Graph-Based State Spaces

Arend Rensink
University of Twente

Graph formalism
- Graphs in this presentation:
  - flat (i.e., not hierarchical), untyped
  - directed, edge-labelled, no parallel edges
  - self-edges depicted as node labels
- Formally: \( G = (L,N,E) \) with
  - \( L \) set of labels
  - \( N \) finite set of nodes
  - \( E \subseteq N \times L \times N \) finite set of labelled edges
- Partial morphisms
  - structure-preserving node mappings

Graphs as states
Graph Productions

Production rule

Graph transition

(SPO = Single Pushout Approach)

Example production rule

```java
public void put(Object val) {
    if (last.next.val == null) {
        last = last.next;
        last.val = val;
    }
}
```

Example rule application

Graphs as states

Transitions carry partial morphisms

Not inverse!
Aim: software model checking

- Construct graph production system from
  - UML diagrams / other specifications
  - Programs to be checked
- Generate state space
  - States=graphs, transitions=transformations
- Formulate properties
  - invariants/reachability (safety)
  - liveness
  - full temporal logic
- Check properties on the model

Example cases [GraBaTs 2004]

<table>
<thead>
<tr>
<th>Case</th>
<th>states (@)</th>
<th>transitions (@)</th>
<th>time (s)</th>
<th>space (MB)</th>
<th>node count (avg)</th>
<th>edge count (avg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>List append</td>
<td>31104</td>
<td>16658</td>
<td>212</td>
<td>13.9</td>
<td>37.7</td>
<td>113.8</td>
</tr>
<tr>
<td>List append</td>
<td>32903</td>
<td>277634</td>
<td>199</td>
<td>24.8</td>
<td>20.0</td>
<td>55.1</td>
</tr>
<tr>
<td>List append</td>
<td>262054</td>
<td>620284</td>
<td>162</td>
<td>88.7</td>
<td>5.1</td>
<td>14.3</td>
</tr>
</tbody>
</table>

- List append: highly dynamic, hardly symmetric
- Philosophers: not at all dynamic, highly symmetric
- Ring mutex: somewhat dynamic, rather symmetric

Envisaged tool chain

- State properties
- Semantic rules
- Program
- Specification

Issues to be addressed

- Time consumption (complexity)
  - graph matching
  - isomorphism
- Space consumption (memory usage)
  - state and transition storage
  - symbolic techniques (BDDs) not applicable
- Problem size
  - state size not a priori fixed (generally unbounded)
  - state spaces generally infinite
- Propositional logic not suitable
- Model checking algorithms not suitable
- Verification not generic (problem size 4, 5, ...)

= implemented
= planned
**Time consumption (1)**

- **Graph matching**
  - Needed to find production rule matchings
  - Complexity: NP-complete
- **Alleviating circumstances**:
  - Graphs to be matched are LHSs
    - Typically small
  - Host graphs are software models
    - Mostly deterministic
    - Transformations only at "locus of control"

**Time consumption (2)**

- **Graph isomorphism**
  - Used to collapse states
  - Complexity: between P and NP (!)
- **Approximation techniques**
  - Over-approximation: graph certificates
    - Excellent precision (> 99%)
    - Still requires isomorphism check afterwards
  - Under-approximation: equality
    - Mediocre precision (10-50%)
    - Very fast; useful as initial filter

**Issues to be addressed**

- **Time consumption (complexity)**
  - Graph matching
  - Isomorphism
- **Space consumption (memory usage)**
  - State and transition storage
  - Symbolic techniques (BDDs)?
- **Problem size**
  - State size not a priori fixed (generally unbounded)
  - State spaces generally infinite
- **Propositional logic not suitable**
- **Model checking algorithms not suitable**
- **Verification not generic (problem size 4, 5, …)**
**Space consumption**

- Symbolic methods (BDDs) not suitable
  - No fixed state vector
  - Idea: Store "deltas" between graphs
  - Average delta: 2-7 elements
- Transition storage also expensive
  - Idea: Store "boundaries" of LHS matching
  - Average boundary: 2-3 elements
- Current implementation:
  - Overhead per state/transition > 75%
  - Java quite memory generous

**Issues to be addressed**

- Time consumption (complexity)
  - graph matching
  - isomorphism
- Space consumption (memory usage)
  - state and transition storage
  - symbolic techniques (BDDs) not applicable
- Problem size
  - State size not a priori fixed (generally unbounded)
  - State spaces generally infinite
- Propositional logic not suitable
- Model checking algorithms not suitable
- Verification not generic (problem size 4, 5, ...)

