AssemblyHierarchical – Connecting Devices through Cables

Gregor Behnke
University of Freiburg
Freiburg im Breisgau, Germany
behnkeg@informatik.uni-freiburg.de

Abstract

We report on the AssemblyHierarchical domain, which encodes the task of connecting devices through a set of availables cables.

Introduction

We report on the idea and structure behind the domain AssemblyHierarchical, which was part of the benchmark set of the total order track of the IPC 2020. The AssemblyHierarchical domain was inspired by the Assembly Assistant (Bercher et al. 2014; 2015; 2017; 2018) developed in the Transregional Collaborative Research Centre SFB/TRR 62 "Companion-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG). The assistant was designed to help a (usually novice) user to set up his or her home entertainment system. In this setting, there are multiple signal sources (satellite receiver, VCR, DVD player, ...) as well as multiple signal sinks (TV, speaker, ...). These devices need to be connected using only the available cables and intermediate devices (e.g. amplifiers). Adapters might have to be used to connect a given cable with a device. The assistant determines – based on the available cables and devices – how the cables should be used and instructs its user to plug them into the appropriate devices.

The assistance system implemented within the SFB/TRR 62 used planning to determine how to connect cables to devices. The AssemblyHierarchical domain is more general than the original one used by the assistant in the sense that it describes any arbitrary flow of an opaque signal from one device to another – this can be a video, audio, network, USB, or any other type of signal. The original domain was rather restrictive in the allowed operations and plans. The difficulty in modelling is due to the duality of the problem in this setting – plugging in cables *and* transmitting a signal – which can only be easily modelled using e.g. derived predicates, which in turn are computationally expensive. The problem stems from situations as follows. Consider four devices A, B, C, and D and cables connecting A and B, B and C, and C and D. To transmit a signal from A to D we can plug in these cables in any order. This signal is at any time only transmitted as far as the outgoing connection of A is plugged in. Notably, it is possible to plug in the cable from A to B last which will instantaneously cause the signal to be available at D. As noted before, this can be modelled with derived predicates or additional actions for signal transmission. The latter becomes much more complicated if we also allow for cables to be unplugged since signals also have to be "un-propagates". The AssemblyHierarchical domain we present in this paper actually supports unplugging cables.

Hierarchical planning is rather well suited to model this task. We can use a generator-style recursion of the HTN to allow for cables to be plugged in. The last decomposition of this generator will result in an abstract task that checks whether the connection has actually been established. This can be done solely via method preconditions.

Domain

As all other domains in the IPC, the AssemblyHierarchical domain is formulated in HDDL (Höller et al. 2020). We distinguish three types of `AbstractDevices`: `Devices` (representing larger devices), `Cables`, and `Adapters`. Each `AbstractDevice` has a number of `Ports`, which have a `PlugFace` (either male or female) and a `PlugDirection` (in, out, or both). A `Port` describes any point of an `AbstractDevice` that can be connected to anoter `AbstractDevice`. For example a `Port` of a TV might be connectable to the end of a cable – which is also a `Port`. A male port can only be connected to a female port. Further, an in-port can only be connected to an out- or both-type port and an out-port only to an in- or both-type port. Each `Port` further has a `PlugType` and can only be connected to ports of the same `PlugType`.

To connect ports, the AssemblyHierarchical domain has `connect` actions. Since the IPC 2020 does not allow for arbitrary formulae in preconditions, there are in total eight versions of the `connect` actions (named `connect_1` to `connect_8`). Given two ports `?p1` and `?p2` they represent the different possible configuration regarding the `PlugFace` and `PlugDirection` of the two ports. The domain contains an abstract task `connect` that can be decomposed into any of the eight concreate connect actions. The domain further contains a `disconnect` action that disestablishes an existing port connection.

The remainder of the AssemblyHierarchical domain features just three further abstract tasks: `ConnectDevices`, `ValidateDeviceConnection`, and `ValidatePortConnection`. `ConnectDevices` ensures that a signal can be transmitted between the two `AbstractDevices` `?d1` and `?d2` which are its parameters. In order to do so, it first allows – via methods – to generate an arbitrarily long sequence of `connect` and `disconnect` tasks. This recursion is ended with a method adding the action `guard` to the plan (for an explanation of this action see below). The method ending the recursion also inserts an instance of the abstract task `ValidateDeviceConnection` with the two arguments `?d1` and `?d2`. This task starts the validation of the (hopefully) established connection between `?d1` and `?d2`. To this end, it has to select an (outgoing-)port `?p1` of `?d1` and an (ingoing-)port `?p2` of `?d2`. These two ports are then passed on as arguments to the only subtask: `ValidatePortConnection`. This task validates via a recursive decomposition that there is infact a path from `?p1` to `?p3` via properly connected cables. If so, the last decomposition method simpy adds the action `ok`, which makes the goal fact `pAim` true.

