Synthesis, Ladder-diagrams, Manufacturing

Sayan Mitra and ...
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
Email: {mitras}@illinois.edu

I. Introduction

Motivate with the need for fast adaptation in manufacturing systems. Adaptation with respect to what? Changes in demand, product and processing specifications (e.g., drill 3 holes instead of 2, a part needs to finish is shorter time), parts supplied (e.g., the new blocks are of different size). Failures. Replacement with slightly different machines and parts. Updates and patches.

Debugging ladder logic is easy (Filipe Lopez).

The synthesis problem addressed here: Given hardware configuration H, and given set of requirements R, our algorithm will automatically generate the ladder-logic program P, such that P running on H is guaranteed to meet the requirement R. Ideally, the algorithm will also give an output **Fail**, if it is impossible to create a program P that makes H meet R.

Variations of the above theme: generate program P that is optimal with respect to some metric, e.g., minimized processing time for a given path. Minimizes changes with respect to a previous program P' (patch P').

Promise of synthesis in adaptation: automatic generation of correct by construction programs.

II. Related Work

Synthesis for routers - Synthesis for SDN/network updates [1]. Main problem solved here is the avoidance of 2-phase updates. Key differentiator: data packets are mutable, i.e., packet headers can be used for routing.

- Arvind's work

Synthesis for traffic lights [2]. Temporal logic.

Lots of work on program synthesis and synthesis of controllers for mobile robots. These are probably less related.

Formal verification of DES model of manufacturing system [**?**].

III. Modeling a Manufacturing System

A. Manufacturing System Abstraction

A manufacturing system consists of a plant; a set of requirements detailing sequences of operations; a changing set of widgets moving through the plant; and a controller orchestrating the operations of machines and movement of widgets.

The plant of a manufacturing system can be thought of as a floor plan that details the layouts of machines and the connections between them. A plant is specified by a directed graph $G = (V, E)$ and a set of operations O.

Each vertex $v \in V$ models a *cell* that can perform a set of operations unto widget $o \subseteq O$. Cells represent either a single machine or a set of machines in the manufacturing floor plan. Additionally, there are special cells that correspond to sources, sinks, and forks. Sources have a single operation to produce new widgets, sinks have a single operation to consume widgets, and forks select the next conveyor a widget should be placed on when the path splits up.

Each $e \in E$ models a uni-directional conveyor with a specified length and speed. A controller may set the speed of the conveyor to a pre-defined speed or 0.

Let R be the set of requirements. Then R is a partially ordered set on O. It specifies the sequences of operations that are performed on widgets flowing through the system.

A widget w models a physical good that is moving through a manufacturing system. Widgets are specified a unique identifier *w.id*, a requirement *w.req* $r \in R$, a sequence of operations, that have to be performed on it, and a empty list *w.ops*. A widget is instantiated at a source cell and traverses G, the graph of the manufacturing system per the instructions issued by the controller. Whenever a widget visits a particular cell, an operation is performed on the widget and is appended to the *w.ops*. When a widget reaches a sink cell, *w.ops* is compared to *w.req* for equality and the widget is removed from the manufacturing system.

A controller for the plant is a program that controls the flow of widgets through G such that each widget's *w.req* is satisfied. A controller can do the following operations:

- Control the speed of conveyors (0 or max speed).
- Move widgets from cells to conveyors and vice versa.
- Order cells to perform operations.

B. Example

IV. Algorithms

A. Static

COMPLETE ME

B. Dynamic

Given G, the graph of the plant, and R, the set of requirements, the goal is to minimize some arbitrary cost function C. An example of such a cost function can be to maximize the throughput of widgets through the plant.

Algorithm 1 Dynamic Controller

```
input: $G = (V, E), R$
$G_r \leftarrow \{\}$
for $r \in R$ do
    if $\neg$ checkFeasibility($r, G$) then
        return
    end if
    $G_r \leftarrow G_r +$ generateDCG($r, G$)
end for

for $v \in V$ do
    $v.weight \leftarrow 0$
end for

while true do
    for $g_r \in G_r$ do
        dijkstra($g_r, C$)
    end for
    while $\neg \exists w$ exit cell do
        simulate($G$)
    end while
    for $v \in V$ do
        update($v.weight$)
    end for
end while
```

The algorithm for the controller takes in as input:

- G, the graph of the plant
- R, the set of requirements
- C, a cost function

For every requirement r in R, the algorithm uses the *checkFeasibility* function to check whether G can actually satisfy r or in other words, there exists a sequence of cells in G that can complete all operations indicated by any path in r. If r can be satisfied by G, then the *generateDCG* subroutine generates a DAG g_r which corresponds to all the paths in G that satisfy r. Each node in g_r references one of the nodes in G. Note that multiple nodes in g_r may reference the same node in G.

Next, the algorithm assigns weights to all the nodes in G according to C.

The algorithm then loops infinitely until a termination signal is sent to it or for a specified duration of time. In each iteration of the loop, Dijkstra's algorithm is run for each $g_r \in G_r$ to compute the shortest path from a source to a sink. The system is simulated until any widget in G completes in operations in a cell and is moved from a cell to a conveyor. Finally, the weights of all the nodes in G are updated according to C.

All subroutines detailed in this algorithm are described in the Appendix.

