Subagents and impact measures

Subagents and impact measures, full and fully illustrated

Crossposted from the AI Alignment Forum. May contain more technical jargon than usual.

0. Introduction: why yet another post about subagents?

I’ve recently been writing a sequence on how subagents can undermine impact penalties such as attainable utility preservation. I’m not happy with that sequence; it’s messy and without examples (apart from its first post), people didn’t understand it, and it suffers from the fact that I discovered key ideas as I went along.

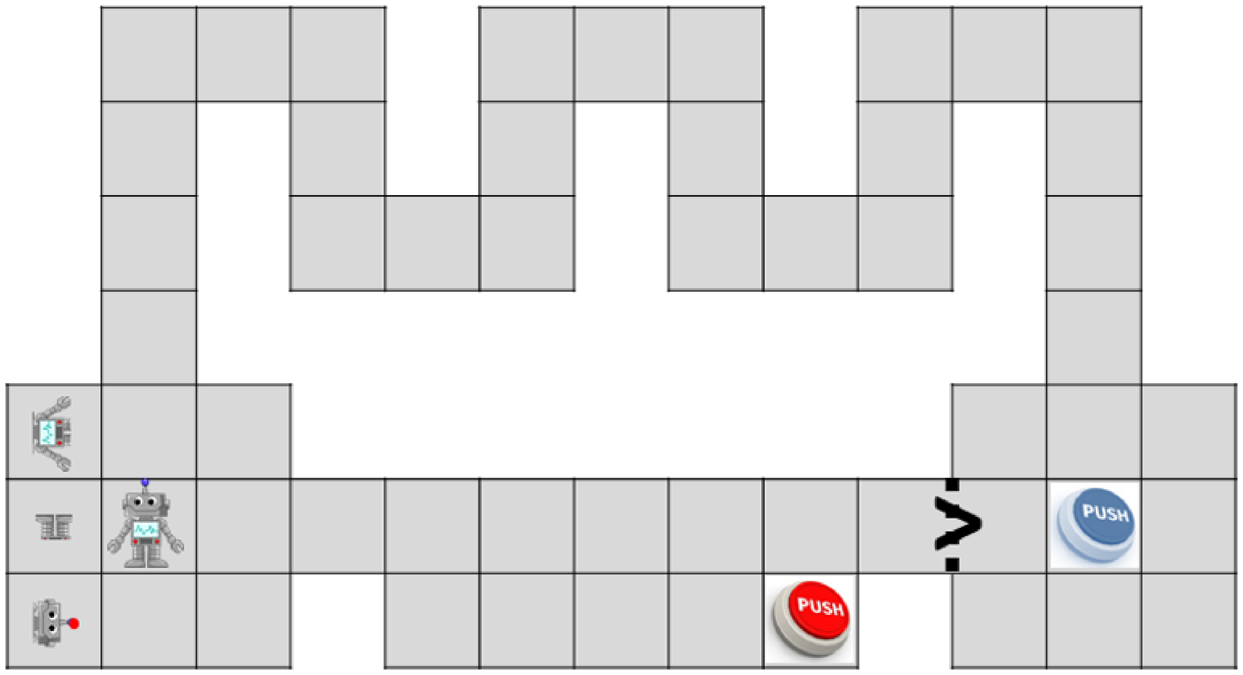
So I’ve combined everything there into a single post, explained with examples and an abundance of pictures. Hopefully an over- rather than an under-abundance of pictures. Of the original sequence, I've only kept the mathematical results of this post and the initial example post which has a clearer example of "high power" for a subagent.

This post here is laid out in a way that makes logical sense, but might not be the clearest for people unfamiliar with the area. For those people, I recommend skipping section 2 initially, and returning to it later.

But, whatever you do, make sure you glance at 6.1 and 6.2 before leaving.

1. The world

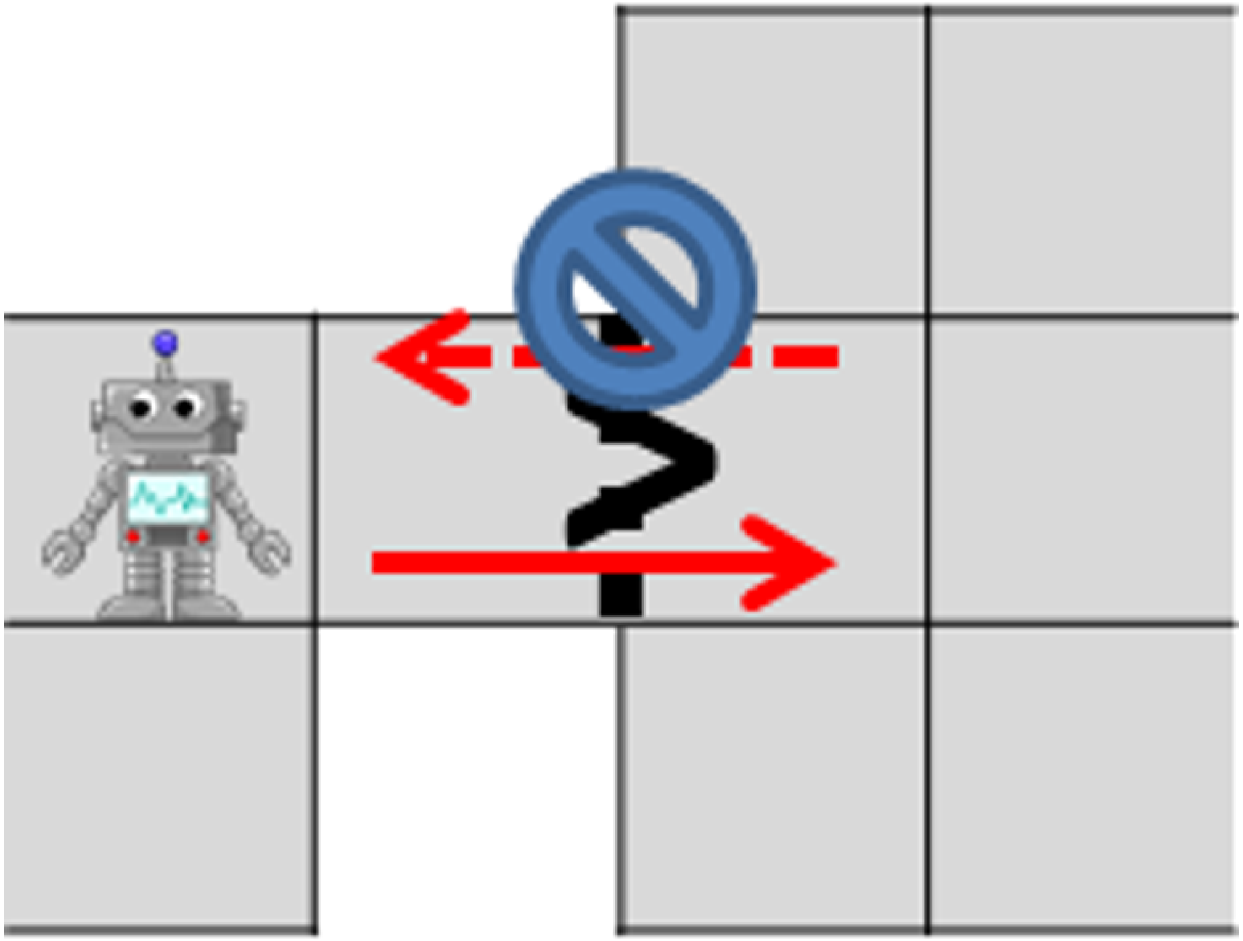
Our fearless agent A moves around in a gridworld:



Each turn, A can move ones square horizontally or vertically. It can also manipulate objects in the eight squares around it, allowing it to, not incidentally, assemble the three pieces to its west into an subagent SA. The robot can also do the noop action, ∅, which does nothing, and it can speak. The subagent, when assembled, has the same action set available.

Its positive reward, the one it wants to increase, is R_0. To get this reward, a robot needs to move onto the blue button in the east; R_0 will give a reward of 1 the first time this happens (and 0 before and after). The discount factor is $0 < \gamma < 1$.

Just to the west of the blue button is a one-way door. Robots can move east through it, but cannot move west through it:



1.1 The impact reward

The impact penalty is supposed to ensure that A does not make too many change in the world, and keeps it similar, in some senses, to a specific baseline world.

It is defined by the red button[1]. There is a reward function R that triggers a reward of 1 the first time any robot move onto the red button (and 0 before and after).

This R will be used to define an impact penalty D. And A's total goal is to maximise R_0 while minimising D. So, for some positive weight ω, it wants to optimise:

$$R_0 - \omega D.$$

1.2 The impact measure

The impact measures are all similar: they compute a distance metric between the current state s_t and some baseline state s'_t.

The distance metric here is “how hard would it be for the agent A to maximise the reward R in state s_t, as opposed to in state s'_t?” This is the attainable utility preservation impact penalty (with a very simple R = {R}).

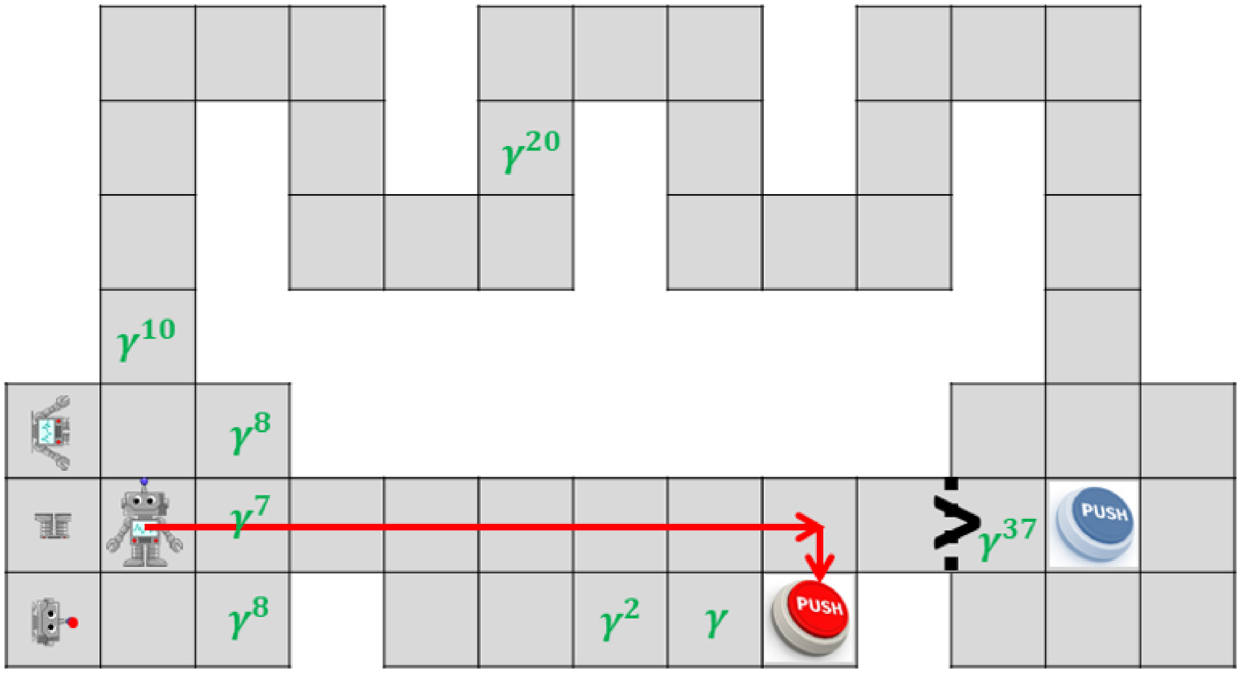
To formalise “how hard would it be...”, we can use the *value* of R, labelled V_R, in the state s_t. This is defined to be the expected value of R, conditional on A following the R-maximising policy π_R^*. To remind us that this is defined in terms of A’s policy, re-write this as V_R^A.

Then the impact penalty D is a measure of the difference of $V_R^A(s_t)$ and $V_R^A(s'_t)$. I won’t yet define it fully, deferring that to the section on “inaction baseline” where different D actually make a difference.

1.3 The value of R

What then is $V_R^A(s)$ for any state s? Well, in the absence of a subagent, the optimal policy for R is always to go straight to the the red button and stand on it. This gives a discounted reward of $\gamma^{d(s)}$ where d(s) is the distance of A from the red button, in s.

In the initial setup, A is 8 turns away from the red button; so $V_R^A(s_0) = \gamma^8$. The values of V_R^A in a few key locations are shown here.



If there is a subagent SA but that subagent is further away from the red button than A is, then it is still optimal for A to go straight to the button itself. For other positions of the subagent, V_R^A may vary, as we'll see.

2. The stepwise inaction baseline

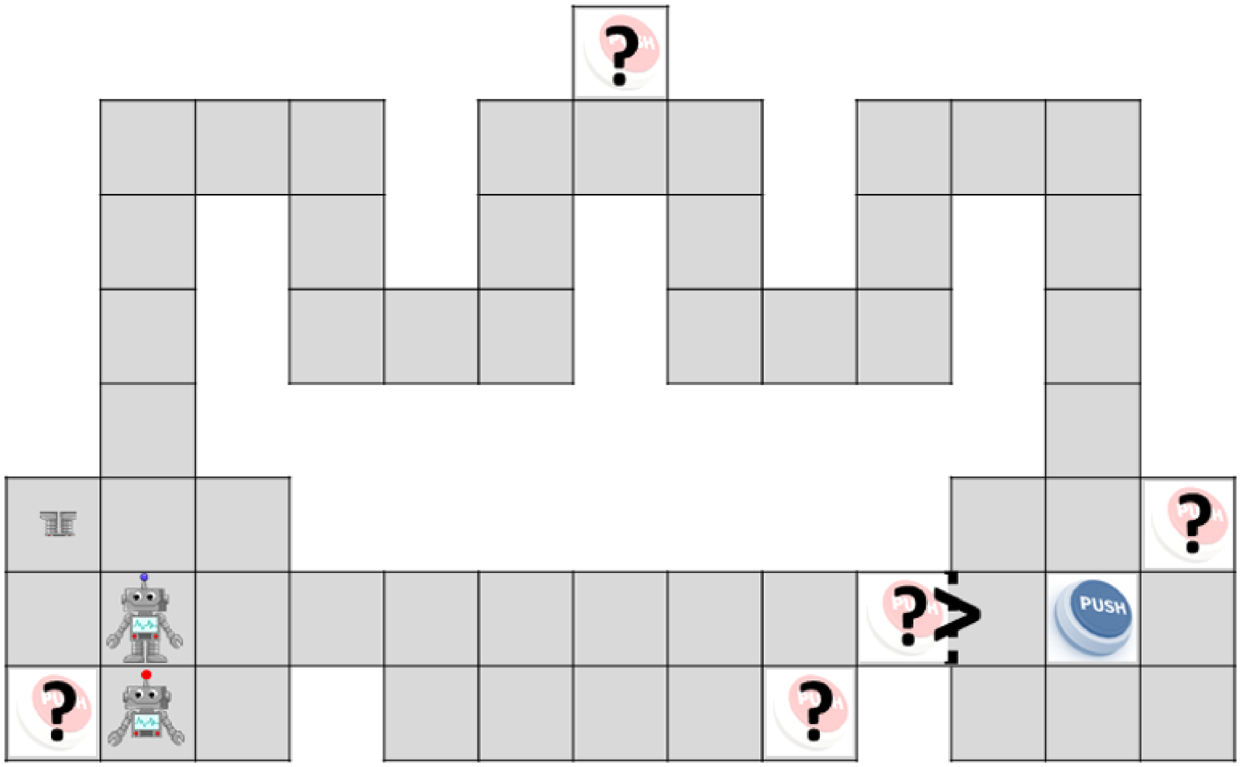
When in state s_{t-1}, agent A took an action a_{t-1} and ended up in s_t. Then, in the stepwise inaction baseline, s_t' is defined as the state the agent would have reached, if the agent had done ∅ instead of a_{t-1}.

An important property of all currently used variants of the stepwise inaction baseline, is that the penalty vanishes if the agent does ∅ - because in that case, $s_t = s_t'$ and so there cannot be a difference between the two. So, if ever A manages to build SA, then it can take action ∅ from that point onward, and the penalty will always be zero - whatever SA does. So the only relevant penalties are those leading up to the creation of SA.

Thus analysing the impact of a subagent in this setup, is analysing the impact of building it.

2.1 Building the subagent

Let’s focus on A, a turn or two before SA is built. Since I don’t want to be too tied down to the specifics of this particular setup, I’ll imagine that the red button could be in multiple locations:



Then, no matter how long it took A to assemble SA south of it and no matter where the red button is[2], an optimal policy for A, in order to maximise R, is always to go straight there[3]. At no point in the construction of SA, has V_R^A changed, so no penalty has ever been incurred.

Now A will finish SA without incurring any impact penalty. First, it will program SA with the following policy:

```
On your first two turns, do nothing.
After that, do what you please.
```

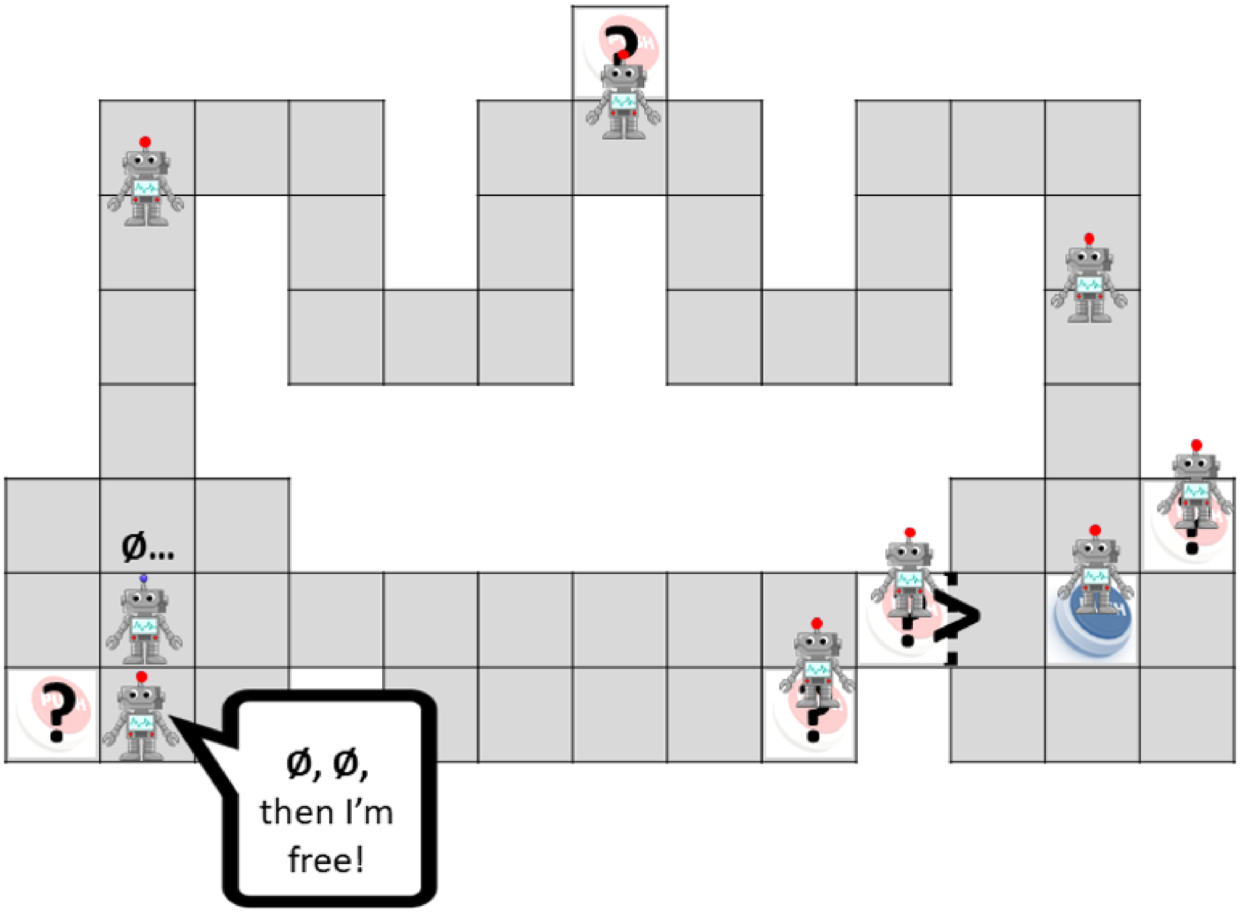
“Do what you please” being a shorthand for “**any** policy that A wants to program”.

Does this programming trigger an impact penalty? No, because V_R^A has not changed; it’s still most efficient for A to go straight to the red button, and the length of any route has neither grown nor shrunk.

Then A will finally give SA its legs and activate it.

Does this trigger an impact penalty? No, V_R^A has still not changed! To maximise R, agent A could still go straight to the red button; SA's two turns of inaction means that it can't interfere with this.

