See the Forest for the Trees – How to Effectively Model and Randomize a DRT Structure

Harry Duque and Lars Viklund
Graphs and Trees

• Directed Rooted Tree (DRT)
  • Connected
  • Oriented
  • Acyclic
  • Each vertex has a single parent
  • Single start point, a root
  • All edges point away from root
Highly Configurable Datapath Unit
Modelling of Connectivity
Challenges

<table>
<thead>
<tr>
<th>Validity</th>
<th>Controllability</th>
<th>Performance</th>
<th>Scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Rules</td>
<td>• Steer</td>
<td>• Solver</td>
<td>• Vertices</td>
</tr>
<tr>
<td>• Constraints</td>
<td>• Direct</td>
<td></td>
<td>• Entry</td>
</tr>
</tbody>
</table>
How hard can it be?

- We have excellent support from SV language
- We can make use of complexity reduction techniques
- Number of re-use mechanisms available
See the Forest from the Trees

• Tree has a recursive, self-similar shape
• Recursion
• Search problem
• Backtracking
Model of Solution Space

- SRC_<*>: A primary input
- DC_<*>: A converter sub-unit
- SAMP_<*>: A sampler sub-unit
- Helper START and STOP vertices
A possible solution

- **SRC_<*>**: A primary input
- **DC_<*>**: A converter sub-unit
- **SAMP_<*>**: A sampler sub-unit
- Helper START and STOP vertices
Model of Solution Space in Constraints

```java
class vertex extends vertex_base#(id_t, config_t);

constraint c_connectivity {
    this.id inside {START} -> this.child_id inside {SRC_A, SRC_B, SRC_C, SRC_D};
    this.id inside {SRC_A} -> this.child_id inside {DC_X, DC_Y, SAMP_0, SAMP_1, SAMP_2};
    this.id inside {SRC_B} -> this.child_id inside {DC_X, DC_Y};
    this.id inside {SRC_C, SRC_D} -> this.child_id inside {DC_X, DC_Y, DC_Z};
    this.id inside {DC_X} -> this.child_id inside {DC_Y, SAMP_0};
    this.id inside {DC_Y} -> this.child_id inside {DC_X, SAMP_0, SAMP_1};
    this.id inside {DC_Z} -> this.child_id inside {DC_X, DC_Y, SAMP_0};
    this.id inside {SAMP_0, SAMP_1, SAMP_2} -> this.child_id inside {STOP};
}
```
Backtracking

• Incrementally builds solution candidates
• Abandons potential candidates when not meeting criteria
• Guaranteed to produce a valid solution

function find(vertex v);
  if abandon(v) prune(v)
  if leaf(v) output(v)
  foreach children(v)[i]
    find(i)
endfunction : find
Building the Tree

- Separate allocator class
- Implements the different described functions for backtracking
- For implementation refer to the paper.

```python
function find(vertex v);
    if abandon(v) prune(v)
    if leaf(v) output(v)
    foreach children(v)[i]
        find(i)
endfunction
```

find
if (!current.randomize() with {!((this.child_id inside {local::withdrawn})||)}) begin
    this.withdrawn.push_front(current.id);
    return 0;
end

if (current.child_id == stop) begin
    this.withdrawn.push_front(current.id);
    return 1;
end

child = vertex_base#(id_t, config_t)::type_id::create();
child.init(current.child_id, current.id, this);
this.withdrawn.push_back(current.id);
ok = find_path(child, stop);
this.withdrawn.pop_back(current.id);
if (ok) begin
    this.withdrawn.push_front(current.id);
    current.add_child(child);
    return 1;
end
An Example
An Example

