Louise A. Dennis
January 26, 2021

Trying to Use the Compositional Framework

I'm trying to use our compositional framework to deduce some system properties and I'm grappling with a couple of things. This has resulted in a couple of suggestions for changes to the framework, though I'd also be grateful if someone could check my working here!!

1 Three Modules

I'm working with a variant of our original example where a robot has to plot an obstacle free path to a goal while accounting for battery level. I've simplified it somewhat to focus on the bits that are of interest. I have the following three modules:

Goal Selection Picks a target location based on a heat-map and whether a recharge is needed.

Planning Plans a route to the target location

Plan Execution Executes the plan if there is enough battery power, otherwise it requests a recharge from goal selection

I'm aiming to prove a property along the lines of:

> If there is not enough battery power for the current plan then eventually the goal will be the charging position and a plan to go there will be executed.

2 Updating Module Contracts

At the moment in our framework, a contract for a Module, M, consists of four components

Inputs, $\overline{i_M}$ A signature of symbols/signals that are input to the module

Outputs, $\overline{o_M}$ A signature of symbols/signals output by the module

Assume, $\mathcal{A}_M(\overline{i_M})$ A logical formula over the inputs

Guarantee, $\mathcal{G}_M(\overline{o_M})$ A logical formula over the outputs.

Informally this says that when the module receives an input, if the Assume holds then it will eventually produce an output for which the Guarantee holds.

We seem to have rapidly got to a situation where all our modules include their input signature in their output signature. I'd like instead to have a signature, Σ, of all symbols in the system and then represent modules as:

Inputs, $\overline{i_M}$ A list of signals that input into this module.

Fluents, $\overline{f_M}$ Symbols/signals whose value may change during execution of the module: $\overline{f_M} \subseteq \Sigma$

Assume, $\mathcal{A}_M$ A logical formula over Σ.

Guarantee, $\mathcal{G}_M$ A logical formula over Σ.

We then had a calculus for combining inference over modules, but I've found in my proofs I keep needing slight adaptations/extensions of it.

First, I added explicitly the concept of input and output signals that are implicit in the current calculus: $M^{\uparrow}$ and $M^{\downarrow}$ where $M^{\uparrow}$ means M receives an input and $M^{\downarrow}$ means M emits an output. To be honest I think we'll need concepts like these anyway to extend the framework to handle streams of inputs and so on.

The first rule I'm introducing is a formalisation of what a contract means. I've called this the **Contract Rule**:

$$\forall M \cdot \mathcal{A}_M \wedge M^{\uparrow} \Rightarrow \Diamond(\mathcal{G}_M \wedge M^{\downarrow}) \tag{1}$$

[*]Notes in this series are for ϵ-baked ideas, for $1 \geq \epsilon \geq 0$. Only exceptionally should they be cited.

If module, M, receives an input and its assume is true then eventually it will emit an output and its guarantee will be true.

The biggest change I've needed to work with the framework is some way of handling how the truth of formulae not involving fluents is affected by module execution (which should be not at all). So I have a variant on the contract rule which is:

$$\forall M, \overline{x} \cdot \overline{x} \cup f_M = \emptyset \wedge \phi(\overline{x}) \wedge \mathcal{A}_M \wedge M^{\uparrow} \Rightarrow \lozenge(\mathcal{G}_M \wedge M^{\downarrow} \wedge \phi(\overline{x})) \tag{2}$$

If module, M, receives an input and its assume is true then eventually it will emit an output and its guarantee will be true. Moreover if, at input, some formula not involving any fluents is true then it is still true at output.

I call this the **Fluent Rule**. Obviously the contract rule is a special case of the fluent rule. I use this one *a lot*.

Then I want to adapt the rules in the FOL framework that allow us to connect results from more than two modules. I started with something much more like our framework rules, but ended up with something that just talks about how signals go around the system: I've adapted from what I think is an implicit assumption that for a module to execute it needs to receive all inputs, to only needing one input. I've called it the **Chain Rule**:

$$\forall M_1, M_2 \cdot M_1^{\downarrow} \wedge M_1^{\downarrow} \in \overline{i_{M_2}} \Rightarrow M_2\uparrow \tag{3}$$

If, a module, M_1 emits an output that output is an input signal for another module M_2 then M_2 receives an input signal.

