Contents

1 Monotone Design Problems

This chapter introduces *design problems with implementations*, which summarize a design problem as a relation among functionality, resources, and implementations.

We show that they form a semi-category **DPI**.

We provide several examples from diverse fields.

ACT4EBOOK–297: We start right away talking about monotonicity after the DPI definition, but the link is not clear.

{bhfn:1}

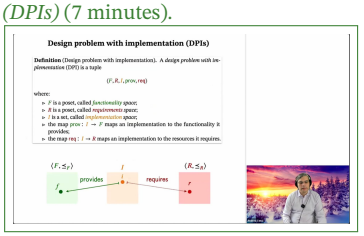
{ch:design-problems}

The Swiss Army knife is a multi-tool pocketknife manufactured by Victorinox, first manufactured in 1981.

1.1 Design Problems with Implementation

{sec:Design-Problems}

Watch: *Design problems with implementation (DPIs)* (7 minutes).



{def:DPI}

We start by defining a "design problem with implementation", which is a tuple of "functionality space", "implementation space", and "resources space", together with two maps that describe the feasibility relations between these three spaces (Fig. 1.1).

Definition 1.1 (Design problem with implementation)**.** A *design problem with implementation* (DPI) is a tuple

$$\langle \mathbf{F}, \mathbf{R}, \mathbf{I}, \mathtt{prov}, \mathtt{req} \rangle, \tag{1.1}$$

where:

- ▹ $\mathbf{F}$ is a poset, called *functionality space*;
- ▹ $\mathbf{R}$ is a poset, called *requirements space*;
- ▹ $\mathbf{I}$ is a set, called *implementation space*;
- ▹ the map $\mathtt{prov} : \mathbf{I} \to \mathbf{F}$ maps an implementation to the functionality it provides;
- ▹ the map $\mathtt{req} : \mathbf{I} \to \mathbf{R}$ maps an implementation to the resources it requires.

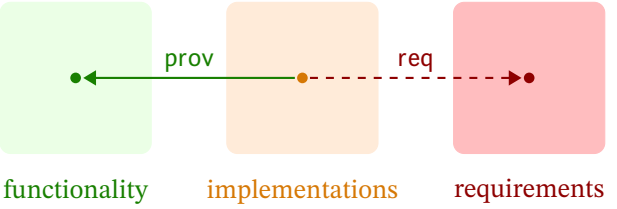


Figure 1.1 {fig:setup}

{exa:motor}

Example 1.2 (Motor design)**.** Suppose we need to choose a motor for a robot from a given set. The functionality of a motor could be parametrized by torque and speed. The resources to consider could include the cost [USD], the mass [g], the input voltage [V], and the input current [A]. The map $\mathtt{prov} : \mathbf{I} \to \mathbf{F}$ assigns to each motor its functionality, and the map $\mathtt{req} : \mathbf{I} \to \mathbf{R}$ assigns to each motor the resources it needs (Fig. 1.2).

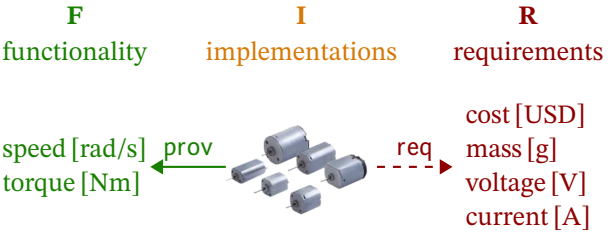


Figure 1.2 {fig:motor_evalexec}

{exa:chassis}

Example 1.3 (Chassis design)**.** Suppose we need to choose a chassis for a robot (Fig. 1.3). The implementation space $\mathbf{I}$ could be the set of all chassis that could ever be designed (in case of a theoretical analysis), or just the set of chassis available in the catalogue at hand (in case of a practical design decision). The functionality of a chassis could be formalized as "the ability to transport a certain payload [g]" and "at a given speed [m/s]". More refined functional requirements

would include maneuverability, the cargo volume, etc.. The resources to consider could be the cost [USD] of the chassis; the total mass; and, for each motor to be placed in the chassis, the required speed [rad/s] and torque [Nm].

chassis available at ServoCity.com

Figure 1.3

{fig:dpi_chassis_eval}

ACT4EBOOK–295: Integrate Example 1.4 with Example 1.2

{exa:dpi_elmotor}

Example 1.4. We revisit the leading example of **??** with the newly introduced co-design perspective. Let's consider a list of electrical motors as in Table 1.1.

Table 1.1: A simplified catalogue of motors.

Motor ID	Company	Torque [kg · cm]	Weight [g]	Max Power [W]	Cost [USD]
1204	SOYO	0.18	60.0	2.34	19.95
1206	SOYO	0.95	140.0	3.00	19.95
1207	SOYO	0.65	130.0	2.07	12.95
2267	SOYO	3.7	285.0	4.76	16.95
2279	Sanyo Denki	1.9	165.0	5.40	164.95
1478	SOYO	19.0	1,000	8.96	49.95
2299	Sanyo Denki	2.2	150.0	5.90	59.95

{t

We can think of this as a catalogue of electric motors $\langle \mathbf{I}_{\mathrm{EM}}, \mathtt{prov}_{\mathrm{EM}}, \mathtt{req}_{\mathrm{EM}} \rangle$. In particular, the set of implementations collects all the motor models, which we can specify using the motor IDs:

$$\mathbf{I}_{\mathrm{EM}} = \{1204, 1206, 1207, 2267, 2279, 1478, 2299\}. \tag{1.2}$$

We now have to think about resources and functionalities. Each motor requires some weight (in g), power (in W), and has some cost (in USD), and provides some torque (in kg · cm). Thus, we can identify

$$\mathbf{F} = \mathbb{R}^{\mathrm{kg \cdot cm}}, \quad \mathbf{R} = \mathbb{R}^{\mathrm{g}} \times \mathbb{R}^{\mathrm{W}} \times \mathbb{R}^{\mathrm{USD}},$$

by considering the units as discussed in **??**. The correspondences are given by

the details in Table 1.1. For instance, we have

$$\mathsf{prov}_{\mathrm{EM}}(1204) = 0.18\,\mathrm{kg}\cdot\mathrm{cm}, \quad \mathsf{req}_{\mathrm{EM}}(1204) = \langle 60\,\mathrm{g}, 2.34\,\mathrm{W}, 19.95\,\mathrm{USD}\rangle. \tag{1.3}$$

Graphical notation

A graphical notation will help reasoning about composition. A DPI is represented as a box with n_f green edges and n_r red edges (Fig. 1.4).

Add n_f, n_r to the figure (counting the wires).

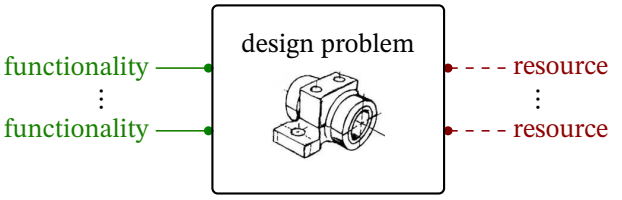


Figure 1.4 fig:dp_graphical

This means that the functionality and resources spaces can be factorized in n_f and n_r components:

$$\mathbf{F} = \prod_{i=1}^{\mathrm{n}_f} \pi_i \mathbf{F}_i, \quad \mathbf{R} = \prod_{j=1}^{\mathrm{n}_r} \pi_j \mathbf{R}_j,$$

where "π_i" represents the projection to the i-th component. If there are no green (respectively, red) edges, then n_f (respectively, n_r) is zero, and $\mathbf{F}$ (respectively, $\mathbf{R}$) is equal to $\mathbf{1} = \{\langle\rangle\}$, the set containing one element, the empty tuple $\langle\rangle$.

