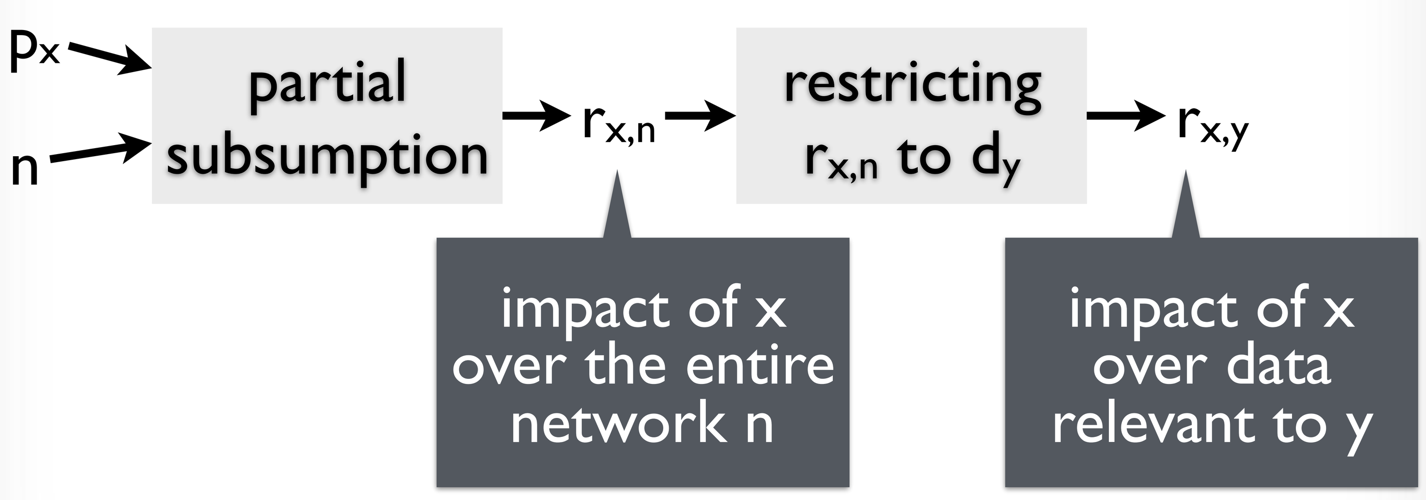
- all network state as relations
- reconfigure is minimal (smallest relational updates that restore a property)

residue generator

given modules x,y

- p_x : integrity constraint imposed by x
- d_y : (derived) network data updated by y
- n : (underlying) network data shared by all

find residue $r_{x,y}$ — x's impact over y



consistency and termination analyzer

consistency

- modules agree on one single unique operation over the network (data)
  - when modules are minimal, their updates are characterized by their residues — logical spec of what constitutes valid network state (after updates)
- analyze consistency by residues: modules are consistent if
  - either all module's residues are redundant (always evaluates to true, or maximal) — no interaction between the module and d — except one module's residue
  - or, any residues that are not redundant are logically equivalent

termination

- the triggering graph among modules are acyclic
- derive triggering relations by analyzing the modules' residues
  - x (d_x\{$head_x$←$body_x$\}) triggers y (d_y\{…\}) if $head_x \cap d_y \neq \varnothing$

example: consistency analysis with residues