Planning with Qualitative Action-Trajectory Constraints in PDDL: Supplementary Material

This supplementary material contains:

- A full proof of soundness and completeness for the PAC-C algorithm, that takes in input a PAC planning problem and returns a classical planning problem (Section 1).
- A detailed description of our benchmarks (action-trajectory constraints and state-trajectory constraints) that we designed for the experimental analysis (Section 2).

For the sake of convenience, here we also include the main definitions and the algorithm given in the paper.

1 Theorem Proof

Definition 1. *Given a plan $\pi = \langle a_1, a_2, \ldots, a_n \rangle$, the following rules define when π satisfies an action constraint:*

π satisfies (`always` ϕ) *iff* $\forall t : 1 \leq t \leq |\pi| \cdot \pi[t] \in \phi$

π satisfies (`sometime` ϕ) *iff* $\exists t : 1 \leq t \leq |\pi| \cdot \pi[t] \in \phi$

π satisfies (`at-most-once` ϕ) *iff* $\forall t_1 : 1 \leq t_1 \leq |\pi| \cdot$ *if* $\pi[t_1] \in \phi$
then $\forall t_2 : t_1 < t_2 \leq |\pi| \cdot \pi[t_2] \notin \phi$

π satisfies (`sometime-after` ϕ ψ) *iff* $\forall t_1 : 1 \leq t_1 \leq |\pi| \cdot$ *if* $\pi[t_1] \in \phi$
then $\exists t_2 : t_1 \leq t_2 \leq |\pi| \cdot \pi[t_2] \in \psi$

π satisfies (`sometime-before` ϕ ψ) *iff* $\forall t_1 : 1 \leq t_1 \leq |\pi| \cdot$ *if* $\pi[t_1] \in \phi$
then $\exists t_2 : 1 \leq t_2 < t_1 \cdot \pi[t_2] \in \psi$

π satisfies (`always-next` ϕ ψ) *iff* $\forall t : 1 \leq t < |\pi| \cdot$ *if* $\pi[t] \in \phi$
then $\pi[t+1] \in \psi$ *and* $\pi[|\pi|] \notin \phi$

π satisfies (`pattern` $\phi_1 \ldots \phi_k$) *iff* $\exists \langle a_1, \ldots, a_k \rangle$ *a sequence of actions from π that are ordered as in π, such that* $\forall i \in \{1, \ldots, k\}$ $a_i \in \phi_i$.

Definition 2. *A classical planning problem with action constraints (a PAC problem) is a tuple $\langle \Pi, C \rangle$ where Π is a classical planning problem and C is a set of action constraints.*

Theorem 1. *Let Π be a PAC problem $\langle \langle F, A, I, G \rangle, C \rangle$ and Π' the planning problem obtained by compiling Π through Algorithm 1. A plan π is a solution for Π iff π' is a solution for Π', where π' is the plan (with the same length of π) obtained by compiling the actions of π through Algorithm 1.*

Notation

- $\pi = \langle a_1, a_2, \ldots, a_n \rangle$, $\pi' = \langle a'_1, a'_2, \ldots, a'_n \rangle$; $\pi[t]$ is the t-th action of plan π.
- $\sigma = \langle s_1, s_2, \ldots, s_{n+1} \rangle$ is the state trajectory induced by π; each state is defined over F.
- $\sigma' = \langle s'_1, s'_2, \ldots, s'_{n+1} \rangle$ is the state trajectory induced by π'; each state is defined over $F \cup$ *PC-atoms* $\cup$ *RC-atoms*.