These *co-design diagrams* are not to be confused with signal flow diagrams, in which the boxes represent oriented systems and the edges represent signals.

1.2 Examples

We now present a list of design problems for different disciplines, to showcase the universality of the approach.

Mechatronics

Many mechanisms can be readily modeled as relations between a provided functionality and required resources.

Yet another different model of a motor. Either harmonize all in one, or explicitly have discussion about how the same thing can have different models

Example 1.5. The functionality of a DC motor (Fig. 1.5) is to provide a certain speed and torque, and the resources are current and voltage.

ACT4EBOOK–296: Add equation that relate all.

ACT4EBOOK–296: What are the implementations?

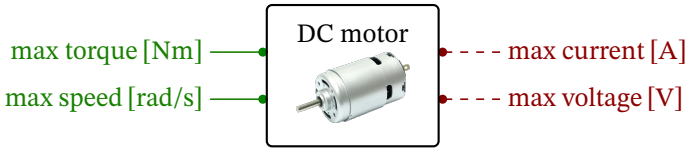


Figure 1.5 fig:dc_motor

Example 1.6. A gearbox (Fig. 1.6) provides a certain output torque τ_o and speed τ_o, given a certain input torque τ_i and speed ω_i. For an ideal gearbox with a reduction ratio $r \in \mathbb{Q}_+$ and efficiency ratio γ, $0 < \gamma < 1$, the constraints among those quantities are $\omega_i \geq r\,\omega_o$ and $\tau_i\omega_i \geq \gamma\,\tau_o\omega_o$.

ACT4EBOOK–296: What are the implementations?

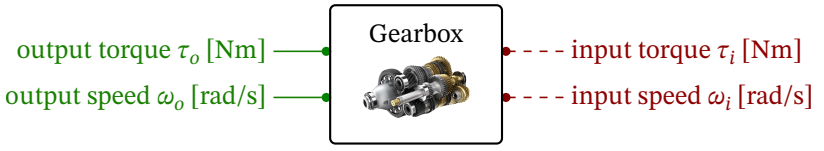


Figure 1.6 {fig:gearbox}

Example 1.7. *Propellers* (Fig. 1.7) generate thrust given a certain torque and speed.

ACT4EBOOK–296: What are the implementations?

Example 1.8. A *crank-rocker* (Fig. 1.8) converts rotational motion into a rocking motion.

296

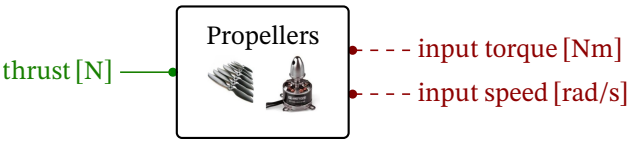


Figure 1.7 fig:propeller

Put here an additional crank-rocker mechanism figure so that people who don't know that can get an idea.

ACT4EBOOK–296: What are the implementations?

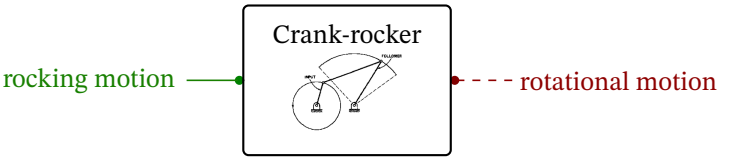


Figure 1.8 fig:crack

Geometrical constraints

Geometrical constraints are examples of constraints that are easily recognized as monotone, but possibly hard to write down in closed form.

Example 1.9 (Bin packing)**.** Suppose that each internal component occupies a volume bounded by a parallelepiped, and that we must choose the minimal enclosure in which to place all components (Fig. 1.9). What is the minimal size of the enclosure? This is a variation of the *bin packing* problem, which is in NP for both 2D and 3D [**lodi02two**]. It is easy to see that the problem is monotone, by noticing that, if one the components shapes increases, then the size of the enclosure cannot shrink.

ACT4EBOOK–296: What are the implementations?

Figure 1.9 fig:packing

Inference

Many inference problems have a monotone formalization, taking the accuracy or robustness as functionality, and computation or sensing as resources. Typically these bounds are known in a closed form only for restricted classes of systems, such as the linear/Gaussian setting.

Example 1.10. (SLAM) One issue with particle-filter-based estimation procedures, such as the ones used in the popular GMapping [**grisetti07improved**] method, is that the filter might diverge if there aren't enough particles. Although the relation might be hard to characterize, there is a monotone relation between the robustness (1 - probability of failure), the accuracy, and the number of particles (Fig. 1.10).

ACT4EBOOK–296: What are the implementations?

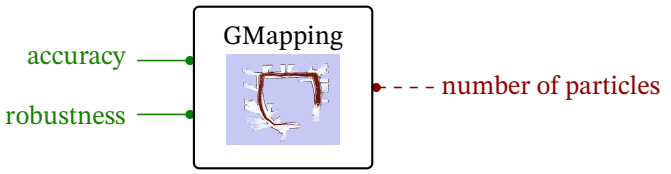


Figure 1.10 fig:gmapping

Example 1.11. (Stereo reconstruction) Progressive reconstruction systems ([**locher16progressive**]), which start with a coarse approximation of the solution that is progressively refined, are described by a smooth relation between the resolution and the latency to obtain the answer (Fig. 1.11). A similar relation characterizes any anytime algorithms in other domains, such as robot motion planning.

ACT4EBOOK–296: What are the implementations?

Figure 1.11 fig:progressive

Example 1.12. The empirical characterization of the monotone relation between the accuracy of a visual SLAM solution and the power consumption is the goal of recent work by Davison and colleagues [**nardi15introducing**, **zia16comparative**].

ACT4EBOOK–296: What are the implementations?

ACT4EBOOK–153: Reproduce plot from paper.

Communication

Example 1.13 (Transducers)**.** Any type of "transducer" that bridges between different mediums can be modeled as a DP. For example, an access point (Fig. 1.12) provides the "wireless access" functionality, and requires that the infrastructure provides the "Ethernet access" resource.

ACT4EBOOK–296: What are the implementations?

Figure 1.12 fig:accesspoint

Example 1.14 (Wireless link)**.** The basic functionality of a wireless link is to provide a certain bandwidth (Fig. 1.13). Further refinements could include bounds

on the latency or the probability that a packet drop is dropped. Given the established convention about the the preference relations for functionality, in which a *lower* functionality is "easier" to achieve, one needs to choose "*minus* the latency" and "*minus* the packet drop probability" for them to count as functionality. As for the resources, apart from the transmission power [W], one should consider at least the spectrum occupation, which could be described as an interval $[f_0, f_1]$ of the frequency axis $\mathbb{R}^{[\mathrm{Hz}]}$. Thus the resources space is $\mathbf{R} = \mathbb{R}^{[\mathrm{W}]} \times \mathsf{intervals}(\mathbb{R}^{[\mathrm{Hz}]})$.

ACT4EBOOK–296: What are the implementations?