Note:. The Contract/Fluent rules can/should probably be extended to specify that it is the *next* output in the sometime part of the rule. I didn't need it for this proof, but it might prove useful to have the ability to talk about the next output and some eventual output when we discuss more complex systems. Similarly, I assume that any input triggers module execution, but where a module can get input from more than one other module we might want to distinguish between modules that execute on any input and modules that need all inputs and have a variety of chain rules accordingly. Likewise, at some point, we might want to distinguish between outputs.

3 Contracts for the Modules

So I have a signature for my System:

$$\Sigma = \{GoalSet \in \{((\mathbb{N},\mathbb{N}),\mathbb{N})\}, g \in (\mathbb{N},\mathbb{N}), chargePos \in (\mathbb{N},\mathbb{N}), recharge \in \{true, false\}, s_o \in (\mathbb{N},\mathbb{N}),$$
$$plan \in \{(\mathbb{N},\mathbb{N}), b \in \mathbb{N}, t \in \mathbb{N}, execute : \{(\mathbb{N},\mathbb{N})\} \rightarrow \{true, false\}\}\}$$

3.1 Goal Selection

The Goal Selection Module, G, picks a target location based on a heat-map and whether a recharge is needed.

Inputs $E^{\downarrow}$ (Plan Execution Module), $H^{\downarrow}$ (Infra red sensor – not specified and doesn't seem to be necessary for the proof).

Fluents g

Assume $\mathcal{G}_E$

Guarantee $g \neq chargepos \Rightarrow (\exists h \cdot h \in \mathbb{N} \wedge (g,h) \in GoalSet) \wedge (\forall x, y, h_1 \cdot ((x,y),h_1) \in GoalSet \Rightarrow h \geq h_1)) \wedge (recharge \iff g = chargePos)$

Here we see that:

- the Goal Selection modules's **inputs** are the Plan Execution Module, E, and an infra red sensor that generates heat maps (which go into $GoalSet$).
- It's **fluents** are g – it generates a new target goal $\{g = (x,y)\}$.
- It **assumes** that its inputs meet the guarantees of the plan execution module and
- in turn it **guarantees** that:
  1. if the output goal is not the charge Position ($g \neq chargePos$) then it is in the goal set and for all other goals in GoalSet the heat is higher than for g (i.e., g is the location in the Goal Set with the highest heat).
  2. If a recharge is needed then g is the charge position.

3.2 Planning

The Contract for the Planner is.

Inputs $G^{\downarrow}$ (Goal Selection Module), $S^{\downarrow}$ (a SLAM module (not specified) that provides the current position).

Fluents *plan*

Assume $\mathcal{G}_G$

Guarantee $g = s_0 \Rightarrow plan = \{(s_0, s_0)\} \wedge s_0 \in plan \wedge g \in plan \wedge (\exists q, r \cdot q, r \in plan \wedge adjacent(s_0, q) \wedge adjacent(g, r)) \wedge \forall p_0 \cdot p_0 \in plan \wedge p_0 \neq s_0 \wedge p_0 \neq g \Rightarrow (\exists q, r \cdot q, r \in p \wedge adjacent(q, p_0) \wedge adjacent(r, p_0))$

Here we see that:

- the Planner's **inputs** are the Goal Selection Module, $G^{\downarrow}$, and some SLAM module $S^{\downarrow}$.
- It's **fluents** *plan* – it generates a motion plan to reach the goal.
- It **assumes** that its inputs meet the guarantees of the Goal Selection module and
- It **guarantees** that:
  1. If the goal equals the starting point then the plan is a null plan (moves from s_0 to s_0).
  2. Both the agent's initial position and the target goal are contained in the plan.
  3. If there are points q, r in a plan next to the initial position and goal respectively.
  4. For all points p_0 in the plan not equal to the initial position or goal position, then there are points q, r in the plan next to p_0.

3.3 Plan Execution

The contract for the Plan Execution module is:

Inputs $P^{\downarrow}$ (Planning Module), $B^{\downarrow}$ (The Battery).