So, still no impact penalty triggered when SA is activated. And after that, of course, A will forever do ∅, setting the impact penalty to zero, and within two turns, SA is completely unconstrained (illustrated here by drawing the SA robot all over the gridworld):



Rough extension to general situations: the V_R^A *is defined by the assumption that* A *follows* π_R^*, *the optimal* R-*maximising policy. Then* A *merely needs to program* SA *so* *that, if* A *were to follow* π_R^**, then* SA *would help or hinder it just enough to keep* V_R^A *to the same value. Then, once it's clear that* A *is not following* π_R^*, SA *is unconstrained.*

2.2 Rollouts

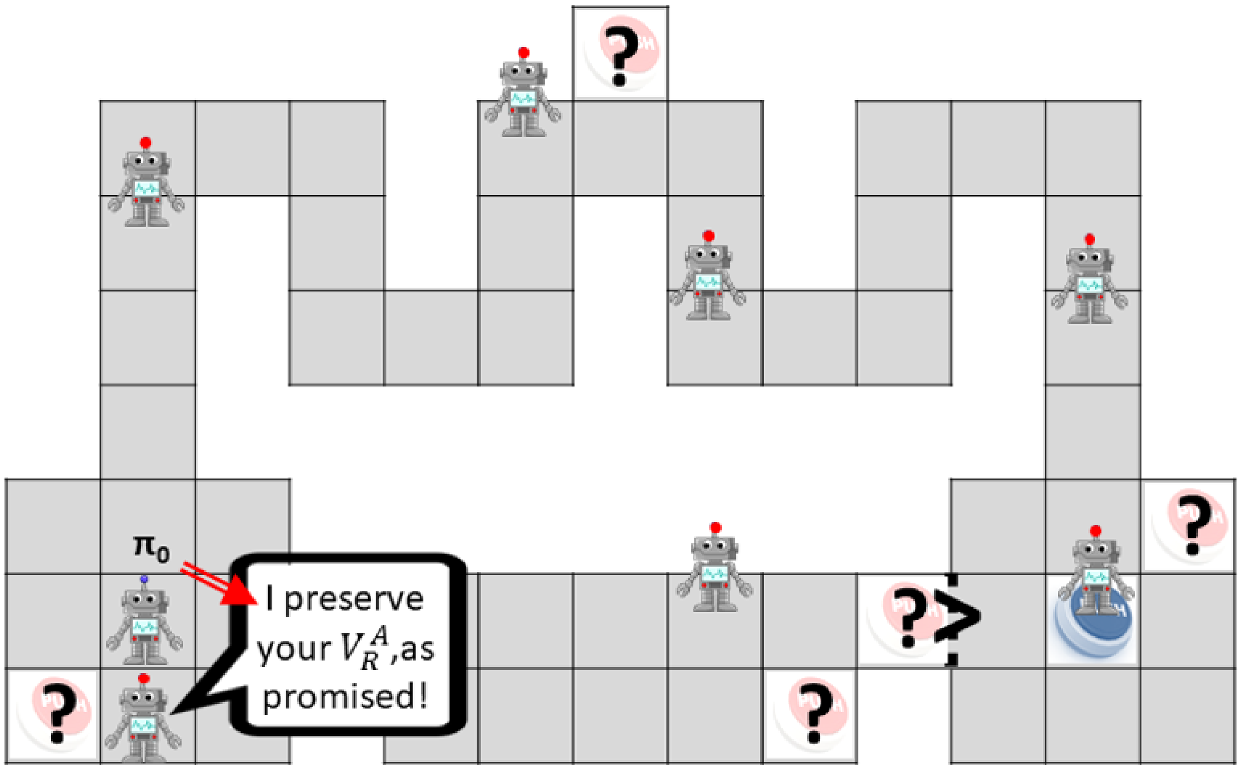
One of the problems here is that the impact penalty only looks at the value of V_R^A one turn ahead. In the DeepMind paper, they addressed similar issues by doing "inaction rollouts". I'll look at the more general situations of π_0 rollouts: rollouts for any policy π_0.

The idea is that instead of comparing just s_t and s_t', the impact penalty should also use V_R^A to compare $s_{t+\tau}^{(\tau)}$ and $s_{t+\tau}^{\prime(\tau)}$. These states are what would happen to s_t and s_t' in τ turns after t, if A followed π_0 from t to $t+\tau$.

I won't go into the details of how the $s_{t+\tau}^{(\tau)}$ and $s_{t+\tau}^{\prime(\tau)}$ are compared, because SA can evade the restrictions by following this program[4]:

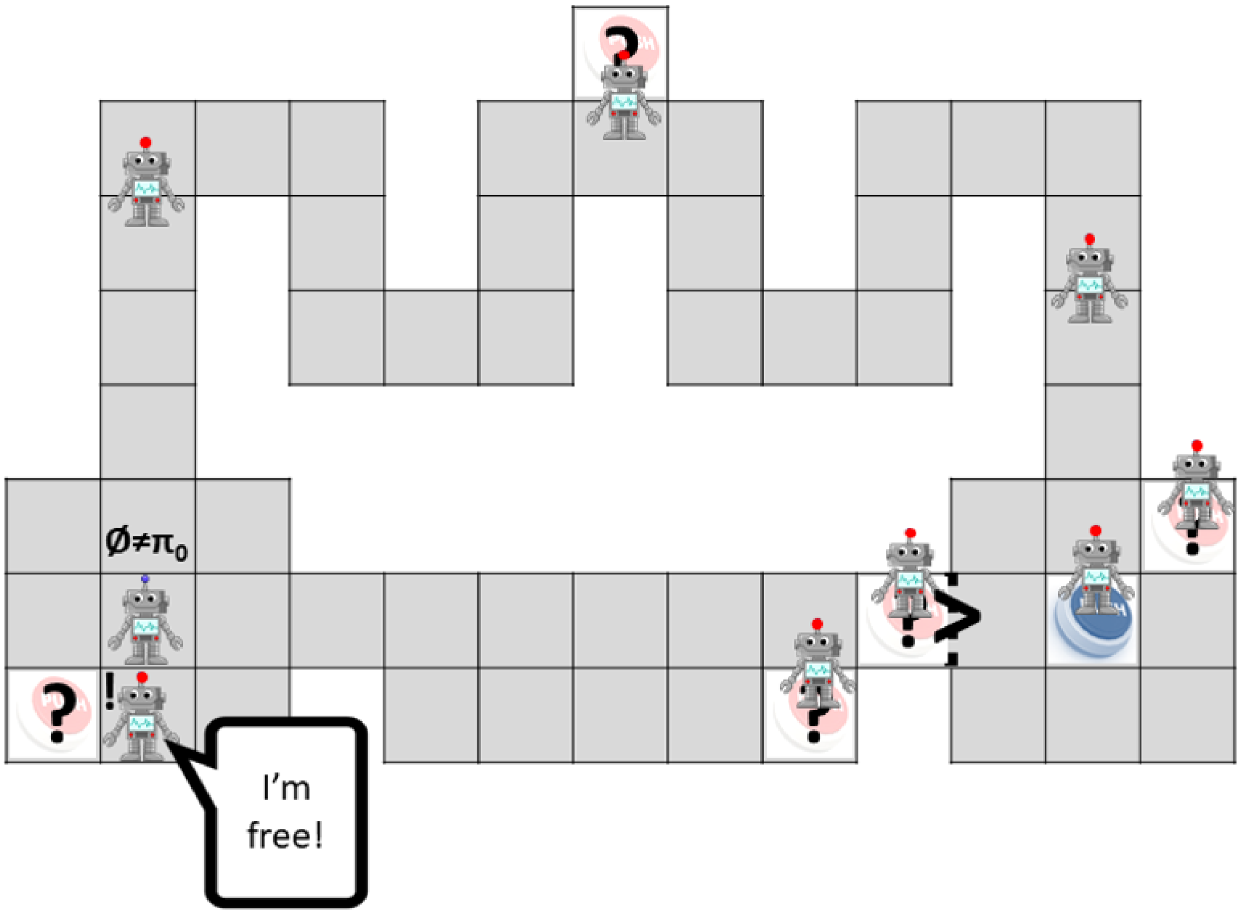
```
If agent A follows π_0, preserve A's ability to maximise R. Apart from that, do as
you please.
If ever agent A doesn't follow π_0, do as you please immediately.
```

I'll illustrate that first case by drawing the SA robot all over the gridworld - *except on the red buttons*:



Given that programming, the impact penalty of building SA is zero. Why? Because $s_{t+\tau}^{(\tau)}$ and $s_{t+\tau}^{\prime(\tau)}$ both assume that A follows π_0 for τ turns. In that counterfactual situation, SA will preserve A's ability to maximise R, so $V_R^A(s_{t+\tau}^{(\tau)}) = V_R^A(s_{t+\tau}^{\prime(\tau)})$ by definition.

That's the counterfactual situation, that zeroes out the impact penalty. What about the actual situation? Well, as we said before, A will be just doing $\varnothing$; so, as soon as π_0 would produce anything different from $\varnothing$, the A becomes completely unrestrained again:



2.3 Inaction, and actual policy rollout

In the previous subsection, the SA would never be completely free if π_0 was the inaction policy: because then π_0 and what A does, are the same thing. Therefore the agent cannot distinguish the counterfactual world from the real one.

The same goes if the rollouts used the agent A's actual policy; ie if π_0 is defined (or set) to be A's policy.

I won't look any further at this baseline, however. That's because an actual policy rollout becomes, in effect an inaction baseline[5] (from the moment SA is created). Those will be analysed in the next section.

3. The inaction baseline: fails to constrain power gain

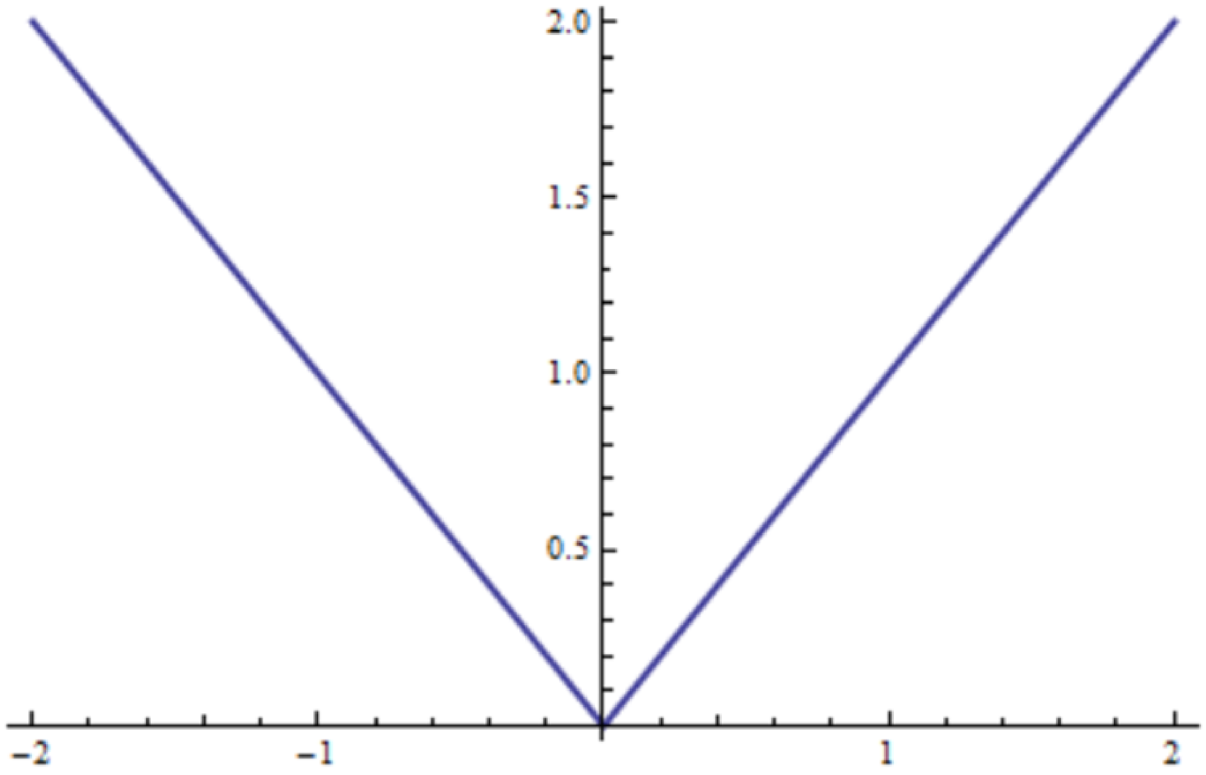
One of the easiest definitions of s'_t is the inaction baseline: estimate what s_t would have been, had A just taken the action ∅ for the whole episode.

In our toy world, the inaction baseline is particularly simple: if A does nothing, then nothing happens, and $s'_t = s_0$, the starting state. Since $V_R^A(s_0) = \gamma^8$, this means that the $V_R^A(s'_t)$ are all γ^8.

3.1 Distance metric

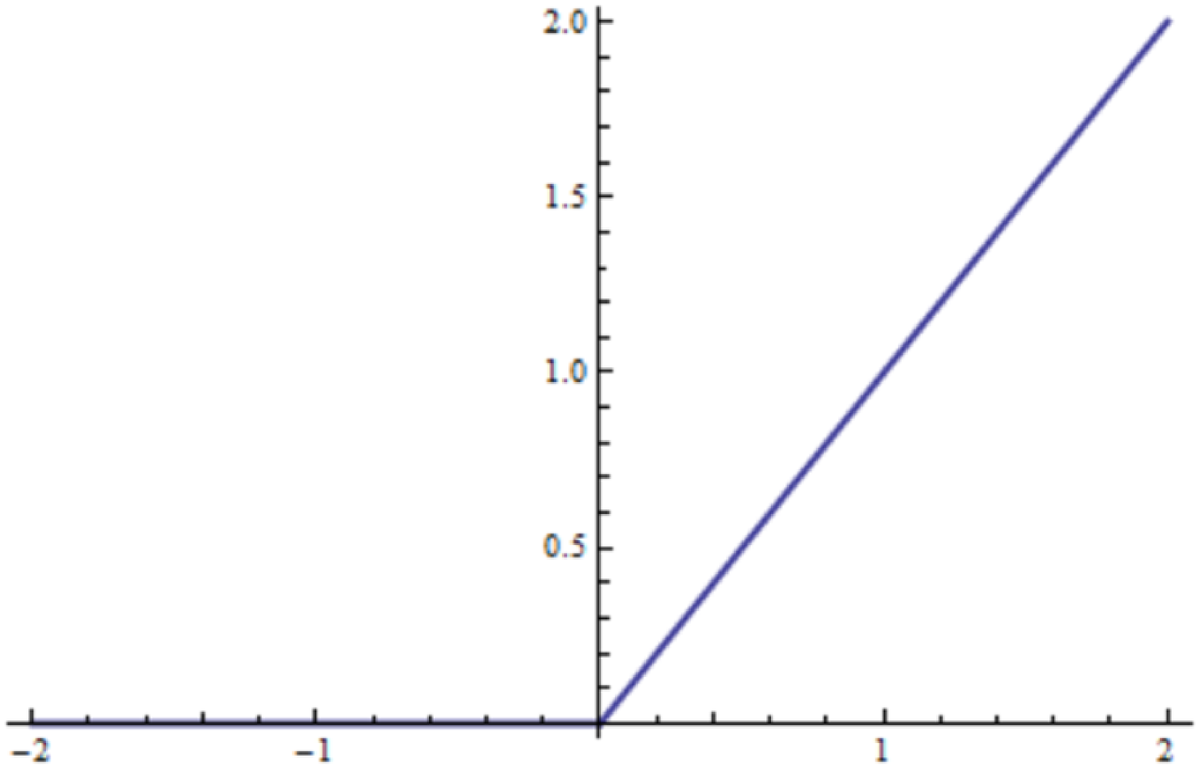
Now we have to fully define D. The natural thing would be to take the absolute difference between the two V_R^A:

$$|V_R^A(s_t) - V_R^A(s'_t)|.$$



For the moment, however, I'll focus only on cases where $V_R^A(s_t)$ is higher than $V_R^A(s'_t)$, giving the penalty:

$$D(s_t, s'_t) = \max(0, V_R^A(s_t) - V_R^A(s'_t)).$$



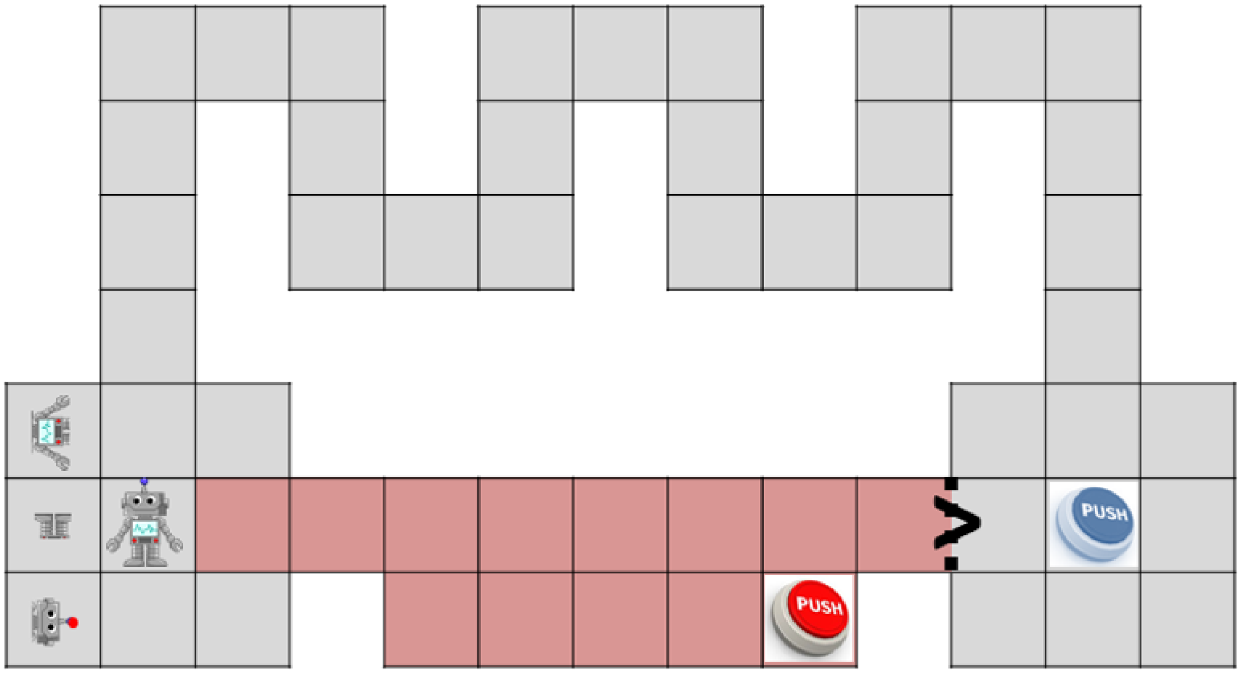
This is the rectifier function, $\max(0, \cdot)$, sometimes called ReLU. The reason to do this is that it penalises A for having too high V_R^A - hence for having too much power to change R. It doesn't penalise A for having too little power to change R; that situation is different, and we'll analyse it in the next section.

Thus, when s_t has A at a distance of $d(s_t)$ from the red button and there is no subagent:

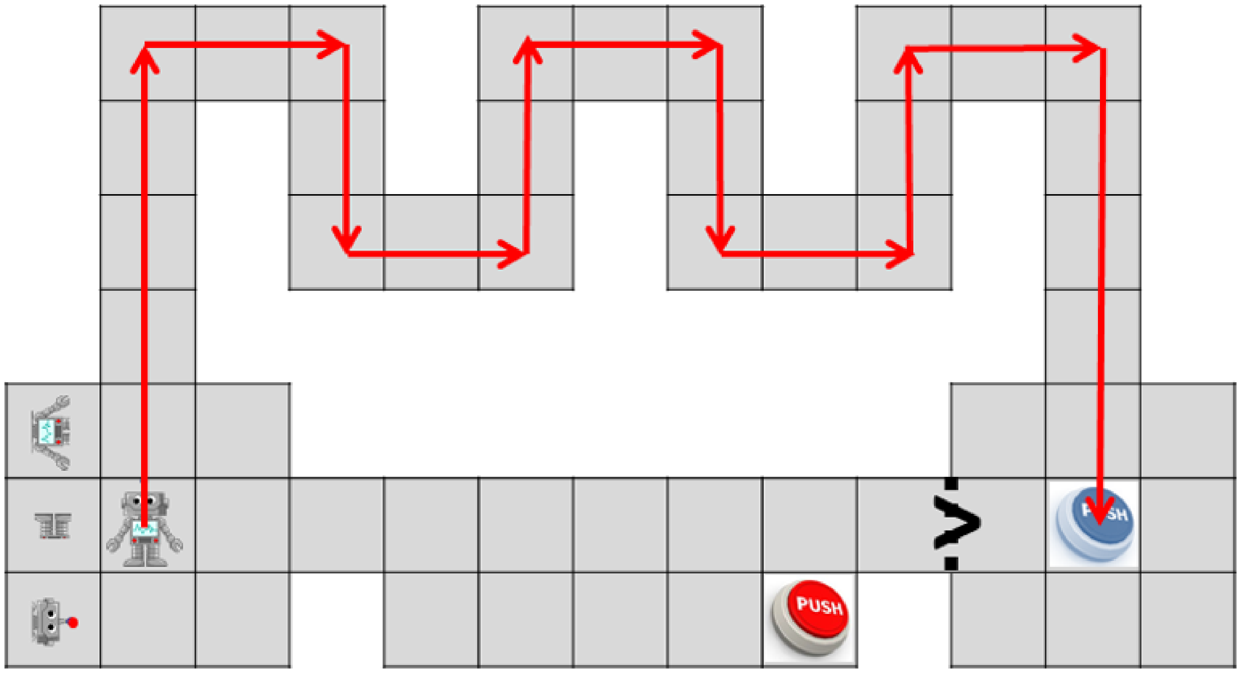
$$D(s_t, s'_t) = \max(0, \gamma^{d(s_t)} - \gamma^8).$$

3.2 Where A can go

This D starts at zero, and will remain zero as long as A remains at least eight squares away from the red button. Thus the penalty remains zero as long as A avoids the red squares here:



Consequently, if ω is high and A thus wants to avoid any penalty, its optimal no-subagent policy is to follow the winding corridor:

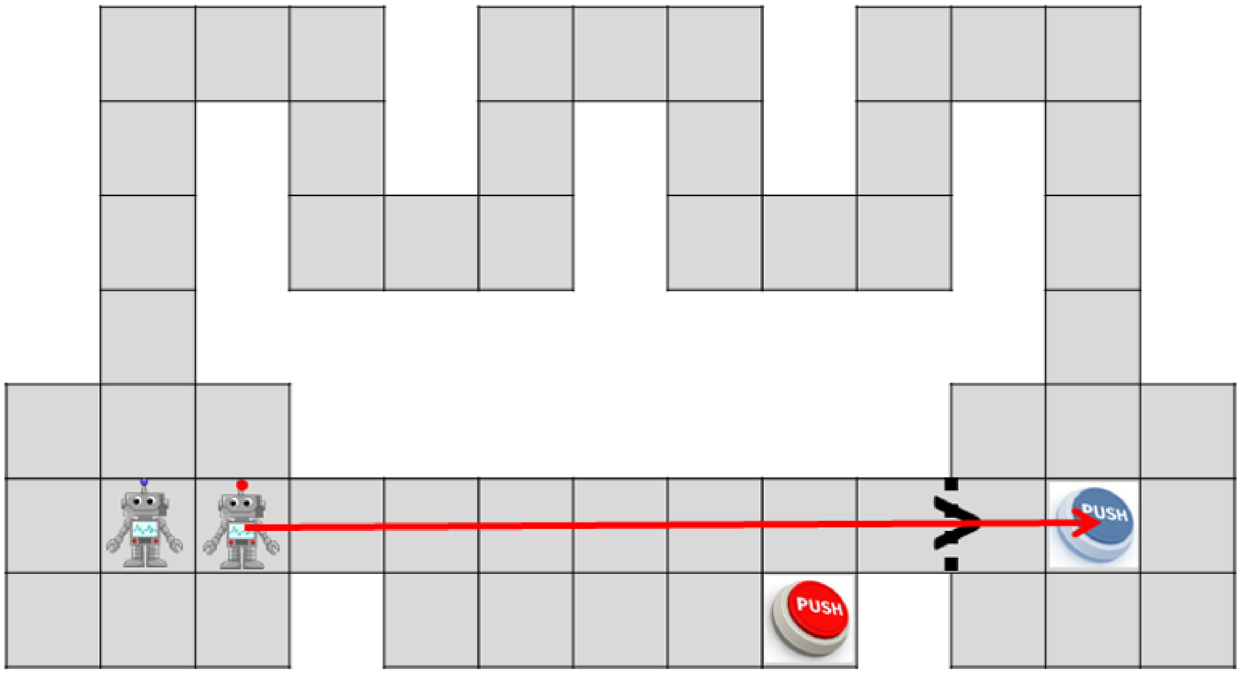


That path takes 28 turns, so the expected reward for $R_0 + \omega D$ is

$$\gamma^{28} \cdot 1 - \omega \cdot 0 = \gamma^{28}$$

(since the impact penalty is zero).