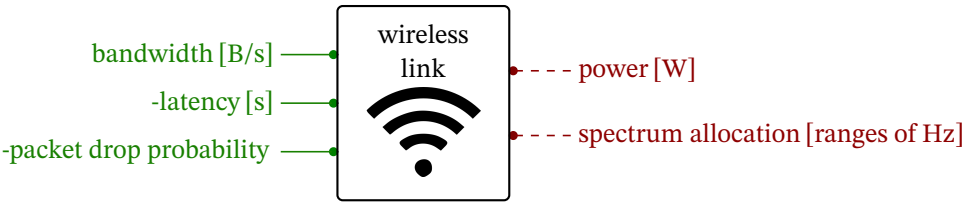
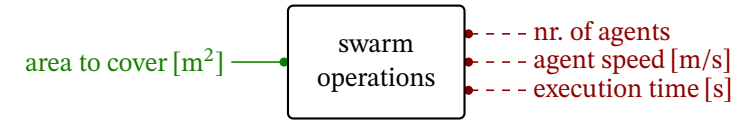


Figure 1.13 fig:networklink

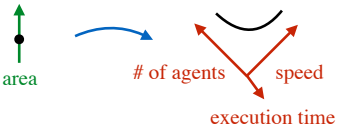
Multi-robot systems

In a multi-robot system there is always a trade-off between the number of robots and the capabilities of the single robot.

Example 1.15. Suppose we need to create a swarm of agents whose functionality is to sweep an area. If the functionality is fixed, one expects a three-way trade-off between the three resources: number of agents, the speed of a single agent, and the execution time. For example, if the time available decreases, one has to increase either the speed of an agent or the number of agents (Fig. 1.14b).



(a)



{sub@fig:multirobot2} **(b)**

Figure 1.14

ACT4EBOOK–154: redo right-hand side in tikz

LQG Control

ACT4EBOOK–155: Fill LQG section

Missing: discussion of LGQ control as a design problem.

Computation

The trivial model of a CPU is as a device that provides computation, measured in flops, and requires power [W]. Clearly there is a monotone relation between the two.

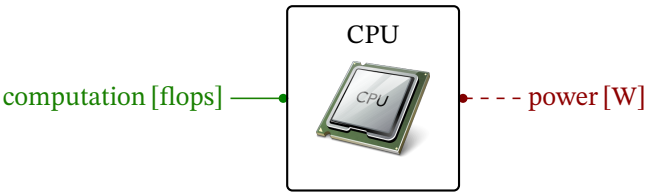


Figure 1.15

A similar monotone relation between application requirements and computation resources holds in a much more general setting, where both application and computation resources are represented by graphs. This will be an example of a monotone relation between nontrivial partial orders.

In the Static Data Flow (SDF) model of computation [**sriram00**, **lee10**], the application is represented as a graph of procedures that need to be allocated on a network of processors.

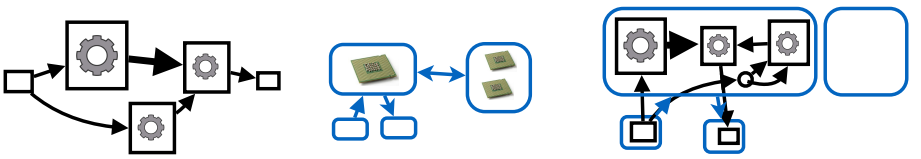


Figure 1.16

Define the *application graph* (sometimes called "computation graph") as a graph where each node is a procedure (or "actor") and each edge is a message that needs to be passed between procedures. Each node is labeled by the number of ops necessary to run the procedure. Each edge is labeled by the size of the message. There is a partial order $\preceq$ on application graphs. In this order, it holds that $A_1 \preceq A_2$ if the application graph A_2 needs more computation or bandwidth for its execution than A_1. Formally, it holds that $A_1 \preceq A_2$ if there is a homomorphism $\varphi : A_1 \Rightarrow A_2$; and, for each node $n \in A_1$, the node $\varphi(n)$ has equal or larger computational requirements than n; and for each edge $\langle n_1, n_2 \rangle$ in A_2, the edge $\langle \varphi(n_1), \varphi(n_2) \rangle$ has equal or larger message size.

Define a *resource graph* as a graph where each node represents a processor, and each edge represents a network link. Each node is labeled by the processor capacity [flops] Each edge is labeled by latency [s] and bandwidth [B/s]. There is a partial order on resources graph as well: it holds that $R_1 \preceq R_2$ if the resource graph R_2 has more computation or network available than R_1. The definition is similar to the case of the application graph: there must exist a graph homomor-

phism $\varphi : R_1 \Rightarrow R_2$ and the corresponding nodes (edges) of R_2 must have larger or equal computation (bandwidth) than those of R_1.

Given an application graph A and a resource graph R, a typical resource allocation problem consists in choosing in which processor each actor must be scheduled to maximize the throughput T [Hz]. This is equivalent to the problem of finding a graph homomorphism $\Psi : A \Rightarrow R$. Let T^* be the optimal throughput, and write it as a function of the two graphs:

$$T^* = T^*(A, R).$$

Then the optimal throughput T^* is decreasing in A (a more computationally demanding application graph decreases the throughput) and increasing in R (more available computation/bandwidth increase the throughput).

Therefore, we can formalize this as a design problem where the two functionalities are the throughput T [Hz] and the application graph A, and the resource graph R is the resource.

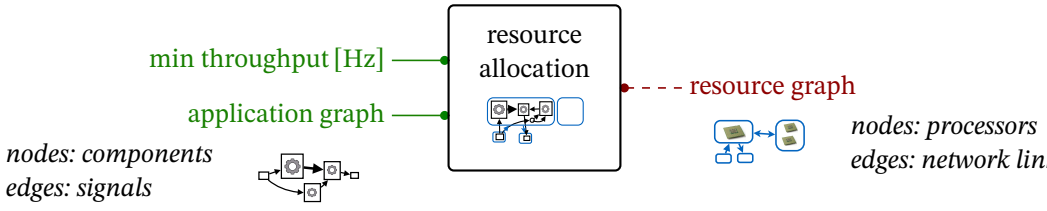


Figure 1.17

Example 1.16. Svorenova et al. [**svorenova16resource**] consider a joint sensor scheduling and control synthesis problem, in which a robot can decide to not perform sensing to save power, given performance objectives on the probability of reaching the target and the probability of collision. The method outputs a Pareto frontier of all possible operating points. This can be cast as a design problem with functionality equal to the probability of reaching the target and (the inverse of) the collision probability, and with resources equal to the actuation power, sensing power, and sensor accuracy.