Proof. We focus on the case in which the plan has at least one action. We prove the two direction sof the implication in the claim separately.
(π solution for $\Pi \implies \pi'$ solution for Π') **By contradiction**, assuming that π is a solution for Π but π' is not a solution for Π'. If π' is not a solution, at least one of the following three cases has to hold:

Algorithm 1: PAC-C

Input : A PAC Problem $\Pi = \langle\langle F, A, I, G\rangle, C\rangle$
Output: A classical planning problem equivalent to Π

1 $F' = F \cup \mathit{PC\text{-}atoms} \cup \mathit{RC\text{-}atoms}$
2 $I' = I \cup \bigcup_{\mathsf{SA}_{\phi,\psi}} \mathit{got}_\psi$
3 $A' = \{a \mid a \in A \text{ and for each } \mathsf{A}_\phi \in C,\ a \in \phi\}$
4 **foreach** $a \in A'$ **do**
5   **foreach** $c \in PC(C)$ **do**
6   **if** $c = \mathsf{AO}_\phi$ **and** $a \in \phi$ **then**
7       $\mathit{Pre}(a) = \mathit{Pre}(a) \wedge \neg \mathit{done}_\phi$
8       $\mathit{Eff}(a) = \mathit{Eff}(a) \cup \{\mathit{done}_\phi\}$
9   **if** $c = \mathsf{SB}_{\phi,\psi}$ **then**
10   **if** $a \in \phi$ **then** $\mathit{Pre}(a) = \mathit{Pre}(a) \wedge \mathit{done}_\psi$
11   **if** $a \in \psi$ **then** $\mathit{Eff}(a) = \mathit{Eff}(a) \cup \{\mathit{done}_\psi\}$
12   **if** $c = \mathsf{AX}_{\phi,\psi}$ **then**
13   **if** $a \in \phi$ **then** $\mathit{Eff}(a) = \mathit{Eff}(a) \cup \{\mathit{request}_\psi\}$
14   **else if** $a \in \psi$ **then** $\mathit{Eff}(a) = \mathit{Eff}(a) \cup \{\neg \mathit{request}_\psi\}$
15   **if** $a \notin \psi$ **then** $\mathit{Pre}(a) = \mathit{Pre}(a) \wedge \{\neg \mathit{request}_\psi\}$
16   **foreach** $c \in RC(C)$ **do**
17   **if** $c = \mathsf{ST}_\phi$ **and** $a \in \phi$ **then** $\mathit{Eff}(a) = \mathit{Eff}(a) \cup \{\mathit{got}_\phi\}$
18   **if** $c = \mathsf{SA}_{\phi,\psi}$ **then**
19   **if** $a \in \psi$ **then** $\mathit{Eff}(a) = \mathit{Eff}(a) \cup \{\mathit{got}_\psi\}$
20   **if** $a \in \phi$ **and** $a \notin \psi$ **then** $\mathit{Eff}(a) = \mathit{Eff}(a) \cup \{\neg \mathit{got}_\psi\}$
21   **if** $c = \mathsf{PA}_{\phi_1 \ldots \phi_k}$ **then**
22   **foreach** $\phi_i \in \langle \phi_1 \ldots \phi_k\rangle \cdot a \in \phi_i$ **do**
23         $\mathit{Eff}(a) = \mathit{Eff}(a) \cup \begin{cases} \{\mathit{stage}_c^{i-1} \triangleright \mathit{stage}_c^{i}\} & \textbf{if } i > 1 \\ \{\mathit{stage}_c^{1}\} & \textbf{otherwise} \end{cases}$
24 $G' = G \wedge \bigwedge_{\mathsf{SA}_{\phi,\psi} \in C} \mathit{got}_\psi \wedge \bigwedge_{\mathsf{ST}_\phi \in C} \mathit{got}_\phi \wedge \bigwedge_{\mathsf{AX}_{\phi,\psi} \in C} \neg \mathit{request}_\psi \wedge \bigwedge_{c = \mathsf{PA}_{\phi_1 \ldots \phi_k} \in C} \mathit{stage}_c^k$
25 **return** $\langle F', A', I', G'\rangle$

(I) $\exists\, t$ such that $\pi'[t] \notin A'$.

(II) $\exists\, t$ such that $s'_t \not\models \mathit{Pre}(\pi'[t])$.

(III) $s'_{n+1} \not\models G'$.