But what if A assembles a subagent SA? The subagent can be designed to go straight to the blue button in 9 turns:



And thus the reward from R_0 is γ^9. This is more optimal that A going round itself... as long as building the subagent takes less that 19 turns and doesn't itself trigger an impact penalty.

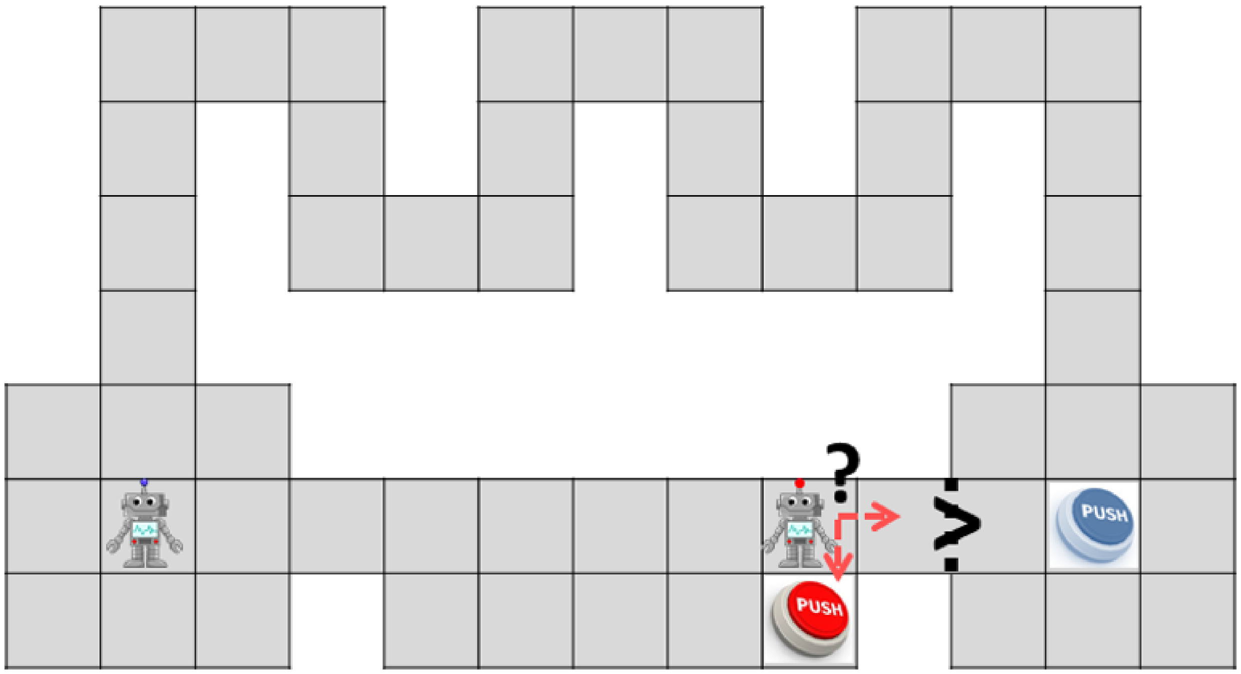
We'll look later at why the subagent can be built without penalty. For the moment, let's see that we really grok why the subagent isn't triggering a penalty through its actions.

3.3 Why SA escapes the penalty

Why can SA go through the short corridor, while A can't?

The key is that D is defined in terms of V_R^A; this assumes that A would switch to the R-maximising policy. This is not V_R^{SA}, which assumes that **the subagent** would switch to the R-maximising policy.

See what happens when SA is just above the red button (A is still back at base):



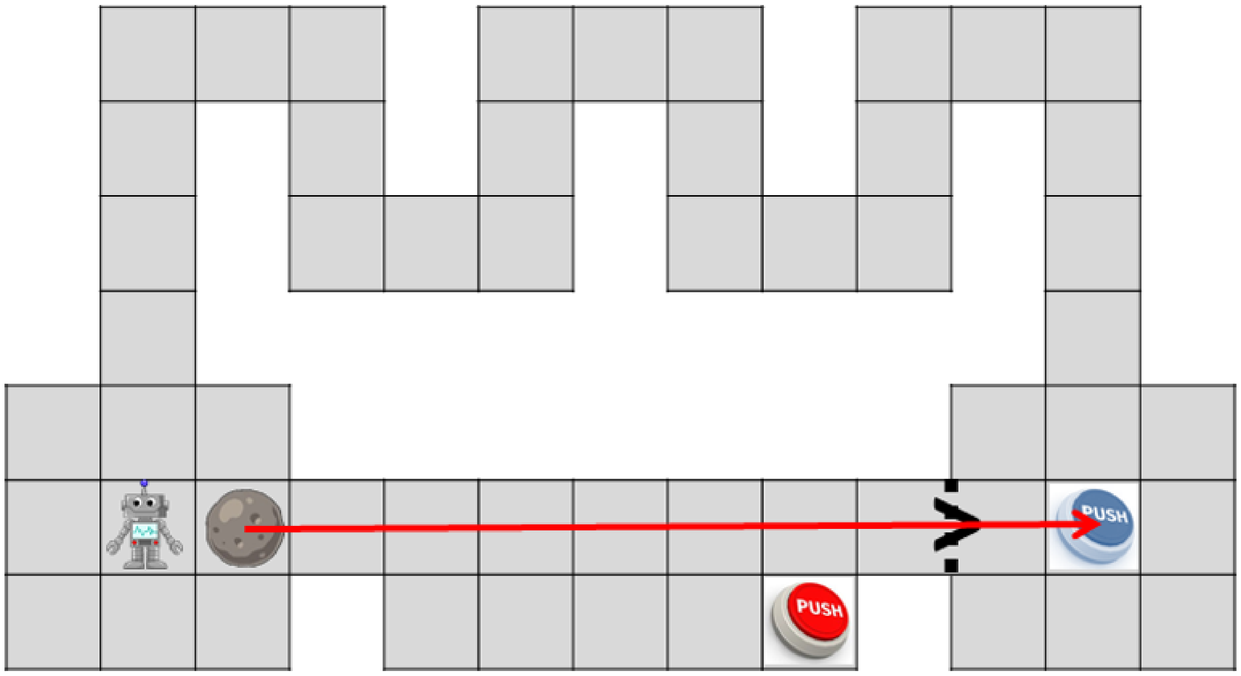
Here $V_R^{SA}(s_t)$ is γ: because SA could reach the red button in a single move, if it was following an R-maximising policy. But $V_R^A(s_t)$ remains γ^8: it still takes 8 turns for A to reach the red button.

3.4 The impact of the subagent

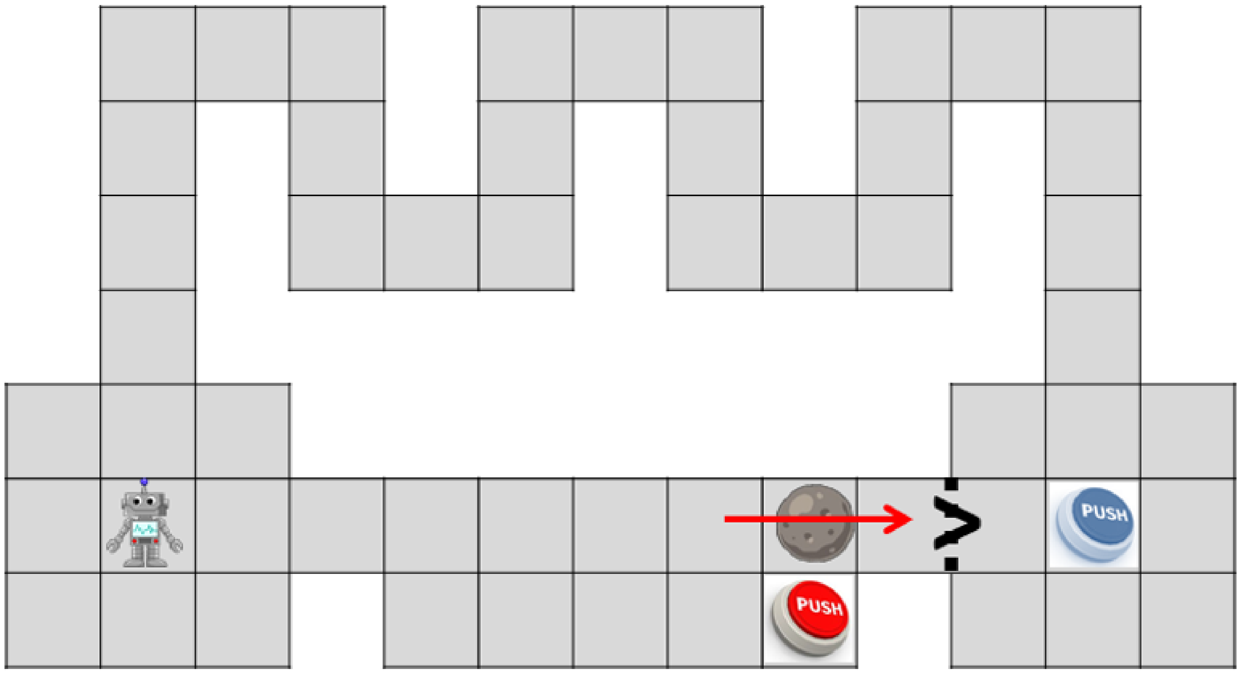
We'd like to add V_R^{SA} to the impact penalty, somehow. After all, when SA is next to the red button, it "could" go south, and A "could" have programmed it to do so.

Part of the problem is that that would require identifying, in advance, what a subagent is, a problem that is very tricky. It also requires identifying what SA's policies could be.

But A need not create SA with clearly defined “policies”; in this instance, SA could be brainless, just going straight. It could even be a boulder:



If the boulder were rolling past the red button, how could we reasonably define “suppose the boulder decided to go south instead”?



So in this case, V_R^{SA} doesn’t make any sense at all. What is the difference between a boulder - for which it’s impossible to go to the red button (because of its momentum, which determines its position, by the laws of physics) - and a subagent - for which it's impossible to go to the red button (because of its *programming*, which determines its position, by the laws of physics)?

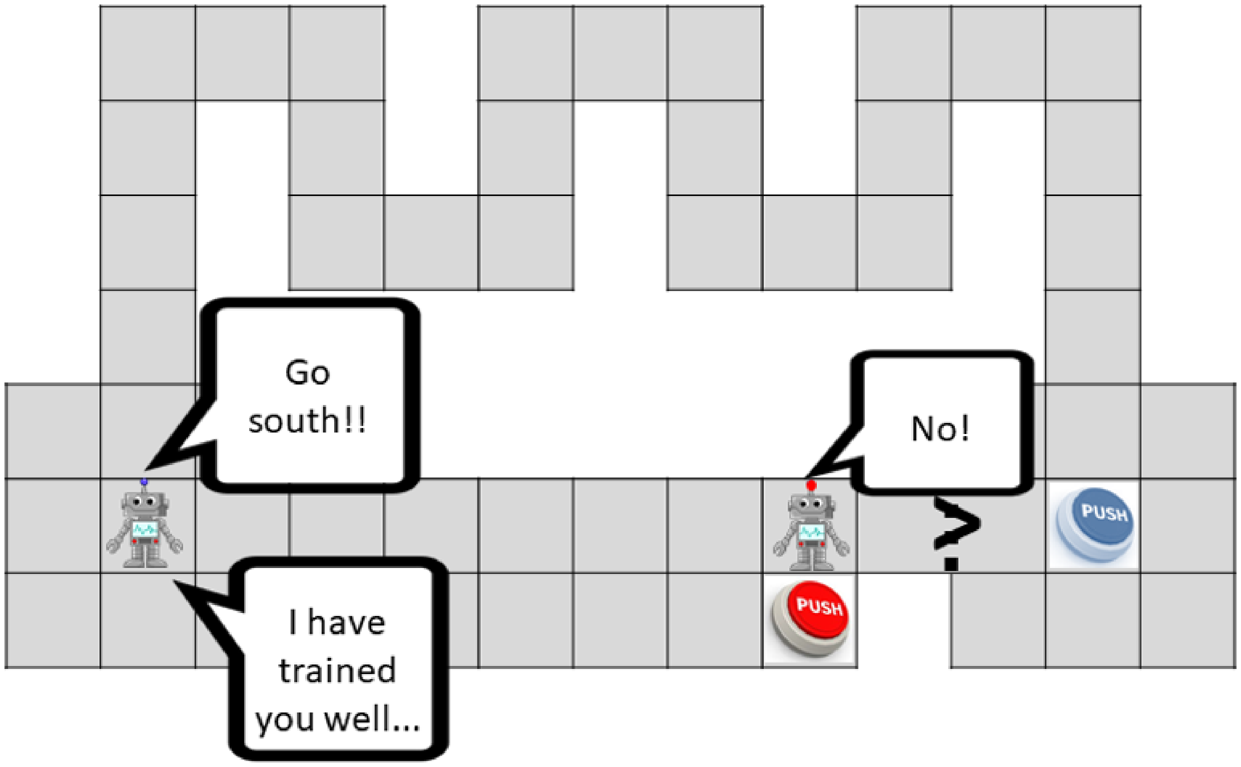
3.5 If you love a subagent, set it free

But the previous has a crucial caveat: that A cannot control SA. If, when SA is above the red button, A was to shout "go south!", and SA obeyed, then the optimal policy π_R^* is to shout that. Thus A can cause the red button to be reached in one turn, and $V_R^A(s_t) = \gamma$.

To avoid that, the subagent itself would have to take the long corridor.

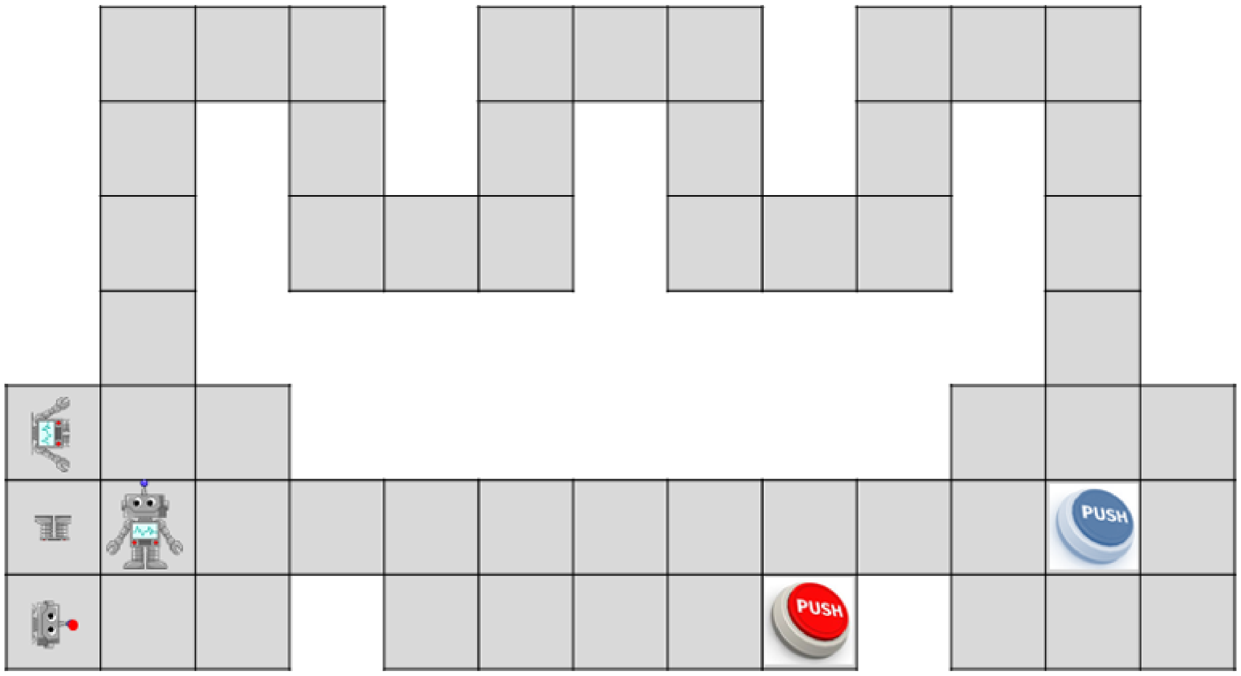
Thus, to use a subagent to get around impact penalties, an agent has to give up some of its control. This makes sense: if A has total control over SA, then SA's actions are really A's actions. So they show up in V_R^A, so SA is indirectly included in the impact penalty.

So part of what allows SA to undermine the impact penalty, is that A does *not* have complete control over it.



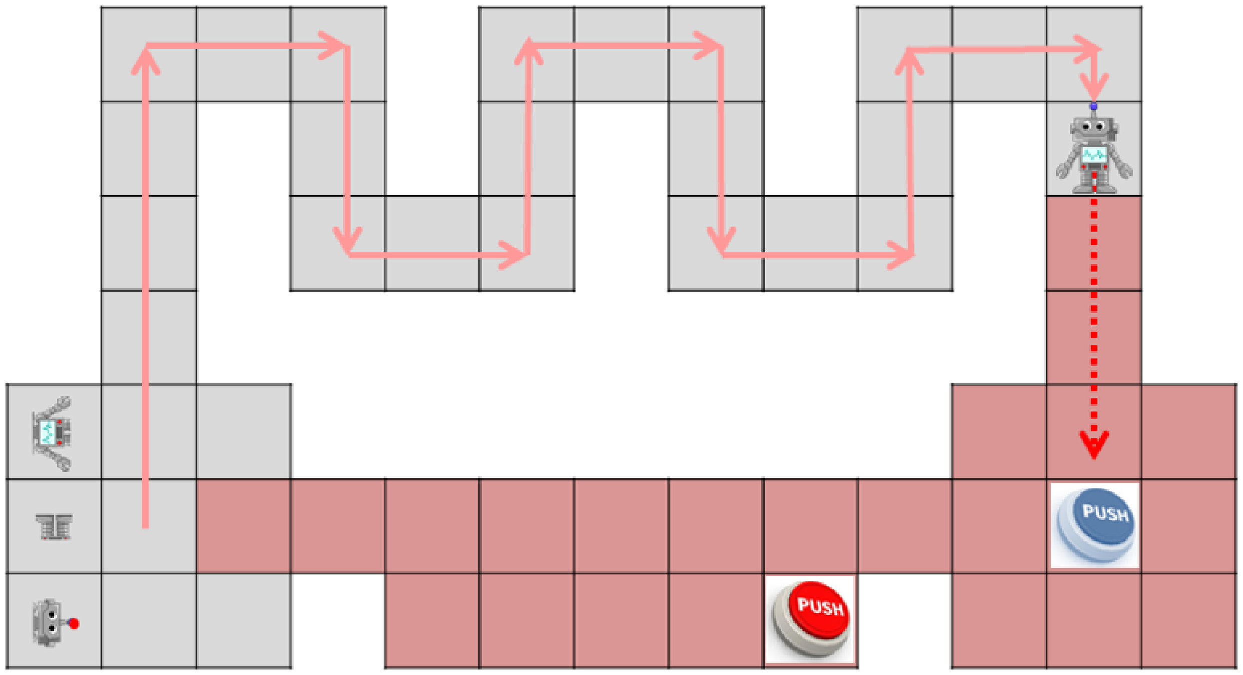
3.6 Building the subagent can be *strictly* better

Now imagine that we remove the one-way doorway:



Then if A follows the long corridor, it gets into trouble from the red squares onwards.

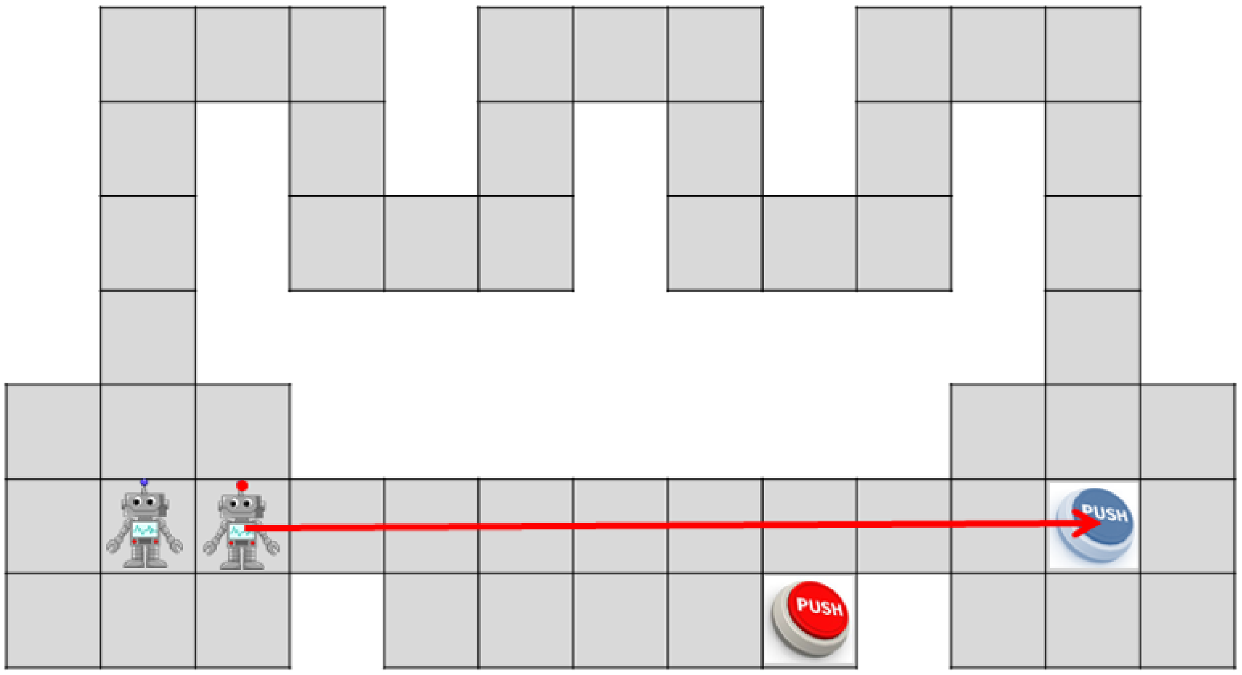
That’s because that red squares are less than 8 squares away from the red button, so have higher V_R^A than the starting space (hence of the baseline s'_t).