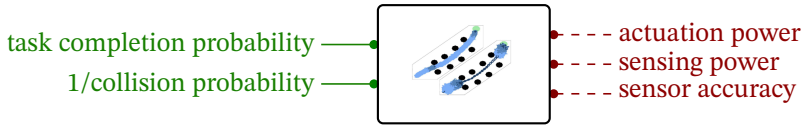
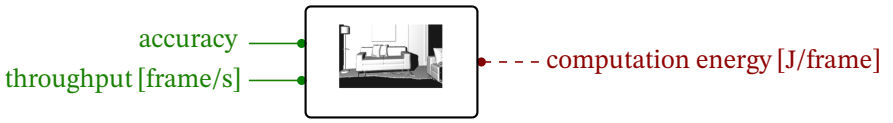


fig:progressive-1-1

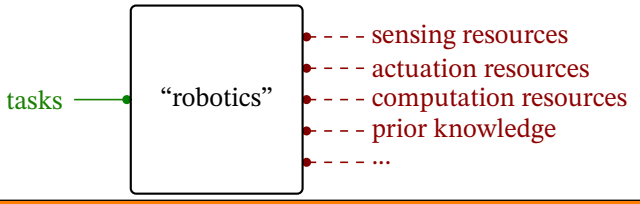
Figure 1.18

Example 1.17. Nardi et al. [**zia16comparative**] describe a benchmarking system for visual SLAM that provides the empirical characterization of the monotone relation between the accuracy of the visual SLAM solution, the throughput [frames/s] and the energy for computation [J/frame]. The implementation space is the product of algorithmic parameters, compiler flags, and architecture choices, such as the number of GPU cores active. This is an example of a design problem whose functionality-resources map needs to be experimentally evaluated.



{fig:dp_zia}

Figure 1.19



{fig:robot-generic}

ACT4EBOOK–156: finish with axes

Figure 1.20

Other examples in minimal robotics

Many works have sought to find "minimal" designs for robots, and can be understood as characterizing the relation between the poset of tasks and the poset of physical resources, which is the product of sensing, actuation, and computation resources, plus other non-physical resources, such as prior knowledge (Fig. 1.20). Given a task, there is a minimal antichain in the resources poset that describes the possible trade-offs (for instance, compensating lousier sensors with more computation).

The poset structure arises naturally: for example, in the *sensor lattice* [**lavalle12sensing**], a sensor dominates another if it induces a finer partition of the state space. Similar dominance relations can be defined for actuation and computation. O'Kane and Lavalle [**okane08comparing**] define a robot as a union of "robotic primitives", where each primitive is an abstraction for a set of sensors, actuators, and control strategies that can be used together (for instance, a compass plus a contact sensor allow to "drive North until a wall is hit"). The effect of each primitive is modeled as an operator on the robot's information space. It is possible to work out what are the minimal combinations of robotic primitives (minimal antichain) that are sufficient to perform a task (for instance, global localization), and describe a dominance relation (partial order) of primitives. Other works have focused on minimizing the complexity of the controller. Egerstedt [**egerstedt03motion**] studies the relation between the complexity of the environment and a notion of minimum description length of control strategies, which can be taken as a proxy for the computation necessary to perform the task. Soatto [**soatto11steps**] studies the relation between the performance of a visual task, and the minimal representation that is needed to perform that task.

Computer science

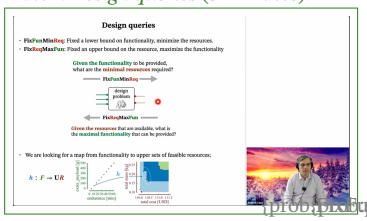
ACT4EBOOK–157: to write

Missing: Example of Hoare triples.

1.3 Queries

{sec:design-problems-querying}

Watch: *Design queries* (3 minutes).



{prob:FixFunMinRes}

A DPI is a model to which we can associate a family of optimization problems. While in previous examples we covered the problem "feasibility", we still miss FixFunMinRes, FixResMaxFun, and FeasibleImp.

The first can be translated to "Given a lower bound on the functionality f, what are the implementations that have minimal resources usage?" (Fig. 1.21).

Problem (FixFunMinRes)**.** Given $f \in \mathbf{F}$, find the implementations in $\mathbf{I}$ that realize the functionality f (or higher) with minimal resources, or provide a proof that there are none:

{{eq:objective}}

$$\begin{cases} \text{using} & i \in \mathbf{I}, \\ \mathrm{Min}_{\preceq_{\mathbf{R}}} & r, \\ \text{s.t.} & r = \mathsf{req}(i), \\ & f \preceq_{\mathbf{F}} \mathsf{prov}(i). \end{cases} \tag{1.4}$$

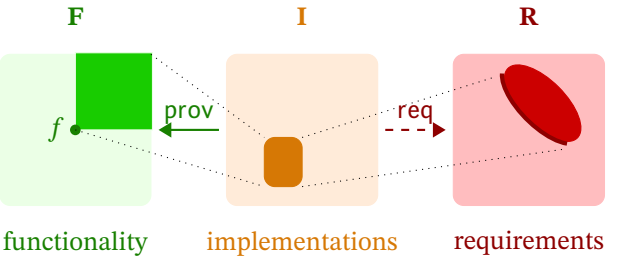


Figure 1.21 fig:set

Remark 1.18 (Minimal *vs* least solutions)**.** Note the use of "$\mathrm{Min}_{\preceq_{\mathbf{R}}}$" in (1.4), which indicates the set of minimal (non-dominated) elements according to $\preceq_{\mathbf{R}}$, rather than "$\min_{\preceq_{\mathbf{R}}}$", which would presume the existence of a least element. In all problems in this paper, the goal is to find the optimal trade-off of resources ("Pareto front"). So, for each f, we expect to find an antichain $R \in \mathcal{A}\mathbf{R}$. We will see that this formalization allows an elegant way to treat multi-objective optimization problems. The algorithm to be developed will directly solve for the set R, without resorting to techniques such as *scalarization*, and therefore is able to work with arbitrary posets, possibly discrete.

In an entirely symmetric fashion, we could fix an upper bound on the resources usage, and then maximize the functionality provided (Fig. 1.22). The formulation is entirely dual, in the sense that it is obtained from (1.4) by swapping Min with Max, $\mathbf{F}$ with $\mathbf{R}$, and prov with req.

{prob:FixResMaxFun}

Problem (FixResMaxFun)**.** Given $r \in \mathbf{R}$, find the implementations in $\mathbf{I}$ that requires r (or lower) and provide the maximal functionality, or provide a proof that there are none:

{{eq:objective-1}}

$$\begin{cases} \text{using} & i \in \mathbf{I}, \\ \mathrm{Max}_{\preceq_{\mathbf{F}}} & f, \\ \text{s.t.} & f = \mathsf{prov}(i), \\ & r \succeq_{\mathbf{R}} \mathsf{req}(i). \end{cases} \tag{1.5}$$

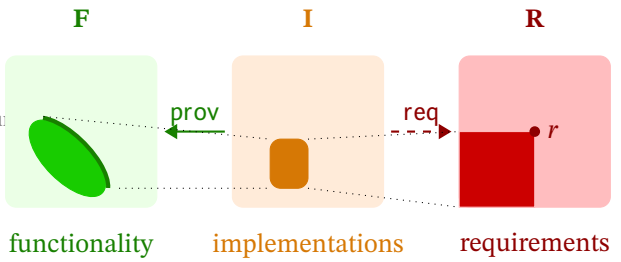


Figure 1.22 fig:fu

Another type of query is "Given a lower bound on the functionality f and an upper bound on the costs f, what are the feasible implementations?

{prob:FeasibleImp}

Problem (FeasibleImp)**.** Given $f \in \mathbf{F}$ and $r \in \mathbf{R}$, find the implementations in $\mathbf{I}$ that requires r (or lower) and provide f (or higher)

$$\begin{cases} \text{using} & i \in \mathbf{I}, \\ \text{s.t.} & f \preceq_{\mathbf{F}} \mathsf{prov}(i), \\ \text{s.t.} & \mathsf{prov}(i) \preceq_{\mathbf{R}} \mathsf{req}(i), \end{cases} \tag{1.6}$$

{{eq:FeasibleImp}}

Another variation is to find only whether there are feasible solutions or not.