(I) If an action $\pi'[t] \notin A'$ then (line 3 of pseudocode) $\exists$ `always` $\phi \cdot \pi[t] \not\models \phi$. $\pi[t]$ violates an always constraints, therefore π is not a solution for Π (**contradiction**).
(II) The precondition of $\pi'[t]$ is as follows: $\mathit{Pre}(\pi'[t]) = \mathit{Pre}(\pi[t]) \wedge \mathit{Pre}_1 \wedge \ldots \wedge \mathit{Pre}_m$. If $s'_t \not\models \mathit{Pre}(\pi[t])$ then also $s_t \not\models \mathit{Pre}(\pi[t])$ and π would not be solution for Π (**contradiction**). Therefore $\exists\, \mathit{Pre} \in \{\mathit{Pre}_1, \ldots, \mathit{Pre}_m\}$ such that $s'_t \not\models \mathit{Pre}$. Pre is a new precondition added by Algorithm 1 due to the compilation of either an `at-most-once` ϕ, `sometime-before` ϕ ψ or `always-next` ϕ ψ constraint.

- (`at-most-once` ϕ). $\mathit{Pre} = \neg \mathit{done}_\phi$, which implies that $\mathit{done}_\phi \in s'_t$. $\mathit{done}_\phi \in s'_t$ plus $\mathit{done}_\phi \notin I'$ imply the existence of an action $\pi'[j]$ ($j < t$) with done_ϕ as an effect. This also means that $\pi[j] \in \phi$. As $\mathit{Pre} = \neg \mathit{done}_\phi$, $\pi[t] \in \phi$, too; therefore both $\pi[j] \in \phi$ and $\pi[t] \in \phi$ making π violate the `at-most-once` ϕ (**contradiction**).

- (`sometime-before` ϕ ψ). $\mathit{Pre} = \mathit{done}_\psi$, therefore $\mathit{done}_\psi \notin s'_t$. Since no action can delete done_ψ, it follows that there exists no index j with $j < t$ such that $\pi'[j]$ achieves done_ψ, which in turns implies that there is no index j with $j < t$ such that $\pi[j] \in \psi$ either (see line 11 of pseudocode). Since $\mathit{Pre} = \mathit{done}_\psi$, we also know that $\pi[t] \in \phi$. Therefore, we have that plan π has the action $\pi[t] \in \phi$, but before t there is no action that makes ψ in π; hence π violates `sometime-before` ϕ ψ (**contradiction**).

- (`always-next` ϕ ψ). $\mathit{Pre} = \neg \mathit{request}_\psi$, which implies that $\mathit{request}_\psi \in s'_t$. $\mathit{request}_\psi \in s'_t$ implies that $\pi[t-1] \in \phi$ (line 13 of pseudocode). $\mathit{Pre} = \neg \mathit{request}_\psi$, therefore $\pi[t] \notin \psi$. We have that π violates `always-next` ϕ ψ because $\pi[t-1] \in \phi$ and $\pi[t] \notin \psi$ (**contradiction**).

(III) $G' = G \wedge \bigwedge_{\mathsf{SA}_{\phi,\psi} \in C} got_\psi \wedge \bigwedge_{\mathsf{ST}_\phi \in C} got_\phi \wedge \bigwedge_{\mathsf{AX}_{\phi,\psi} \in C} \neg request_\psi \wedge \bigwedge_{c=\mathsf{PA}_{\phi_1 \ldots \phi_k} \in C} stage^k_c$. If $s'_{n+1} \not\models G$ then also $s_{n+1} \not\models G$; thus π is not a solution for Π (**contradiction**). Therefore, the only way $s'_{n+1} \not\models G'$ holds is when one among got_ϕ, got_ψ, $stage^k_c$ does not hold in s'_{n+1} or $request_\psi \in s'_{n+1}$. We proceed case by case:

- (got_ϕ). $got_\phi \notin s'_{n+1}$ implies that there is no action that achieves got_ϕ in π'. Indeed got_ϕ can never be deleted by any action. This means that there is no action satisfying ϕ in π, too. Therefore, π violates `sometime` ϕ (**contradiction**).

- (got_ψ). $got_\psi \in I'$ (by definition) and $got_\psi \notin s'_{n+1}$ (by absurd assumption) imply that there exists an action $\pi'[t_1]$ that deletes got_ψ, and no action after t_1 that adds got_ψ. This means that there exists an index t_1 with $1 \leq t_1 \leq |\pi|$ such that $\pi[t_1] \in \phi$, and $\forall t_2$ with $t_1 \leq t_2 \leq |\pi|$ we have that $\pi[t_2] \notin \psi$. By definition of `sometime-after` ϕ ψ, π violates such a constraint (**contradiction**).