- Vertex under randomization - Successful
- Vertex under randomization - Failure
- Randomized child vertex

withdrawn = {}

this.id inside {START}  ->  this.child_id inside {SRC_A, SRC_B, SRC_C, SRC_D};
An Example

withdrawn = {}

this.id inside {START} -> this.child_id inside {SRC_A, SRC_B, SRC_C, SRC_D};
An Example

withdrawn = \{SRC_A\}

\text{this.id inside} \{SRC_A\} \rightarrow \text{this.child_id inside} \{DC_X, DC_Y, SAMP_0, SAMP_1, SAMP_2\}
An Example

withdrawn = \{SRC_A, DC_Y\}

this.id inside \{DC_Y\} \rightarrow this.child_id inside \{DC_X, SAMP_0, SAMP_1\};
An Example

withdrawn = \{SRC_A, DC_Y, SAMP_0\}

this.id inside \{SAMP_0, SAMP_1, SAMP_2\} -> this.child_id inside \{STOP\};
An Example

withdrawn = \{SRC_A, DC_Y, SAMP_0\}

this.id inside \{DC_Y\} -> this.child_id inside \{DC_X, SAMP_0, SAMP_1\};
An Example

 withdrawn = {SRC_A, DC_Y, SAMP_0, DC_X}

this.id inside {DC_X} -> this.child_id inside {DC_Y, SAMP_0};
An Example

\[
\text{withdrawn} = \{\text{SRC}_A, \text{DC}_Y, \text{SAMP}_0, \text{DC}_X\}
\]

\[
\text{this.id inside } \{\text{DC}_Y\} \rightarrow \text{this.child_id inside } \{\text{DC}_X, \text{SAMP}_0, \text{SAMP}_1\};
\]
An Example

withdrawn = \{SRC\_A, DC\_Y, SAMP\_0, DC\_X, SAMP\_1\}

this.id inside \{SAMP\_0, SAMP\_1, SAMP\_2\} -> this.child.id inside \{STOP\};
An Example

withdrawn = \{SRC_A, DC_Y, SAMP_0, DC_X, SAMP_1\}

this.id inside \{DC_Y\} -> this.child_id inside \{DC_X, SAMP_0, SAMP_1\};
An Example

withdrawn = {SRC_A, DC_Y, SAMP_0, DC_X, SAMP_1}

this.id inside {SRC_A}  ->  this.child_id inside {DC_X, DC_Y, SAMP_0, SAMP_1, SAMP_2};
An Example

withdrawn = \{SRC\_A, DC\_Y, SAMP\_0, DC\_X, SAMP\_1, SAMP\_2\}

this.id \text{ inside } \{SAMP\_0, SAMP\_1, SAMP\_2\} \rightarrow this.child.id \text{ inside } \{STOP\};
An Example

withdrawn = {SRC_A, DC_Y, SAMP_0, DC_X, SAMP_1, SAMP_2}

this.id inside {SRC_A} -> this.child_id inside {DC_X, DC_Y, SAMP_0, SAMP_1, SAMP_2};
An Example

withdrawn = \{SRC_A, DC_Y, SAMP_0, DC_X, SAMP_1, SAMP_2\}

this.id inside \{START\}  \rightarrow  this.child_id inside \{SRC_A, SRC_B, SRC_C, SRC_D\};
A Tree

```
SRC_A
 /    \
|     |
DC_Y  SAMP_2
 /      /
|       |
SAMP_0  SAMP_1
```
Results

• Validity
• Controllability
• Performance
• Scalability
Results

• Validity
• Controllability
• Performance
• Scalability

```java
class sampler0 extends vertex#(id_t, config_t);

constraint c_samp_0 {
    this.id inside {SRC_A} -> this.child_id inside {DC_X,
      DC_Y,
      SAMP_0};

    this.id inside {DC_X} -> this.child_id inside {SAMP_0};
    this.id inside {DC_Y} -> this.child_id inside {SAMP_0};
    this.id inside {DC_Z} -> this.child_id inside {SAMP_0};
}
...
```
Results

• Validity
• Controllability
• Performance
• Scalability
Summary

• Modelling of connectivity
• Recursive, backtracking randomization
• Criteria
• Generic framework
Q&A