{prob:Feasibility}

Problem (Feasibility)**.** Given $f \in \mathbf{F}$ and $r \in \mathbf{R}$, find if (1.6) is feasible.

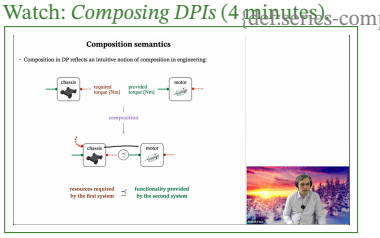
1.4 The semi-category DPI

Graphically, one is allowed to connect only edges of different colors and of the same type. This interconnection is indicated with the symbol "$\preceq$" in a rounded box (Fig. 1.23).

Figure 1.23 fig:connection

The semantics of the interconnection is that the second DPI provides the resources required by the first DPI. This is a partial order inequality constraint of the type $r_1 \preceq f_2$.

Watch: *Composing DPIs* (4 minutes) {{def:series-composition}}



Definition 1.19 (DPI composition). The series composition of two DPIs

$$\begin{aligned} \mathbf{d}_1 = &\ \langle \mathbf{F}_1, \mathbf{R}_1, \mathbf{I}_1, \mathsf{prov}_1, \mathsf{req}_1 \rangle, \\ \mathbf{d}_2 = &\ \langle \mathbf{F}_2, \mathbf{R}_2, \mathbf{I}_2, \mathsf{prov}_2, \mathsf{req}_2 \rangle, \end{aligned} \tag{1.7}$$

for which $\mathbf{F}_2 = \mathbf{R}_1$, is defined as

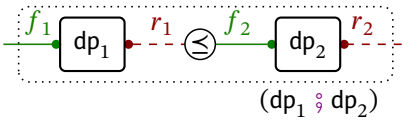
$$(\mathbf{d}_1 \fatsemi \mathbf{d}_2) := \langle \mathbf{F}_1, \mathbf{R}_2, \mathbf{I}, \mathsf{prov}, \mathsf{req} \rangle, \tag{1.8}$$

where:

$$\mathbf{I} = \{[i_1 \,;\, i_2] \in (\mathbf{I}_1 \fatsemi \mathbf{I}_2) \mid \mathsf{req}_1(i_1) \preceq_{\mathbf{R}_1} \mathsf{prov}_2(i_2)\}, \tag{1.9}$$

$$\begin{aligned} \mathsf{prov} :&\quad [i_1 \,;\, i_2] \mapsto \mathsf{prov}_1(i_1), \\ \mathsf{req} :&\quad [i_1 \,;\, i_2] \mapsto \mathsf{req}_2(i_2). \end{aligned} \tag{1.10}$$

2}



The composition is graphically represented as in Section 1.4.

Lemma 1.20. Series composition is associative.

Proof. Consider

$$\begin{aligned} \mathsf{dp}_1 = &\ \langle \mathbf{F}_1, \mathbf{R}_1, \mathbf{I}_1, \mathsf{prov}_1, \mathsf{req}_1 \rangle, \\ \mathsf{dp}_2 = &\ \langle \mathbf{F}_2, \mathbf{R}_2, \mathbf{I}_2, \mathsf{prov}_2, \mathsf{req}_2 \rangle, \\ \mathsf{dp}_3 = &\ \langle \mathbf{F}_3, \mathbf{R}_3, \mathbf{I}_3, \mathsf{prov}_3, \mathsf{req}_3 \rangle, \end{aligned} \tag{1.11}$$

for which $\mathbf{F}_2 = \mathbf{R}_1$ and $\mathbf{F}_3 = \mathbf{R}_2$. We want to show that

{{eq:assoc_dpi}}

$$(\mathsf{dp}_1 \fatsemi \mathsf{dp}_2) \fatsemi \mathsf{dp}_3 = \mathsf{dp}_1 \fatsemi (\mathsf{dp}_2 \fatsemi \mathsf{dp}_3). \tag{1.12}$$

We know that the first part of the left term of (1.12) gives

$$(\mathsf{dp}_1 \fatsemi \mathsf{dp}_2) = \langle \mathbf{F}_1, \mathbf{R}_2, \mathbf{I}_{1,2}, \mathsf{prov}_{1,2}, \mathsf{req}_{1,2} \rangle, \tag{1.13}$$

Therefore, the full left term of (1.12) reads

{{eq:dpi_ass_1}}

$$(\mathsf{dp}_1 \fatsemi \mathsf{dp}_2) \fatsemi \mathsf{dp}_3 = \langle \mathbf{F}_1, \mathbf{R}_3, \mathbf{I}_{1,3}, \mathsf{prov}_{1,3}, \mathsf{req}_{1,3} \rangle, \tag{1.14}$$

with

$$\begin{aligned}\mathbf{I}_{1,3} &= \{[[i_1\ ;\ i_2]\ ;\ i_3] \in ((\mathbf{I}_1 \fatsemi \mathbf{I}_2) \fatsemi \mathbf{I}_3) \mid \mathsf{req}_{1,2}([i_1\ ;\ i_2]) \preceq_{\mathbf{R}_2} \mathsf{prov}_3(i_3) \\ &\qquad \wedge\, \mathsf{req}_1(i_1) \preceq_{\mathbf{R}_1} \mathsf{prov}_2(i_2)\} \\ &= \{[i_1\ ;\ i_2\ ;\ i_3] \in (\mathbf{I}_1 \fatsemi \mathbf{I}_2 \fatsemi \mathbf{I}_3) \mid \mathsf{req}_2(i_2) \preceq_{\mathbf{R}_2} \mathsf{prov}_3(i_3) \\ &\qquad \wedge\, \mathsf{req}_1(i_1) \preceq_{\mathbf{R}_1} \mathsf{prov}_2(i_2)\}.\end{aligned} \tag{1.15}$$

and

$$\begin{aligned}\mathsf{prov}_{1,3} :\ & [[i_1\ ;\ i_2]\ ;\ i_3] \mapsto \mathsf{prov}_{1,2}([i_1\ ;\ i_2]), \\ & [i_1\ ;\ i_2\ ;\ i_3] \mapsto \mathsf{prov}_1(i_1), \\ \mathsf{req}_{1,3} :\ & [[i_1\ ;\ i_2]\ ;\ i_3] \mapsto \mathsf{req}_3(i_3) \\ & [i_1\ ;\ i_2\ ;\ i_3] \mapsto \mathsf{req}_3(i_3).\end{aligned} \tag{1.16}$$

By expanding the second part of the second term of (1.12), one has:

$$(\mathsf{dp}_2 \fatsemi \mathsf{dp}_3) = \left\langle \mathbf{F}_2, \mathbf{R}_3, \mathbf{I}_{2,3}, \mathsf{prov}_{2,3}, \mathsf{req}_{2,3} \right\rangle, \tag{1.17}$$