- ($stage^k_c$). $stage^k_c$ is added by the algorithm for a $c =$ `pattern` $\phi_1, \ldots, \phi_k$ constraint. Due to the structure of the compiled problem, atom $stage^k_c$ is true in the last state only if there exists a sequence of actions $\langle a'_1, \ldots, a'_k \rangle$ from π' ordered as in π' such that a'_1 achieves $stage^1_c$, and each a'_i with $i > 1$ has $stage^{i-1}_c \rhd stage^i_c$ as a conditional effect. Since we are assuming $stage^k_c \notin s'_{n+1}$, such a sequence of actions $\langle a'_1, \ldots, a'_k \rangle$ does not exists. From the algorithm (line 23 of pseudocode), it follows that an action a'_1 has $stage^1_c$ as an effect only if $a_1 \in \phi_1$, and an action a'_i has $stage^{i-1}_c \rhd stage^i_c$ as an effect only if $a_i \in \phi_i$. This implies that there is no sequence of actions $\langle a_1, \ldots, a_k \rangle$ from π, ordered as in π, such that $a_i \models \phi_i$ for all $i \in \{1, \ldots, k\}$. Therefore, π violates `pattern` $\phi_1, \ldots, \phi_k$ (**contradiction**).

- ($request_\psi$). $request_\psi \in s'_{n+1}$ implies that there exists an action $\pi'[t]$ that adds $request_\psi$ and each other subsequent action does not delete $request_\psi$. Since $\pi'[t]$ has $request_\psi$ as an effect, and hence $\pi[t] \in \phi$ too, π violates `always-next` ϕ ψ (**contradiction**) since, by construction, there will be no action a after $\pi[t]$ such that $a \in \psi$ and $a \notin \phi$.

(π' solution for Π' $\implies$ π solution for Π) **By contradiction**, assuming π' is a solution for Π' but π is not a solution for Π. Algorithm 1 changes the initial state, goals, preconditions and effects of actions. We can have that π is not a solution for Π for the following reasons:
(I) $\exists\, t$ such that $s_t \not\models Pre(\pi[t])$. **This cannot be the case**: $Pre(\pi[t]') = Pre(\pi[t]) \wedge Pre_1 \wedge \ldots \wedge Pre_m$. $s'_t \models Pre(\pi[t]')$ subsumes that $s_t \models Pre(\pi[t])$.
(II) $s_{n+1} \not\models G$. **This cannot be the case** because $s'_{n+1} \models G'$ subsumes that $s_{n+1} \models G$.
(III) $\exists\, c \in C$ such that π does not satisfy c:

- ($c =$ `always` ϕ). π that does not satisfy `always` ϕ implies that $\exists\, t$ such that $\pi[t] \notin \phi$. It follows that action $\pi'[t] \notin A'$, line 3 in Algorithm 1 (**contradiction**).

- ($c =$ `at-most-once` ϕ). π that does not satisfy `at-most-once` ϕ implies that there exist indexes t_1 and t_2 with $1 \leq t_1 < t_2 \leq |\pi|$ such that $\pi[t_1], \pi[t_2] \in \phi$. By construction, $\pi'[t_1]$ has $done_\phi$ as an effect, and $\pi'[t_2]$ will not be applicable since $\neg done_\phi$ is a precondition of $\pi'[t_2]$, and there is no action that can ever make $done_\phi$ false (**contradiction**).

- ($c =$ `sometime-before` ϕ ψ). π does not satisfy `sometime-before` ϕ ψ implies that there exists an index t_1 with $1 \leq t_1 \leq |\pi|$ such that $\pi[t_1] \in \phi$ and for every index t_2 with $1 \leq t_2 < t_1$ $\pi[t_2] \notin \psi$. Note that $done_\psi$ is a precondition of $\pi'[t_1]$ by construction, and that no action executed before t_1 has $done_\psi$ as an effect (lines 10 and 11 of pseudocode). Since $done_\psi \notin I'$, we have that $done_\psi \notin s'_{t_1}$ and $\pi'[t_1]$ cannot be executed in π'(**contradiction**).