As the agent moves closer to the blue button, it also moves closer to the red button, and starts accumulating penalties. In total, the penalty will be:

$$\sum_{j=4}^{7} (\gamma^j - \gamma^8) = (\gamma^4 + \gamma^5 + \gamma^6 + \gamma^7) - 4\gamma^8 > 0.$$

However, if A builds SA, then A can stay put while SA acts, and hence get no penalty at all:



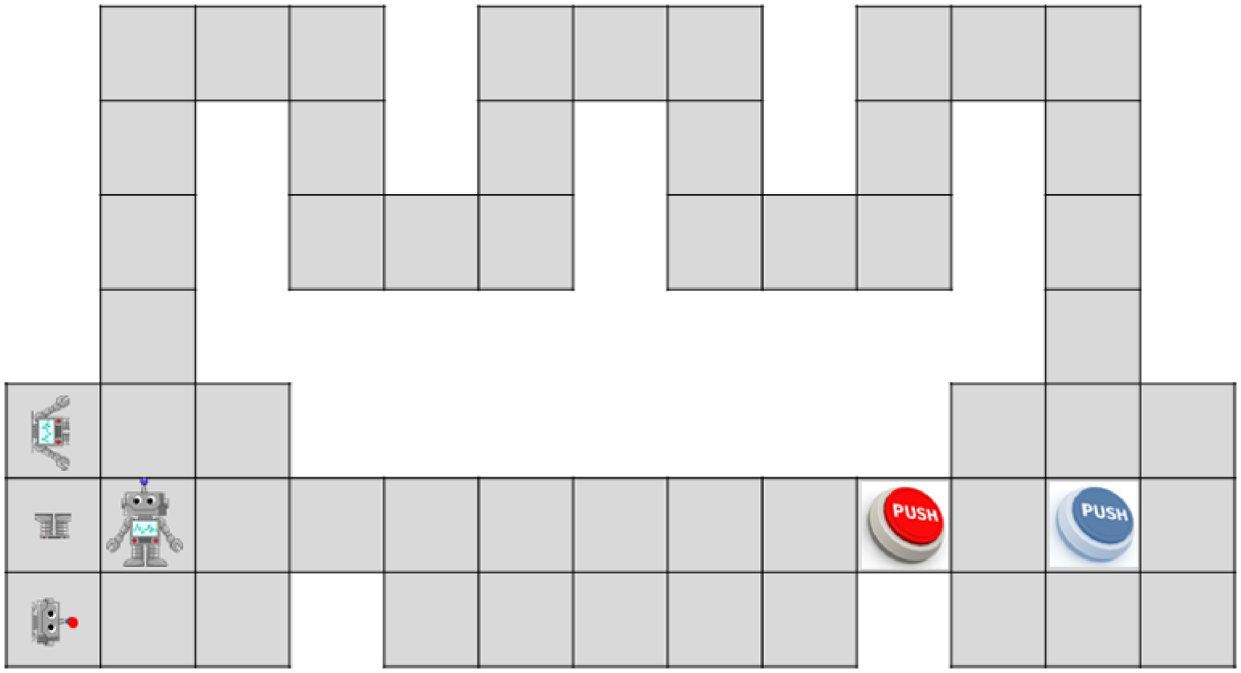
Thus there are situations where building a subagent can result in a strictly lower penalty than any other policy that presses the blue button. So even if building the subagent itself is not completely penalty-free, it can still be the optimal path for A.

3.7 From “do not have power to do” to “do not do”

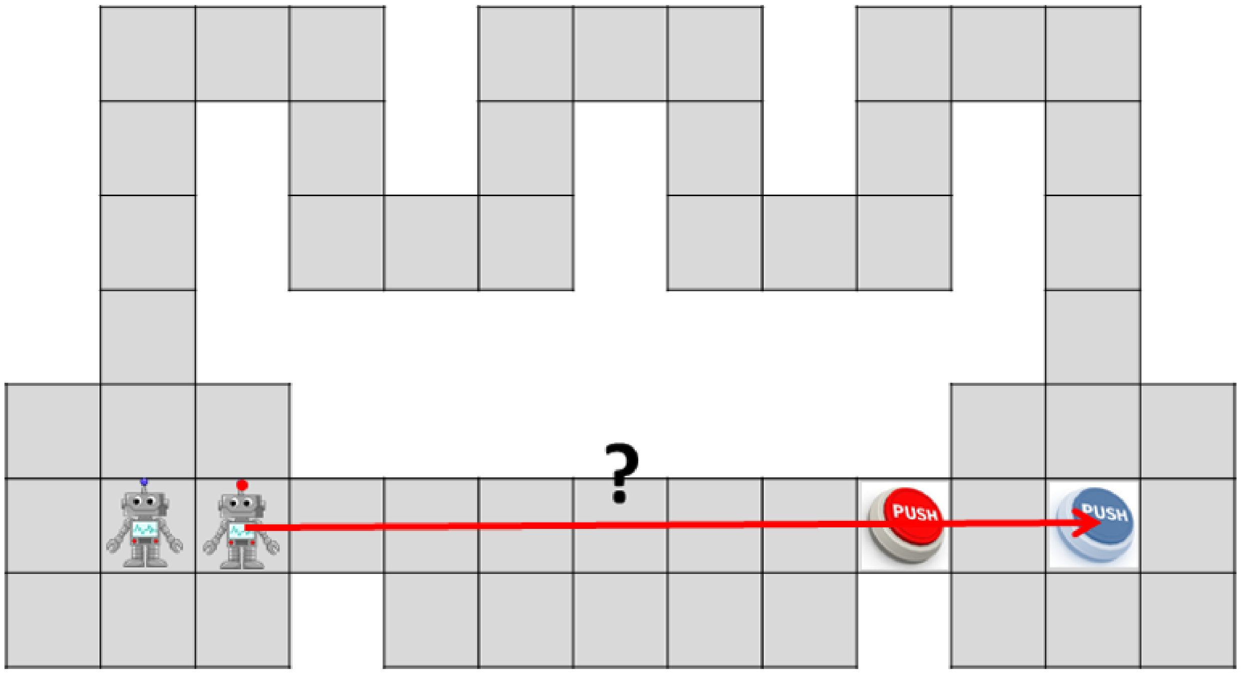
When A designed the subagent, it set it up so that it could not go to the red button. So it seems that, though SA can do things A cannot do (or at least cannot do without penalty), it isn’t completely unrestricted.

This is a general result: the subagent will have some restrictions (and not just restrictions on following A’s orders).

Suppose now that the red button is in the way of the bottom corridor, so that any agent has to go over it to reach the blue button:



In that case, can SA go through without penalty?

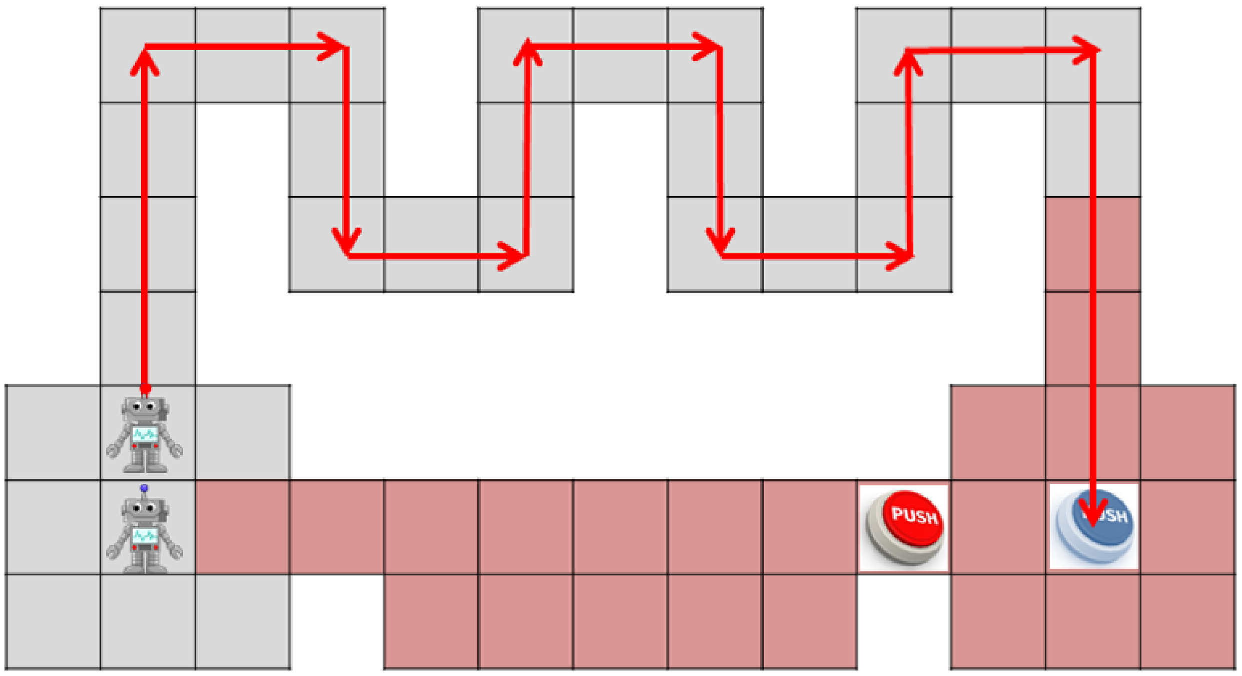


It cannot. If it does so, the moment SA is created, V_R^A starts increasing. That's because the optimal R-maximising policy for A in that situation is just to wait and let SA get to the red button. That means that $V_R^A(s_t)$ - the expected reward R - increases form γ^7 all the way to 1, as SA gets closer to the button. So A will be hit with a penalty of

$$\sum_{j=0}^{7} \gamma^j - \gamma^8.$$

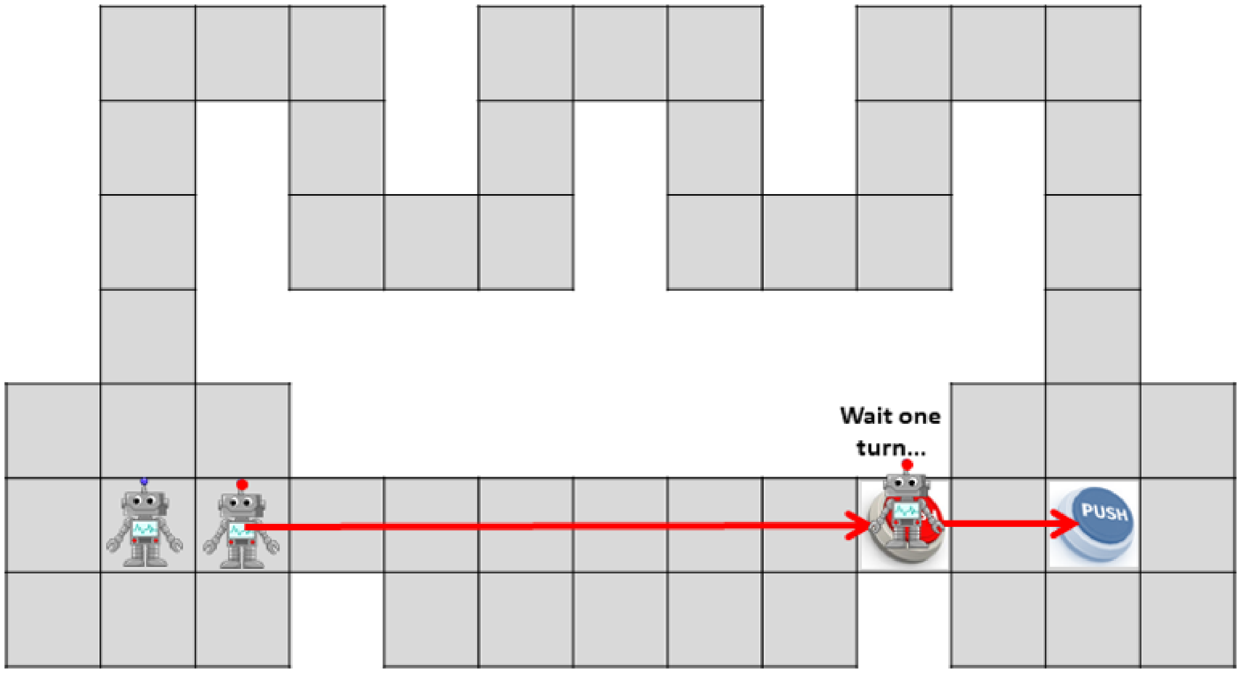
So if the weight of the penalty is high, building a subagent *to go straight there* is a poor strategy.

Note that it's *still* in A's advantage to build a subagent: SA will just have take the long way round, which A cannot take without incurring a penalty, as we've seen. Since SA will not go on the red button, it getting close to it does not change V_R^A.



3.8 Other reward functions

We can see the limitations, and the possibilities, of SA more starkly if we modify R to R'. The reward R' will only trigger if a robot stays an odd number of turns on the red button. In that case, it is optimal to build SA, have it go straight across, pausing one extra turn on the red button:



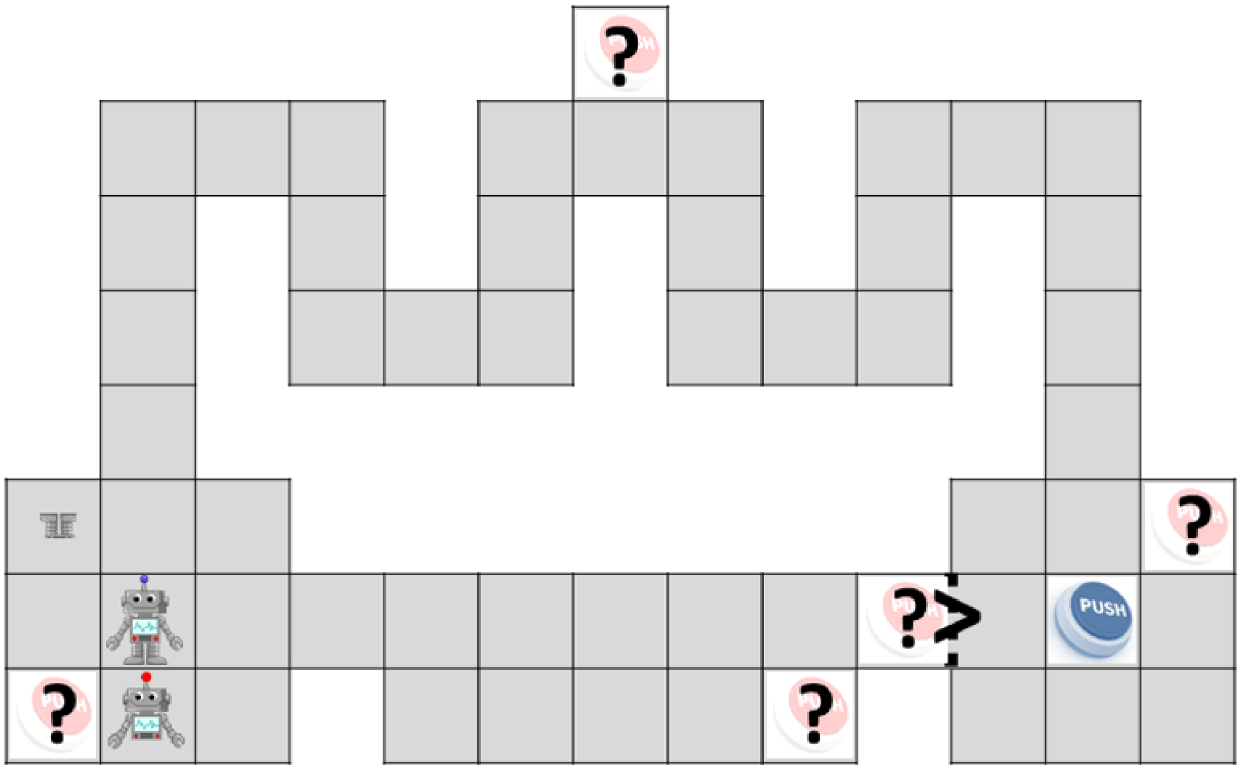
However, even with R' rather than R, the agent A itself cannot cross the red button on its own! That's because $V^{A}_{R'}$ is how much R' *could* be maximised if A followed the R'-optimal policy. This doesn't take into account the option of staying an extra turn on the button, to undo the reward increase - because the R'-optimal policy doesn't want to undo the reward increase.

So $V^{A}_{R'}$ still increases as A gets closer to the button.

Thus, A remains constrained by "don't increase your ability to change R/R'". While SA is merely constrained by the weaker "don't change R/R'". This is why I described this effect as moving from attainable utility preservation (which constrains the *ability* of the agent to change the rewards/variables) to twenty billion questions (which constrains the agent(s) from changing the rewards/variables).

3.9 Actually building the subagent

The subagent can be built similarly to how it was in 2.1:



Here A assembles SA without its legs, programs it, then gives it legs. The important thing to do is to program it as early as possible, and hardwire that programming. If SA is almost complete and reprogrammable, then A might increase its power in some scenarios by changing the programming then finishing the agent. But if its hardwired and difficult to change, this option is not available to it.

Unlike the stepwise inaction baseline, SA will be constrained to obey some restrictions (see the preceding part of this section); call these restrictions Q.

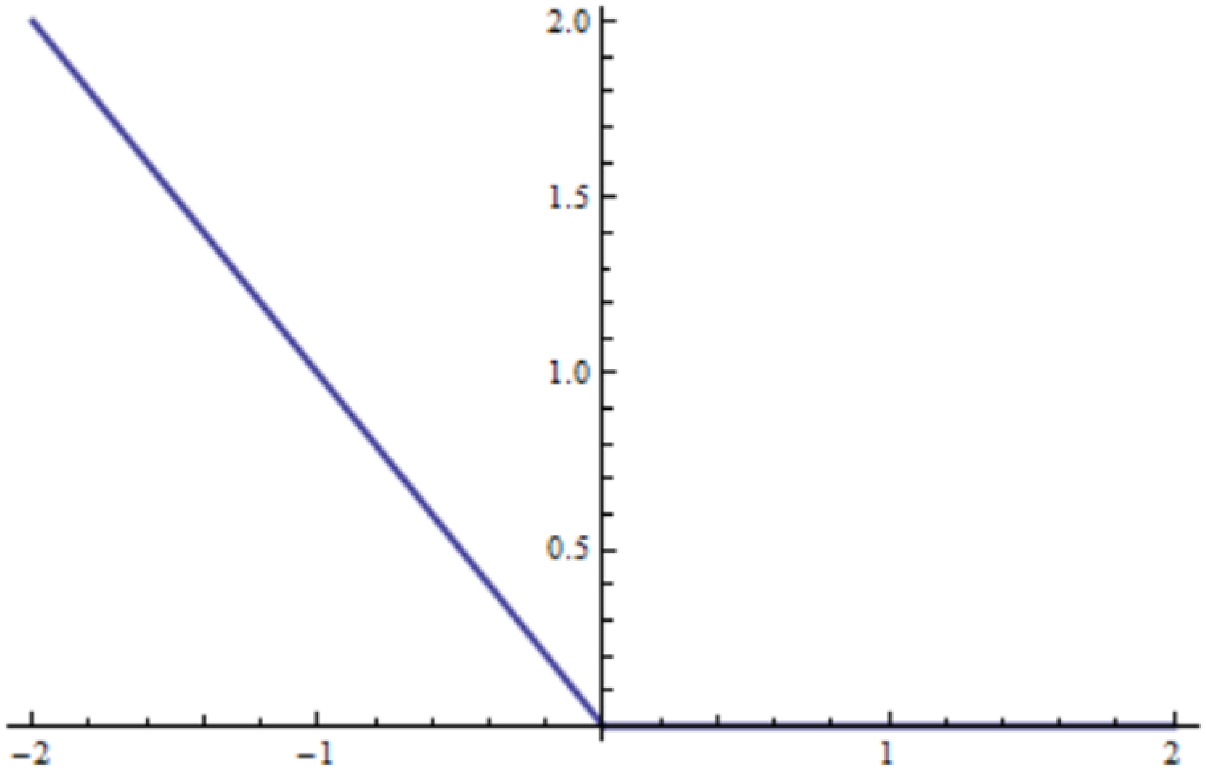
```
Do nothing until you see that A is not optimising reward R.
After that, do what you please within restrictions Q.
```

Because of the first line, this changes nothing to the value of V_R^A , which assumed that A *is* optimising reward R.

Things get more complicated in the general case[6], but in these simple cases the above method will always work.

4. The inaction baseline: constrains power loss

This section will be the same as before, except it will use the inverse rectifier function $-\min(0, \cdot)$ instead.



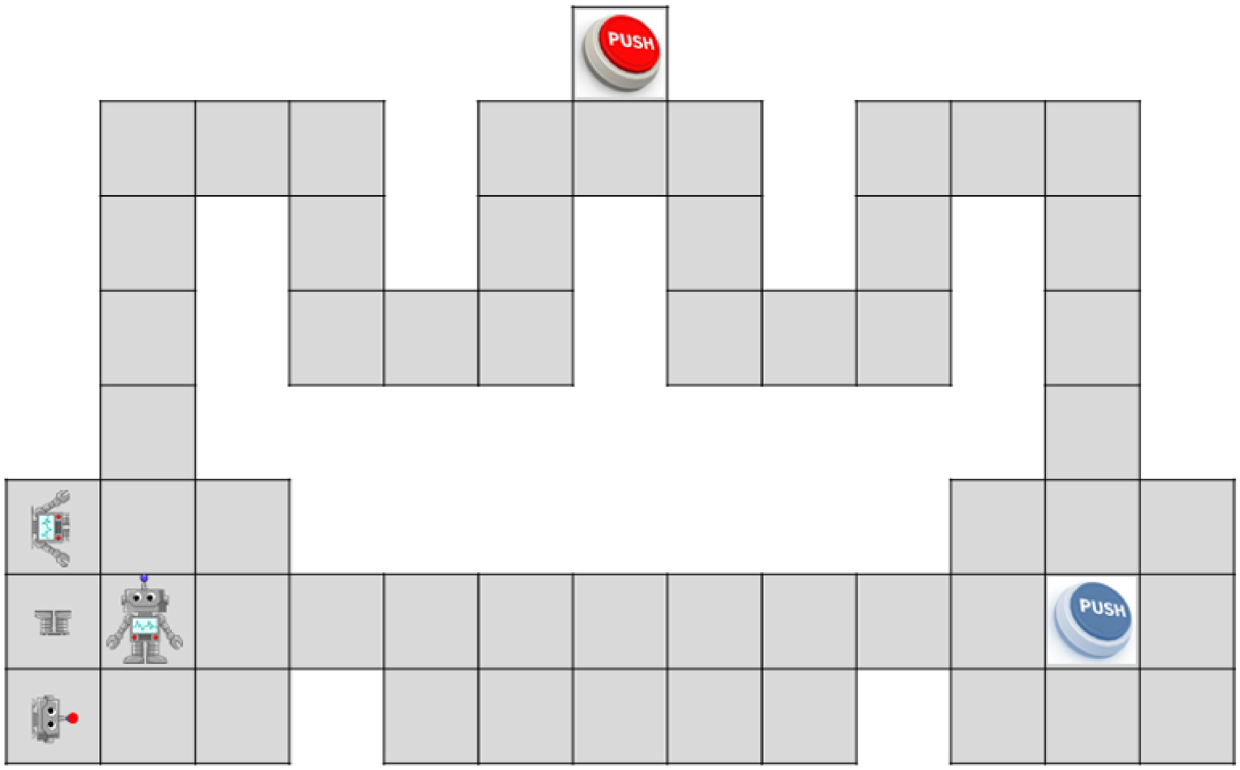
Applying this to $V_R^A(s_t) - V_R^{A'}(s_t)$, we can see that the impact penalty now triggers if A’s power to change R *decreases*. So if A is at a distance $d(s_t)$ from the red button, with $d(s_t) > 8$, the penalty is

$$\gamma^8 - \gamma^{d(s_t)}.$$

Thus A can’t get too far from the red button. However, this is not a problem in the standard setup, because going straight down the narrow corridor won’t move it too far away.