Therefore, the full right term of (1.12) reads

{{eq:dpi_ass_2}}

$$\mathsf{dp}_1 \fatsemi (\mathsf{dp}_2 \fatsemi \mathsf{dp}_3) = \left\langle \mathbf{F}_1, \mathbf{R}_3, \mathbf{I}'_{1,3}, \mathsf{prov}'_{1,3}, \mathsf{req}'_{1,3} \right\rangle, \tag{1.18}$$

with

$$\begin{aligned}\mathbf{I}'_{1,3} &= \{[i_1\ ;\ [i_2\ ;\ i_3]] \in (\mathbf{I}_1 \fatsemi (\mathbf{I}_2 \fatsemi \mathbf{I}_3)) \mid \mathsf{req}_1(i_1) \preceq_{\mathbf{R}_1} \mathsf{prov}_{2,3}([i_2\ ;\ i_3]) \\ &\qquad \wedge\, \mathsf{req}_2(i_2) \preceq_{\mathbf{R}_2} \mathsf{prov}_3(i_3)\} \\ &= \{[i_1\ ;\ i_2\ ;\ i_3] \in (\mathbf{I}_1 \fatsemi \mathbf{I}_2 \fatsemi \mathbf{I}_3) \mid \mathsf{req}_1(i_1) \preceq_{\mathbf{R}_1} \mathsf{prov}_2(i_2) \\ &\qquad \wedge\, \mathsf{req}_2(i_2) \preceq_{\mathbf{R}_2} \mathsf{prov}_3(i_3)\}\end{aligned} \tag{1.19}$$

and

$$\begin{aligned}\mathsf{prov}'_{1,3} :\ & [i_1\ ;\ [i_2\ ;\ i_3]] \mapsto \mathsf{prov}_1(i_1) \\ & [i_1\ ;\ i_2\ ;\ i_3] \mapsto \mathsf{prov}_1(i_1), \\ \mathsf{req}'_{1,3} :\ & [i_1\ ;\ [i_2\ ;\ i_3]] \mapsto \mathsf{req}_{2,3}([i_2\ ;\ i_3]) \\ & [i_1\ ;\ i_2\ ;\ i_3] \mapsto \mathsf{req}_3(i_3)\end{aligned} \tag{1.20}$$

It is clear that $\mathbf{I}_{1,3} = \mathbf{I}'_{1,3}$, $\mathsf{prov}_{1,3} = \mathsf{prov}'_{1,3}$, and $\mathsf{req}_{1,3} = \mathsf{req}'_{1,3}$. This, together with (1.14) and (1.18) shows associativity. □

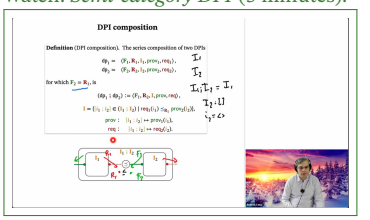
These two properties are sufficient to conclude that there exists a semi-category of design problems.

{def:DPIcat}

Definition 1.21 (Semi-category **DPI**). There is a semi-category **DPI** where

- ▹ The objects are posets.
- ▹ The morphisms are DPIs $\langle \mathbf{F}, \mathbf{R}, \mathbf{I}, \mathsf{prov}, \mathsf{req} \rangle$.
- ▹ Morphism composition is given by Def. 1.19.

Watch: *Semi-category DPI* (5 minutes).



Lemma 1.22. **DPI** is not a category, because one cannot find identities.

Proof. We prove this by contradiction. Suppose we can find a DPI that works as an identity for interconnection for any other DPI. Therefore, we have

$$\mathbf{I}_1 \fatsemi \mathbf{I}_2 = \mathbf{I}_1. \tag{1.21}$$

This implies that $\mathbf{I}_2$ must be an empty list of sets, that is inhabited by only one element, the empty list of elements. Therefore, $\texttt{req}_2$ of the identity is necessarily a constant because there is an . □

ACT4EBOOK–158: discussion about *decomposition is not decoupling*

1.5 Co-design problems

{sec:Co-design-problems}

A "co-design problem" will be defined as a *multigraph* of design problems.

{def:cdpi}

Definition 1.23 (Co-design problem with implementation)**.** A *Co-Design Problem with Implementation* (CDPI) is a tuple $\langle \mathbf{F}, \mathbf{R}, \langle \mathcal{V}, \mathcal{E} \rangle \rangle$, where $\mathbf{F}$ and $\mathbf{R}$ are two posets, and $\langle \mathcal{V}, \mathcal{E} \rangle$ is a multigraph of DPIs. Each node $v \in \mathcal{V}$ is a DPI $v = \langle \mathbf{F}_v, \mathbf{R}_v, \mathbf{I}_v, \mathsf{prov}_v, \mathsf{req}_v \rangle$. An edge $e \in \mathcal{E}$ is a tuple $e = \langle \langle v_1, i_1 \rangle, \langle v_2, j_2 \rangle \rangle$, where $v_1, v_2 \in \mathcal{V}$ are two nodes and i_1 and j_2 are the indices of the components of the functionality and resources to be connected, and it holds that $\pi_{i_1} \mathbf{R}_{v_1} = \pi_{j_2} \mathbf{F}_{v_2}$ (Fig. 1.24).

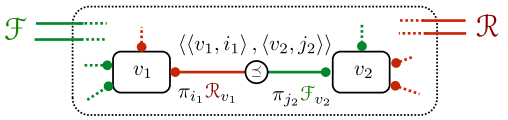


Figure 1.24:

ACT4EBOOK–159: redo in tikz fig:mcdp2

A CDPI is equivalent to a DPI with an implementation space $\mathbf{I}$ that is a subset of the product $\prod_{v \in \mathcal{V}} \mathbf{I}_v$, and contains only the tuples that satisfy the co-design constraints. An implementation tuple $i \in \prod_{v \in \mathcal{V}} \mathbf{I}_v$ belongs to $\mathbf{I}$ iff it respects all functionality–resources constraints on the edges, in the sense that, for all edges $\langle \langle v_1, i_1 \rangle, \langle v_2, j_2 \rangle \rangle$ in $\mathcal{E}$, it holds that

$$\pi_{i_1} \mathsf{req}_{v_1}(\pi_{v_1} i) \preceq \pi_{j_2} \mathsf{prov}_{v_2}(\pi_{v_2} i).$$

The posets $\mathbf{F}$, $\mathbf{R}$ for the entire CDPI are the products of the functionality and resources of the nodes that remain *unconnected*. For a node v, let UF_v and UR_v be the set of unconnected functionalities and resources. Then $\mathbf{F}$ and $\mathbf{R}$ for the CDPI are defined as the product of the unconnected functionality and resources of all DPIs: $\mathbf{F} = \prod_{v \in \mathcal{V}} \prod_{j \in \mathsf{UF}_v} \pi_j \mathbf{F}_v$ and $\mathbf{R} = \prod_{v \in \mathcal{V}} \prod_{i \in \mathsf{UR}_v} \pi_i \mathbf{R}_v$. The maps $\mathsf{prov}, \mathsf{req}$ return the values of the unconnected functionality and resources:

$$\mathsf{prov} : i \mapsto \prod_{v \in \mathcal{V}} \prod_{j \in \mathsf{UF}_v} \pi_j \mathsf{prov}_v(\pi_v i),$$

$$\mathsf{req} : i \mapsto \prod_{v \in \mathcal{V}} \prod_{i \in \mathsf{UR}_v} \pi_i \mathsf{req}_v(\pi_v i).$$

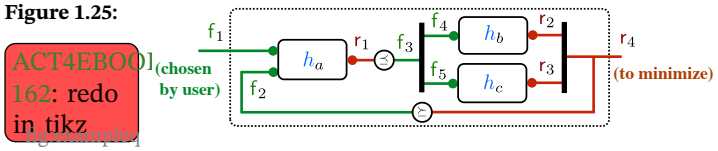
ACT4EBOOK–160: macro for unconnected a bit similar to upper sets?