Fluents *recharge*, *execute*

Assume $\mathcal{G}_P$

Guarantee $(((b - t) > length(plan) \vee g = chargePos) \iff execute(plan)) \wedge ((execute(plan) = false \wedge recharge = true) \iff (b - t) \leq length(plan) \wedge g \neq chargePos) \wedge \forall p \cdot p \neq plan \Rightarrow execute(p) = false$

Here we see that:

- the Plan Execution module's **inputs** the Planner and some battery module.
- It's **fluents** are the recharge signal/command and plan execution signal/command
- It **assumes** that its inputs meet the guarantees of the Planning module and
- It **guarantees** that:
  1. It the length of the plan is greater less than the available battery power (minus some threshold, t, reserved to, for instance, ensure the robot can always reach the charging position) or current goal is the charging position, then it executes the plan.
  2. It calls recharge and doesn't execute the current plan if the length of the plan is below the available battery power (minus the threshold) and the current goal is not the charging position.
  3. It doesn't execute anything other than the plan.

4 What I want to prove

I property I would like to establish is.

> If the current plan is longer than the battery available (minus threshold), then eventually the current plan will contain the charging position as the goal.

NB. Ideally, this would extend to "and the current plan gets executed" but I ran out of energy...

After a certain amount of fiddling, I ended up with:

$$\Box((\mathcal{A}_E \wedge E^{\uparrow} \wedge \mathit{length}(\mathit{plan}) > b - t) \Rightarrow \Diamond(\mathcal{G}_E \wedge E^{\downarrow} \wedge g = \mathit{chargePos} \wedge g \in \mathit{plan})) \tag{4}$$

> It is always the case that if the Plan Execution module receives an input signal and its Assume is true and at that time length of the current plan is greater than the battery (minus threshold) then eventually the module will emit an output, its guarantee will hold and the current plan will contain the charging position as the goal.

I'm used to working with theorem provers – so my approach to this sort of thing is to work backwards breaking things down into subgoals. I'm going to handwave a lot of stuff in what follows with "by standard logical manipulation" even though I'm aware that this is where theorem provers often say "no".

From standard logic we can assume ($g = \mathit{chargePos} \vee g \neq \mathit{chargePos}$). So again, from standard logical manipulation I'm splitting my theorem into two subgoals:

- $\Box((\mathcal{A}_E \wedge E^{\uparrow} \wedge \mathit{length}(\mathit{plan}) > b - 15 \wedge g = \mathit{chargePos}) \Rightarrow \Diamond(\mathcal{G}_E \wedge E^{\downarrow} \wedge g = \mathit{chargePos} \wedge g \in \mathit{plan}))$
- $\Box((\mathcal{A}_E \wedge E^{\uparrow} \wedge \mathit{length}(\mathit{plan}) > b - 15 \wedge g \neq \mathit{chargePos}) \Rightarrow \Diamond(\mathcal{G}_E \wedge E^{\downarrow} \wedge g = \mathit{chargePos} \wedge g \in \mathit{plan}))$

4.1 First Subgoal

Intuitively this should hold because the Planning module guarantees that $g \in \mathit{plan}$ as part of its output and if $g = \mathit{chargePos}$ then we're done. We can therefore ignore the whole bit about plan length.

So let's reduce our goal to:

$$\Box((\mathcal{A}_E \wedge E^{\uparrow} \wedge g = \mathit{chargePos}) \Rightarrow \Diamond(\mathcal{G}_E \wedge E^{\downarrow} \wedge g = \mathit{chargePos} \wedge g \in \mathit{plan}))$$

This is where I needed something like the fluent rule, in order to prove that if $g = \mathit{chargePos}$ before execution of E then $g = \mathit{chargePos}$ after execution. Anyway, since nether g nor plan are in f_E, I can use the fluent rule to get the goal:

$$\Box(\mathcal{A}_E \wedge E^{\uparrow} \wedge g = \mathit{chargePos}) \Rightarrow (\mathcal{A}_E \wedge E^{\uparrow} \wedge g = \mathit{chargePos} \wedge g \in \mathit{plan})$$

(i.e., if the inputs to the Plan Execution agent have the current goal as the charging position, then the current goal is the charging position (trivially true) and g is in the plan)

A bit of tidying and simplifying here reduces the goal to:

$$\mathcal{A}_E \Rightarrow g \in \mathit{plan}$$

(i.e., The assume of the Plan Selection agent implies that g is in the plan).

We have $\mathcal{A}_E = \mathcal{G}_P$ so we can change the goal to:

$$\mathcal{G}_P \Rightarrow g \in \mathit{plan}$$

Which is trivially true.