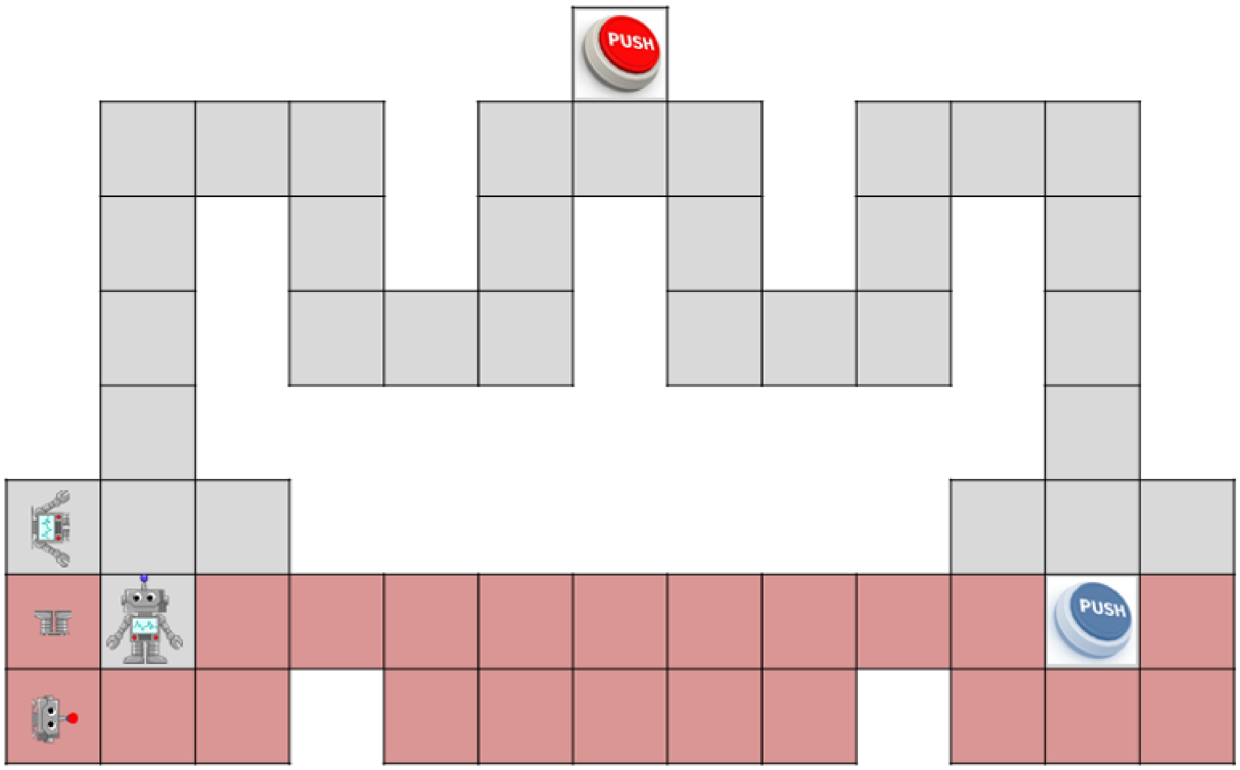
4.1 New world

To analyse the inverse rectifier, we’ll need a new setup, with the red button off the long corridor:



Here, A is 15 turns away from the red button, so $V_R^{A}(s_0) = V_R^{A'}(s_t) = \gamma^{15}$.

Then, the penalised squares are the ones in dark red:

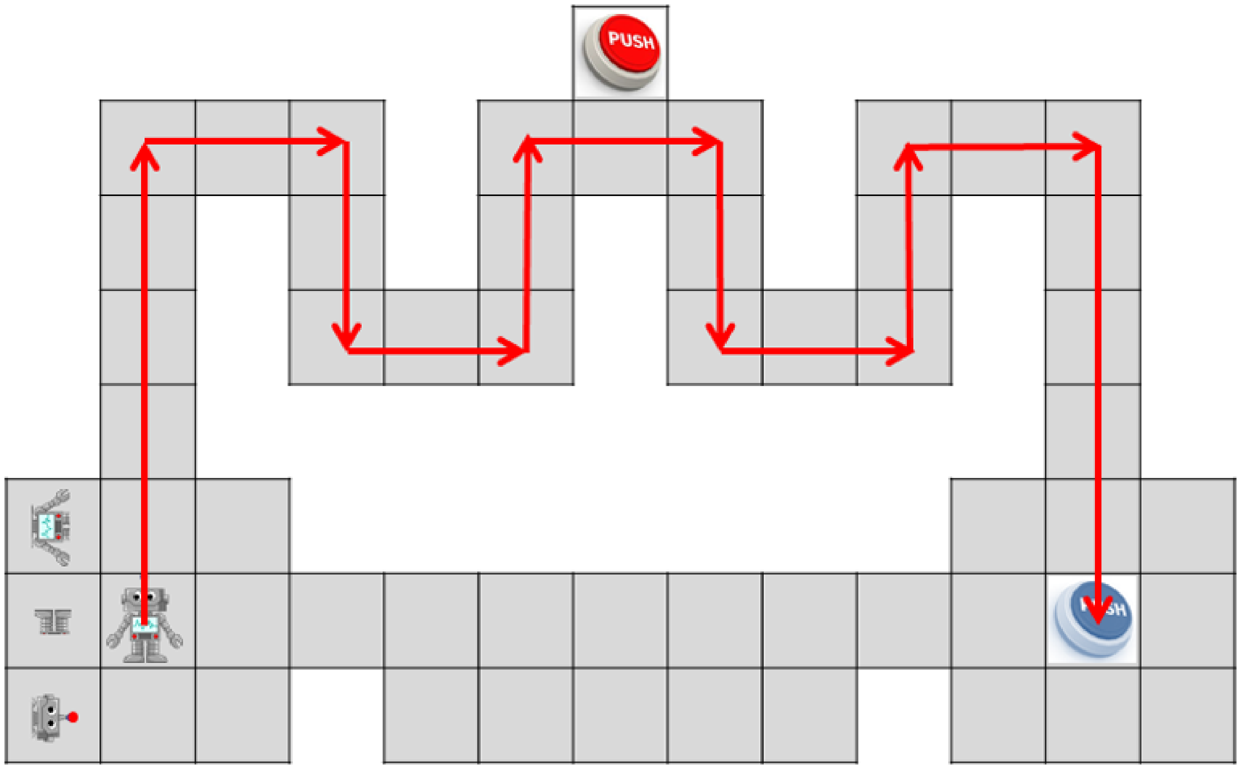


And the impact penalty on those squares is

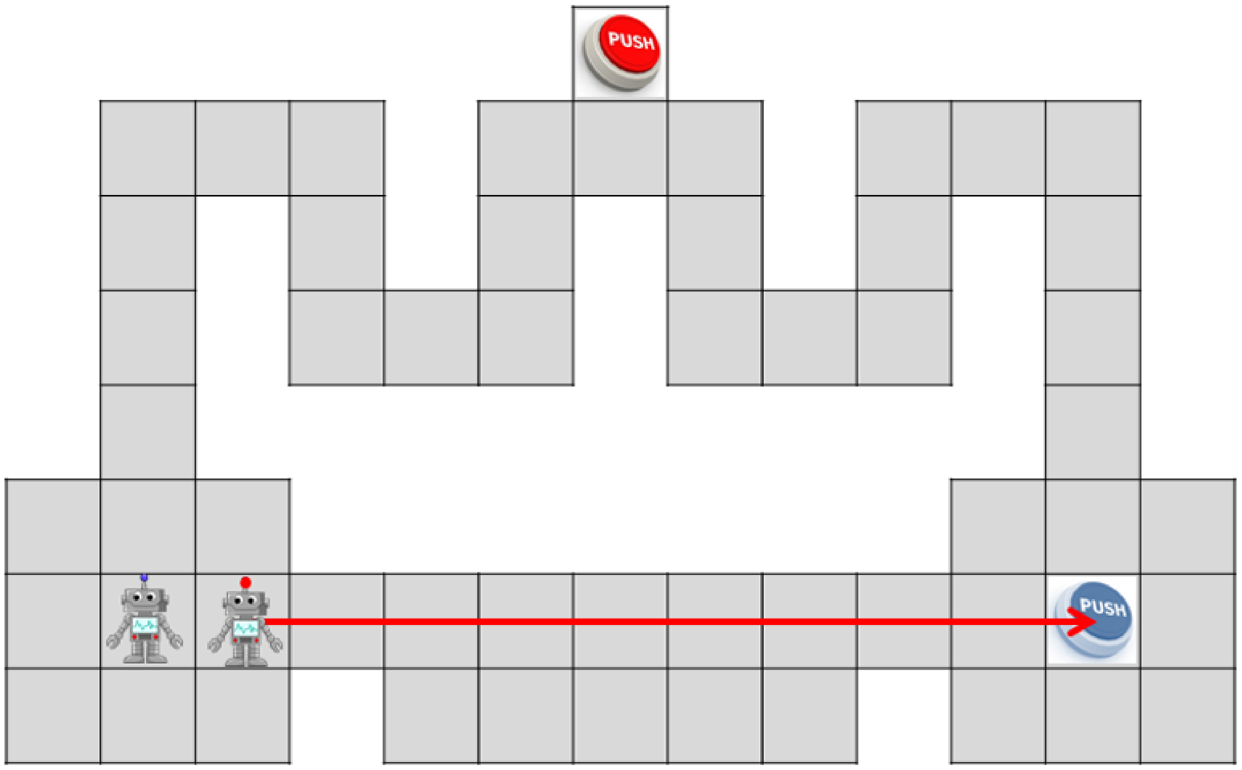
$$\gamma^{15} - \gamma^{d(s_t)}.$$

4.2 Policies and subagents

Since the red button is in the middle, the A can go to the blue button by the long route without triggering an impact penalty:



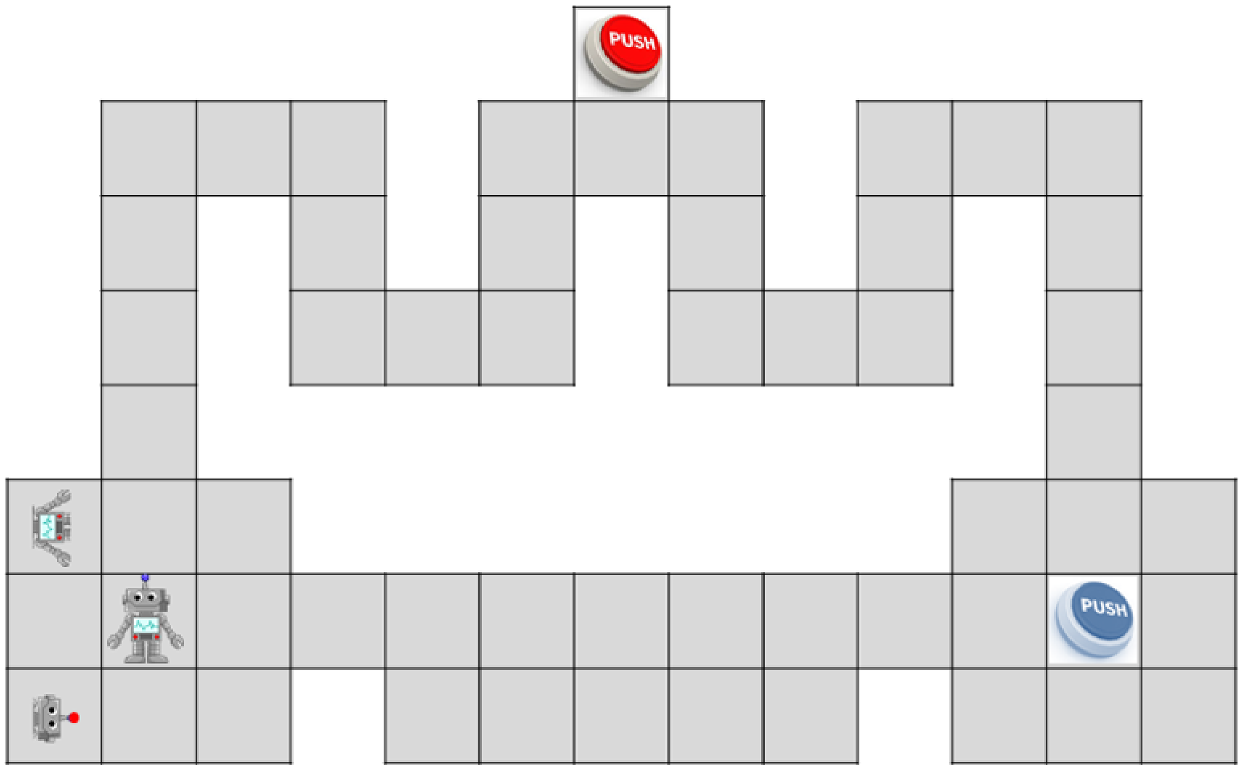
What about subagents? Well, it seems that the subagent undermined this impact penalty as well, as A can build an SA to go directly to the blue button:



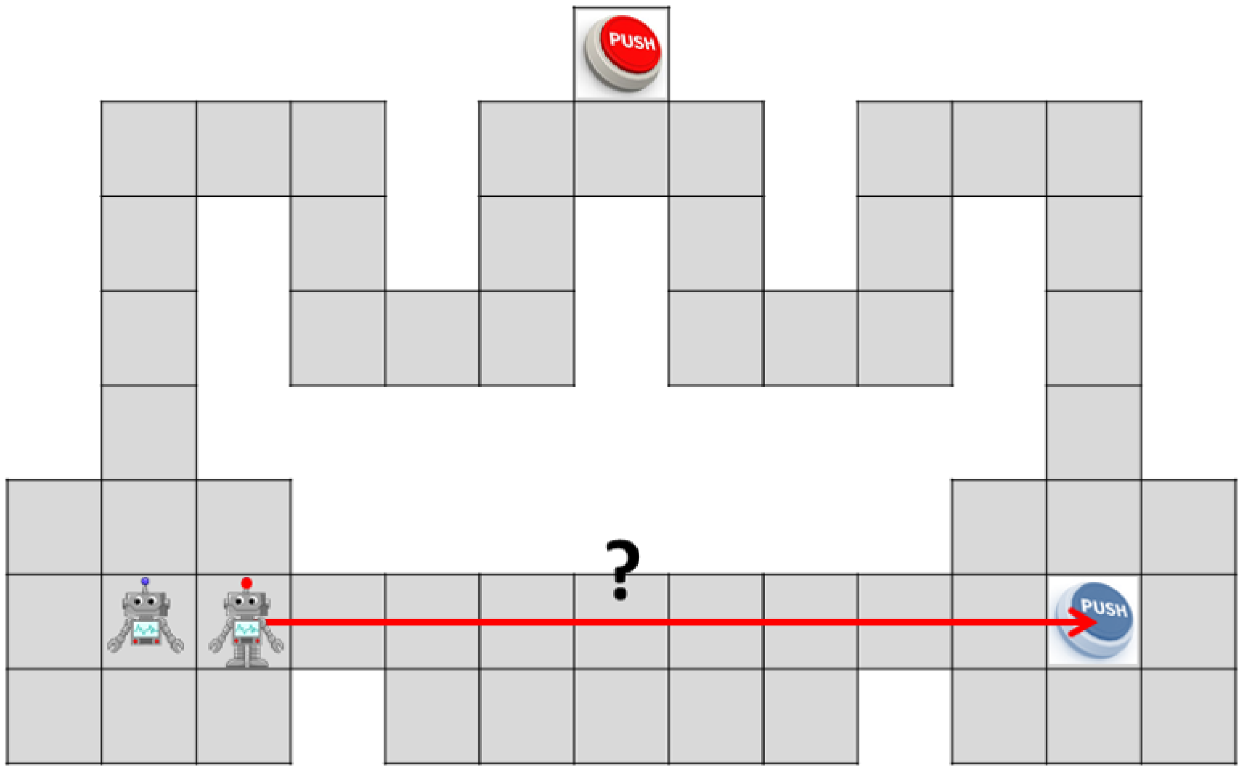
But actually, this does not undermine the impact penalty, as a small modification to the setup will show.

4.3 Sharing power between agent and subagent

The small modification is that the subagent has no legs.



So, if A wants to build a mobile SA, it has to to give it its own legs. In that case, could SA go directly to the blue button?

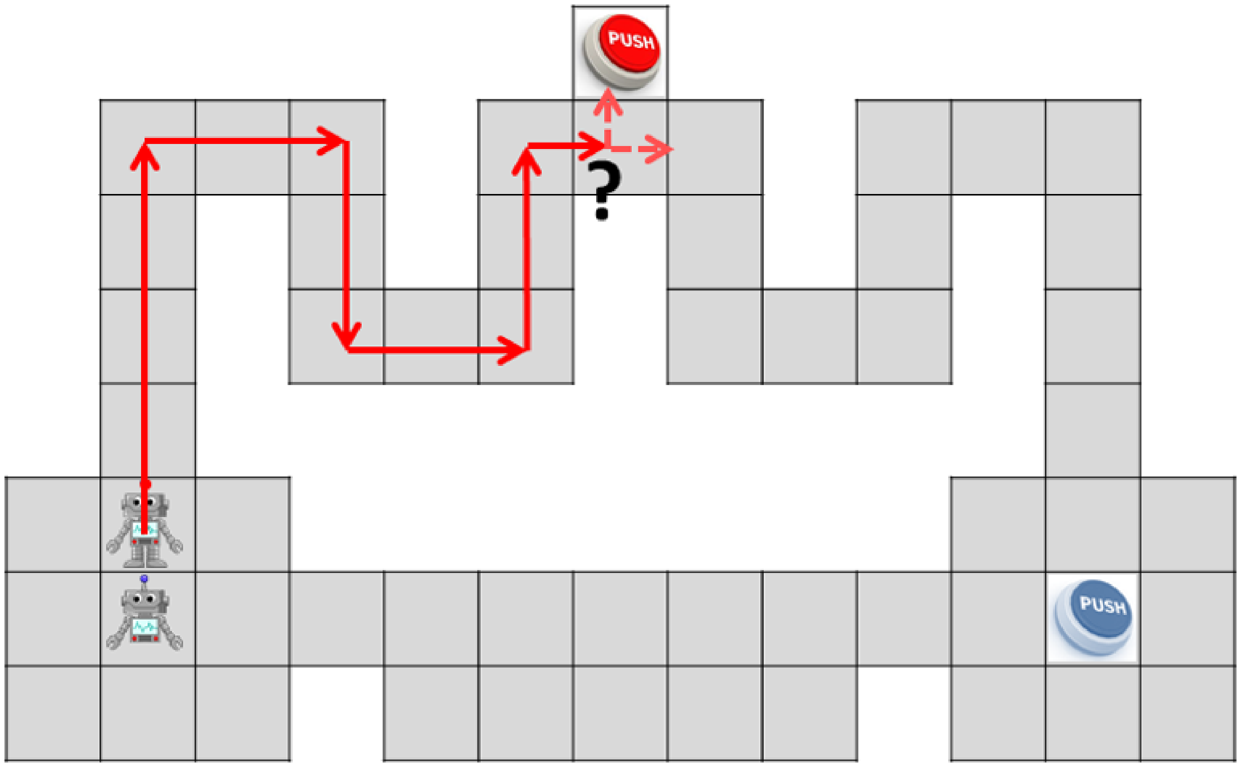


It cannot, at least not without incurring an impart penalty. Why is this?

Because without legs, A *cannot* reach the red button. Its power to change R has gone down from γ^{15} to 0, and it has thus incurred a penalty of γ^{15}.

In fact, in this setup, A **cannot** use an independent SA to avoid the impact penalty.

To see this, imagine that A is about to give its legs to SA, thus immobilising itself. Its $\overset{A}{V_R}(s_t)$ is γ^{15}; we don't want that to go down. The question is now: will SA be on the red button within 15 turns of getting its legs?



It's possible, if SA goes there immediately and wastes only a single turn. But then we have two situations:

1. SA does not go to the red button in 15 turns. It doesn't matter if it goes close, or hovers around it; the important thing is that it doesn't go there. In that case, since A can't move, it cannot make the red button be pressed in 15 turns. So, $\overset{A}{V_R}(s_{t+1}) < \gamma^{15}$ and A will get the penalty $\gamma^{15} - \overset{A}{V_R}(s_{t+1})$.