Since the AssemblyHierarchical domain was only used in the total order track and all methods in the domain are totally ordered, the validation of the connection always happens after all cables have been plugged in. This would however not be the case if the initial plan was to contain multiple `ConnectDevices` actions – as disconnect actions of the second, if ordered after the first in the initial plan – might invalidate the connection established by the first. If multiple devices shall be connected, the respective `ConnectDevices` tasks must therefore be partially ordered in the initial plan. This however also not gurantees that connections are only validated after the last `connect` or `disconect`. For this purpose, the `guard` action adds the fact `pGuard` to the state. All `connect` and `disconect` actions have `(not (pGuard))` as their preconditions and the `ValidateDeviceConnection` task always occurs strictly after the `guard` action. This way, validation always happens after all `connect` and `disconnect` actions.

Instances

In the IPC 2020, only totally ordered instances of the AssemblyHierarchical domain were included in the benchmark set. As such, each instance contained only a single task in its inital plan, namely one `ConnectDevices` task for two specific devices. Each instance is solely described by a natural number i. Each instance has only two true devices called `pc` and `printer`. The instance numer i contains i additional cables, called `cableWithPlugTypeX` where `X` $\in \{1, \dots i\}$. Each cable has two bi-directional ports. One port of cable one is male and fits into the `pc`'s sole port while one port of the last cable is also male and fits into the `printer`'s sole port. Apart from that, cable i always has a port with which it can be connected to cable i, i.e. every cable has exactly two ports. There are also i `PlugTypes` named `plugTypeX` for `X` $\in \{1, \dots i\}$. Both of the ports of cable i have plug type i. The `pc`'s sole port always has plug type 1 and the `printer` always has plug type i. For each $j \in \{1, \dots, i-1\}$ there is an adapter called `adapterFromPlugTypeXToPlugTypeY` (with `X` $= j$ and `Y` $= j + 1$) which as two ports: one of type j and one of type $j + 1$.

With this setup there is only a single possible way to use all cables and adapters to connect the `pc` to the `printer`. As such, each problem of the AssemblyHierarchical domain has only one "true" solution. The domain however introduces a factorial amount of symmetric solutions – as the order in which the cables and adapters are plugged into one another can be choosen freely.

The domain in the IPC 2020 contained in total 30 instances from $i = 1$ to $i = 30$.

Performance in the IPC

Even though the setup of the AssemblyHierarchical is very simple, all participating planners in the IPC 2020 struggled with solving more than a few instances. The best planner on this domain – Lilotane (Schreiber 2021b; 2021a) solved only five out of the 30 total instances.

References

Bercher, P.; Biundo, S.; Geier, T.; Hörnle, T.; Nothdurft, F.; Richter, F.; and Schattenberg, B. 2014. Plan, repair, execute, explain – How planning helps to assemble your home theater. In *Proc. of the 24th Int. Conf. on Autom. Plan. and Sched. (ICAPS 2014)*, 386–394. AAAI Press.

Bercher, P.; Richter, F.; Hörnle, T.; Geier, T.; Höller, D.; Behnke, G.; Nothdurft, F.; Honold, F.; Minker, W.; Weber, M.; and Biundo, S. 2015. A planning-based assistance system for setting up a home theater. In *Proc. of the 29th AAAI Conf. on AI (AAAI 2015)*, 4264–4265. AAAI Press.

Bercher, P.; Richter, F.; Hörnle, T.; Geier, T.; Höller, D.; Behnke, G.; Nielsen, F.; Honold, F.; Schüssel, F.; Reuter, S.; Minker, W.; Weber, M.; Dietmayer, K.; and Biundo, S. 2017. *Advanced User Assistance for Setting Up a Home Theater*. Cognitive Technologies. Springer. chapter 24, 485–491.

Bercher, P.; Richter, F.; Honold, F.; Nielsen, F.; Schüssel, F.; Geier, T.; Hörnle, T.; Reuter, S.; Höller, D.; Behnke, G.; Dietmayer, K.; Minker, W.; Weber, M.; and Biundo, S. 2018. A companion-system architecture for realizing individualized and situation-adaptive user assistance. technical report, Ulm University.

Höller, D.; Behnke, G.; Bercher, P.; Biundo, S.; Fiorino, H.; Pellier, D.; and Alford, R. 2020. HDDL – A language to describe hierarchical planning problems. In *Proc. of the 34th AAAI Conf. on AI (AAAI 2020)*, 9883–9891. AAAI Press.

Schreiber, D. 2021a. Lifted logic for task networks: TOHTN planner lilotane in the IPC 2020. In *Proceedings of 10th International Planning Competition: planner and domain abstracts (IPC 2020)*.

Schreiber, D. 2021b. Lilotane: A lifted SAT-based approach to hierarchical planning. *Journal of Artificial Intelligence Research* 70:1117–1181.