**State space reduction (1)**

- Existing techniques:
  - Symmetry recognition
  - Partial order reduction
  - Abstraction, e.g. slicing (property-driven)
- Symmetry recognition: here automatic
  - Implied by isomorphism check
  - Dining philosophers: linear reduction
  - Expectation: little symmetry in real life

**State space reduction (2)**

- Partial order reduction
  - Linearization of confluent rule applications
  - Theory:
    - Exponential "best case" improvement
    - Restricted applicability, especially with NACs
    - Practice: ???
- Abstraction
  - Approximative results (false negatives)
  - Very promising, not just for this purpose
Experimentation (1)

Dining philosophers
- get hungry
- get left fork, get right fork (in sequence)
- drop both forks (atomically) and think

<table>
<thead>
<tr>
<th>#phils</th>
<th>#states</th>
<th>#trans</th>
<th>space (MB)</th>
<th>time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>117</td>
<td>481</td>
<td>0.1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>3,261</td>
<td>2,1536</td>
<td>2.9</td>
<td>19</td>
</tr>
<tr>
<td>10</td>
<td>32,903</td>
<td>27,1634</td>
<td>24.8</td>
<td>199</td>
</tr>
<tr>
<td>12</td>
<td>347,337</td>
<td>3,449,980</td>
<td>267.0</td>
<td>3,712</td>
</tr>
</tbody>
</table>

Comparison [ICGT 2004]

- CheckVML (Varró)
  - Encode graphs in SPIN
  - Choose fixed node identities
  - Predict rule applications

<table>
<thead>
<tr>
<th>#phils</th>
<th>#states</th>
<th>#trans</th>
<th>space(MB)</th>
<th>exec(s)</th>
<th>prep(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>3,261</td>
<td>2,1536</td>
<td>2.9</td>
<td>19</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>32,903</td>
<td>27,1634</td>
<td>24.8</td>
<td>199</td>
<td>7</td>
</tr>
<tr>
<td>12</td>
<td>347,337</td>
<td>3,449,980</td>
<td>267.0</td>
<td>3,712</td>
<td>10</td>
</tr>
</tbody>
</table>

Issues to be addressed
- Time consumption (complexity)
  - graph matching
  - isomorphism
- Space consumption (memory usage)
  - state and transition storage
  - symbolic techniques (BDDs) not applicable
- Problem size
  - state size not a priori fixed (generally unbounded)
  - state spaces generally infinite
- Propositional logic not suitable
- Model checking algorithms not suitable
- Verification not generic (problem size 4, 5, …)

Property specification
- State-based properties
  - Invariants, liveness properties
  - Expressible by graph predicates
  - Mechanism: graph embedding (+ NACs)
- Temporal logic properties
  - Existing MC logics are propositional (L/CTL)
  - Graph properties are FOL formulae
  - Dynamic allocation/deallocation
Graph Temporal Logic
- Navigation using regular expressions
  \[ \text{path ::= a | path}.path | path* \]
- Second-order expressions for node sets
  \[ \text{set ::= Z | x | set}.path | \text{set for } \exists x: x \in \text{set} \]
- Linear temporal logic with predicates
  \[ \text{form ::= } x \in \text{set} | \neg \text{form} | \text{form} \land \text{form} \]
  \[ | \forall x: \text{form} | \text{let Z=set in form} | X \text{form} | \text{form U form} \]