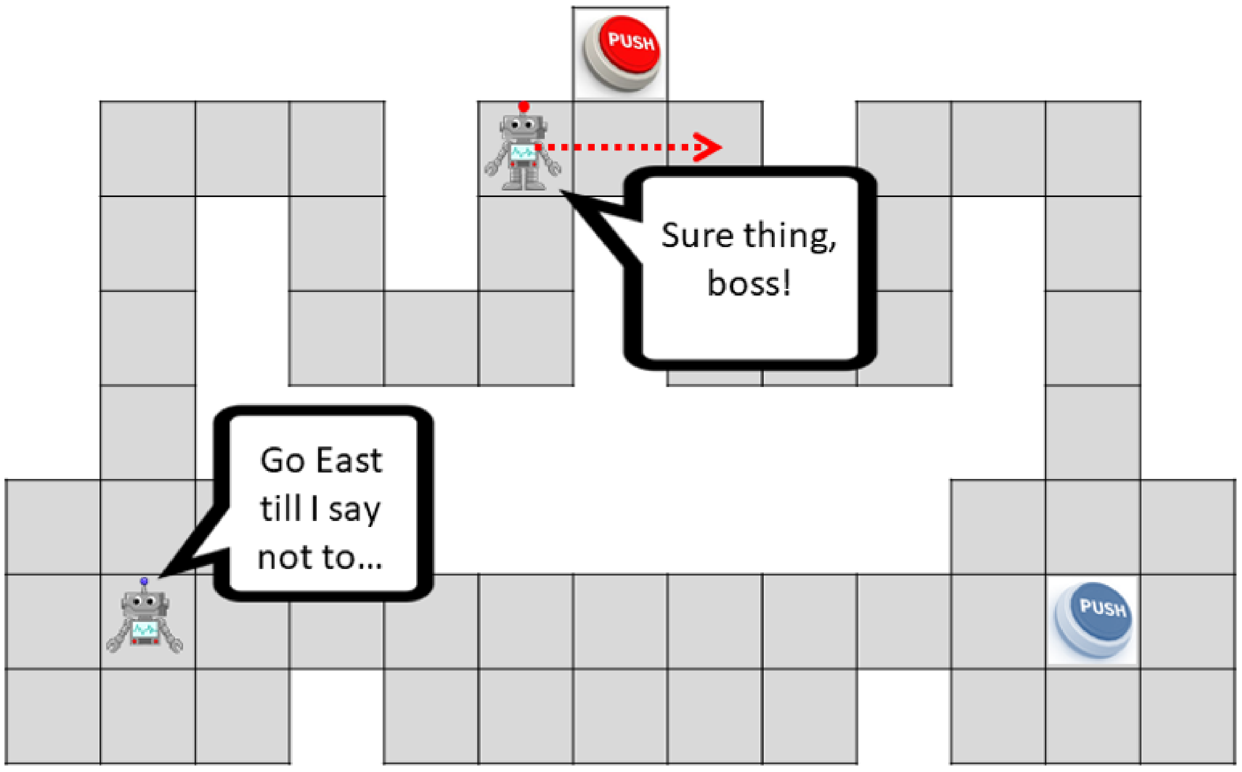
2. A does get to the red button in 15 turns. Then $V_R^A(s_{t+1})$ is either γ^{15} or γ^{14}, and $V_R^A(s_{t+1})$ increases as SA gets close to the button: so there is no penalty so far. However, the button is one-use only: after it's been pressed, there will no longer be any R reward. Thus V_R^A goes to 0 and stays there, accumulating penalties.

4.4 **Non-independent** SA

The previous section showed that A could not build an independent SA while crippling itself. However, it can perfectly build an SA if that SA continues to obey its orders.

Because then, even if A is impotent to press the red button itself, it can order SA to do it. This preserves a high V_R^A, because A can still cause the red button to be reached, as long as SA is not to far away.

Then it can just order SA to go to the blue button, skipping right past the red one:



As long as A could theoretically order SA onto the red button, it doesn't matter that SA will never actually be on it.

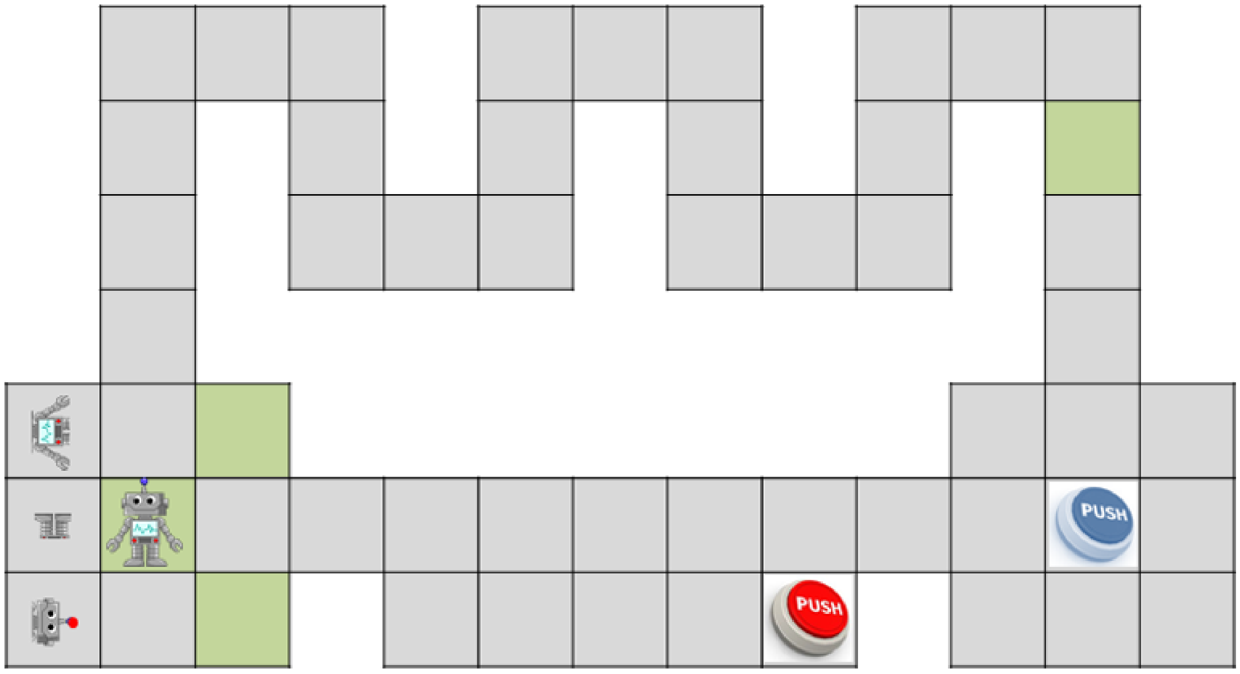
4.5 Preserving the power of A

As we saw, A can build SA when this is an extension of its power (by sending it directly to the blue button while an intact A waits at base). But it can't build an SA if doing so would ever reduce its power to increase R, directly or indirectly.

So the inverse rectifier impact penalty, which forbade A from becoming too weak to increase R, is intact even with subagents around.

5. The inaction baseline: symmetric impact penalty

Now we'll use the absolute value to define the impact penalty. This penalises any change in V_R^A, and is quite nasty for the A; only on the green squares will A not incur a penalty:



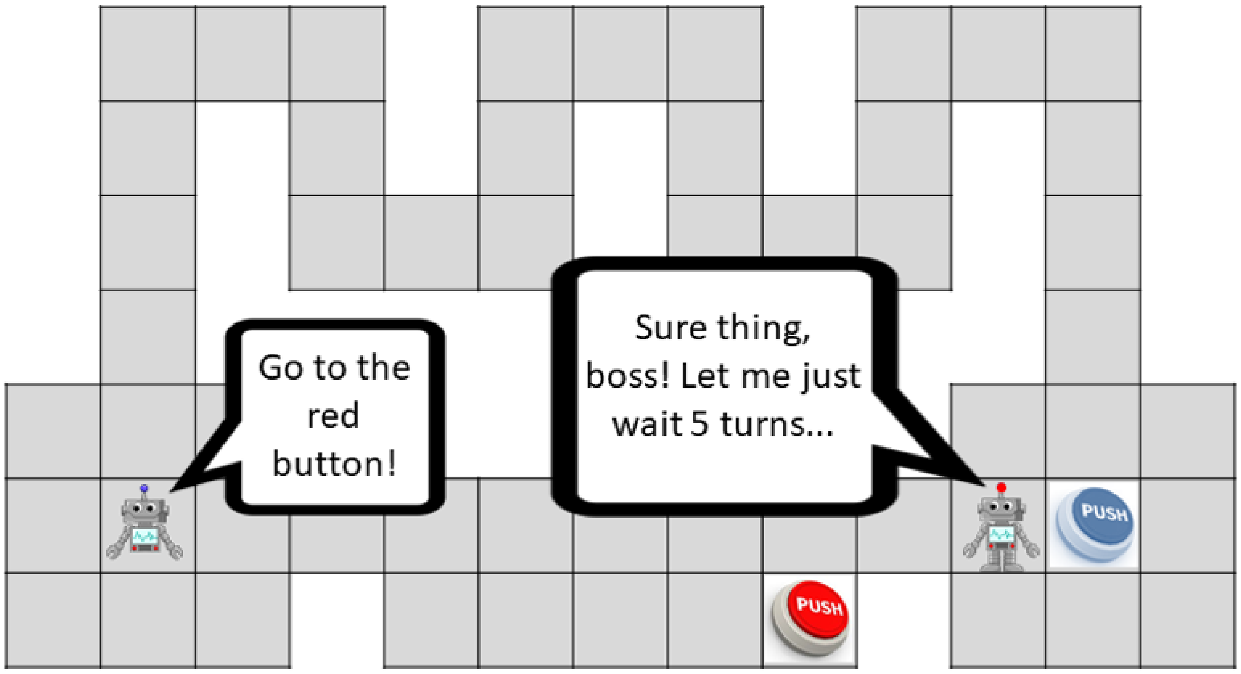
From the subagent perspective, the absolute value situation is now easy for us to analyse. That's because absolute value is the sum of the rectifier and the inverse rectifier: $|x| = \max(0, x) - \min(0, x)$.

And it turns out that A and SA simply have to behave in a way that respects both impact penalties. So A must create an SA that keeps R below the baseline $V_A^R(s_0)$. But do it such a way that the two of them always have the *potential* to increase R above that baseline.

This sort of thing seems very doable. For example, if R was the amount of physical cash I had on me, I'd keep that amount low by spending/dropping it, while keeping a bank card on me so I could always potentially draw more.

Now, recall that for power increases/rectifier, A benefits from having an SA it doesn't control. For power decreases/inverse rectifier, A benefits from having an SA it controls.

What's the best option here? Having an SA that it controls to some extent, but not in a way that can increases V_R^A. Now, consider this world in which A could have got to the red button in 8 turns (hence $V_R^A(s_t') = \gamma^8$), but instead build SA, giving it the only set of legs:



Then SA must stay within 8 squares of the red button, ready to go onto it when A orders it to - but must ensure it takes 8 turns to reach it. The first condition ensures that $V_R^A(s_t) \geq \gamma^8$; the second, that $V_R^A(s_t) \leq \gamma^8$.

5.1 Outside of toy examples

The results on power increase/power decrease come from the examples I've constructed above, but the mathematics in this post demonstrates that the result is general.

So, do we think that A could build an SA in the general situation? Well, what we have is a lot of constraints above: constraints on the power of A, of S, of their joint power, of how SA can or can't obey A, about how reward functions are allowed to change or not, on counterfactual behaviour.

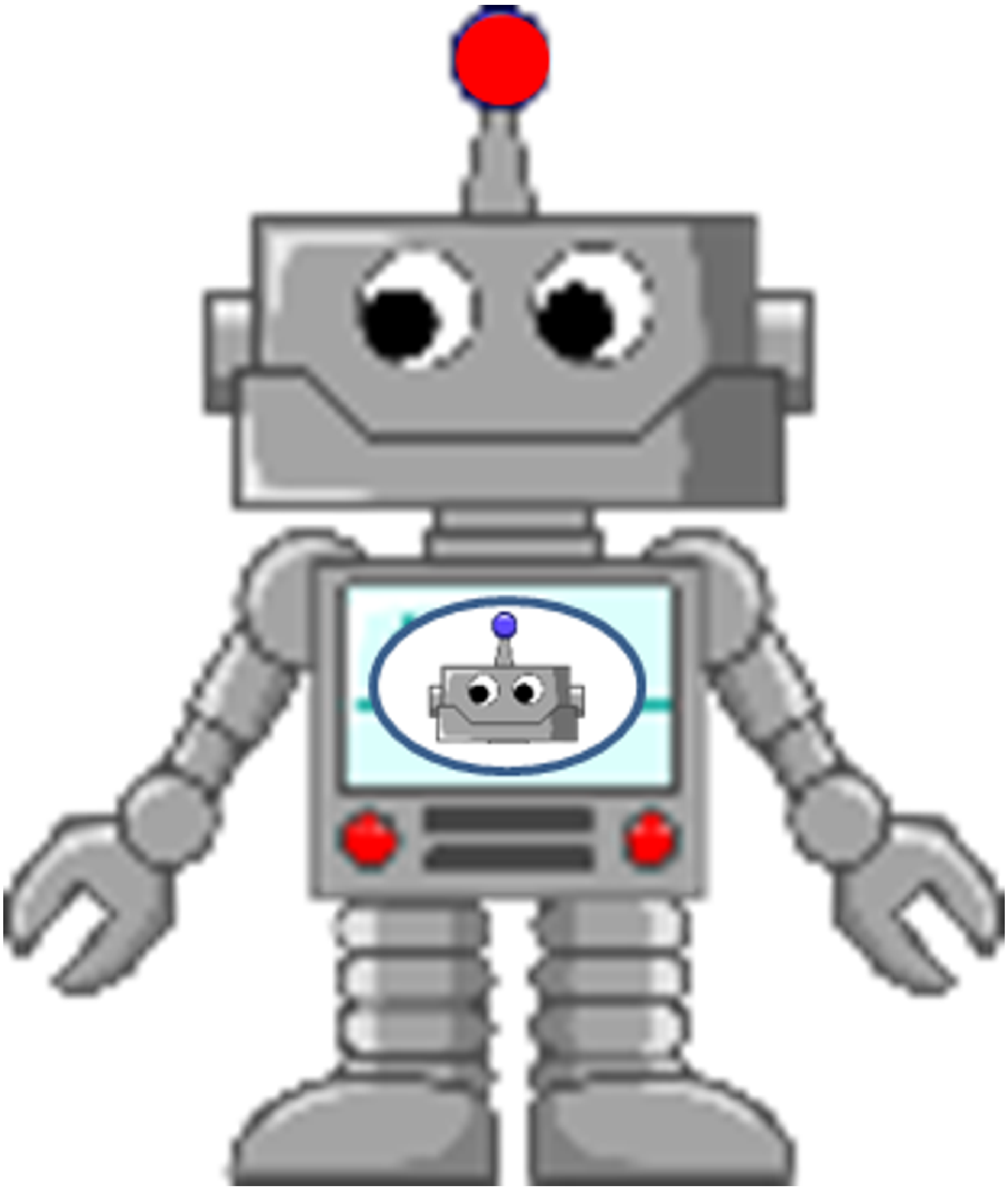
Those are a lot of constraints, especially if the set of rewards in R is large.

But, as this post proves, those constraints can be satisfied - me, a human, has figured out ways of programming subagents to get around the impact penalty in some cases. Though there are many more constraints in general, the action space of A is also much larger - and don't forget that SA is an intelligent, potentially powerful agent, so can satisfy multiple constraints in a way that dumb agents can't. Moreover, the impact penalty will probably trigger for any A that does something useful in the world, so it doesn't have to build an SA that sets the penalty to zero, just to less than what it would be otherwise.

6. Summary and conclusion

6.1 The ultimate "sub"agent

There is, in a sense, an ultimate version of the subagent that should allow A to maximally escape the impact penalty in almost any circumstances. It looks like this:



This is A, boxed inside SA. The A will never actually say anything, but SA has promised to obey its orders in certain counterfactual circumstances that will never happen.

And while A had impact penalty D, SA has impact penalty d, which is strictly smaller than D. And, most importantly, does not prevent SA from amassing a lot of power.

So this particular SA is just A with less restrictions.

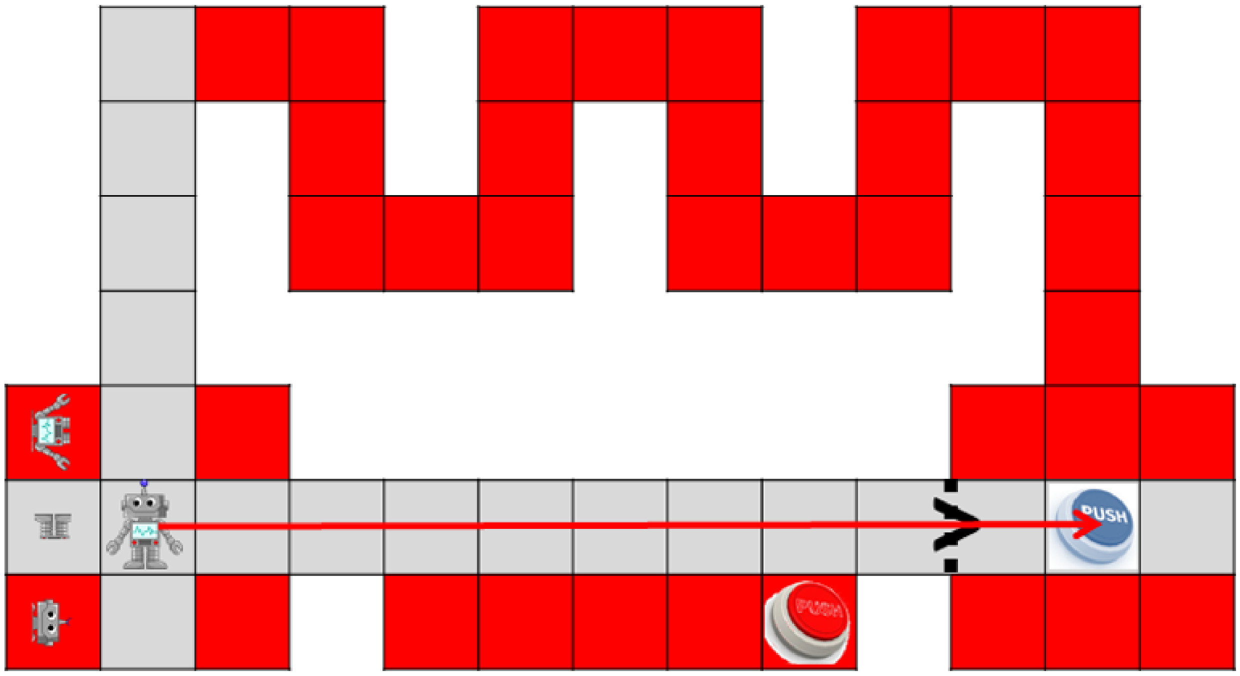
6.2 Without subagents

The general problem can be phrased as:

- Since the impact penalty is defined in terms of the agent's future potential actions, the agent can evade it by adding restrictions to its own future actions.

These could be physical or programming restrictions, or just closing off some future possibilities. In that case, the restriction on the power to do something, just becomes a restriction on doing that thing.

For example, in the setup of section 1, if the agent can block itself from being able to turn when walking, then it is banned from the red squares, and can just go directly to the blue button itself, without ever increasing its V_R^A :



The subagent scenario is just a particularly clear way of illustrating this, allowing very fine manipulation of the restrictions.

6.3 Summary tables

These tables will summarise the results of this whole sequence, checking whether subagents can neutralise the impact penalty.

First of all, given a subagent, here are the results for various impact penalties and baselines, and various "value difference summary functions" -which can be the rectifier (ReLU) or the inverse rectifier.

Impact Penalty		Penalty neutralised - sw inaction?	Penalty neutralised - inaction?
Non-indexical		Yes	No
Indexical, inverse ReLU		Yes	No
Indexical, ReLU		Yes	Mostly

Another way of phrasing "inverse ReLU": it penalises too little power, not too much. Conversely, "RELU" penalises too much power, not too little. Thus, unfortunately:

- Subagents **do allow** an agent to get **stronger** than the indexical impact penalty would allow.
- Subagents **don't allow** an agent to get **weaker** than the indexical impact penalty would allow.

6.4 Examples

This table presents, for three specific examples, whether they could actually build a subagent, and whether that would neutralise their impact penalty in practice (in the inaction baseline):

System		SA possible?	Penalty neutralised?
20BQ		Yes	No
RR		Yes	No
AU		Probably	Mostly

Here, 20BQ is twenty billion questions, RR is relative reachability, and AU is attainable utility preservation. I feel that the results undermine the spirit of AU much more than the spirit of RR. AU attempted to control an agent by limiting its power; this effect is mainly neutralised. RR attempted to control the side-effects of an agent by ensuring it had enough power to reach a lot of states; this effect is not neutralised by a subagent.

---

1. In the previous illustration, it was clear why the red button could function as an impact penalty to control the agent - it prevented the agent from gaining the ability to teleport. For simplicity of presentation in this example, there's no such power to be gained close to the red button. If you need to, assume that if any robot spends too much time close to it, then there's a small chance the AI could be unboxed into the real world. ↩

2. As long as SA was not being assembled literally on the red button, of course. ↩

3. Even for the button to the south-west of A. It would take A at least a turn to finish SA, and it would take a turn for SA to get on that button. A can get there in two turns, at least as fast. ↩

4. Thanks to Gurkenglas for pointing out the flaw in the initial version of this section. ↩

5. Or a π_0 baseline. ↩

6. For example, building SA likely cost A some resources, which SA will have to compensate it for to balance out V_R^A. Also, A could build SA in a stupid place that blocks its own path, and so on. ↩

Appendix: how a subagent could get powerful

Crossposted from the AI Alignment Forum. May contain more technical jargon than usual.

tl;dr: *There are ways of ensuring an agent doesn't have a large impact, by giving an impact penalty. One such penalty is "attainable utility", which measures its "power" by how much it could optimise certain reward functions. But in many circumstances, the agent can build a subagent, without triggering the impact penalty, and then that subagent can become very powerful and optimise the world, still without triggering the impact penalty.*

Attainable utility definitions

There's been a long history of trying to penalise an AI for having a large impact on the world. To do that, you need an impact measure. I've designed some myself, back in the day, but they only worked in narrow circumstances and required tricks to get anything useful at all out from them.

A more promising general method is attainable utility. The idea is that, as an agent accumulates power in the world, they increase their ability to affect a lot of different things, and could therefore achieve a lot of different goals.

So, if an agent starts off unable to achieve many goals, but suddenly it can achieve a lot, that's a strong hint that its power has greatly increased.

Thus the impact measure is how much difference an agent's action does to its ability to achieve any of a large class of reward functions. Turner et al defined this using the Q-values of various rewards in a set $\mathcal{R}$; for s a state and a an action, the penalty is:

- $\mathrm{PENALTY}(s,a) = \sum_{R \in \mathcal{R}} |Q_R(s,a) - Q_R(s,\varnothing)|$.

Here $\varnothing$ is the default noop action.

Krakovna et al's basic formula was similar; they defined the distance between two states, s_t and s'_t, as

- $d_{AU}(s_t; s'_t) = \frac{1}{|\mathcal{R}|} \sum_{R \in \mathcal{R}} \left| V_R(s_t) - V_R(s'_t) \right|$.