C. Correctness

The correctness of the algorithm depends subroutine *generateDCG* because each widget with requirement r traverses G according to a path in a corresponding $g_r \in G_r$. Each path in g_r specifies a path in G that satisfies requirement r which implies that any widget that traverses any path in g_r will have all its operations satisfied.

V. Preliminaries

A. Setup the manufacturing system

The overall system consist of a plant, a changing set of widgets moving through the plant and being operated on by the machines in the plant, and a controller orchestrating the movement and the operation.

Our *plant* is specified by a conveyance graph G and a set of operation labels L. A *conveyance graph* is an undirected $G = (V, E)$. Each vertex $v \in V$ is a *cell*. Each cell v is labeled by a subset $label(v)$ of operation labels L that can be performed by the machines in that cell. If $\ell \in label(v)$, then operation ℓ can be performed by some machine at cell v.

For example, one cell in the manufacturing floor could have a CNC machine A which could do operations A_1, A_2, A_3 and a lathe B which could do operations B_1, B_2. Then the vertex corresponding to this cell in the graph will be labeled by the set $\{A_1, A_2, A_3, B_1, B_2\}$. As we will see in Section **??**, when a widget appears in a cell, and the controller outputs a label ℓ for the cell, then the widget gets operated on by that label. That is, the state of the widget will change, specifically, a queue maintained in the widget will have ℓ enqueued.

There will be special cells corresponding to source, sink, and forks which do not perform operations. Sources have a single operation that produce a new widget; sinks have a single operation that consumes widgets, and forks have operations that decide the placement of widgets on (possibly multiple) conveyors.

Each edge in E corresponds to a conveyor. Conveyors may have length / capacity, and can be seen as a FIFO queue. Each conveyor can be controlled by the controller to move (in one of two directions).

A *widget* w is specified by a unique identifier $w.id$ and a list $w.op$ of labels indicating the sequence of operations performed on w. The list $w.op$ starts as empty, and as w visits a cell at which ℓ is performed by the controller, ℓ gets appended to $w.op$. We also have to define the position of w in G. (how?)

We have to define sensors: vector of positions or occupancy of all widgets in the system (?)

We have to define actuation: vector of commands sent to conveyors, machines, and distributors. For simplicity conveyors could be just 1,0,-1, for example.

Define a control program: its a finite state machine mapping sensor input, current state to the actuator outputs and updated state. This is the program that will be synthesized, as a ladder diagram.

B. Complete System

The complete closed-loop system is a discrete transition system (finite state) with the following components: The state has the following components:

(i) Set of widgets in the system, and the states of each widget;
(ii) State of the controller, including the output labels being produced.

The transitions

(i) Conveyors that are moving, move the widgets;
(ii) Widgets at the end of conveyors, get transferred to cells;
(iii) Cells doing operations update widget $w.op$;
(iv) Sensors update, controller computes new output;
(v)

A *requirement* is a partially ordered set on L.

C. *Example*

D. *Problem statement*

Simple manufacturing. A system is said to meet a requirement R, if for every widget w at a sink, $w.op$ is complete path in the requirement R.

Given a system specification and a requirement, synthesize the controller (ladder diagram) that meets the requirement.

VI. Synthesis Algorithm

A. *Analysis of Algorithm*

B. *Implementation*

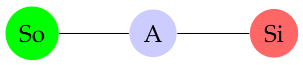
VII. Experiments

A. *Examples*

This section details three examples. Each example is defined by a graph, a list of operations performed the cells in the graph, and requirements that each widget must satisfy upon exiting the system.

1) Example 1: Single cell system

Graph G:



Cell operations:

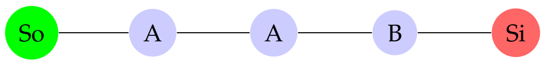
$L := a$

- $A := a$

Requirements R:

2) Example 2: Duplicate sequential system

Graph G:

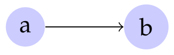


Cell operations:

$L := a, b$

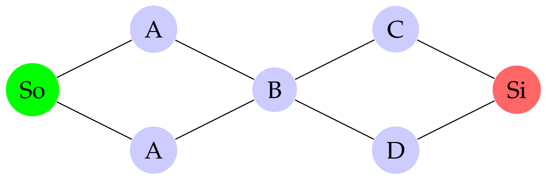
- A $:= a$
- B $:= b$

Requirements R:



3) Example 2: Forked system with duplicates

Graph G:

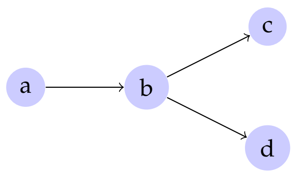


Cell operations:

$L := a, b, c, d$

- A $:= a$
- B $:= b$
- C $:= c$
- D $:= d$

Requirements R:



B. *Simulations*

Simulations in Factory I/O demonstrating performance of synthesized controllers.

C. *FischerTechnik*

Demonstration of automation.

References

[1] J. McClurg, H. Hojjat, P. Černý, and N. Foster, "Efficient synthesis of network updates," *SIGPLAN Not.*, vol. 50, no. 6, pp. 196–207, Jun. 2015. [Online]. Available: http://doi.acm.org/10.1145/2813885.2737980

[2] S. Coogan, M. Arcak, and C. Belta, "Formal methods for control of traffic flow: Automated control synthesis from finite-state transition models," *IEEE Control Systems*, vol. 37, no. 2, pp. 109–128, 2017.