Truss: A Standard Verification Framework

CHAPTER 6

Truss, and verify.
Anon.

Have you ever watched a building being constructed? Early in the project, when the frame of the building is just a skeleton, it's not clear what the finished building will look like. However, as construction continues, from the windows down to the cubicles that are our workplaces, the intent of the framework becomes clear. In fact, a large part of the building's presence depends on the fundamental structure.

This same basic process occurs when we build a verification system. Early in the project, the application framework is built. The result of years of best practices from both the verification and software fields, Truss is an application framework for verification. It is an *implementation*, and therefore makes some decisions about how things should be structured. With verification as with construction, the framework sets the tone for the system.

Truss is a layered architecture, so you can choose how to implement the layers. Although it makes very minimal assumptions, Truss does provide some base classes and conventions as a guide.

Overview

This chapter presents three main topics:

- The roles and responsibilities of the various major Truss components
- How these components work together
- How you adapt this framework for your verification system

This chapter builds on the two previous chapters of the handbook. It implements an open-source verification infrastructure based on the discussion in the Layered Approach chapter. It also uses the Teal library described in the last chapter as a connection between C++ and the simulation.

Teal provides the fundamental elements of a verification system and supports a wide array of methodologies. Truss, on the other hand, provides the infrastructure layers above Teal, adding a set of classes, templates, idioms, and conventions to facilitate the construction of an adaptable verification system.

One of the tricks in building a reasonable system is to find the *key algorithm*. The rest of the algorithms can usually fit around that key algorithm. For example, in a video editing program the key algorithm is all about getting the pace of the edits right. When you watch a movie, that happy, sad, or scared feeling you get comes from how well-timed and precise the changes in scene are.[1] The authors, having developed software for video editing systems, know that in this domain the key algorithm is implemented by adjusting the edit points of a few seconds of video while the video is constantly looping around those edits. This is not a trivial thing to do, because multiple streams of video and audio,

1. Okay, emotions also come from the music, but everything works together.