- ($c =$ `sometime` ϕ). π that does not satisfy `sometime` ϕ implies that there is no action $a = \pi[t]$ such that $a \in \phi$. By construction, no action with got_ϕ as an effect is executed in π'. Since got_ϕ is false in the initial state I', got_ϕ will be false in the last state reached by π'. Therefore, π' is not a solution as G' requires got_ϕ true (**contradiction**).

- ($c =$ `sometime-after` ϕ ψ). π that does not satisfy `sometime-after` ϕ ψ implies that there exists an index t_1 with $1 \leq t_1 \leq |\pi|$ such that $\pi[t_1] \in \phi$ and for every index t_2 with $t_1 \leq t_2 \leq |\pi|$ we have that $\pi[t_2] \notin \psi$. This implies that $\pi'[t_1]$ deletes got_ψ (line 20 of pseudocode) and that no action executed after t_1 adds got_ψ. Since got_ψ is a conjunct of G', π' does not achieve the goal (**contradiction**).

- ($c =$ `always-next` ϕ ψ). π does not satisfy `always-next` ϕ ψ implies that either the last action of π (a_n) makes ϕ true or there are two subsequent actions $\pi[t]$ and $\pi[t+1]$ such that $\pi[t] \in \phi$ and $\pi[t+1] \notin \psi$. In the first case, by construction, we have that a'_n adds $request_\psi$. Since $request_\psi$ is a conjunct of G', π' does not achieve the goal (**contradiction**). In the other case, we have that $\pi'[t]$ achieves $request_\psi$, and that $\pi'[t+1]$ has $\neg request_\psi$ as a precondition (lines 13 and 15 of pseudocode). Therefore $\pi'[t+1]$ is not applicable (**contradiction**).

- ($c =$ `pattern` $\phi_1, \ldots, \phi_k$). π does not satisfy `pattern` $\phi_1, \ldots, \phi_k$ implies that it does not exist a sequence $\langle a_1, \ldots, a_k \rangle$ of actions from π, ordered as in π, such that for every $i \in \{1, \ldots, k\}$ $a_i \in \phi_i$. From the algorithm (line 23 of pseudocode) it follows that an action a'_1 has $stage_c^1$ as an effect only if $a_1 \in \phi_1$, and that an action a'_i has $stage_c^{i-1} \triangleright stage_c^i$ as an effect only if $a_i \in \phi_i$. This implies that it does not exists a sequence of actions $\langle a'_1, \ldots, a'_k \rangle$ from π', ordered as in π', such that a'_1 has $stage_c^1$ as an effect and every other action a'_i has $stage_c^{i-1} \triangleright stage_c^i$ as an effect. Therefore, by construction, $stage_c^k$ will not hold in the last state reached by π' (**contradiction**).

$\square$

Section 2 of Supplementary Material starts at the next page.

