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. 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.

IV. Synthesis Algorithm

A. Analysis of Algorithm

B. Implementation

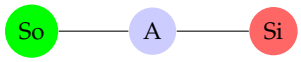
V. Experiments

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:

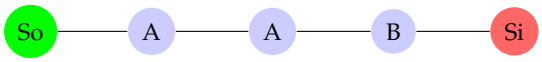
$L := a$

- $A := a$

Requirements R:

2) Example 2: Duplicate sequential system

Graph G:

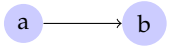


Cell operations:

$L := a, b$

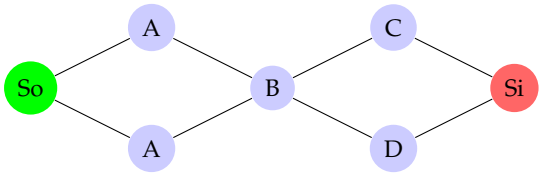
- A $:= a$
- B $:= b$

Requirements R:



3) Example 2: Forked system with duplicates

Graph G:

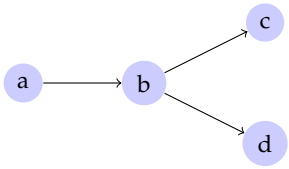


Cell operations:

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

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

VI. Questions/Notes/Etc

Potok:

1) In III, the system is defined as an undirected graph but are there any (dis)advantages to define it as a directed graph whose edges can change direction during system execution? (I will read related literature understand potential differences).
2) Positions of widgets can be tracked by the cells source they were generated, the cells they have visited, and the edges they have taken?
3) How high-level are we considering the systhesis to be? Would the algorithm generate the ladder logic for the individual cells or can we treat them as black boxes that have been verified?
4) Should sensors be considered to be cells as well with operations of scan/weight/detect? This wouldn't change the definitions too much and keep it largely the same.
5) The requirements will have to specify quite a few things to guarantee applicability to real-world. For example, the number of individual machines allowed on the floor.
6) I'm not familiar with synthesis algorithms so going out on a limb here. How do synthesis tools search the space of generating efficient systems? Just a random thought, but would using something like genetic algorithms make sense? After an mutation occurs, the algorithm could check paths satisfy the requirements to discard (potentially) searches.
7) As a starting point, it may be useful to not consider actuation and just have a directed graph(?)
8) Are there any guarantees on the system being fault tolerant or anything related to monitoring?
9) Would this require building a mini-simulator environment similar to the one discussed before factory-io?
10) In factory settings, there are machines that operate on items as they move along conveyors, how would such a system be modeled in this definition? Some ideas include: considering the external operating part as a single cell and edges leading into/out; consider the system as two conveyors with a cell that take 0 time to complete.
11) What type of manufacturing systems would this be targeting? From the definition it seems that widgets are processed sequentially on some path, but some factories process multiple widgets concurrently at essentially the same location on a path (think pea canning factory where a bunch of peas travel on line side-by-side).

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.