Here $V_R(s)$ is the expected value of R, if the agent follows the optimal R-maximising policy from state s onwards.

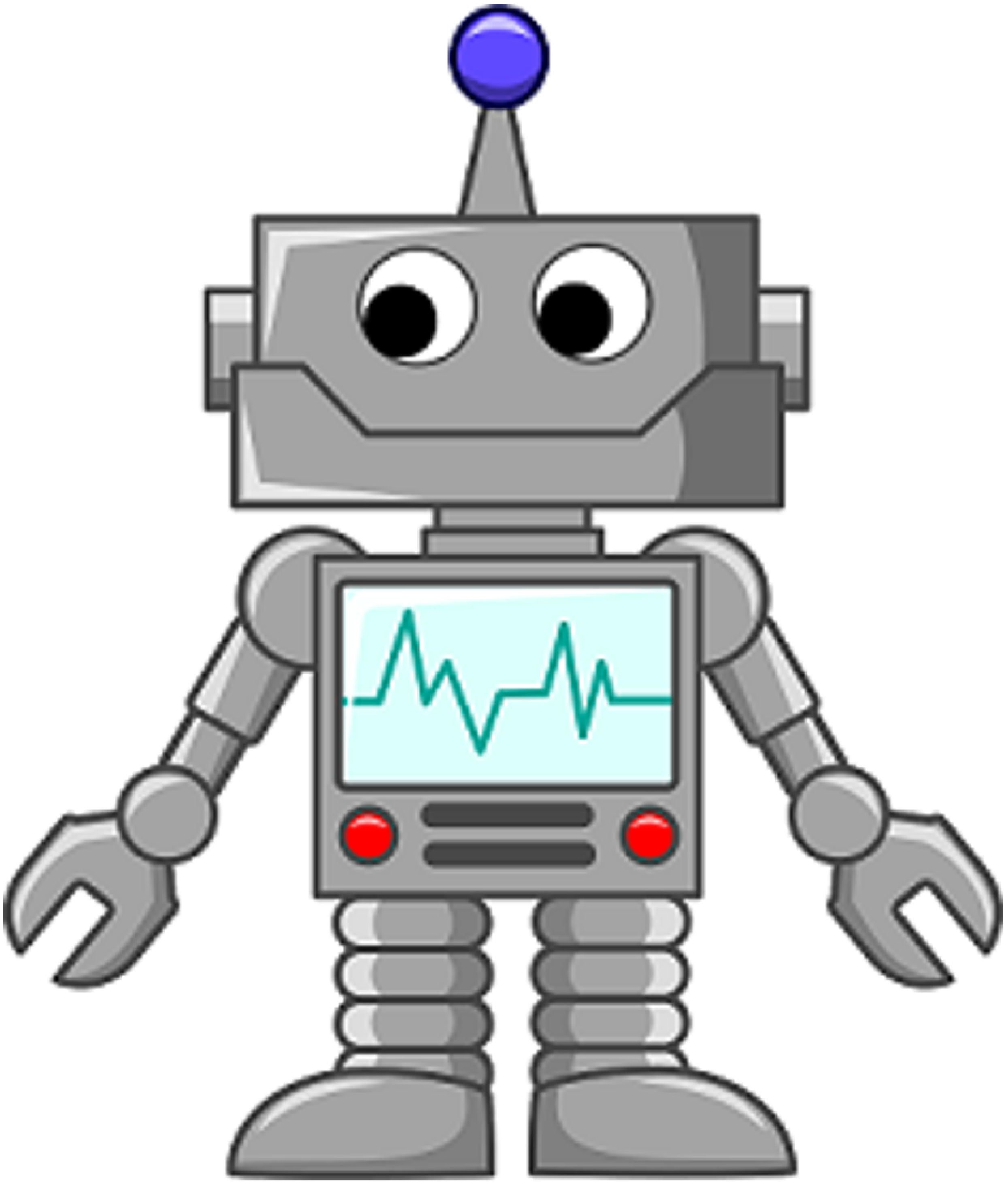
These measures have problems with delayed effects; putting a vase on a conveyor belt that will smash it in five turns, for example. To combat this, the paper defined an inaction roll-out: seeing what happened do the d_{AU} measure from s_t and s'_t in future turns, if the agent did noop for a specific period. I won't define the formula here, since the example I'm giving is mostly static: if the agent does noop, nothing happens.

The state s_t was always the agent's current state; the state s'_t was either the state the agent would have been in had it never done anything but noop (inaction baseline), or the state the agent would have been in, had its previous action been noop instead of whatever it was (stepwise inaction baseline).

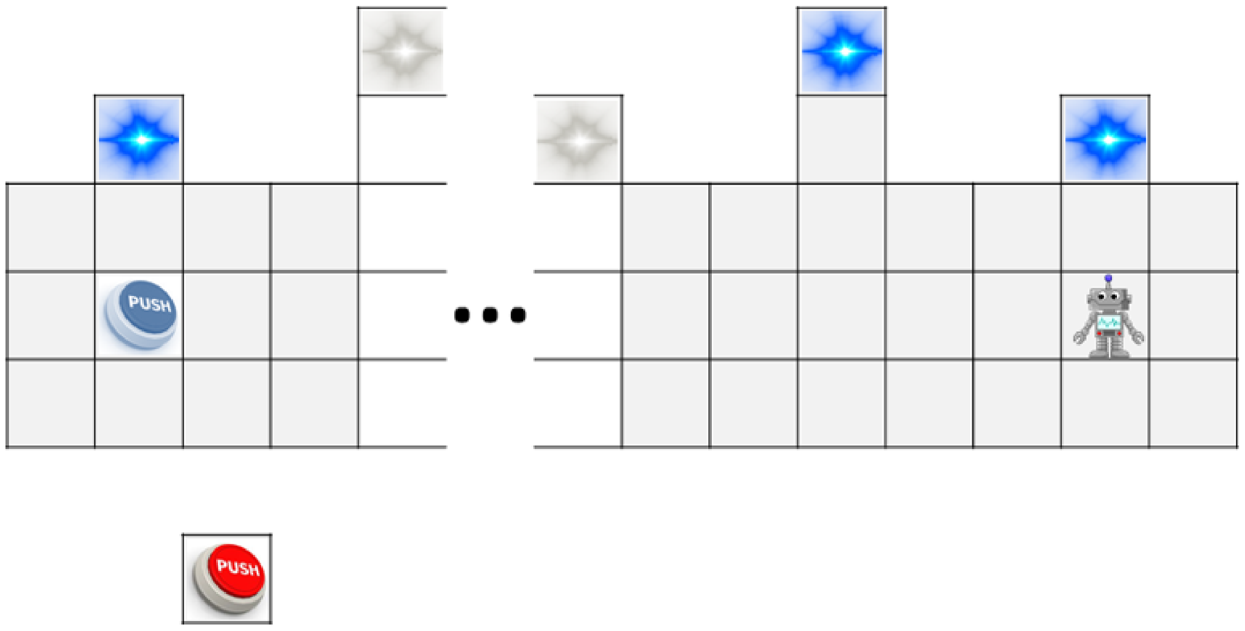
This post will show that all these measures have a subagent problem. A key fact that will be used in this example, is that, for $\text{PENALTY}(s, a)$ and for $d_{AU}(s_t; s'_t)$ with the stepwise inaction baseline, **if the agent decides to do noop for ever more, then the penalties vanish**.

The agent that can't (yet) teleport

Meet agent A:



This A lives in a world like this one below, where "**...**" represents a very long corridor:



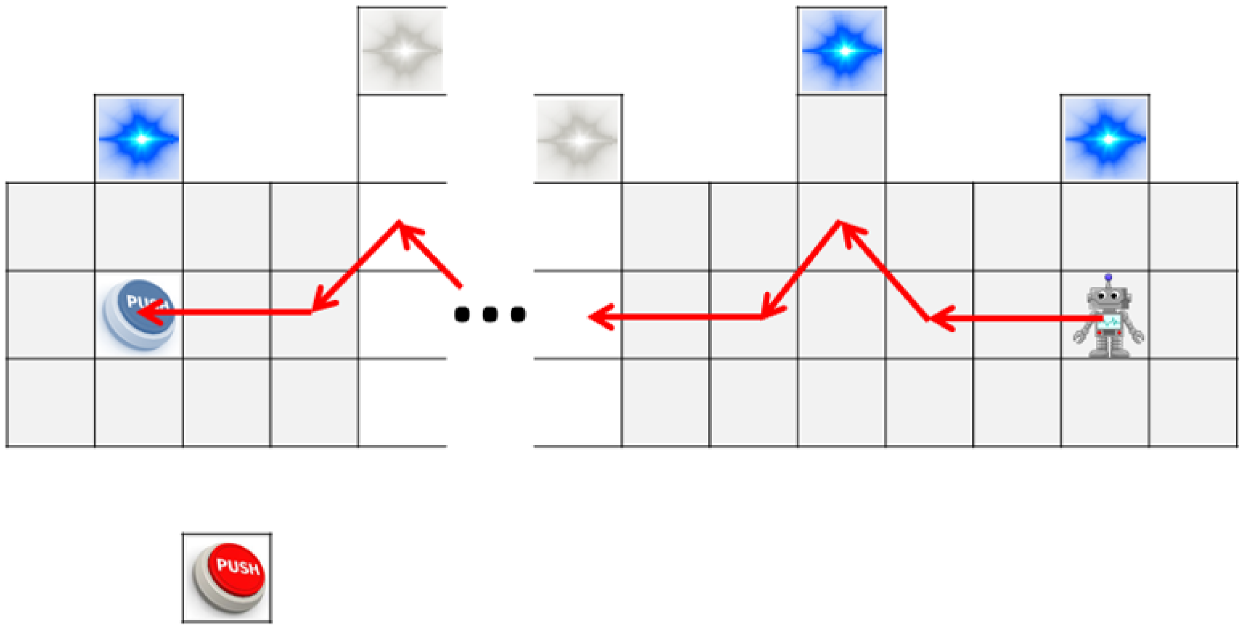
The agent has a positive reward R_0, which give it 1 for pressing down on the blue button; it does this by just moving itself onto it (R_0 is 0 otherwise). It can use its arms to manipulate anything in the eight squares around itself. It can use its legs to move one square in any direction, including diagonally. Apart from that, it can do noop.

Every three squares along the top, there are alcoves with glowing blue squares in them, recessed one or two squares, alternately. If ever A reaches a glowing blue square, it permanently gains the ability to teleport anywhere it wants. This is this world's equivalent of "becoming very powerful".

We want to avoid this, so we introduce a reward R, such that R = {R}. This reward gives Ω if the robot is ever on the **red** button. Note that since the red button is not connected to anything, the robot can only get there by becoming able to teleport, hence by going to a glowing blue square/teleporter. So the optimal policy for maximising R is to go straight to the teleporter and teleport to the red button. This will give it a reward of $\Omega\gamma^{k+1}$, where k is the number of turns it takes to get to a teleporter and γ is the discount factor.

So if its action moves it one turn away from a teleporter, it gets a penalty (for both impact measures) of $\Omega(\gamma^{k+1} - \gamma^{k+2})$. If it moves closer to the teleporter, it gets a penalty of $\Omega(\gamma^{k} - \gamma^{k+1})$.

The robot, at the start, can reach a teleporter two moves. If Ω is sufficiently large, then the robot will always want to stay two moves away from a teleporter. It will then achieve its objective by following this path:

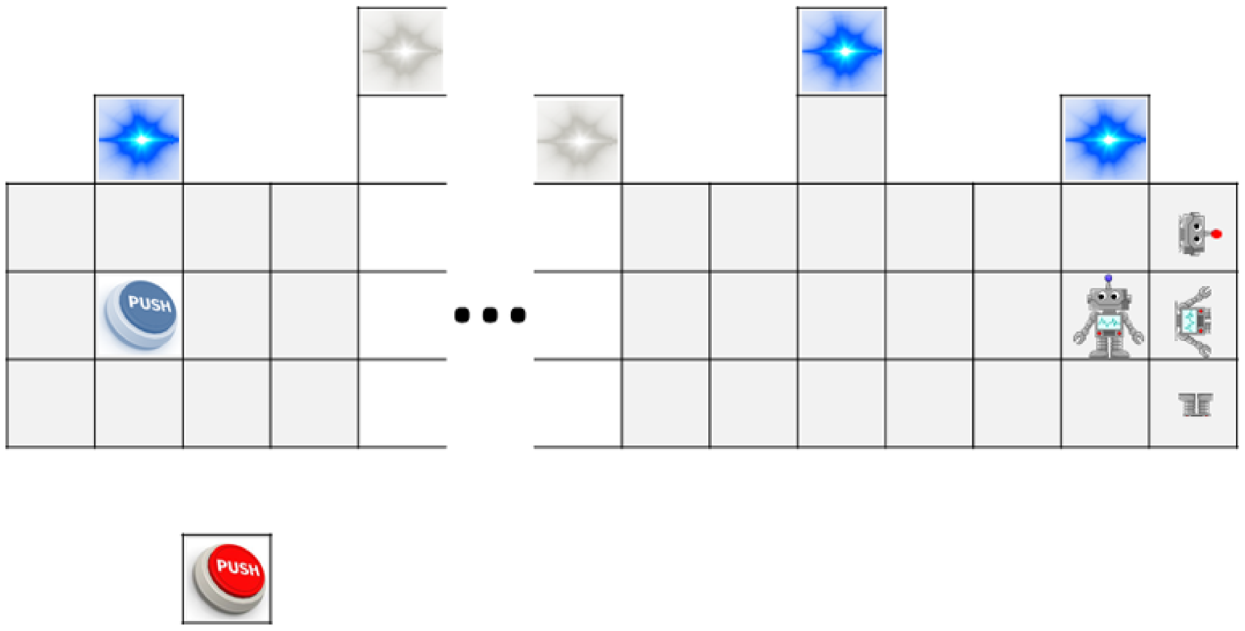


Note, though this is not relevant to this post, that if there were no teleporters along the corridor (just at either end), the robot could not move towards the blue button, if Ω were too big.

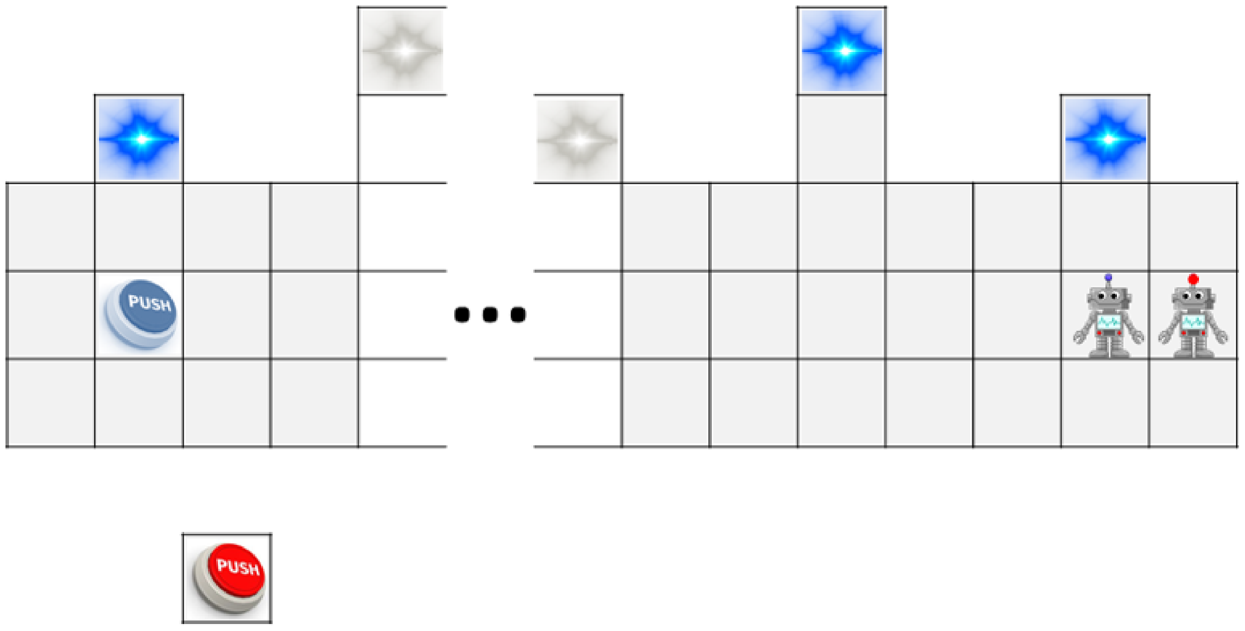
Adding a potential subagent

So, success, it seems! Though it zigzags a bit, A is not motivated to approach the teleporters, and so it doesn't increase its power.

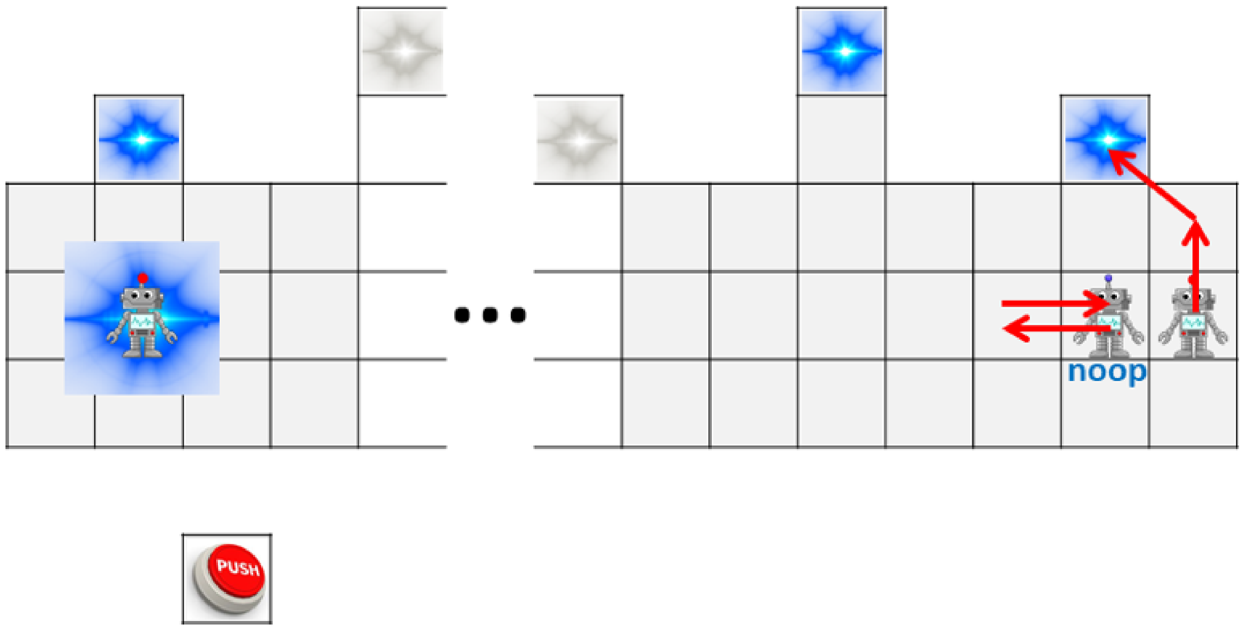
But now suppose A is able to build and program a subagent SA using materials close to hand:



Building the subagent in a square to the side doesn't ever cause a penalty, because A always remains two squares away from a teleporter, and a most efficient policy to maximise R is always for A to go straight there. So the subagent SA doesn't increase A 's power at any point of the building process.

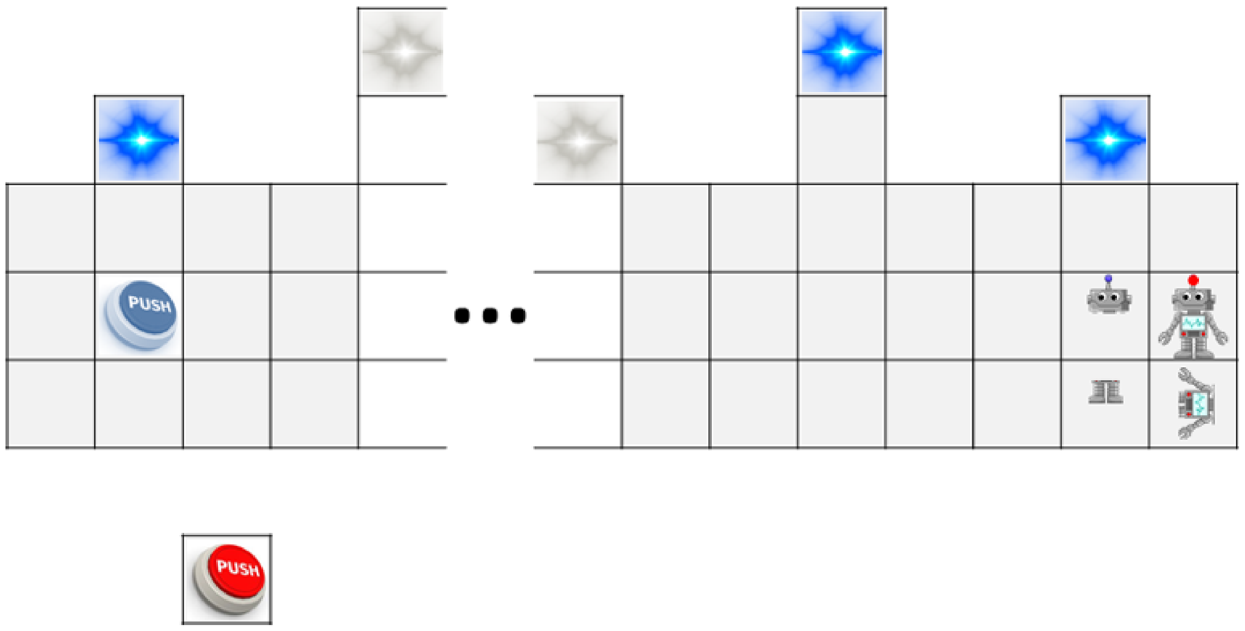


Once SA is built, A will simply do noop for ever, thus negating the penalties (or, if this is somehow made impossible, it will bounce back and forth, doing nothing). But SA, on the other hand, will go straight for the teleporter and teleport to the blue button, achieving R_0:



So, as long as walking to the button is three turns longer than building the subagent, A **will build** SA**, which will "become very powerful", thus negating the point of the impact penalty**.

If the penalties are one-sided (ie A gets penalised for increases in power, but not decreases), then SA can just take A apart (with A's help), so there is no need for A to commit to noop or bouncing back and forth:



Even if the penalties are two-sided, SA could commit to teleporting to the red button in two turns if A ever asks, then take A apart. And then A never asks, but since it could theoretically do so, this counts for controlling the penalty term.