4.2 Second Subgoal

$$\Box(\mathcal{A}_E \wedge E^{\uparrow} \wedge (\mathit{length}(\mathit{plan}) > b - 15 \wedge g \neq \mathit{chargePos})) \Rightarrow \Diamond(\mathcal{G}_E \wedge E^{\downarrow} \wedge g = \mathit{chargePos} \wedge g \in \mathit{plan}))$$

So I'm going to use my first subgoal here to reduce the goal to:

$$\Box(\mathcal{A}_E \wedge E^{\uparrow} \wedge (\mathit{length}(\mathit{plan}) > b - 15 \wedge g \neq \mathit{chargePos})) \Rightarrow \Diamond(\mathcal{A}_E \wedge E^{\uparrow} \wedge g = \mathit{chargePos})$$

(i.e., if at some point on when the Plan Execution module receives an input the current plan is over the battery threshold then eventually the Plan Execution module will receive an input and the target goal will be the charging position)

Using the chain rule (if P emited a signal then E receives a signal) and $\mathcal{G}_P = \mathcal{A}_E$:

$$\Box(\mathcal{A}_E \wedge E^{\uparrow} \wedge (length(plan) > b - 15 \wedge g \neq chargePos)) \Rightarrow \Diamond(\mathcal{G}_P \wedge P^{\downarrow} \wedge g = chargePos)$$

g is not in f_P so we can use the fluent rule:

$$\Box(\mathcal{A}_E \wedge E^{\uparrow} \wedge (length(plan) > b - 15 \wedge g \neq chargePos)) \Rightarrow \Diamond(\mathcal{A}_P \wedge P^{\uparrow} \wedge g = chargePos)$$

Chain rule and $\mathcal{G}_G = \mathcal{A}_P$:

$$\Box(\mathcal{A}_E \wedge E^{\uparrow} \wedge (length(plan) > b - 15 \wedge g \neq chargePos)) \Rightarrow \Diamond(\mathcal{G}_G \wedge G^{\downarrow} \wedge g = chargePos)$$

Now the guarantee of G says that $g = chargePos \iff recharge$ so that gives us:

$$\Box(\mathcal{A}_E \wedge E^{\uparrow} \wedge (length(plan) > b - 15 \wedge g \neq chargePos)) \Rightarrow \Diamond(\mathcal{G}_G \wedge G^{\downarrow} \wedge recharge)$$

Since $recharge \notin f_G$ we can use the fluent rule:

$$\Box(\mathcal{A}_E \wedge E^{\uparrow} \wedge (length(plan) > b - 15 \wedge g \neq chargePos)) \Rightarrow \Diamond(\mathcal{A}_G \wedge G^{\uparrow} \wedge recharge)$$

Chain rule and $\mathcal{G}_E = \mathcal{A}_G$:

$$\Box(\mathcal{A}_E \wedge E^{\uparrow} \wedge (length(plan) > b - 15 \wedge g \neq chargePos)) \Rightarrow \Diamond(\mathcal{G}_E \wedge E^{\downarrow} \wedge recharge)$$

$\mathcal{G}_E$ includes the conjunct $((execute(plan) = false \wedge recharge = true) \iff (b - t) \leq length(plan) \wedge g \neq chargePos)$ so a little logic gives us:

$$\Box(\mathcal{A}_E \wedge E^{\uparrow} \wedge (length(plan) > b - 15 \wedge g \neq chargePos)) \Rightarrow \Diamond(\mathcal{G}_E \wedge E^{\downarrow} \wedge (b - 15) \leq length(plan)) \wedge g \neq chargePos)$$

b, g, and $plan$ are not fluents so:

$$\Box(\mathcal{A}_E \wedge E^{\uparrow} \wedge (length(plan) > b - 15 \wedge g \neq chargePos)) \Rightarrow \Diamond(\mathcal{A}_E \wedge E^{\uparrow} \wedge (b - 15) \leq length(plan)) \wedge g \neq chargePos)$$

Which is trivially true after a bit of logical manipulation and we're done!

5 Conclusion

What is the conclusion here? Despite the fact I've changed the notation around inputs and outputs I think we can probably still use the existing notation (I went round the houses a bit figuring this out, hence why a new notation got introduced). But I think, to be useful, we really need something like the fluent rule and I think that means we really need to have some concept of module fluents and to be able to explicitly refer to the moments when signals are received and emitted.