Example properties
- The buffer is circular
  \[ \forall n \in \text{Cell}: n \in n.next^+ \]
- Cell values are unchanged until consumed
  \[ G(\forall n \in \text{Cell}: \forall x \in \text{val}: x) \]
- Values are consumed in-order
  \[ G(\forall n \in \text{Cell}: n.next.val \Rightarrow (n.next.val U ! n.val)) \]
- New values are created all the time
  \[ G(\text{let Z=val in } F(\exists x \in \text{val}: x \notin \text{Z})) \]

Issues to be addressed
- Time consumption (complexity)
  - graph matching
  - isomorphism
- Space consumption (memory usage)
  - state and transition storage
  - symbolic techniques (BDDs) not applicable
- Problem size
  - state size not a priori fixed (generally unbounded)
  - state spaces generally infinite
- Propositional logic not suitable
- Model checking algorithms not suitable
- Verification not generic (problem size 4, 5, ...)

Model checking algorithms
- More expressiveness means less decidability/higher complexity
- Initial ideas: [FSTTCS 2004]
  - With Distefano & Katoen
    - No edges (multisets of entities)
    - Single outgoing edge
**Issues to be addressed**

- Time consumption (complexity)
  - graph matching
  - isomorphism
- Space consumption (memory usage)
  - state and transition storage
  - symbolic techniques (BDDs) not applicable
- Problem size
  - state size not a priori fixed (generally unbounded)
  - state spaces generally infinite
- Propositional logic not suitable
- Model checking algorithms not suitable
- Verification not generic (problem size 4, 5, ...)

**Abstract interpretation**

- Method consists of:
  - Concrete TS: \((S_c, \rightarrow_c, ic)\)
  - Abstract TS: \((S_a, \rightarrow_a, ia)\)
  - Abstraction function \(\alpha: S_c \rightarrow S_a\) with \(\alpha(\text{ic}) = \text{ia}\)
    - Sound: \(s_c \rightarrow s'_c\) implies \(\alpha(s_c) \rightarrow \alpha(s'_c)\)
    - Weakly complete: \(s_a \rightarrow s'_a\) implies \(s_c \rightarrow s'_c\)
    - for some \(s_c \in \alpha^{-1}(s_a), s'_c \in \alpha^{-1}(s'_a)\)
      (\(\alpha\) is a surjective simulation/homomorphism)
- Property reflecting:
  - \(\alpha(s_c) \models \varphi\) implies \(s_c \models \varphi\) for \(\varphi\) in an appropriate logic
  - not vice versa: verification is approximative

**Abstraction research programme**

- Define graph abstraction
  - Automatically computable
  - Property reflecting
- Lift graph transformations
  - Define effect directly on abstract graphs
- Develop general theory
  - Basic principles to apply to any GT approach
  - Wanted: Algebraic justification

**Graph abstraction [ESOP 2004]**

- First
- List
- Cell
- Val
- Object
- Unshared
- Unused
- Shared

- No nxt
- Nxt
- Cell
- Val
- Object
- Unshared
- Unused
- Shared
**Enriching abstract graphs**

- The following information is added:
  - The (potential) number of node instances
  - The (potential) degree of sharing (in+out)
- Both can be expressed as multiplicities
- Strongly inspired by *shape graphs*
  - Sagiv, Reps, Wilhelm, Benedikt

**Pictorial representation**

- Write edge multiplicities at "ports"
  - Node multiplicities
  - Outgoing edges
  - Incoming edges

**Abstract graph transformation**

- Materialization
  - Matching of left hand side made concrete
  - Result: partially concrete graph
- Transformation
  - Partially concrete graph treated as fully concrete
- Normalization
  - Transformation result is partially concrete
  - Re-apply abstraction principle
What you should take home

• Graphs as states: promising model

• Some inherent benefits
  – Captures dynamic behaviour
  – Implicit symmetries
  – Allows structural abstraction

• Some inherent disadvantages
  – Infinite state space
  – Increased complexity in several issues

• A lot of open issues