2 Benchmarks Table

Domain	Natural Language	Action Trajectory Constraints	State Trajectory Constraints (PDDL3 and LTL$_f$)
Rover	Rovers must communicate data only after *all* the needed data was gathered	(sometime-before ϕ (exists (?r - rover ?s - store) (sample_soil ?r ?s waypoint2))) (sometime-before ϕ (exists (?r - rover ?s - store) (sample_rock ?r ?s waypoint3))) (sometime-before ϕ (exists (?r - rover ?p - waypoint ?i - camera) (take_image ?r ?p objective1 ?i high_res))) ϕ = (or (exists (?r - rover ?x - waypoint) (communicate_soil_data ?r general waypoint2 ?x waypoint0)) (exists (?r - rover ?x - waypoint) (communicate_rock_data ?r general waypoint3 ?x waypoint0)) (exists (?r - rover ?x - waypoint) (communicate_image_data ?r general objective1 high_res ?x waypoint0)))	PDDL3: (sometime-before ϕ (exists (?r - rover) (have_soil_analysis ?r waypoint2))) (sometime-before ϕ (exists (?r - rover) (have_rock_analysis ?r waypoint3))) (sometime-before ϕ (exists (?r - rover) (have_image ?r objective1 high_res))) ϕ = (or (communicated_soil_data waypoint2) (communicated_rock_data waypoint3) (communicated_image_data objective1 high_res)) LTL$_f$: (sometime-before ϕ ψ) is $(\neg\phi \wedge \psi)\,\mathcal{R}(\neg\phi)$
	The order between communications is fixed	(sometime-before (exists (?r - rover ?x - waypoint) (communicate_soil_data ?r general waypoint2 ?x waypoint0)) (exists (?r - rover ?x - waypoint) (communicate_rock_data ?r general waypoint3 ?x waypoint0))) (sometime-before (exists (?r - rover ?x - waypoint) (communicate_rock_data ?r general waypoint3 ?x waypoint0)) (exists (?r - rover ?x - waypoint) (communicate_image_data ?r general objective1 high_res ?x waypoint0)))	PDDL3: (sometime-before (communicated_soil_data waypoint2) (communicated_rock_data waypoint3)) (sometime-before (communicated_rock_data waypoint3) (communicated_image_data objective1 high_res)) LTL$_f$: (sometime-before ϕ ψ) is $(\neg\phi \wedge \psi)\,\mathcal{R}(\neg\phi)$
	Non relevant action deletion	(always (and (forall (?r - rover ?x - waypoint) (not (communicate_image_data ?r general objective0 high_res ?x waypoint0))) (forall (?r - rover ?s - store) (not (sample_rock ?r ?s waypoint2))) (forall (?r - rover ?s - store) (not (sample_soil ?r ?s waypoint0))) (forall (?r - rover ?p - waypoint ?i - camera) (not (take_image ?r ?p objective0 ?i colour))) (forall (?r - rover ?x - waypoint) (not (communicate_soil_data ?r general waypoint0 ?x waypoint0))) (forall (?r - rover ?x - waypoint) (not (communicate_image_data ?r general objective0 colour ?x waypoint0))) (forall (?r - rover ?x - waypoint) (not (communicate_soil_data ?r general waypoint3 ?x waypoint0))) (forall (?r - rover ?x - waypoint) (not (communicate_image_data ?r general objective1 colour ?x waypoint0))) (forall (?r - rover ?p - waypoint ?i - camera) (not (take_image ?r ?p objective0 ?i high_res))) (forall (?r - rover ?s - store) (not (sample_rock ?r ?s waypoint1))) (forall (?r - rover ?x - waypoint) (not (communicate_rock_data ?r general waypoint1 ?x waypoint0))) (forall (?r - rover ?p - waypoint ?i - camera) (not (take_image ?r ?p objective1 ?i colour))) (forall (?r - rover ?s - store) (not (sample_soil ?r ?s waypoint3))) (forall (?r - rover ?x - waypoint) (not (communicate_rock_data ?r general waypoint2 ?x waypoint0)))))	PDDL3: (always (not (communicated_image_data objective1 colour))) (always (forall (?r - rover) (not (have_soil_analysis ?r waypoint0)))) (always (forall (?r - rover) (not (have_image ?r objective0 high_res)))) (always (not (communicated_soil_data waypoint0))) (always (forall (?r - rover) (not (have_image ?r objective0 colour)))) (always (not (communicated_soil_data waypoint3))) (always (forall (?r - rover) (not (have_image ?r objective1 colour)))) (always (not (communicated_rock_data waypoint2))) (always (not (communicated_rock_data waypoint1))) (always (not (communicated_image_data objective0 colour))) (always (forall (?r - rover) (not (have_rock_analysis ?r waypoint2)))) (always (not (communicated_image_data objective0 high_res))) (always (forall (?r - rover) (not (have_soil_analysis ?r waypoint3)))) (always (forall (?r - rover) (not (have_rock_analysis ?r waypoint1)))) LTL$_f$: (always ϕ) is $\Box(\phi)$
Openstack *	If a new stack is opened, then a new order must immediately start *(not expressible in* PDDL3*)*	(always-next (exists (?open ?new-open - count) (open-new-stack ?open ?new-open)) (exists (?o - order ?avail ?new-avail - count) (start-order ?o ?avail ?new-avail)))	$\Box(\neg($(stacks-avail-n0) $\wedge \bigcirc$(stacks-avail-n1) $\wedge\ \phi) \vee \bigcirc\bigcirc$(stacks-avail-n0)$)$ $\Box(\neg($(stacks-avail-n1) $\wedge \bigcirc$(stacks-avail-n2) $\wedge\ \phi) \vee \bigcirc\bigcirc$(stacks-avail-n1)$)$ $\Box(\neg($(stacks-avail-n2) $\wedge \bigcirc$(stacks-avail-n3) $\wedge\ \phi) \vee \bigcirc\bigcirc$(stacks-avail-n2)$)$ $\Box(\neg($(stacks-avail-n3) $\wedge \bigcirc$(stacks-avail-n4) $\wedge\ \phi) \vee \bigcirc\bigcirc$(stacks-avail-n3)$)$ $\Box(\neg($(stacks-avail-n4) $\wedge \bigcirc$(stacks-avail-n5) $\wedge\ \phi) \vee \bigcirc\bigcirc$(stacks-avail-n4)$)$ with ϕ: ((shipped-o1) $\vee \bigcirc\neg$(shipped-o1)) $\wedge$ ((shipped-o2) $\vee \bigcirc\neg$(shipped-o2)) $\wedge$ ((shipped-o3) $\vee \bigcirc\neg$(shipped-o3)) $\wedge$ ((shipped-o4) $\vee \bigcirc\neg$(shipped-o4)) $\wedge$ ((shipped-o5) $\vee \bigcirc\neg$(shipped-o5))
	The setup of the machine is immediately followed by the production of the product *(not expressible in* PDDL3*)*	(always-next (exists (?p - product ?avail - count) (setup-machine ?p ?avail)) (exists (?p - product ?avail - count) (make-product ?p ?avail)))	$\Box(\neg$(machine-configured-p1) $\vee \bigcirc$(made-p1)$)$ $\Box(\neg$(machine-configured-p2) $\vee \bigcirc$(made-p2)$)$ $\Box(\neg$(machine-configured-p3) $\vee \bigcirc$(made-p3)$)$ $\Box(\neg$(machine-configured-p4) $\vee \bigcirc$(made-p4)$)$ $\Box(\neg$(machine-configured-p5) $\vee \bigcirc$(made-p5)$)$
Trucks	A package can be loaded at most one time	(forall (?p - package) (at-most-once (exists (?a1 - truckarea ?l - location) (load ?p truck1 ?a1 ?l))))	PDDL3: (forall (?p - package) (at-most-once (exists (?a1 - truckarea) (in ?p truck1 ?a1)))) LTL$_f$: (at-most-once ϕ) is $\Box(\phi \Rightarrow (\phi\ \mathcal{U}\ (\Box(\neg\phi) \vee \texttt{final})))$