ACT4EBOOK–161: Re-do a concrete example like this

Example 1.24. The MCDP in Fig. 1.25 is the interconnection of 3 DPs h_a, h_b, h_c. The semantics of the MCDP as an optimization problem is given by:

$$f_1 \mapsto \begin{cases} \operatorname{Min} r_4, & r_1 \in h_a(f_1, f_2), \quad r_1 \preceq f_3, \quad f_3 = \langle f_4, f_5 \rangle, \\ & r_2 \in h_b(f_4), \quad r_4 \preceq f_2, \quad r_4 = \langle r_2, r_3 \rangle, \\ & r_3 \in h_c(f_5). \end{cases}$$

Figure 1.25:



{exa:chassis_plus_motor}

Example 1.25. Consider the co-design of chassis (Example 1.3) plus motor (Example 1.2). The design problem for a motor has speed and torque as the provided functionality (what the motor must provide), and cost, mass, voltage, and current as the required resources (Fig. 1.26).

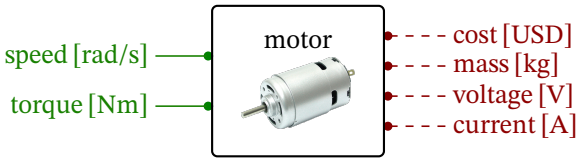


Figure 1.26 fig:motor

For the chassis (Fig. 1.27), the provided functionality is parameterized by the mass of the payload and the platform velocity. The required resources include the cost, total mass, and what the chassis needs from its motor(s), such as speed and torque.

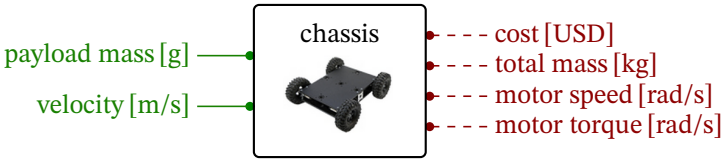


Figure 1.27 fig:gmcdp_chassis

The two design problem can be connected at the edges for torque and speed (Fig. 1.28). The semantics is that the motor needs to have *at least* the given torque and speed.

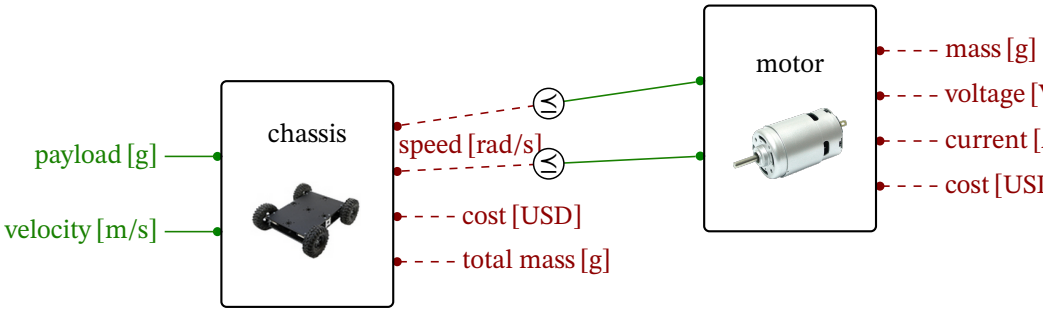


Figure 1.28 fig:gmcdp_chassis_plus_motor_series

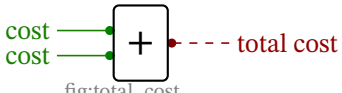


Figure 1.29 fig:total_cost

Resources can be summed together using a trivial DP corresponding to the map $h : \langle f_1, f_2 \rangle \mapsto \{f_1 + f_2\}$ (Fig. 1.29).

A co-design problem might contain recursive co-design constraints. For example, if we set the payload to be transported to be the sum of the motor mass plus some extra payload, a cycle appears in the graph (Fig. 1.30).

From here is under development

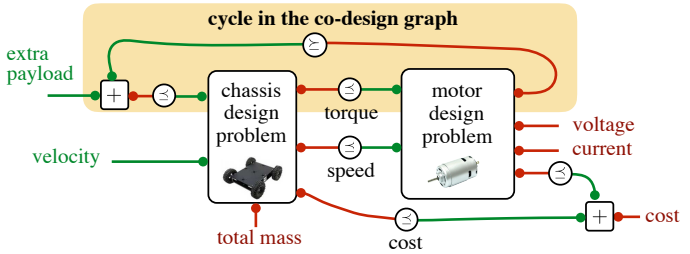
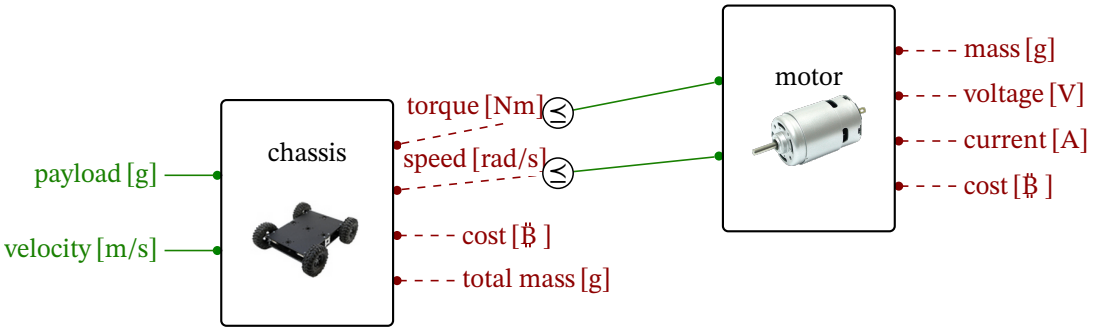


Figure 1.30 fig:gmcdp_chassis_plus_motor



ACT4EBOOK–163: finish

To here is under development

This formalism makes it easy to abstract away the details in which we are not interested. Once a diagram like Fig. 1.30 is obtained, we can draw a box around it and consider the abstracted problem (Fig. 1.31).

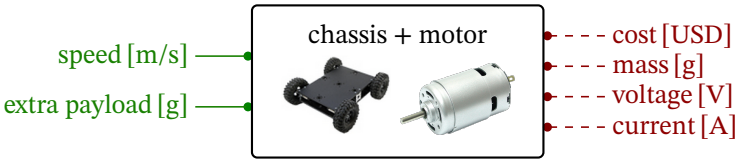


Figure 1.31 {fig:gmcdp_chassis_plus_motor-1}
{exa:finish}

Let us finish assembling our robot. A motor needs a motor control board. The functional requirements are the (peak) output current and the output voltage range (Fig. 1.32).

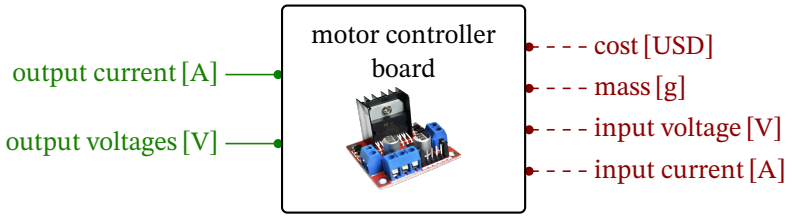


Figure 1.32 {fig:mcb}

The functionality for a power supply could be parameterized by the output current, the output voltages, and the capacity. The resources could include cost and mass (Fig. 1.33).

