2011-07-21

Safe Compositional Network Sketches: NetSketch Tool Implementation

Soule, Nate

http://hdl.handle.net/2144/1430

Boston University
Safe Compositional Network Sketches:
NetSketch Tool Implementation
Nate Soule  Azer Bestavros  Assaf Kfoury  Andrei Lapets

Overview

Numerous problems exist that can be modeled as traffic through a network in which constraints exist to regulate flow. Vehicular road travel, computer networks, and cloud based resource distribution, among others all have natural representations in this manner. As these networks grow in size and/or complexity, analysis and certification of the safety invariants becomes increasingly costly.

The NetSketch formalism introduces a lightweight verification framework that allows for greater scalability than traditional analysis methods.

The NetSketch tool was developed to provide the power of this formalism in an easy to use and intuitive user interface.

NetSketch Formalism

- A type-theoretic framework for network analysis
- Domain Specific Language for describing constrained flow networks
- Allows for compositional (vs whole-system) analysis
- Ability to analyze networks that contain unknowns
- Customizable analysis granularity levels
- Types, akin to those in programming languages applied/inferred to subgraphs of network
- Typed modules can be abstracted, with inner topology/constraints no longer required for analysis

- Using the NetSketch formalism networks can be described in a conceptually consumable manner, and efficiently composed and analyzed for satisfaction of safety constraints.

NetSketch Tool

The NetSketch Tool was developed to expose the formalism in a user friendly, accessible environment.

- Allows creation of networks through definition of nodes, links, and constraints over those links
- Exposes a library of pre-built node types for rapid development of networks
- Allows for the creation of new arbitrary node types for custom requirements
- Generates interval based typings on user selected subgraphs of the network

- The tool operates on top of a domain specific language, and type inference/generation engine, but exposes this power in a simple and intuitive graphical user interface.

- Users can:
  - Drag and drop nodes in the network
  - Visually draw links between nodes
  - Point-and-click to apply constraints
  - Select a sub-graph for type generation by dragging nodes onto a type generation panel
  - Select common nodes from a palette of pre-configured options

- The linear constraints form a convex hull in n-dimensional space representing the feasible values.

Type Generation

Each node may have

- A set of ports allowing flow to enter
- A set of ports allowing flow to exit

Constraints may be defined at each of these ports as a set of linear equalities/inequalities. Generating types from sets of untyped gadgets involves transforming the linear constraints on ports into intervals over the real numbers.

Input Type Generation

- Requires determining a maximally enclosing hyper-rectangle.
- Unique without further user input.

Output Type Generation

- Requires determining a maximally enclosing hyper-rectangle.
- Unique without further user input.

Architecture

Two-tier client/server architecture:

- Client tier: 100% JavaScript/HTML
  - JavaScript graphics (JSGL) and widget (ExtJS) libraries provide cross-browser compatibility of visual components
  - HTTP AJAX communication with back-end processes
- Server tier: Haskell and C++
  - A Haskell based server accepts and parses client requests, traverses and manipulates networks, and invokes type generation engine
  - Recognizes and accepts the domain specific language of the NetSketch formalism, thereby completely decoupling the core of the processing from any particular client
  - A set of C++ based libraries to perform the heavier mathematical computations (Simplex, vertex enumeration, constraint dimensional projection)

Future Work and Direction

- Extending the tool to allow for full feature set, such as definition and type inference of network "holes" (unknowns)
- Work is currently underway on a variation of the formalism that will allow input type generation to occur in an optimally unique way while still allowing representation of real-world applicable networks