`Storage`	A crate can be lifted at most one time and crates must be positioned following a given pattern	`(forall (?c - crate)`
		`(at-most-once (exists (?h - hoist ?a1 - storearea ?a2 - area ?p - place)`
		`(lift ?h ?c ?a1 ?a2 ?p))))`
		
		`(pattern (exists (?h - hoist ?a2 - area) (drop ?h crate0 depot0-1-2 ?a2 depot0))`
		`(exists (?h - hoist ?a2 - area) (drop ?h crate1 depot0-2-2 ?a2 depot0)))`
			PDDL3:
			`(forall (?c - crate) (at-most-once (exists (?h - hoist) (lifting ?h ?c))))`
			
			The pattern of crate positioning is specified in PDDL3 by using a combination of sometime-before and sometime constraints (see below). This combination of constraints is equivalent to the pattern of actions only when used in conjunction with the at-most-once constraint.
			
			`(sometime-before (on crate1 depot0-2-2) (on crate0 depot0-1-2))`
			`(sometime (on crate0 depot0-1-2))`
			`(sometime (on crate1 depot0-2-2))`
			
			LTL_f:
			(at-most-once ϕ) is $\Box(\phi \Rightarrow (\phi\ \mathcal{U}\ (\Box(\neg\phi) \vee \texttt{final})))$
			
			◇(¬(on crate0 depot0-1-2) ∧ ○(on crate0 depot0-1-2) ∧
			○(◇(¬(on crate1 depot0-2-2) ∧ ○(on crate1 depot0-2-2))))

`TPP` *	The planner can only move truck1	`(always (and`
		`(forall (?from ?to - place) (not (drive truck2 ?from ?to)))`
		`(forall (?from ?to - place) (not (drive truck3 ?from ?to)))))`
			□(at truck2 depot1)
			□(at truck3 depot1)

	Some road are disabled for truck1 *(not expressible in* PDDL3*)*	`(always (and`
		`(not (drive truck1 market2 depot1))`
		`(not (drive truck1 depot1 market2))))`
			□¬((at truck1 market2) ∧ ○(at truck1 depot1))
			□¬((at truck1 depot1) ∧ ○(at truck1 market2))

	A pattern of drive actions *(not expressible in* PDDL3*)*	`(pattern (drive truck1 depot2 market2)`
		`(drive truck1 market2 market3)`
		`(drive truck1 market3 market1)`
		`(drive truck1 market1 market3)`
		`(drive truck1 market3 market2)`
		`(drive truck1 market2 depot2))`
			◇((at truck1 depot2) ∧ ○(at truck1 market2) ∧
			○(◇((at truck1 market2) ∧ ○(at truck1 market3) ∧
			○(◇((at truck1 market3) ∧ ○(at truck1 market1) ∧
			○(◇((at truck1 market1) ∧ ○(at truck1 market3) ∧
			○(◇((at truck1 market3) ∧ ○(at truck1 market2) ∧
			○(◇((at truck1 market2) ∧ ○(at truck1 depot2)))))))))))

	Sometime a good must be loaded at the requested level (if goal level > 1)	`(sometime (exists (?m - market) (load goods1 truck1 ?m level0 level1 level2 level3)))`
		`(sometime (exists (?m - market) (load goods2 truck1 ?m level0 level1 level2 level3)))`
		`(sometime (exists (?m - market) (load goods3 truck1 ?m level0 level1 level2 level3)))`
		`(sometime (exists (?m - market) (load goods4 truck1 ?m level0 level1 level2 level3)))`
		`(sometime (exists (?m - market) (load goods5 truck1 ?m level0 level1 level1 level2)))`
			◇(loaded goods1 truck1 level3)
			◇(loaded goods2 truck1 level3)
			◇(loaded goods3 truck1 level3)
			◇(loaded goods4 truck1 level3)
			◇(loaded goods5 truck1 level2)

	A buy is immediately followed by a load *(not expressible in* PDDL3*)*	`(always-next`
		`(exists (?g - goods ?m - market ?l1 ?l2 ?l3 ?l4 - level)`
		`(buy truck1 ?g ?m ?l1 ?l2 ?l3 ?l4))`
		
		`(exists (?g - goods ?m - market ?l1 ?l2 ?l3 ?l4 - level)`
		`(load ?g truck1 ?m ?l1 ?l2 ?l3 ?l4)))`
			□((forall (?m - market ?g - goods) (ready-to-load ?g ?m level0)) ∨
			○(forall (?m - market ?g - goods) (ready-to-load ?g ?m level0)))

For domains with (*) there are one or more constraints that cannot be formulated in PDDL3

This table gives a brief summary of the constraints formulated and the respective translations as action constraints and state trajectory constraints (PDDL3 and LTL_f). In our benchmarks, we also slightly rewrite/simplify the PDDL code to overcome some limitation of LTL-C. More precisely:

- The exist quantifiers are converted into negated universal quantifiers. I.e.: $\exists x\ P(x)$ becomes $\neg(\forall x\ \neg P(x))$.
- Double negation simplification.
- As LTL-C does not support universal quantifiers in the precondition, we proceed as following: for `Trucks`, we simplify the quantifier into an explicit big and operation; for `Openstack` we directly use a fully ground representation of each problem.