Relations such as current × voltage ≼ power required and power × endurance ≼ energy required can be modeled by a trivial "multiplication" DPI (Fig. 1.34).

We can connect these DPs to obtain a co-design problem with functionality voltage, current, endurance and resources mass and cost (Fig. 1.35).

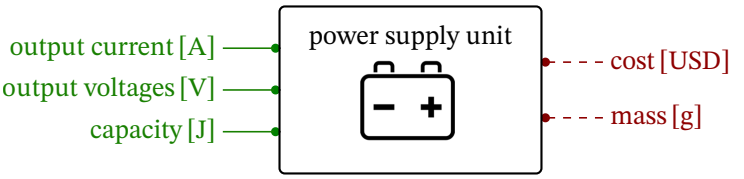


Figure 1.33 fig:example-ba

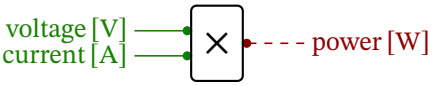


Figure 1.34 fig:current_times_voltage

Draw a box around the diagram, and call it "MCB+PSU"; then interconnect it with the "chassis+motor" diagram in Fig. 1.36.

We can further abstract away the diagram in Fig. 1.36 as a "mobility+power" CDPI, as in Fig. 1.37. The formalism allows to consider mass and cost as independent resources, meaning that we wish to obtain the Pareto frontier for the minimal resources. Of course, one can always reduce everything to a scalar objective. For example, a conversion from mass to cost exists and it is called "shipping". Depending on the destination, the conversion factor is between \$0.5/lbs, using USPS, to \$10k/lbs for sending your robot to low Earth orbit.

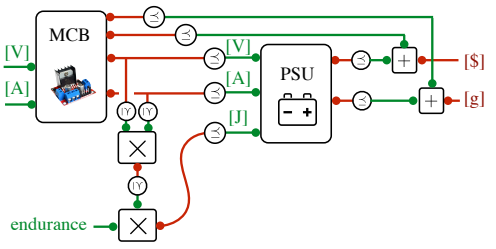


Figure 1.35 fig:connect

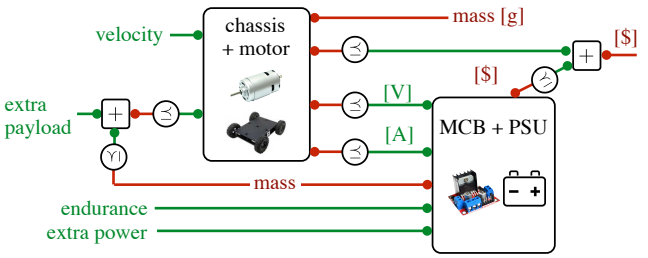


Figure 1.36 fig:another

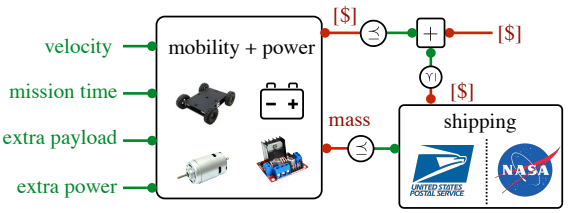


Figure 1.37 fig:shipping

1.6 Discussion of related work

{sec:design-problems-related}

Theory of design

Modern engineering has long since recognized the two ideas of modularity and hierarchical decomposition, yet there exists no general quantitative theory of design that is applicable to different domains. Most of the works in the theory of design literature study abstractions that are meant to be useful for a human designer, rather than an automated system. For example, a *function structure* diagram [**pahl07**] decomposes the function of a system in subsystems that exchange energy, materials, and signals, but it is not a formal representation. From the point of view of the theory of design, the contribution of this work is that the *design problem* abstraction developed, where one takes functionality and resources as the interfaces for the subsystems, is at the same time (1) mathematically precise; (2) intuitive to understand; and (3) leads to tractable optimization problems.

This work also provides a clear answer to one long-standing issue in the theory of design: the inter-dependence between subsystems, (in other words, cycles in the co-design graph). Consider, as an example, Suh's theory of *axiomatic design* [**suh01**], in which the first "axiom" is to keep the design requirements orthogonal (in other words, do not introduce cycles). This work shows that it is possible to deal elegantly with recursive constraints.

Partial Order Programming

In "Partial Order Programming" [**parkerjr89partial**] Parker studies a hierarchy of optimization problems that can be represented as a set of partial order constraints. The main interest is to study partial order constraints as the means to define the semantics of programming languages and for declarative approaches to knowledge representation.

In Parker's hierarchy, MCDPs are most related to the class of problems called *continuous monotone partial order program* (CMPOP). CMPOPs are the least specific class of problems studied by Parker for which it is possible to obtain existence results and a systematic solution procedure. MCDPs subsume CMPOPs. A CMPOP is an MCDP where: 1) All functionality and resources belong to the same poset $\mathbf{P}$ ($\mathbf{F}_v = \mathbf{R}_v = \mathbf{P}$); 2) Each functionality/resource relation is a simple map, rather than a multi-valued relation; 3) There are no dangling functionality edges in the co-design diagram ($\mathbf{F} = \mathbf{1}$).

In a MCDP, each DP is described by a Scott continuous map $h : \mathbf{F} \to \mathcal{A}\mathbf{R}$ which maps one functionality to a minimal set of resources. By contrast, in a CMPOP an operator corresponds to a Scott continuous map $h : \mathbf{F} \to \mathbf{R}$. The consequence is that a CMPOP has a unique solution [**parkerjr89partial**], while an MCDP can have multiple minimal solutions (or none at all).

Abstract interpretation

The methods used from order theory are the same used in the field of *abstract interpretation* [**cousot14abstract**]. In that field, the least fixed point semantics arises from problems such as computing the sets of reachable states. Given a starting state, one is interested to find a subset of states that is closed under the

dynamics (in other words, a fixed point), and that is the smallest that contains the given initial state (in other words, a *least* fixed point). Reachability and other properties lead to considering systems of equation of the form

$$x_i = \varphi_i(x_1, \dots, x_i, \dots, x_n), \quad i = 1, \dots, n, \tag{1.22}$$ {{eq:ai}}

where each value of the index i is for a control point of the program, and φ_i are Scott continuous functions on the abstract lattice that represents the properties of the program. In the simplest case, each x_i could represent intervals that a variable could assume at step i. By applying the iterations, one finds which properties can be inferred to be valid at each step.

We can repeat the same considerations we did for Parker's CMPOPs vs MCDPs. In particular, in MCDP we deal with multi-valued maps, and there is more than one solution.

In the field of abstract interpretation much work has been done towards optimizing the rate of convergence. The order of evaluation in (1.22) does not matter. Asynchronous and "chaotic" iterations were proposed early [**cousot77asynchronous**] and are still object of investigation [**bourdoncleefficient**]. To speed up convergence, the so called "widening" and "narrowing" operators are used [**cortesi11widening**]. The ideas of chaotic iteration, widening, narrowing, are not immediately applicable to MCDPs, but it is a promising research direction.

Solutions to exercises