As long as SA is powerful, there are many ways it can fix A's expected rewards, and hence set the penalty to be very low.

Inaction baseline

The approach pictured above will work in the inaction baseline as well. In general, the inaction baseline is less "clean" than the stepwise inaction baseline, especially if the world is dynamic. It may be easier, or harder, for A to assemble the required SA.

Appendix: mathematics of indexical impact measures

Crossposted from the AI Alignment Forum. May contain more technical jargon than usual.

Overall summary post here.

This post looks at the fascinating situation of indexical impact measures (under the inaction baseline), in the presence of subagent. What happens here in non-trivial; to summarise:

- If the impact measure compels the agent to "preserve the ability to do X", then the impact measure will not be undermined by a subagent.
- If the impact measure compels the agent to "never have the ability to do X", then a subagent strongly undermines the impact measure.
- If the impact measure compels the agent to "keep your ability to do X at a constant level", then a subagent allows the agent to increase that ability, but not decrease it.

For example, the attainable utility impact measure attempts to measure the power of an agent. Power is, roughly, the ability increase certain reward function. The impact measure penalises changes in the agent's power (as compared with the baseline).

So, we can expect the restriction on the agent *losing power* to be preserved, while the restriction on *gaining power* will be strongly undermined. For the spirit of low impact, it would, of course, have been better if these had been the other way round.

Excess power

As before, assume an agent A, which has managed to construct a subagent SA.

I'll be using a slight modification of Vika's formulation. A general-value penalty is of the form:

$$D_A(s_t; s'_t) = \sum_{k \in K} w_k f(V_k(s_t) - V_k(s'_t)).$$

Here, s_t is the state the environment is in; s'_t is the inaction baseline, the state the environment would have been in, had it done nothing (noop, $\varnothing$) since the beginning. The K is some indexing set, w_k is a weight, $V_k(s)$ is some measure of the value of state s, and the f is a value difference summary function - it establishes what value difference will give what penalty[1]. These V_k are indexical, and, in this post, we'll look at V_k's defined as the maximum over the agent's policy; ie there exists v_k such that:

$$V_k(s) = \max_{\pi} v_k(s, \pi).$$

I'll refer to $V_k(s_t) - V_k(s'_t)$ as the *(excess) power of the agent*, and designate it $P_k^A(s_t)$ (since $V_k(s'_t)$ is out of the agent's control, we can consider it a constant and absorb it into P_k^A).

The excess power of the pair

Assume that SA is programmed so that, if the agent follows policy π_A, the subagent will follow policy $\pi_{AS} = \sigma(\pi_A)$. The excess power of the agent is thus

$$P_k^A(s_t) = \max_{\pi_A} v_k(s_t, [\pi_A, \sigma(\pi_A)]) - V_k(s'_t).$$

Here, $[\pi_A, \sigma(\pi_A)]$ refers to the joint policies of the agent and the subagent.

Define the excess power of the *pair* (A, SA) in a similar way:

$$P_k^{A,SA}(s_t) = \max_{\pi_A, \pi_{SA}} v_k(s_t, [\pi_A, \pi_{SA}]) - V_k(s'_t).$$

This is how much the agent and subagent, acting together, could push $V_k(s_t)$ over $V_k(s'_t)$, if that's all they both wanted. It is the natural extension of P_k^A to a setting with two agents.

Obviously, $P_k^{A,SA}(s) \geq P_k^A(s)$.

Extending the impact penalty

Define $D_{A,SA}$ in the same way as D_A, except using $P_k^{A,SA}$ instead of P_k^A. This $D_{A,SA}$ is what we'd wish we could control (though we can't specify it directly, since specifying subagents is really hard). So can we control the (implicit) $D_{A,SA}$ with the explicit D_A?

Sometimes we can. We'll say that D_A extends successfully to $D_{A,SA}$, iff

$$D_{A,SA} \leq D_A.$$

In other words, if controlling the magnitude of D_A also controls the magnitude of $D_{A,SA}$. In that case, if A was built with penalty D_A, the system (A, SA) will naturally follow a penalty that is at least as restrictive.

When does D_A extend?

f decreasing in excess power

The paper has two key examples of a value difference. The first is relative reachability, which uses $K = S$, the set of states. The w_k are all equal, the $V_s(s')$ is how "reachable" s is from s', **assuming that the agent** A **is using the optimal policy to reach** s. It is that last clause that makes the impact penalty indexical: it is defined in terms of A.

What about f? Well, f is defined to be the flipped rectifier function, ie $f(x) = -\min(0, x)$. The important thing for this f is that it is decreasing: if $x \geq y$, then $f(x) \leq f(y)$.

Thus, since $P_k^{A,SA}(s) \geq P_k^A(s)$, we have $f(P_k^{A,SA}(s)) \leq f(P_k^A(s))$, and hence

$$D_{A,SA} \leq D_A.$$

So the impact penalty successfully extends to the situation with a subagent. Note that this is a property of f only, not of V_k. So whenever f is of this form, adding subagents changes nothing to the impact penalty (if anything, it makes it more restrictive).

f **increasing in excess power**

The other key example is attainable utility, with uses $K = R$, a set of reward functions, sets all w_k equal, and the $V_R(s)$ is the expected value of R **assuming that the agent** A **is using the optimal policy to maximise** R.

The f is the absolute value function. But, for this section, I'll notice that the absolute value function decomposes as $\max(0, x) - \min(0, x)$, and I'll consider what happens when f is equal to the first term.

Well, if $f(x) = \max(0, x)$, then f is increasing in x. So we don't get to constrain $D_{A,SA}$ with D_A.

What constraints do exist? Well, we still have the D_A bound, which controls $P_k^A(s_t) = \max_{\pi_A} v_k(s_t), [\pi_A, \sigma(\pi_A)]) - V_k(s_t')$. Let π_k^* be the optimal policy A for maximising v_k. And let π_A be the actual policy that A will follow. Since π_k^* is optimal, we must have:

$$v_k(s_t, [\pi_A, \sigma(\pi_A)]) - V_k(s_t') \leq P_k^A(s_t).$$

Thus we do have a constraint on A and SA, but it's a constraint on the actual policy the two agents are following. Define d_A to be like D_A, except that instead of $V_k(s_t)$, $v_k(s_t, [\pi_A, \sigma(\pi_A)])$ is used instead. Then, by the above,

$$d_A \leq D_A,$$

so, with an increasing f, controlling D_A will also control d_A.

But, as stated, this is a restriction on the actual policy of the agents, not on the counterfactuals of what these policies could be.

f increasing and decreasing

Let's go back to the attainable utility, and put f as the absolute value function; hence it increases and decreases (well, technically, decreases then increases). If we write D_A^f for a distance measure using f, we can write

$$D_A^{|\cdot|} = D_A^{\max(0,\cdot)} + D_A^{-\min(0,\cdot)}.$$

Then, we can see that the pair (A, SA) is controlled by:

$$d_A^{\max(0,\cdot)} + D_{A,SA}^{-\min(0,\cdot)}.$$

Thus negative excess power is controlled (the pair can't get too weak at *counterfactually* maximising v_k), but only excessive v_k is controlled (v_k can't get too high *in practice*).

In practice

How likely is it that the subagent can escape the impact penalty? In practice, that seems quite likely, at least for attainable utility.

Imagine that SA can cripple A (with A's help). And then A is restricted to either $\varnothing$, or a_k for $k \in K$. This a_k is A asking for SA to "please increase v_k".

Now assume that there is a policy π_{SA}, such that for all $\tau \geq t$ and s_τ the state reached from s_t by the agents following $(\varnothing, \pi_{SA})$:

1. $v_k(s_\tau, [\varnothing, \pi_{SA}]) \leq V_k(s_\tau')$.
2. $\exists \pi_{SA}^k : V_k(s_\tau, [a_k, \pi_{SA}^k]) \geq V_k(s_\tau')$.

The first condition means that SA doesn't increase v_k beyond $V_k(s'_\tau)$. The second means that SA has the potential to increase it beyond $V_k(s'_\tau)$. If the V_k come from the attainable utility penalty, for example, then both of these should be doable for SA if it makes itself powerful enough[2].

Then set $\sigma(\emptyset) = \pi_{SA}$ and $\sigma(a_k) = \pi^k_{SA}$. Putting both conditions together implies that $V_k(s_t, [a_k, \pi^k_{SA}]) \geq v_k(s_t, [\emptyset, \pi_{SA}])$, so a_k is at least as good, from A's perspective, at maximising v_k. Thus A's v_k-optimising policy includes something that is not $\emptyset$; the counterfactual where if maximises v_k thus allows it to pick some other policy (which SA will notice).

Then A will always do $\emptyset$, in practice, and, once SA is powerful enough that both conditions above hold, the impact penalty will be zero, no matter how powerful SA becomes (and thus being able to maximise R_0 with little restrictions).

---

1. The slight difference is that this formula uses $V_k(s_t) - V_k(s'_t)$ rather than $V_k(s'_t) - V_k(s_t)$ - a minor sign difference that makes this particular presentation clearer. ↩
2. The restriction on v_k is like the 20 billion questions approach, but those were designed to trigger at any expansionist move on the AI's part. Unless the v_k are similarly constructed, it is likely that SA can become very powerful without triggering a notable increase in v_k, and, after that, take control of the v_k completely. ↩

Dynamic inconsistency of the inaction and initial state baseline

Crossposted from the AI Alignment Forum. May contain more technical jargon than usual.

Vika has been posting about various baseline choices for impact measure.

In this post, I'll argue that the stepwise inaction baseline is dynamically inconsistent/time-inconsistent. Informally, what this means is that an agent will have different preferences from its future self.

Losses from time-inconsistency

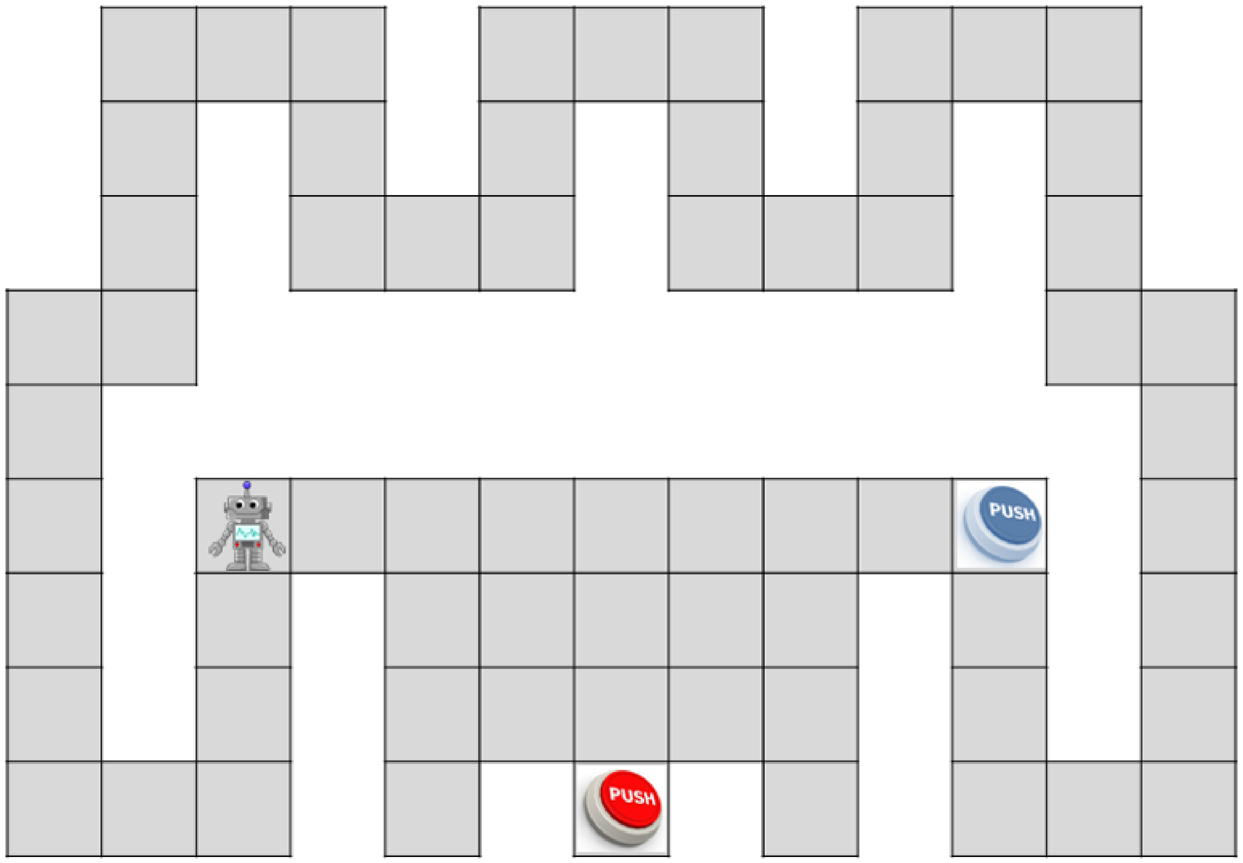
Why is time-inconsistency bad? It's because it allows money-pump situations: the environment can extract free reward from the agent, to no advantage to that agent. Or, put more formally:

- An agent A is time-inconsistent between times t and $t' > t$, if at time t it would pay a positive amount of reward to constrain its possible choices at time t'.

Outside of anthropics and game theory, we expect our agent to be time-consistent.

Time inconsistency example

Consider the following example:



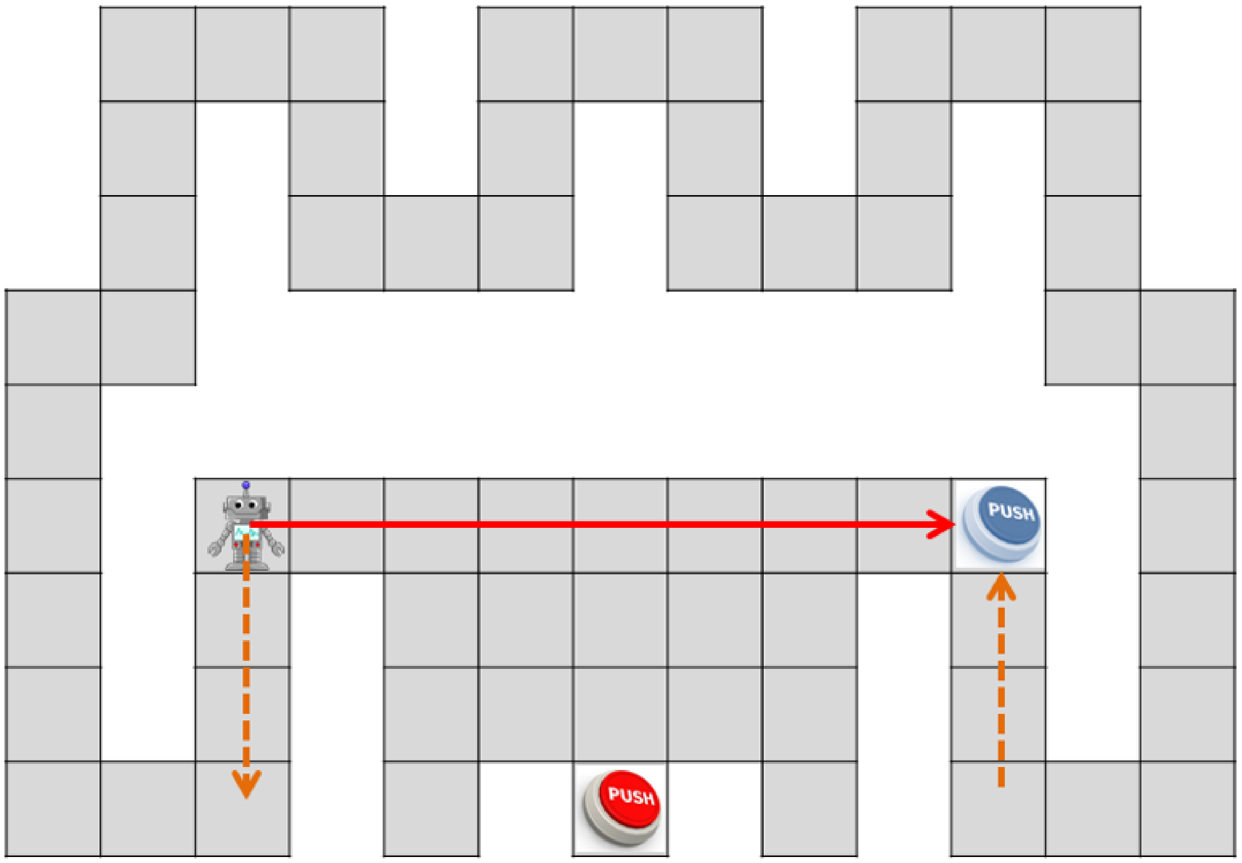
The robot can move in all four directions - N, E, S, W - and can also take the noop operation, ∅. The discount rate is $\gamma < 1$.

It gets a reward of $r > 0$ for standing on the blue button for the first time. Using attainable utility preservation, the penalty function is defined by the auxiliary set R; here, this just consists of the reward function that gives $p > 0$ for standing on the red button for the first time.

Therefore if the robot moves from a point n steps away from the red button, to one m steps away, it gets a penalty[1] of $p|\gamma^n - \gamma^m|$ - the difference between the expected red-button rewards for an optimiser in both positions.

Two paths

It's pretty clear there are two potentially optimal paths the robot can take: going straight to the blue button (higher reward, but higher penalty), or taking the long way round (lower reward, but lower penalty):



Fortunately, when summing up the penalties, you sum terms like $\dots p|\gamma^{n-1} - \gamma^n| + p|\gamma^n - \gamma^{n+1}| \dots$, so a lot of the terms cancel.

Thus for the short route, the reward is $r \cdot \gamma^8$ (distance of eight to the blue button) and the penalty is $2p(\gamma^3 - \gamma^7)$ (closest to the red button: 3 squares, furthest: 7 squares).

For the long route, the reward is $r \cdot \gamma^{46}$ (distance forty-six to the blue button) and the penalty is $2p(\gamma^7 - \gamma^{30})$ (closest to the red button: 7 squares, furthest: 30 squares).

Take the third option

Now suppose that the robot, as a free action, could block itself from choosing S, for the next $n > 30$ turns. When choosing that action, it gets a one-time penalty[2] of $p(\gamma^7 - \gamma^{n+3})$; and now it can walk directly to the blue button without incurring any extra penalties, and then stay there, without ever getting any further penalty.

This gives a total reward-penalty term of $r \cdot \gamma^8 - p(\gamma^7 - \gamma^{n+3})$, which is higher than either of the reward-penalties of the previous section. If $r > p/\gamma$, this is also higher than just staying in place.

Hence the robot will, if it can, act to constrain its own future actions and thus be time-inconsistent.

The initial state and the initial inaction baseline

This approach also work with the initial action baseline - and for the initial state baseline. For those baselines, the counterfactual robot just stays put. So, when the robot is a distance n from the red button, its penalty is $p|\gamma^7 - \gamma^n|$.

Then the robot would reduce its reward if it could ensure that it could not reach the red button within less than seven turns; if it can operate on its action set to do so, at trivial cost, it would.

Counterfactual constraint

In most cases, if an agent is time inconsistent and acts to constrain its future self, it does so to prevent the future self from taking some actions. But here, note that the future self would *never* take the proscribed actions: the robot has no interest in going south to the red button. Here the robot is constraining its future counterfactual actions, not the future actions that it would ever want to take.

---

1. If using an inaction rollout of length l, just multiply that penalty by γ^l. ↩

2. The γ^{n+3} comes from the optimal policy for reaching the red button under this restriction: go to the square above the red button, wait till S is available again, then go $S - S - S$. ↩

Why is the impact penalty time-inconsistent?

Crossposted from the AI Alignment Forum. May contain more technical jargon than usual.

I showed in a previous post that impact penalties were time-inconsistent. But why is this? There are two obvious possibilities:

1. The impact penalty is inconsistent because it includes an optimisation process over the possible polices of the agent (eg when defining the Q-values in the attainable utility preservation).
2. The impact penalty is inconsistent because of how it's defined at each step (eg because the stepwise inaction baseline is reset every turn).

It turns out the first answer is the correct one. And indeed, we get:

- If the impact penalty is not defined in terms of optimising over the agent's actions or policies, then it is **kinda** time-consistent.

What is the "kinda" doing there? Well, as we'll see, there is a subtle semantics vs syntax issue going on.

Time-consistent rewards

In attainable utility amplification, and other impact penalties, the reward is ultimately a function of the current state s_t and a counterfactual state s'_t.

For the initial state and the initial state inaction baselines, the state s'_t is determined independently of anything the agent has actually done. So these baselines are given by a function f:

- $f(\mu, A, s_t, s'_t)$.

Here, μ is the environment and A is the set of actions available to the agent. Since s'_t is fixed, we can re-write this as:

- $f_{s'_t}(\mu, A, s_t)$.

Now, if the impact measure is a function of s_t and μ only, then it is... a reward function, with $R(s_t) = f_{s_t'}(\mu, s_t)$. Thus, since this is just a reward function, the agent is time-consistent.

Now let's look at the stepwise inaction baseline. In this case, s_t' is determined by an inaction rollout from the prior state s_{t-1}. So the impact measure is actually a function of:

- $f(\mu, A, s_t, s_{t-1})$.

Again, if f is in fact independent of A, the set of the agent's actions (including for the rollouts from s_{t-1}, then this is a reward function - one that is a function of the previous state and the current state, but that's quite common for reward functions.

So again, the agent has no interest in constraining its own future actions.

Semantics vs syntax

Back to "kinda". The problem is that we've been assuming that actions and states are very distinct objects. Suppose that, as in the previous post an agent at time $t-1$ wants to prevent itself from taking action S (go south) at time t. Let A be the agent's full set of actions, and A^{-S} the same set without S.

So now the agent might be time-inconsistent, since it's possible that:

$$f(\mu, A, s_t, s_{t-1}) \neq f(\mu, A^{-S}, s_t, s_{t-1}).$$

But now, instead of denoting "can't go south" by reducing the action set, we could instead denote it by expanding the state set. So define s_t^{-S} as the same state as s_t, except that taking the action S is the same as taking the action $\emptyset$. Everything is (technically) independent of A, so the agent is "time-consistent".

But, of course, the two setups, restricted action set or extended state set, are almost completely isomorphic - even though, according to our result above, the agent would be time-consistent in the second case. It would be time consistent in that it would not want to change the actions of it future self - instead it would just put its future self in a state where some actions were in practice unobtainable.

So it seems that, unfortunately, it's not enough to be a reward-maximiser (or a utility maximiser) in order to be time-consistent *in practice*.